Fachhochschule Wiesbaden
Fachbereich Design Informatik Medien
Studiengang Allgemeine Informatik

Diplomarbeit
zur Erlangung des akademischen Grades
Diplom-Informatiker/in (FH)

Implementierung eines webbasierten EPCIS-Clients mit EPCglobal-konformen Erweiterungen

vorgelegt von Johannes Thumser
am 21. November 2006

Referent: Prof. Dr. Werntges
Korreferent: Prof. Dr. Panitz

Inhaltsverzeichnis

Erklärung

Erklärung gemäß Prüfungsordnung – Teil A – §6.4.2

Ich versichere, dass ich die Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Ort, Datum Unterschrift Diplomandin/Diplomand

Hiermit erkläre ich mein Einverständnis mit den im folgenden aufgeführten Verbreitungsformen dieser Diplomarbeit:

Verbreitungsform	**Ja**	**Nein**
Einstellung der Arbeit in die Bibliothek der FHW		
Veröffentlichung des Titels der Arbeit im Internet		
Veröffentlichung der Arbeit im Internet		

Ort, Datum Unterschrift Diplomandin/Diplomand

1. Einleitung

Einführung in das Thema der Arbeit

Mit diesem einführenden Kapitel soll diese Diplomarbeit einen ersten, kurzen Überblick über das zugrunde liegende Thema, die Idee für ein begleitendes Projekt und natürlich über den strukturellen Aufbau erhalten.

Immer öfter fällt im Zusammenhang mit Technologien für eine eindeutige Identifikation von Objekten ein spezieller Begriff: *RFID*[1]. Diese Technologie ermöglicht ein berührungsloses Auslesen von Daten und wird bereits für unterschiedliche Zwecke eingesetzt.

Diese Diplomarbeit wird sich auf der einen Seite mit der Theorie der sich aus dieser Technologie ergebenden Möglichkeiten beschäftigen, auf der anderen Seite aber auch versuchen, ein Konzept für ein begeleitendes Projekt zu entwickeln und eine entsprechende Implementierung anzufertigen.

1.1. Thematischer Hintergrund

Motivation für die Wahl von EPCIS als Thema einer Diplomarbeit

Im Zusammenhang mit der Technologie des berührungslosen Lesens wird sehr häufig ein weiterer Begriff genannt: *Elektronischer Produktcode (EPC)*. Dieser Code ist in der Lage, jedes Objekt weltweit durch eine eindeutige Nummer zu beschreiben. Da er ohne größeren Aufwand mit Hilfe der RFID-Technik gelesen werden kann (also ohne direkten Sichtkontakt), ist die Idee entstanden, an bestimmten Stellen der globalen Versorgungsketten einzelne Daten von Produkten zu erfassen und zu speichern.

[1] Abkürzung für *Radio Frequency Identification*, eine eindeutige Erkennung von Objekten mit Hilfe von Funktechnik.

Diese Daten sollen dann – auch unternehmensübergreifend – über eine eigene Netzwerk-Infrastruktur abgefragt werden können, so dass in Echtzeit immer Informationen zum Produkt selbst, aber auch seinem Status verfügbar sind. Eine für die Realisierung notwendige, weite Verbreitung soll dabei durch eine offene Standardisierung der dazugehörigen Systeme geschaffen werden.

Da dieser Standardisierungsprozess stufenweise erfolgt, können Teile der Technologien bereits heute eingesetzt werden. Gerade für die Identifikation von Paletten werden EPCs bereits genutzt. Die Informationsdienste des Netzwerkes, die künftig die entsprechenden Daten liefern sollen, befinden sich allerdings erst in einer fortgeschrittenen Entwicklungphase: Zum Zeitpunkt dieser Arbeit ist die eigentliche Standardisierung noch nicht abgeschlossen, trotzdem liegt eine Spezifikation in einer späten Working-Draft-Version vor.

Gerade die Tatsache, dass durch den bisher fehlenden Standard kaum Implementierungsversuche im Informationsbereich des EPC-Umfeldes unternommen worden sind, macht es besonders reizvoll, einen eigenen Versuch zu unternehmen, mit Hilfe eines Programmes die Funktionsweisen eines solchen Informationssystemes demonstrieren zu können.

1.2. Idee für ein begleitendes Projekt

Implementierung eines webbasierten EPCIS-Clients

Für diesen Zweck soll im Rahmen dieser Arbeit eine Client-Applikation entwickelt werden, mit deren Hilfe Abfragen an die Informationsdienste gestellt werden können. Die Ergebnisse, welche der Server entsprechend zurückliefert, sollen aufbereitet und interpretiert an passender Stelle wieder angezeigt werden.

Das Problem für eine solche Implementierung stellt die noch fehlende Netzwerk-Infrastruktur dar. Neben dem eigentlichen EPC-Netzwerk sind auch die für Testzwecke notwendigen Informationsdienste noch nicht verfügbar. Ledig-

lich die (noch nicht öffentliche) EPCIS-Spezifikation[2], welche unter anderem die Schnittstellen und die dazugehörigen Typen definiert, kann für eine Realisierung herangezogen werden.

Aus diesem Grund muss das Projekt dieser Diplomarbeit um weitere Komponenten ergänzt werden. Daraus ergeben sich folgende Aufgaben, die es im Laufe dieser Arbeit zu lösen gilt:

1. Entwicklung eines Konzeptes zur Simulation des EPC-Netzwerkes.

2. Entwicklung und Implementierung eines Informationsservers, der über die durch die EPCIS-Spezifikation definierte Fassade verfügt und in der Lage ist, repräsentative Testdaten auszuliefern.

3. Entwicklung und Implementierung einer webbasierten Client-Anwendung, über die sich spezielle Abfragen für die Informationsdienste mittels eines HTML-Frontends formulieren lassen, die dann an den zuvor erstellten Testserver geschickt werden können.

4. Entwicklung und Integration eines Sicherheitskonzeptes für einen geschützten Datenaustausch.

Dieser beschriebene EPCIS-Client könnte durchaus auch für zukünftige Projekte von Nutzen sein. Als eigenständige Applikation sind unterschiedliche Anwendungsfälle denkbar. Mögliche Beispiele dafür sind:

- Der Einsatz für beispielhafte Abfragen zu Demonstrationszwecken, unter anderem während Präsentationen.

- Nutzung für Abfragetests bei realen Informationssystemen.

- Unterstützung beim Verständnis des komplexen Aufbaus der unterschiedlichen Abfragetypen.

[2]EPCIS: *EPC Information Services*, Informationsdienste innerhalb des EPC-Netzwerkes.

Der webbasierte Ansatz soll eine möglichst schnelle, übersichtliche Entwicklung und Implementierung der Applikation ermöglichen. Hinzu kommt, dass für den eigentlichen Datenaustausch zwischen den Systemen ohnehin das Internet (oft in Verbindung mit SOAP[3]) vorgesehen ist, wodurch ein webbasierter Ansatz für die Entwicklung der Client-Applikation natürlich weitere Vorteile bringt.

1.3. Inhaltlicher Aufbau der Arbeit

Übersicht über die einzelnen Kapitel

Für eine Umsetzung dieser groben Planung wird die vorliegende Arbeit in unterschiedliche Teile gegliedert. Dieser Abschnitt soll eine Übersicht über die Struktur geben, die sich aus folgenden Punkten zusammensetzt:

- *Theoretischer Teil:* Dieser Teil der Arbeit soll das theoretische Hintergrundwissen schaffen, das zum Verständnis der Technologien nötig ist. Da sich diese Theorie auf zwei logische Bereiche verteilt, soll auch dieses Kapitel in zwei einzelne Teile gegliedert werden:
  - Zunächst eine grundlegende Einführung in das Themengebiet der RFID-Technologie, dem elektronischen Produktcode und dem Aufbau des dazugehörigen Netzwerkes.
  - Der zweite Teil soll einen Überblick über die Funktionsweise und die technischen Grundlagen des Informationsdienstes EPCIS verschaffen. Diese bilden die Basis des zu implementierenden Systems.
- *Konzeptioneller Teil*: Innerhalb dieses Abschnittes der Arbeit soll die eigentliche Entwicklung des zu erstellenden Systems gemacht werden. Dafür wird ein logischer Aufbau für eine Testumgebung und der erforderlichen Komponenten erstellt. Außerdem soll versucht werden, Lösungen für die oben genannten Fragestellungen zu finden.

[3]Kommunikationsprotokoll, das beim Gebrauch von Web Services eingesetzt wird.

- *Implementierungsteil*: Dieser Teil beschäftigt sich mit der Umsetzung des zuvor entwickelten Konzeptes. Dazu gehört auch eine Beschreibung der technischen Hilfsmittel, die für die Implementierung eingesetzt werden.

- *Auswertungsteil*: Innerhalb dieses Teils sollen die Ergebnisse der Implementierung im Hinblick auf die gestellten Anforderungen genauer betrachtet werden. Ergänzt wird dies um eine kritische Auseinandersetzung mit dem gegebenen Umfeld.

2. RFID, EPC und das EPCglobal-Netzwerk

Eine Einführung

Das erste theoretische Kapitel dieser Arbeit soll eine grundlegende Einführung in die Gebiete *RFID* und *EPC* darstellen. Zusätzlich soll es einen Überblick über den allgemeinen Aufbau, den Einsatz des EPCglobal-Netzwerkes und dessen Komponenten geben.

Am 26. Juni 1974 wurde in Ohio, USA, zum ersten Mal der Strichcode einer Kaugummipackung durch den Kassen-Scanner eines Supermarktes gelesen [1]. In den darauf folgenden Jahren revolutionierte der wahrscheinlich jedem bekannte Barcode die standardisierte Identifikation von Objekten und ist heute kaum noch wegzudenken: Identifikationssysteme wie dieser Strichcode sind für Industrie und Handel unverzichtbar geworden.

Die Möglichkeiten dieses weit verbreiteten Codes sind heute aber in vielen Fällen nicht mehr ganz ausreichend. In Zeiten, in denen eine unternehmensübergreifende Kommunikation zwischen verschiedenen Handelspartnern immer wichtiger wird und auch gesetzlich bedingt die Rückverfolgbarkeit von Produkten aus bestimmten Sparten – gerade im Lebensmittelbereich – gewährleistet sein muss, stößt dieser Code an seine Grenzen: Er ist zwar durch seine hohe Verbreitung äußerst günstig herzustellen und zu pflegen, allerdings hat er nur eine geringe Speicherfähigkeit und seine Daten können neuen Gegebenheiten nur schwer angepasst werden. Hinzu kommt, dass der enthaltene Code nur die Produktart des Objektes beschreibt, für die oben genannte Rückverfolgung hingegen wäre ein eindeutiger Code für jedes einzelne Produkt vorteilhaft.

[1] *Strichcode*, aus Wikipedia, der freien Enzyklopädie. http://de.wikipedia.org/wiki/Strichcode. Aktualisierungsdatum: 28. September 2006.

Wünschenswert wäre also ein Code, der in der Lage ist, jedes Produkt weltweit eindeutig zu identifizieren und der sich auf möglichst einfache Weise auslesen lässt. Somit hätte man die Möglichkeit, an sehr vielen verschiedenen Punkten der globalen Versorgungsketten diese Codes zu scannen und zu speichern. Mit dem zusätzlichen Aufbau einer gemeinsamen Netzwerk-Infrastruktur könnten dann alle beteiligten Handelspartner diese Daten abrufen und somit auf diesem Weg Informationen über das Produkt selbst, seinen Status und seine Historie erhalten.

2.1. RFID und EPC in den globalen Versorgungsketten

Standardisierung der Radiofrequenztechnologie für Identifikationszwecke

Die Realisierung dieser Ideen wurde durch die Standardisierung der RFID-Technologie in Verbindung mit dem elektronischen Produktcode (EPC) bereits begonnen. Dieses automatische Identifikationsverfahren ermöglicht es, Daten berührungslos und ohne Sichtkontakt zu übertragen, und ist somit eine äußerst effiziente Lösung für die Erfassung von Objekten innerhalb der Versorgungskette.

In Verbindung mit dem Elektronischen Produktcode bildet diese Schlüsseltechnologie die Grundlage zu der oben beschriebenen, vernetzten Infrastruktur.

2.1.1. Radio Frequency Identification (RFID)

Die bereits erwähnte RFID-Technologie zählt zu den automatischen Identifikationsverfahren: Dabei werden die Daten auf einem Mikrochip gespeichert, der diese dann selbständig an seine Umwelt versendet. Ein großer Vorteil hierbei ist, dass für das Auslesen des Chips mittels dieses Weges weder Sichtkontakt noch Berührung notwendig ist. Somit können die technisch optimalen Bedingungen einer Speicherung von Daten auf einem Chip (ähnlich einer Telefon- oder Bankkarte) genutzt werden, ohne auf ein umständliches, mechanisches

Ausleseferfahren angewiesen zu sein. Gerade in der Masse – man stelle sich zum Beispiel einen vollen Einkaufswagen im Supermarkt vor – wäre ein solches mechanisches Leseverfahren äußerst uneffektiv.

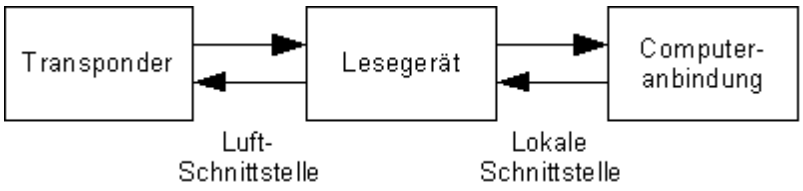
Ein RFID-System setzt sich normalerweise immer aus drei Komponenten zusammen:

- Dem *Transponder*, der auch oft als RFID-Tag bezeichnet wird. Er besteht aus einem Mikrochip, auf dem der EPC gespeichert ist und der zusätzlich mit einer Sende- und Empfangseinheit versehen wird, so dass er über eine Antenne Daten an seine Umgebung abgeben kann.

- Einem *Lesegerät*, welches die Daten empfängt, die vom Transponder gesendet wurden. Je nach Ausführung kann es auch zum Beschreiben des Tags verwendet werden.

- Und zu guter Letzt enthält es normalerweise noch eine *Computer-Anbindung*, um die durch das Lesegerät empfangenen Daten an andere Systeme weiterzuleiten. Diese Anbindung kann unterschiedlichster Art sein (zum Beispiel eine einfache Kasse im Supermarkt, ein Waren-Wirtschaftssystem, aber auch eine komplexe Server-Infrastruktur).

Abbildung 2.1. Bestandteile eines RFID-Systems[2]



Zwischen den einzelnen RFID-Systemen gibt es allerdings unterschiedliche Ausprägungen der vorgestellten Komponenten: So wird beispielsweise unterschieden, auf welche Weise die Daten auf dem Chip gespeichert werden, wie

[2]Die Abbildung wurde frei nach dem Vorbild folgender Webseite von GS1 Germany gestaltet: http://gs1-germany.de/internet/content/produkte/epcglobal/rfid_epc/technik/index_ger.html (Stand: 11. April 2006).

groß die Speicherkapazität ist und ob man ihn mehrmal beschreiben kann. Komplexere Transponder können die Daten auch selbstständig weiterverarbeiten. Weitere Merkmale sind die Energieversorgung (aktiv oder passiv, je nachdem ob der Transponder eine eigene Batterie besitzt) und die Frequenz, auf der gesendet wird. Auch sind unterschiedliche Bauformen wie Folien, Karten, Glasröhren und Scheiben verbreitet.

Wie bereits angedeutet bringt die berührungslose Übertragung dieses Verfahrens enorme Vorteile mit sich: EPCs, welche auf den RFID-Tags gespeichert werden, können problemlos an jeder Stelle der Vorsorgungskette ausgelesen und weiter verarbeitet werden. So ist es beispielsweise möglich, den gesamten Inhalt eines Kartons komplett zu erfassen, ohne dass er vorher ausgepackt werden muss. Für ein Netzwerk, welches eine vollständige Rückverfolgung einzelner Objekte gewährleisten soll, indem diese an verschiedenen Punkten registriert werden, kann dies als durchaus solide Grundlage angesehen werden.

Die ersten RFID-Systeme wurden bereits Ende des Zweiten Welkrieges zu militärischen Zwecken eingesetzt. Dass erst jetzt an einer Nutzung dieser Technologie für die Versorgungsketten gearbeitet wird, liegt daran, dass bisher die Kosten für eine Herstellung zu hoch waren.

2.1.2. Electronic Product Code (EPC)

Hauptbestandteil der auf dem RFID-Tag gespeicherten Daten ist der bereits erwähnte elektronische Produktcode. Er ist daher stark mit der RFID-Technologie verknüpft und sie bilden in Kombination die Grundlage für die angestrebte Netzwerkstruktur.

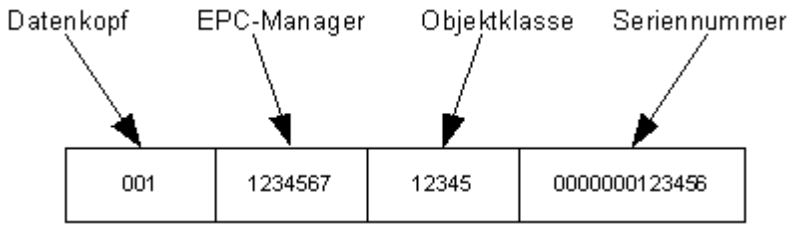
Die Besonderheit an diesem Code ist seine weltweite Eindeutigkeit, die jedes Produkt explizit identifizierbar machen kann. Dies bedeutet, dass er zum Beispiel nicht nur eine Produktsorte eines bestimmten Herstellers beschreiben kann, sondern vielmehr, dass auch alle Produkte dieser Sorte untereinander klar zu unterscheiden sind. Weiter kann dieser Code natürlich nicht nur für einzelne

Produkte eingesetzt werden, sondern auch für logistische Einheiten, wie Paletten und Umkartons, was einen Einsatz auch innerhalb der Wertschöpfungskette möglich macht.

2.1.2.1. Aufbau eines EPCs

Technisch gesehen besteht der EPC aus einer fest definierten Ziffernfolge in einer typabhängigen Länge von 64 – 204 Bit. Diese setzt sich aus folgenden international vereinbarten Komponenten zusammen, die in Kombination den eindeutigen Code bilden:

Abbildung 2.2. EPC-Bestandteile[3]



Datenkopf — Der Datenkopf beschreibt die verwendete EPC-Version und wie die Informationsdaten verschlüsselt wurden. Sehr geläufig ist auch die englische Bezeichnung *Header*.

EPC-Manager — Die Manager-Nummer kennzeichnet den Nummerngeber (wie zum Beispiel den Hersteller). Dieser Wert wird auch als *Unternehmenskennung* bezeichnet.

Objektklasse — Die Objektklasse bezeichnet das Produkt, also zum Beispiel ein Paket Kaffee einer bestimmten Sorte. Dies entspricht normalerweise der Artikelnummer.

[3]Die Abbildung wurde frei nach dem Vorbild folgender Webseite von GS1 Germany gestaltet: http://www.gs1-germany.de/internet/content/produkte/epcglobal/rfid_epc/der_epc/index_ger.html (Stand: 11. April 2006).

Seriennummer	Die Seriennummer ist eine fortlaufende Nummer, die in Kombination mit den anderen Bestandteilen eines EPCs jedes Objekt eindeutig identifizierbar macht.

Die genannten Komponenten können noch detaillierter dargestellt werden: So gibt es noch einen Filterwert, der den Typ der Einheit genauer spezifiziert (ob es sich zum Beispiel um einen Artikel oder eine Palette handelt) und einen Partitionswert, welcher beschreibt, an welcher Stelle die Manager-Nummer und die Objektklassen-Nummer voneinander getrennt werden.

Für die Unternehmenskennung wird normalerweise die *Global Trade Number (GTN)* verwendet, über die Unternehmen weltweit eindeutig zu identifizieren sind. Diese Nummer wird normalerweise ausgehändigt, wenn sich ein Unternehmen für die Nutzung des EAN-Systems[4] anmeldet. Da dieses weit verbreitet ist, kann davon ausgegangen werden, dass in der Regel eine solche Nummer verfügbar ist.

Wichtig ist, zu beachten, dass innerhalb der Ziffernfolge keine weiteren Produkteigenschaften wie Haltbarkeitsdatum oder aktueller Aufenthaltsort gespeichert werden können. Solche Informationen werden von auswertenden Applikationen geschrieben und können dann mit dem EPC als eindeutiger Schlüssel wieder abgerufen werden.

2.1.2.2. Vergleich zum EAN-Code

Der EAN-Code ist der wohl am meisten verwendete Code in Barcode-Systemen. Er ist weltweit anerkannt und etabliert.

Im Gegensatz zum verhältnismäßig langem EPC besteht er nur aus dreizehn Ziffern, die wiederum folgende Komponenten beschreiben:

- Eine Länderkennung,

[4]EAN: *International Article Number* (ehemals: *European Article Number*). Produktkennzeichnung für einen Handelsartikel, normalerweise dargestellt duch einen Strichcode.

- die Bundeseinheitlichen Betriebsnummmer (bbn),
- die Artikelnummer
- und eine Prüfziffer.

An dieser Stelle wird deutlich, dass diese Komponenten im Gegensatz zu denen des EPCs nur in der Lage sind, einzelne Artikelarten voneinander zu unterscheiden (da die Seriennummer fehlt). Eine eindeutige Identifikation jedes einzelnen Artikels ist auf diesem Weg nicht möglich.

Da sich der EAN-Code weltweit durchsetzen konnte und aus diesem Grund zu einer enormen Verbreitung gefunden hat, wurde er auch innerhalb des EPCs zur Kennzeichnung von Objekten integriert. Somit ist die Weiternutzung bestehender Systeme gewährleistet und es ist möglich, die Technologien parallel zu betreiben.

2.1.3. Standardisierungsprozess

Wie wichtig eine Standardisierung der eingesetzten Technologien ist, hat die Entwicklung des EAN-Codes gezeigt. Man kann annehmen, dass er ohne Standardisierung nicht seine weite Verbreitung gefunden hätte. Natürlich setzt auch die Idee, zwischenbetriebliche Kommunikation durch ein globales Netzwerk zu vereinfachen und zu vereinheitlichen, zwingend Standards voraus. Auch wenn Standardisierungsverfahren in der Regel sehr langwierig sind, bringen sie doch enormen Nutzen mit sich.

GS1 und *GS1 US* (ehemals *EAN International* und *Uniform Code Council, Inc.*) gründeten im Jahre 2003 die Non-for-profit-Organisation *EPCglobal Inc.*, die seitdem wirtschaftliche und technische Standards für eine gemeinsame Netzwerk-Infrastruktur entwickelt. Diese befassen sich sowohl mit Hardware- als auch mit Software-Komponenten.

Die eigentliche Standardisierung soll stückweise während drei Phasen entstehen, in denen einzelne Spezifikationen verabschiedet werden, die aufeinander

aufbauen. Diese betreffen hauptsächlich die Schnittstellen der einzelnen Komponenten und entsprechen den drei Schichten des Netzwerkes[5]:

1. *EPC-Standards für den Austausch von physischen Objekten:* Dieser Bereich deckt die eigentliche Identifikation physischer Objekte ab. Es wird sichergestellt, dass ein Anwender den Transponder auslesen und den resultierenden EPC richtig interpretieren kann. Dazu gehört auch die Beschreibung der Luftschnittstelle zwischen RFID-Tag und Lesegerät.

2. *EPC-Standards für die unternehmensinterne Infrastruktur:* Dieser Bereich betrifft Abläufe innerhalb eines Unternehmens. Es werden Schnittstellen zwische Hardware und Software definiert, die dem Anwender die Möglichkeit geben sollen, in einer geschlossenen Infrastruktur Daten zu verarbeiten und zu speichern.

3. *EPC-Standards für den Datenaustausch:* Der dritte Bereich definiert Schnittstellen und Formate für den überbetrieblichen Datenaustausch. Dadurch wird die Transparenz an der Versorgungskette erhöht. Dazu gehören auch eine formale Beschreibung der einzelnen Kerndienste des Netzwerkes und Spezifikationen für deren Nutzung.

Ein Teil dieser Standards wurde bereits fertig gestellt und wird auch schon von verschiedenen Unternehmen eingesetzt. Die anderen befinden sich noch in der Entwicklung und sollen auch nach ihrer Fertigstellung interessierten Anwendern öffentlich zur Verfügung gestellt werden.

2.2. Das EPCglobal-Netzwerk

Das Internet der Dinge

An dieser Stelle soll nun auch das bereits mehrfach angesprochene EPCglobal-Netzwerk etwas genauer betrachtet werden. Dafür werden insbesondere die einzelnen Komponenten und Dienste des Netzwerkes vorgestellt und deren Funk-

[5]Frei nach der Management-Information *EPC-Kommunikation* [Kommunikation], Seite 19.

tionsweise beschrieben. Beispiele für einen Praxiseinsatz, welche das Zusammenspiel der einzelnen Bestandteile verdeutlichen sollen, folgen dann im nächsten Abschnitt.

2.2.1. Grundlegendes Konzept

Hinter dem Begriff *EPCglobal-Netzwerk* (auch *EPC-Netzwerk*) steht die Idee, Informationen zu Produkten und anderen Objekten mit Hilfe des Internets jederzeit verfügbar zu machen. Aus diesem Grund wird dieses Informationsnetzwerk auch immer wieder als *Internet der Dinge* bezeichnet.

Zur Realisierung dieses Netzwerkes werden unterschiedlichste EPC-Informationen – also Daten zu den Produkten selbst und ihrer Bewegungen innerhalb der Versorgungsketten – lokal auf Servern gespeichert, die dann über eine fest definierte Netzwerkstruktur miteinander verbunden werden, so dass jedes berechtigte Mitglied auch auf die Daten des anderen zugreifen kann.

Die dafür nötige Steuerung wird von bestimmten Netzwerk-Komponenten geregelt, für die eigentliche Datenübermittlung kommt das Internet zum Einsatz. Ergänzt wird diese Infrastruktur durch Sicherheitsmechanismen, so dass für jedes Mitglied individuell entschieden werden kann, welche Daten gelesen werden dürfen und welche nicht.

Die eigentliche Netzwerk-Architektur folgt einigen prinzipellen Richtlinien. Diese sind im Wesentlichen:

1. *Globale Standards:* Alle Komponenten werden standardisiert. Hierbei wird auf bereits bestehende, branchenspezifische Standards und Architekturen Rücksicht genommen, auch wenn die Arbeiten grundsätzlich anbieterneutral geschehen.

2. *Modularer Aufbau:* Das Netzwerk selbst wird in einzelne Module aufgeteilt, welche Komponenten mit ähnlichen Funktionen sammeln. Diese wer-

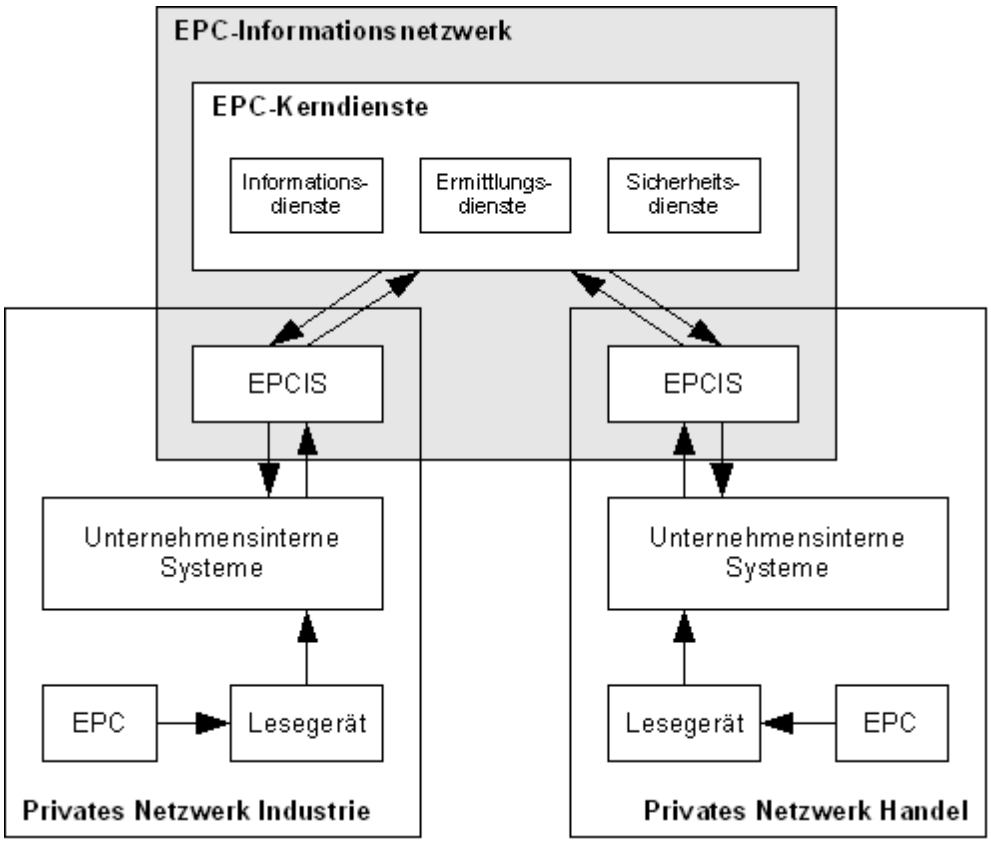
den einzeln spezifiziert, so dass eine Implementierung stufenweise geschehen kann.

3. *Erweiterbarkeit:* Neben der Festlegung von Grunddatentypen und Grundfunktionalitäten definiert die Kernspezifikation der Architektur auch Vorgehensweisen für eine Erweiterung dieser Elemente, so dass die Möglichkeit besteht, diese auch anwenderspezifisch auszubauen.

2.2.2. Netzwerk-Komponenten

Die einzelnen Teile der Netzwerk-Architektur sind in Module zusammengefasst, die in diesem Abschnitt vorgestellt werden sollen. Die nachfolgende Abbildung soll einen schematischen Überblick über die Architektur geben.

Abbildung 2.3. Das EPCglobal-Netzwerk[6]



[6]Die Abbildung wurde frei nach dem Vorbild aus der Management-Information *EPC-Kommunikation* [Kommunikation] gestaltet.

Hinweis:

Die EPCIS-Systeme innerhalb der lokalen Netzwerke entsprechen den zu den EPC-Kerndiensten gehörenden Informationsdiensten. Da mittels dieser Abbildung auf der einen Seite die einzelnen Komponenten des Netzwerkes, auf der anderen aber auch das Zusammenspiel der lokalen und der netzbasierten Systeme gezeigt werden soll, ist eine andere Darstellung leider nicht möglich.

2.2.2.1. EPC-Informationsdienste

Die EPC-Informationsdienste[7] stellen die Schnittstelle eines lokalen Systems zum EPC-Netzwerk dar. Über diese Dienste können unternehmensinterne Systeme EPC-Informationen zu eigenen Produkten speichern und wieder abrufen. Zusätzlich stehen aber auch EPC-Informationen anderer Unternehmen zur Verfügung, wenn eine entsprechende Berechtigung vorhanden ist. Dies ist möglich, weil die lokalen Informationsdienste über das EPC-Netzwerk auch Zugriff auf die anderen EPCIS-Systeme haben.

Somit gehören zu einem EPCIS-System insgesamt zwei grundlegende Funktionen, die durch Schnittstellen dargestellt werden können:

- *Erfassungs-Schnittstelle:* Über diese Schnittstelle können EPC-Informationen, wie Daten zum Produkt selbst und solche, die mit dem Produkt in Beziehung stehen, an das EPCIS-System übermittelt werden. Der EPCIS-Server speichert dann diese Daten derart, dass sie dann dem Netzwerk zur Verfügung stehen.

- *Abfrage-Schnittstelle:* Über die Abfrage-Schnittstelle können die zuvor gespeicherten Informationen wieder abgerufen werden. Dafür wurde ein Abfrage-Konzept entwickelt, über das sich eine Auswahl konfigurieren lässt.

[7]Englisch: *EPC Information Services*, abgekürzt mit *EPCIS*.

Jedes berechtigte Netzwerk-Mitglied hat die Möglichkeit, benötigte EPC-Informationen auf diesem Weg abzurufen.

Idealerweise ist eine EPCIS-Applikation direkt mit den bestehenden Warenwirtschaftssystemen und Datenbanken des Unternehmens verknüpft.

Zur EPCIS-Spezifikation gehört neben der Beschreibung der vorgestellten Schnittstellen auch die Definition sämtlicher Datenelemente, die für die Kommunikation mit diesem Dienst gebraucht werden. Sie bilden also zusammen mit den beiden Schnittstellen die Grundlage für den ganzen Informationsdienst.

Ein Unternehmen kann mehrere EPCIS-Server pflegen. Natürlich ist es auch denkbar, diese über einen externen Dienstleister auszulagern, der dann die erforderlichen Dienste innerhalb des Netzwerkes anbietet, wobei trotzdem die Verantwortung immer beim jeweiligen Unternehmen bleibt.

Da das Thema dieser Arbeit hauptsächlich den gerade vorgestellten EPCIS-Standard betrifft, ist diesem ein eigenes Kapitel gewidmet, welches die einzelnen Bestandteile weiter vertiefen wird.

2.2.2.2. EPC-Ermittlungsdienste

Die EPC-Ermittlungsdienste (englische Bezeichnung: *EPC Discovery Services*) fassen alle Dienste und Funktionalitäten des EPC-Netzwerkes zusammen, mit deren Hilfe all die relevanten EPCIS-Systeme lokalisiert werden können, die Informationen zu bestimmten EPCs enthalten. Auf diesem Weg wird ein gezieltes Auffinden einzelner Informationen möglich gemacht, die nicht lokal verfügbar sind.

Zu diesen Komponenten zählen der zentrale Objektnamensdienst (*Root Object Name Service*, abgekürzt mit *Root ONS*), der lokale Objektnamensdienst (*Local Object Name Service*, abgekürzt mit *Local ONS*) und der Ereignis-Registrierungsdienst (*EPC Event Registry Services*), die folgende Funktion innerhalb der Ermittlungsdienste haben:

- *Zentraler Objektnamensdienst:* Der Objektnamensdienst speichert die Orte für die lokalen Namensdienste eines bestimmten EPC-Managers und liefert sie bei Anfragen zurück. Er bildet also den Kern für eine Lokalisierung.

- *Lokaler Objektnamensdienst:* Dieser Dienst liefert die Orte einzelner EPCIS-Server für einen bestimmten EPC-Manager in Verbindung mit einer Objektklasse. In Kombination mit dem zentralen ONS ist also bereits eine detaillierte Suche nach einzelnen Produktarten möglich.

- *Ereignis-Registrierungsdienst:* Durch diesen Dienst können EPCIS-Server lokalisiert werden, die Ereignis-Daten zu bestimmten EPCs enthalten. Es ist also eine Lokalisierung bis in die unterste Ebene möglich (EPC-Manager, Objektklasse und Seriennummer).

Wichtig ist, dabei zu beachten, dass alle diese Dienste niemals die EPC-Informationen selbst speichern, sondern nur den Standort dieser Informationen.

Um das kritische Volumenmanagement dieser Dienste zu entlasten und die Erweiterbarkeit zu garantieren, ist die Verteilung auf mehrere Server vorgesehen, was mit dem dezentralen Ansatz des Netzwerkes durchaus im Einklang steht. Ereignis-Registrierungsserver könnten zum Beispiel nach Produktgruppen oder Ländern organisiert werden.

2.2.2.3. Sicherheitsdienste

Zu guter Letzt existieren noch zusätzliche Sicherheitsdienste (*Security Services*), die einen sicheren und zuverlässigen Datenaustausch innerhalb des EPC-Netzwerkes bereitstellen sollen. Diese folgen den klassischen Sicherheits-Theorien im Netzwerkbereich und stellen daher folgende Dienste zur Verfügung:

- *Dienste zur Authentifizierung:* Diese Dienste stellen sicher, dass jeder Teilnehmer auch wirklich der ist, für den er sich ausgibt. Dies wird über digitale Zertifikate realisiert, die im Internet weit verbreitet sind.

- *Dienste zur Authorisierung:* Die Authorisierungsdienste regeln die Zugriffsberechtigung einzelner Netzwerk-Mitglieder auf bestimmte Daten. So kann jedes Unternehmen nur die Daten abrufen, für die es auch eine Berechtigung besitzt.

Da diese Dienste noch nicht vollständig spezifiziert wurden, wird innerhalb dieser Arbeit nicht deutlich weiter auf deren Funktionsweise eingegangen.

2.2.3. Vergleich zum Internet

Das EPC-Netzwerk selbst enthält viele Gemeinsamkeiten zu den Strukturen des *World Wide Web*. Schon aus diesem Grund ist die Bezeichnung *Internet der Dinge* durchaus zutreffend.

Da diese Parallelen einiges zum Verständnis des EPC-Netzwerks beitragen können, sollen die entsprechenden Vergleiche an dieser Stelle aufgezeigt werden[8]:

DNS	Dieses zentrale System, welches URLs den entsprechenden IP-Adressen zuordnet, kann auf jeden Fall mit den *Objektnamensdiensten* des EPC-Netzwerkes verglichen werden, welche einzelne EPCs ihren Lokalitäten zuordnen.
Webseiten	Webseiten stellen im übertragenen Sinn einen Ort da, an dem Informationen zu einem bestimmten Thema gespeichert sind. Dies hat große Ähnlichkeiten mit einem *EPCIS-Server*, der Informationen zu einem Produkt verwaltet.
Suchmaschinen	Suchmaschinen werden für das Auffinden von Informationen im Internet eingesetzt, ähnlich den *Ermittlungsdiensten* im EPC-Netzwerk, die nach Informationen zu einzelnen EPCs suchen.

[8]Frei nach dem Vergleich aus der Management-Information *Internet der Dinge* [Dinge].

SSL	SSL sorgt für eine sichere Datenübertragung im Internet und ist somit auch vergleichbar mit den oben beschriebenen *Sicherheitsdiensten*, welche diese Aufgabe im EPC-Netzwerk übernehmen.

2.2.4. Funktionsweise

Die Nutzung der einzelnen Dienste für ein gemeinsames Informationsnetzwerk ist natürlich nur im Zusammenspiel der Komponenten möglich. Dies kann beispielsweise nach folgendem Schema funktioneren:

1. Produziert ein Hersteller Produkte, so kennzeichnet er sie eindeutig mit einem neuen EPC. Zusätzlich speichert er EPC-Informationen zu den Produkten innerhalb seines lokalen EPCIS-Systems. Dazu können Daten wie zum Beispiel das Fertigungs- und das Verfallsdatum gehören. Stammdaten zur Produktsorte werden an dieser Stelle nicht gespeichert.

2. Der EPCIS-Server registriert die neuen Produkte innerhalb des EPC-Netzwerkes, dessen Dienste den Ort des EPCIS-Servers fest mit den jeweiligen EPCs verknüpfen. Dieser Ort steht ab diesem Moment jedem Nutzer des Netzwerkes zur Verfügung.

3. Andere Mitglieder des Netzwerkes, die im Laufe der Zeit etwas mit diesem Produkt zu tun haben, speichern diese Ereignisse ebenso in ihren lokalen EPCIS-Systemen. So könnte ein Händler den Empfang eines Produktes quittieren, indem er ein entsprechendes Ereignis an seinen EPCIS-Server sendet. Auch der Standort dieser Informationen werden durch das EPCIS-System innerhalb des Netzwerkes registriert.

4. Benötigt ein Händler nun vielleicht Informationen zu einem der oben genannten Produkte, so kann er sich mittels seines eigenen EPCIS-Systems vom zentralen ONS des EPC-Netzwerkes den lokalen ONS des Herstellers suchen lassen, der wiederum den Ort des EPCIS-Servers des Herstellers

kennt. An diesen kann es dann die Anfrage nach den gewünschten Informationen stellen.

An dieser Stelle ist noch einmal zu erwähnen, dass alle Zugriffe auf die beschriebene Infrastruktur über die Sicherheitsdienste des EPC-Netzwerkes geregelt und geschützt werden.

2.3. Praxiseinsatz

Das Informationsnetzwerk in Aktion

Der zeitliche Verlauf der Standardisierung der RFID/EPC-Technolgien für den Einsatz in der Wertschöpfungskette geschieht stufenweise. Es ist vorgesehen, dass die neuen Technologien die alten Systeme zunächst ergänzen sollen, um bereits während des Entstehungsprozesses einzelne Techniken nutzen zu können, welche sich Problemen annehmen, die in der Vergangenheit noch nicht lösbar waren. Es wird allerdings angenommen, dass die RFID/EPC-Technologien über längere Zeit betrachtet die heute eingesetzten Barcode-Systeme vollständig ablösen werden.

In der Praxis werden heute zwei Einsatzgebiete unterschieden, in denen EPCs innerhalb der Versorgungsketten eingesetzt werden können:

1. Anwendung von EPCs auf logistischen Einheiten (*Unit-Tagging*).

2. Anwendung auf einzelnen Produkten (*Item-Tagging*).

Es ist möglich, diese beiden Anwendungsbereiche völlig getrennt voneinander zu behandeln, sie können sich aber auch durchaus in ihren Funktionalitäten überschneiden, je nachdem, welche Anforderungen durch den Anwender gesetzt wurden.

2.3.1. Unit-Tagging

Unter dem Begriff *Unit-Tagging* versteht man den Einsatz von EPCs und den damit verknüpften Technologien auf logistischen Einheiten wie Verpackungen, Umkartons und Paletten.

Hierfür werden solche Einheiten mit RFID-Tags ausgestattet, die selbstverständlich auch eindeutigen EPCs enthalten. Somit ist neben den einzelnen Produkten auch jede Versandeinheit eindeutig identifizierbar. Kennt man nun den EPC zu einer gekennzeichneten Einheit, so kann man sich mit Hilfe des EPC-Netzwerkes über deren Inhalt und weitere die Einheit betreffende Daten informieren.

Das Einsatzfeld des Unit-Taggings erstreckt sich über den gesamten Logistikbereich der Versorgungskette. Dazu gehören zum Beispiel der Transport der Waren vom Hersteller zum Handelspartner, die Zwischenlagerung und Weiterverteilung innerhalb von Distributionszentren und die darauf folgende Auslieferung der Waren an die einzelnen Filialen.

An diesen einzelnen Lokationen der Versorgungskette kann dann die EPC-Netzwerkstruktur auf unterschiedlichste Arten genutzt werden, was unter Umständen innerhalb dieser Bereiche immense Vorteile bringen kann[9]:

- *Pulkerfassung beim Wareneingang und beim Warenausgang:* Beim Versenden oder Empfang einer gekennzeichneten Einheit können nahezu zeitgleich alle EPCs einer Lieferung gelesen und gespeichert werden. Dieser Vorgang war bisher zum Teil sehr langwierig, da unter Umständen eine Lieferung vollständig ausgepackt werden musste, um den Sichtkontakt zu den Barcodes zu ermöglichen. Durch die RFID-Technologie kann dieser Vorgang mehr oder weniger automatisiert werden, was einen enormen Zeitgewinn bedeuten kann.

[9]Die Anwendungsfälle wurden frei nach den Ausführungen auf folgender Webseite der Institut, Management + Consulting AG zusammengestellt: http://epc-forum.de/praxis/index1.html (Stand: 21. Juni 2006).

- *Verfolgung und Rückverfolgung von Waren:* Entlang der gesamten Versorgungskette können die einzelnen Lieferungen verfolgt werden, insbesondere auch über mehrere Handelspartner hinweg. Über einfache Fragen wie „Wo ist meine Bestellung?“ oder „Ist meine Bestellung bereits angekommen?“ können auch deutlich komplexere Planungen für das Bestandsmanagement gemacht werden, da beispielsweise ein Distributionszentrum sofort sehen kann, wenn Waren in einer Filiale knapp werden. Selbst der Hersteller kann bereits zu diesem Zeitpunkt mit Hilfe dieser Daten planen. Zusätzlich lässt sich über das EPC-Netzwerk auch rückwirkend der Weg eines Produktes verfolgen. Dies ist gerade bei streng kontrollierten Lebensmitteln wie Fleisch enorm wichtig, zum Teil sogar gesetzlich vorgeschrieben.

- *Statusermittlung von Transportmitteln:* Bisher war es schwierig, Einheiten mit einem Status zu versehen. So war es normalerweise zwar möglich, zum Beispiel den Aufenthaltsort einer Palette zu bestimmen, allerdings war es oft nur sehr schwer zu sagen, ob diese gerade in Gebrauch ist. Dies stellt auch ein großes Problem innerhalb der Getränke-Industrie dar, wo einerseits Kisten für Mehrweg-Flaschen, aber auch die Flaschen selbst verwaltet werden müssen. Da das EPC-Netzwerk umfangreiche Möglichkeiten für Prozessverwaltung bietet, ist durch die Nutzung der RFID-Tags auch eine deutlich bessere Planung für den Einsatz und den Bestand solcher Transporteinheiten möglich.

- *Vereinfachung von Inventuren:* Auch Inventuren können durch die RFID/EPC-Technologie deutlich vereinfacht werden. Es ist einerseits eine ungefähre Ermittlung des aktuellen Bestandes bereits durch die Informationen des EPC-Netzwerkes möglich (man kann bestimmen, welche Waren das Lager erreicht und wieder verlassen haben), auf der anderen Seite können die Regale in den Lagerhallen aber auch einfach mit einem mobilen Lesegerät abgefahren werden, was natürlich einen enormen Zeitvorteil zu einer Inventur im klassischen Stil bringt.

Diese Anwendungsfälle sollen nur als Beispiel dienen. Es existieren natürlich weitere, unzählige Möglichkeiten, die RFID/EPC-Technologie im logistischen Bereich einzusetzen. Eines haben diese Fälle aber immer gemeinsam: Die zeitliche und somit auch finanzielle Optimierung logistischer Vorgänge.

2.3.2. Item-Tagging

Im Gegensatz zum eben vorgestellten Einsatzbereich werden beim *Item-Tagging* einzelne Artikel mit RFID-Tags ausgezeichnet.

Dies bedeutet, dass jedes einzelne Produkt durch einen EPC gekennzeichnet wird und mittels dieses Weges innerhalb des EPC-Netzwerkes identifizierbar bleibt. Dies kann für einzelne Produkte unter Umständen bereits während der zuvor dargestellten logistischen Prozesse sinnvoll sein, allerdings wird das Item-Tagging hauptsächlich auf Filial-Ebene eingesetzt, also vom Einstellen des Produktes in das Regal bis zum Verlassen des Supermarktes.

Aus diesem Grund unterscheiden sich auch die Einsatzmöglichkeiten von denen des Unit-Taggings[10]:

- *Sicherung der Waren vor Diebstahl und Markenfälschung:* RFID-Tags sind für eine Produktsicherung gut geeignet. Sie sind deutlich günstiger als andere Sicherungssysteme und einfacher zu handhaben. Das Prinzip, dass durch eine Sicherheitsschranke am Ausgang ein nicht bezahlter Artikel bewegt wird, was dann wiederum einen Alarm auslöst, dürfte wohl jedem geläufig sein. Zusätzlich kann die RFID/EPC-Technologie auch eingesetzt werden, um sich vor Markenpiraterie zu schützen, was möglich wird, weil ein Artikel mit einem individuellen Code sich auch eindeutig einem Hersteller zuordnen lässt. Eine Fälschung ist zwar theoretisch möglich, kann aber durch eine Abstimmung des Codes innerhalb des Netzwerkes relativ sicher erkannt werden.

[10]Die Anwendungsfälle wurden frei nach den Ausführungen auf folgender Webseite der Institut, Management + Consulting AG zusammengestellt: http://epc-forum.de/praxis/index2.html (Stand: 21. Juni 2006).

- *Pulkerfassung an der Kasse:* Ähnlich der Nutzung beim Unit-Tagging zur Erfassung beim Ein- und Ausgang der Waren kann ein RFID/EPC-System auch den Bezahlvorgang an der Kasse optimieren: Auch hier ist denkbar – die Kennzeichnung jeden Produkts vorausgesetzt – den gesamten Inhalt eines Einkaufswagens auf einmal zu erfassen. Der Wagen müsste hierzu nicht einmal ausgräumt werden. Zusätzlich werden bei diesem automatisierten Vorgang die Daten mit einer extrem hohen Genauigkeit aufgenommen, so dass auch Nachbestellungen deutlich flexibler gestaltet werden können.

- *Umtauschen ohne Kassenbon:* Durch die Eindeutigkeit eines jeden Produktes ist eine vollständige Rückverfolgbarkeit möglich. Eine Einsatzmöglichkeit für diese Tatsache ist der Umtasch von einzelnen Artikeln. Da ein Produkt eindeutig identifizierbar ist und auch Daten wie der Kaufort und das Kaufdatum gespeichert werden können, ist es möglich, ein sicheres Umtauschsystem ohne Kassenbon zu realisieren. Die auf diesem enthaltenen Informationen können dann auch über das EPC-Netzwerk abgefragt werden.

- *Kunden-Kontakt-Pflege:* Einen weiteren wichtigen Punkt stellt die Kunden-Kontakt-Pflege dar (englisch: *Customer Relationship-Management*, abgekürzt mit *CRM*): Durch den Einsatz von RFID-Tags auf Kundenkarten kann das Kaufverhalten des Kunden deutlich weitreichender analysiert werden. Bereits während des Einkaufs wird es möglich, für jeden Kunden individuell Werbung zusammenzustellen. Dies konnte normalerweise erst nach dem Bezahlvorgang gemacht werden, da dies der früheste Punkt war, an dem sich einzelne Waren auch einer speziellen Person zuordnen lassen.

Auch diese Liste an Beispielen könnte fortgesetzt werden. Allerdings ist hier zu sagen, dass alle diese Anwendungsfälle nicht nur – wie beim Unit-Tagging – auf eine Reduzierung der Kosten abzielen, sondern zusätzlich auch den Absatz steigern sollen. Sie betreffen insgesamt gesehen also eher den Kunden.

2.3.3. Vor- und Nachteile

Diese vorgestellten Anwendungszenarien zeigen, dass der Einsatz der RFID/ EPC-Technologie dem Handel viele Vorteile bringen kann. Dabei kann man zwei Punkte besonders hervorheben, deren Eigenschaften sich die meisten der eben vorgestellten Szenarien zu Nutze machen:

- *Bereitstellung von Echtzeit-Informationen:* Alle Daten, die über die Netzwerk-Infrastruktur den Handelspartnern zur Verfügung gestellt werden, können sofort abgerufen werden. Dies ist mittels EDI-Nachrichten nur eingeschränkt möglich.

- *Informationstransparenz an der gesamten Versorgungskette:* Die Daten sind an allen Stellen der Wertschöpfungskette erreichbar, auch übergreifend über mehrere Handelspartner hinweg. In klassischen Systemen war dies oft nur zu benachbarten Partnern der Fall.

Trotzdem ist die RFID-Technologie immer wieder in die Kritik geraten.

Gerade der Anwendungsfall der Kunden-Kontakt-Pflege beim Item-Tagging löst viel Unmut bei den Verbraucherschützern aus, da die Problematik des „Gläsernen Kunden“ dadurch nicht gerade eingedämmt wird. Vielmehr könnten bestehende CRM-Systeme so erweitert werden, dass eine nahezu vollständige Überwachung des Kunden bereits während des Einkaufes möglich ist. Dazu gehören zum Beispiel auch Aufzeichnung des Bewegungsverhaltens und die Einkaufsdauer des Kunden. Die technischen Voraussetzungen für eine solche Überwachung wäre durch die RFID-Technologie auf jeden Fall verfügbar.

Hinzu kommt, dass der Kunde keinen Einfluss darauf hat, wann welche Daten gesendet werden. Dies betrifft dann natürlich auch den Datenschutz des Kunden. Theoretisch wäre es möglich, den Datenblock des Transponders ohne Wissen des Kunden mit weiteren Daten auszustatten, die dann prinzipiell jedes Lesegerät abrufen könnte.

Zu diesen Daten- und Verbraucherschutzfragen kommt hinzu, dass die Technolgie noch nicht vollständig ausgereift ist. So spielen bei der Erfassung der Informationen durch die RFID-Tags auch Störgrößen wie bestimmte Metalle und Flüssigkeiten eine Rolle, was den Einsatz für eine sichere Pulkerfassung noch nicht möglich macht. Auch die Frage der Entsorgung des, bei einem Masseneinsatz der RFID-Tags anfallende, zum Teil giftigen Elektro-Schrotts, konnte bisher nur teilweise geklärt werden. An dieser Stelle ist zum Beispiel von einer möglichen Verseuchung des Trinkwassers die Rede.

3. Die EPC-Informationsdienste

Bestandteile, Aufbau und Funktionsweisen eines EPCIS-Systems

Dieses Kapitel stellt den zweiten Grundlagenteil dieser Arbeit dar. Es soll zeigen, aus welchen Komponenten ein EPCIS-System aufgebaut ist, wie diese miteinander interagieren und auf welche Weise sie dadurch eine geschlossene Informationseinheit für das EPC-Netzwerk bilden. Dadurch soll ein grundlegendes Wissen zu der teilweise sehr abstrakten EPCIS-Theorie geschaffen werden.

Das gesamte Kapitel sehr sehr stark auf der EPCIS-Spezifikation in der Version 1.0 auf, welche zum Zeitpunkt der Verfassung dieser Arbeit in einer fortgeschrittenen Working-Draft-Version vorliegt.

3.1. Einführung in die EPCIS-Architektur

Diese EPCIS-Spezifikation sieht den Aufbau eines Informationssystems nach drei theoretischen Grundprinzipien vor:

1. *Aufteilung in einzelne Schichten:* Die Spezifikation arbeitet mit einem Schichtenmodell, deren einzelne Bereiche aufeinander aufbauen. Die unterste Schicht bildet dabei das abstrakte Datenmodell, welches eine allgemeine Struktur definiert, über die sich in der zweiten Schicht konkrete Datentypen definieren lassen. Zu guter Letzt setzt eine dritte Schicht auf diese Definitionen auf, welche die Service-Zugriffe auf die EPCIS-Daten regelt.

2. *Modularität:* Zusätzlich sind die einzelnen Schichten in unterschiedliche Module gegliedert, die getrennt voneinander spezifiziert werden. Man kann also davon sprechen, dass die EPCIS-Spezifikation aus vielen einzelnen, kleineren Spezifikationen zusammengesetzt ist, die miteinander verknüpft

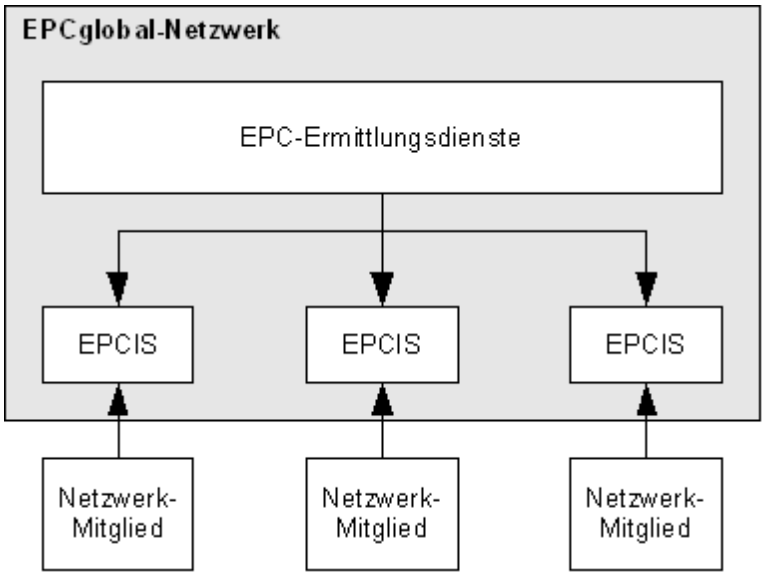
sind. Somit können ganze Bereiche unabhängig voneinander implementiert, ausgetauscht und erweitert werden.

3. *Erweiterbarkeit:* Neben der grundsätzlichen Möglichkeit einer Erweiterung durch Austauschen einzelner Module und sogar ganzer Schichten bietet die Spezifikation noch zusätzliche Erweiterungsarten an: Die Daten der Definitionsschicht werden mit Hilfe von UML (*Unified Modeling Language*, eine standardisierte Sprache zu Modellierung von Software) beschrieben, was eine grundsätzliche Erweiterung durch Vererbung erlaubt. Zusätzlich werden an unterschiedlichen Stellen innerhalb der Elemente durchweg sogenannte Erweiterungspunkte gesetzt, so dass Anwender auch ohne Vererbung die einzelnen Objekte ergänzen können.

3.1.1. Einordnung in das EPC-Netzwerk

Die EPC-Informationsdienste stellen die Schnittstelle des Anwenders zum EPC-Netzwerk dar. Er kann dabei nicht nur auf sein eigenes EPCIS-System zugreifen, sondern über die Netzwerk-Infrastruktur auch auf die Systeme anderer Mitglieder.

Abbildung 3.1. EPCIS im EPC-Netzwerk



Das EPCIS-System selbst gehört dabei zu den Kerndiensten des Netzwerkes und befindet sich in der Hierarchie relativ weit oben: Das bedeutet, dass Daten, die innerhalb eines EPCIS-Systems verarbeitet werden, bereits ein vorläufiges Endstadium erreicht haben. Es handelt sich also nicht mehr um einfache Rohdaten, sondern um solche, die bereits durch RFID-Lesegeräte empfangen, neu codiert, gefiltert und sortiert wurden.

Ein Netzwerk-Mitglied muss einerseits die Möglichkeit bekommen, EPCIS-Daten in seinem lokalen Informationssystem zu speichern, auf der anderen Seite muss auch eine Schnittstelle zur Verfügung stehen, über die sich die Daten wieder abrufen lassen. Weiter benötigt jeder EPCIS-Servers noch eine Verbindung zu den anderen Diensten des Netzwerkes, um auch Zugriff auf entfernte Informationsserver zu bekommen.

3.1.2. Komponenten des Informationsdienstes

Somit besteht ein EPCIS-System im Wesentlichen aus folgenden Komponenten:

1. Aus einer *Erfassungsschnittstelle*, über die sich EPCIS-Daten an das System übertragen lassen. Diese Daten werden dann normalerweise auch dort gespeichert.

2. Aus einer *Abfrage-Schnittstelle*, über die sich dann die Daten mittels vorkonfigurierter Strukturen gefiltert und sortiert wieder auslesen lassen.

3. Dazu kommt eine Möglichkeit zur *Datenhaltung*, wo sich die erfassten EPCIS-Daten speichern lassen.

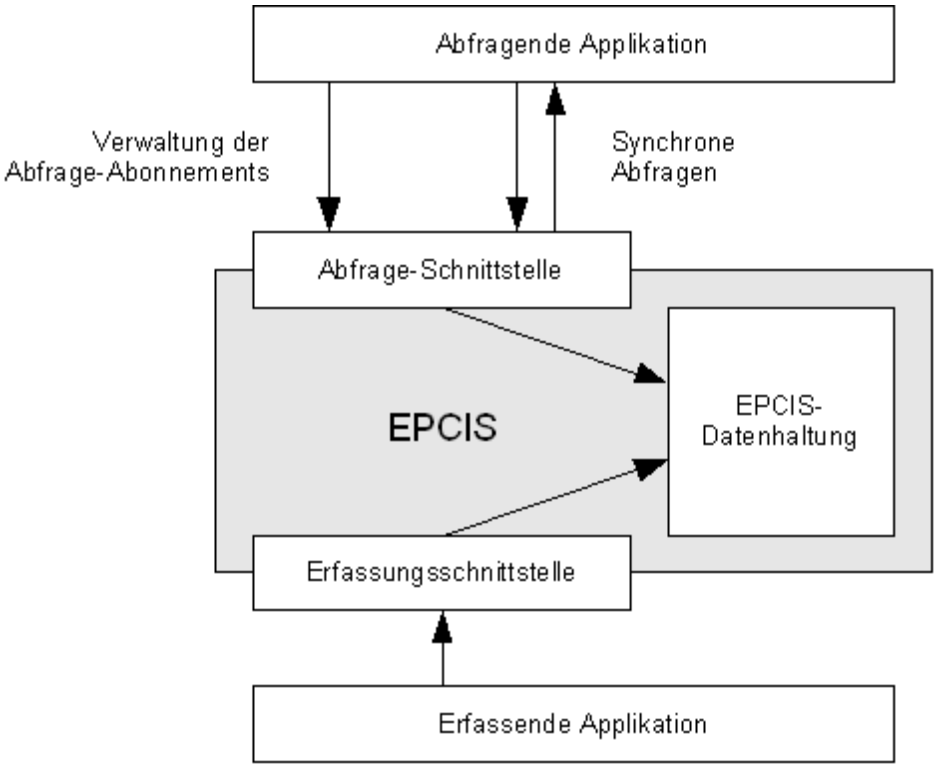
4. Und zu guter Letzt eine beliebige Anzahl von *Client-Applikationen*, die dann mit den beiden genannten Schnittstellen interagieren.

Diese Applikationen, die oft auch als *EPCIS-Clients* bezeichnet werden, realisieren also den Zugriff auf den EPCIS-Server. Da dieser prinzipiell nur die

reinen EPCIS-Daten speichert, ist die eigentliche Auswertung der Daten auf die Seite des Clients ausgelagert. Das bedeutet natürlich, dass ein EPCIS-Client über fundierte Kenntnisse der Geschäftslogik verfügen muss, um die Daten entsprechend interpretieren zu können.

Die Zugriffe auf den EPCIS-Server finden über die jeweilige Schnittstelle statt: Speichernde Zugriffe über die Erfassungs-Schnittstelle, das spätere Abrufen der Daten über die Abfrage-Schnittstelle. Auf letztere kann der Zugriff synchron oder auch asynchron geschehen. Dies wird später noch weiter erläutert.

Abbildung 3.2. EPCIS-Schnittstellen



An dieser Stelle soll noch einmal erwähnt werden, dass durch EPCglobal nicht das Ziel verfolgt wird, eine gemeinsame Referenz-Implementierung für dieses Informationssystem zu erstellen, sondern lediglich die Schnittstellen zwischen den einzelnen Netzwerkkomponenten zu standardisieren und Beschreibungen zu liefern, welche die Funktionalitäten der Komponenten spezifizieren. Die eigentliche Implementierung liegt dann bei den entwickelnden Unternehmen.

Aus diesem Grund werden durch die EPCIS-Spezifikation ausschließlich die beiden Schnittstellen und die Struktur der Daten spezifiziert, die über die Schnittstellen ausgetauscht werden. Zusätzlich liefert die Spezifikation nur theoretische Informationen darüber, was für Komponenten es sonst noch gibt, welche Funktionen sie haben und wie sie sich innerhalb des Systems verhalten. So wird zum Beispiel nicht festgelegt, auf welche Weise die Datenhaltung realisiert wird und wann der Server Zugriffe auf den ONS des EPC-Netzwerkes macht, um Informationen von anderen EPCIS-Servern zu bekommen.

3.2. Abstrakte Darstellung der EPCIS-Daten

Die bereits mehrfach erwähnten *EPCIS-Daten* werden in zwei unterschiedliche Gruppen eingeteilt: In Ereignis- und in Stammdaten. Die EPCIS-Spezifikation klärt innerhalb der Schicht des abstrakten Datenmodells die Begrifflichkeiten dieser Datenformen, ihren grundsätzlichen Aufbau und die Einordnung in das System.

3.2.1. Ereignisdaten

Als *Ereignis*[1] wird eine Struktur bezeichnet, die einen bestimmten Vorfall innerhalb eines Geschäftsprozesses modelliert. Sie hat immer einen bestimmten Typ mit einem eindeutigen Namen und eine Ansammlung von sogenannten *Feldern*, die das Ereignis genauer beschreiben.

In der Daten-Definitions-Schicht werden insgesamt vier unterschiedliche Ereignistypen für die Modellierung einzelner Anwendungsfälle definiert, die dann auch auf sie zugeschnittene Felder enthalten. Diese vier Ereignisarten werden später genauer beschrieben.

Je nach Art des Anwendungsfalles könnten Beispiele für ein Ereignis sein:

[1] Bezeichnung innerhalb der Spezifikation: *Event*. Ereignisdaten werden durchgängig mit *Event Data* bezeichnet.

- „Produkt mit einem bestimmten EPC hat zu einem bestimmten Zeitpunkt einen bestimmten Ort passiert.“

- „Palette mit einem bestimmten EPC wurde zu einem bestimmten Zeitpunkt an einem bestimmten Ort einer bestimmten Bestellung zugeordnet.“

Die informativen Aussagen dieser Beispiele werden dann den eben genannten Feldern des entsprechenden Ereignisses zugeordnet. Im Vergeich zur Objektorientierung entspricht also das Ereignis selbst einer Klasse und die jeweiligen Felder den Attributen.

3.2.2. Stammdaten

Stammdaten werden im Gegensatz zu den Ereignissen nicht dafür eingesetzt, bestimmte Anwendungsfälle selbst zu beschreiben, sondern sie sollen die Ereignisdaten ergänzen und Zusammenhänge definieren, die für die Interpretation der Ereignisse notwendig sind. Durch Kombination von Ereignis- und Stammdaten sollte es grundsätzlich möglich sein, jeden Geschäftsfall exakt zu beschreiben.

Die Stammdaten selbst bestehen aus einer beliebig großen Liste von *Wortschätzen*, die einzelne Einträge, die sogenannten *Wortschatz-Elemente* enthalten. Diese wiederum können beliebig durch Attribute ergänzt werden, die zusätzliche Informationen zu dem jeweiligen Element liefern sollen.[2]

Oft definiert ein Wortschatz durch seine Wortschatz-Elemente eine Liste von möglichen Werten, die Felder von Ereignissen annehmen können. Gekennzeichnet wird dies durch einen eindeutigen Namen des Wortschatzes, der dann dem entsprechenden Feld zugeordnet wird. Die meisten Ereignisfelder enthalten über einen Wortschatz definierte Werte. Ein Beispiel für einen solchen Wortschatz ist jener, der mögliche Werte für die Beschreibung von Orten enthält, an denen ein Ereignis aufgetreten ist. Er stellt also eine Sammlung von

[2]Die Spezifikation spricht an diesen Stellen von *Master Data*, *Vocabulary*, *Vocabulary Element* und *Master Data Attribute*.

Standorten zur Verfügung, die von den ereignisgenerierenden Systemen verwendet werden kann.

Weiter ist es möglich, einzelne Wortschatz-Elemente durch zusätzliche Informationen zu ergänzen. Dies wird mit sogenannte Attributen realisiert, die genau einem Element zugeordnet werden. Man wäre also in der Lage – um das Beispiel von eben wieder aufzugreifen – einen bestimmten Standort durch eine Adresse und eine kurze Beschreibung zu ergänzen.

Eine besondere Form von Attributen stellt eine Liste von Elementen des gemeinsamen Wortschatzes dar, die dann dem eigentlichen Element untergeordnet werden. Auf diesem Weg ist es möglich, eine Hierarchie zwischen den einzelnen Wortschatz-Elementen herzustellen. Dies ist zwar ein zunächst relativ abstrakter Einsatzfall von Attributen, kann aber durchaus Vorteile bringen: Über diese Hierarchien lassen sich deutlich komplexere Anwendungfälle darstellen, wie (wieder in unserem Standort-Beispiel) die Zuordnung einzelner Räume zu einem Gebäude. Solche Hierachien können später auch während Abfragen berücksichtigt werden.

Zusätzlich werden Wortschätze durch die EPCIS-Spezifikation in zwei unterschiedliche Sparten eingeteilt:

1. *Allgemeingültige Stammdaten:* Sie betreffen Daten, die allgemein von jedem Unternehmen verwendet werden können, da sie keine spezifischen Informationen enthalten. Dazu gehören zum Beispiel die Definition einzelner Statusbeschreibungen innerhalb eines Geschäftsprozesses. Gerade hier macht es sogar Sinn, wenn alle Handlespartner den gleichen Satz an Statusdefinitionen verwenden, da diese dann auch unternehmensübergreifend verstanden und interpretiert werden können. Entwickelt werden diese allgemeinen Wortschätze von eigens dafür gebildeten Arbeitsgruppen mit Vertretern aus Industrie und Handel.

2. *Anwenderspezifische Stammdaten:* Im Gegensatz zu den allgemeingültigen Stammdaten betreffen diese nur diejenigen Bereiche, welche entweder nur für ein einzelnes Unternehmen interessant sind, bzw. solche, die nicht allgemein definiert werden können. Das Standort-Beispiel gehört sicherlich zu dieser Form der Stammdaten, da jedes Unternehmen schließlich seine eigenen Lokalitäten besitzt. Diese Wortschätze werden somit von den Unternehmen selbst in Zusammenarbeit mit den von diesen Definitionen betroffenen Partnern entwickelt.

Die Einteilung in die beiden Sparten kann unter Umständen sehr subjektiv sein. Technisch gesehen wird allerdings überhaupt nicht zwischen diesen Einordnungen unterschieden, wodurch es innerhalb eines EPCIS-Systems nicht zu Komplikationen kommen dürfte.

3.3. Definition konkreter Ereignisse

In der Daten-Definitionsschicht der EPCIS-Spezifikation werden dann einzelne Ereignisse und die dazugehörigen Datentypen mit Hilfe der oben beschriebenen Grundlagen definiert. Dabei wird zwischen vier Ereignissen zu verschiedenen Anwendungsfällen unterschieden:

1. *Objekt-Ereignis:* Dieses Ereignis wird verwendet, wenn die Informationen sich auf einen oder mehrere EPCs beziehen. Dies ist zum Beispiel der Fall, wenn ein Lesegerät ein bestimmtes Produkt registriert.

2. *Aggregationsereignis:* Ein Aggregationsereignis beschreibt eine Unterordnung einer bestimmten Menge von EPCs zu einem einzelnen Objekt. Ein Beispiel hierfür wäre das Zusammenfassen einiger Produkte auf einer Palette.

3. *Quantitätsereignis:* Bei diesem Ereignis geht es um eine Mengenangabe für eine bestimmte Produktklasse. Dies kann für Inventur-Zwecke eingesetzt werden.

4. *Transaktionsereignis:* Über das Transaktionsereignis kann eine Menge von EPCs explizit mit einer Transaktion, also zum Beispiel einer offenen Bestellung verknüpft werden.

Zusätzlich existiert noch ein weiteres Ereignis: Das abstrakte *EPCIS-Ereignis* bildet die Basis für alle anderen Ereignisse. Eine direkte Instanz kann somit natürlich nicht erzeugt werden, allerdings ist durch die Vererbung davon auszugehen, dass alle anderen Ereignisse auch gleichzeitig ein EPCIS-Ereignis sind.

Wie bereits beschrieben, besitzt jedes dieser Ereignisse sogenannte Felder, welche die eigentlichen Informationsdaten tragen. Um solche Felder und ihre jeweiligen Datentypen definieren zu können, werden vier Schlüsseldimensionen festgelegt, die Aspekte wie *Bezugsobjekt*, *Zeit*, *Ort* und *Geschäftkontext* berücksichtigen. Jedes definierte EPCIS-Ereignis sollte diese vier Dimensionen besitzen.

3.3.1. Schlüsseldimensionen eines Ereignisses

Jede der Dimensionen erfüllt ihren Zweck, indem sie eine bestimmte Frage zum jeweiligen EPCIS-Ereignis beantwortet:

1. *Was ist passiert?* Diese Dimension beschreibt den eigentlichen Gegenstand des Ereignisses. Dies kann der EPC des betreffenden Objektes, eine Mengenangabe für eine bestimmte Objekt-Klasse (beim Quantitäts-Ereignis), aber auch eine Transaktion (beim Transaktion-Ereignis) sein.

2. *Wann ist es passiert?* An dieser Stelle wird der zeitliche Aspekt berücksichtigt. Dieser Fall ist normalerweise recht einfach zu realisieren, da bei jedem Ereignis das Datum und die Uhrzeit gespeichert wird, zu der es eingetreten ist.

3. *Wo ist es passiert?* Die Ortsdimension wird in zwei einzelne Sichtweisen unterteilt:

- Die retrospektive Sichtweise bezieht sich auf den Ort der Erfassung. Dies ist normalerweise der Punkt, an dem des Lesegerät steht.

- Die prospektive Sichtweise auf ein Ereignis beschreibt den Ort, an dem sich das Objekt nach der Erfassung befindet. Es resultiert also in der Regel aus bereits eingetretenen Ereignissen und kann sich zeitlich über weitere Ereignisse erstrecken. Diese Sichtweise wird als *Geschäftslokation* bezeichnet.

Ganz grob zusammengefasst kann man die Geschäftslokation mit einem *Raum* und den Lesepunkt mit einer *Tür* zu diesem Raum vergleichen.

4. *Warum ist es passiert?* Die letzte Dimension bezieht sich auf den Geschäfts-Kontext des Ereignisses. Auch hier wird erneut zwischen einer rückblickenden und einer vorausschauenden Sichtweise unterschieden:

   - Der retrospektive Aspekt beschreibt den aktuellen Prozessschritt eines Ereignisses. Dieser bezieht sich wiederum genau auf die Stelle, an der das Ereignis eingetreten ist. Beispiele sind „abgezeichnet“ und „verschickt“.

   - Der prospektive Aspekt des Geschäft-Kontextes bezieht sich wieder auf einen weitläufigeren Bereich: Er ist mehr als eine Art Status innerhalb der Prozesskette zu sehen und kann sich somit auch wieder über spätere Ereignisse erstrecken. Beispiel solcher Dispositionschritte sind „in Bearbeitung“ oder „bereit zum Verkauf“.

3.3.2. Ereignisfelder und ihre Datentypen

Diese Dimensionen werden durch entsprechende Felder innerhalb der Ereignisse dargestellt. Daher definiert die EPCIS-Spezifikation verschiedener solcher Eigenschaften mit den dazugehörige Datentypen, um die Schlüsseldimensionen innerhalb der Ereignisse abbilden zu können.

Bis auf wenige Ausnahmen werden möglichen Werte, die ein Feld annehmen kann, durch einen Wortschatz innerhalb der Stammdaten definiert.

3.3.2.1. Felder mit primitiven Datentypen

Viele verwendeten Felder verlangen nicht nach einem aufwendigen Datentyp, sondern es reicht ein relativ einfacher Typ aus.

Es gibt genau drei dieser primitiven Datentypen, die innerhalb der Ereignisse immer wieder Verwendung finden:

Ganzzahl — Für Zahlenangaben existiert der Integer-Typ.

Zeit — Dieser Datentyp wird für Zeitangaben eingesetzt. Dargestellt wird er durch das *xsl:DateTime*-Schema.

EPC — Zu guter letzt fehlt noch ein Typ, um EPCs zu beschreiben. Hierfür wird normalerweise eine Darstellung als URI gewählt, die in der Lage ist, alle verschiedenen EPC-Klassen abzudecken.

3.3.2.2. Das Aktions-Feld

Das sehr spezielle Aktions-Feld (englische Bezeichnung: *Action*) beschreibt, wie sich das Ereignis auf den Lebenlauf des enthaltenen Objektes bezieht.

Der Datentyp dieses Feldes verkörpert technisch gesehen eine Enumeration, die folgende Werte annehmen kann:

`ADD` — Das enthaltene Objekt wird hinzugefügt.

`DELETE` — Das Objekt wird wieder entfernt.

`OBSERVE` — Das Objekt bleibt unverändert (das heißt genauer, dass es weder hinzugefügt, noch entfernt wird).

Eine exaktere Bedeutung bekommen diese Werte erst durch den Zusammenhang mit dem jeweiligen Ereignis.

3.3.2.3. Die EPC-Klasse

Der Datetyp für die EPC-Klasse (englische Bezeichnung: *EPC Class*) beschreibt eine bestimmte Produktsorte, ist also vergleichbar mit einer Artikelnummer.

Da dies sehr abhängig vom jeweiligen Hersteller ist, werden die möglichen Werte auch vom diesem selbst definiert und somit auch den anwenderspezifischen Wortschätzen zugeordnet. Allerdings ist eine allgemeine Schreibweise für diese Elemente vorgesehen, die besagt, dass die Produktklassen wie die späteren EPCs aussehen sollten, deren Seriennummer durch einen * ersetzt wird. Dadurch lassen sich EPCs später eindeutig einer EPC-Klasse zuordnen.

3.3.2.4. Beschreibung der Geschäftstransaktion

Die EPCIS-Spezifikation sieht für die Definition von Geschäftstransaktionen (Bezeichnung in der Spezifikation: *Business Transaction*) zwei einzelne Datentypen vor, die dann in Kombination die eigentliche Transaktion beschreiben:

Transaktionsart	Dieser Datentyp beschreibt, um was für eine Art von Transaktion es sich handelt.
Transaktion	Dieser Typ benennt die Transaktion selbst, deren Typ durch die Transaktionsart beschrieben wurde.

Hierbei ist zu beachten, dass die möglichen Werte für die Transaktionsart durch einen allgemeingültigen und die für die eigentliche Transaktion durch einen anwenderspezifischen Wortschatz definiert werden. Ein Beispiel ist eine allgemeine Definition der Transaktionsart „Offene Bestellung“, dem dann zum Beispiel individuelle Bestellnummern zugeordnet werden.

3.3.2.5. Felder für Lokalitäten

Wie bereits beschrieben, können Orte in einem EPCIS-System in zwei unterschiedlichen Sichtweisen beschrieben werden, je nachdem, ob der retrospektive oder der prospektive Aspekt hervorgehoben werden soll.

Aus diesem Grund werden zwei Felder für die Beschreibung von Lokalitäten eingeführt:

Lesepunkt	Er beschreibt die rückblickende Sicht auf den Ort, also auf einen festen Punkt, an dem das Ereignis eingetreten ist. Die englische Bezeichnung für diesens Feld lautet *Read Point*.
Geschäftslokation	Im Gegensatz zum Lesepunkt wird hier die vorausblickende Sicht auf die Lokalität beschrieben. Sie wird mit *Business Location* bezeichnet.

Keines dieser beiden Felder enthält die genaue Bezeichnung oder Nummer des physikalischen Lesegerätes. In der EPCIS-Spezifikation werden ausschließlich Lokalitäten verwendet, die sich auf Punkte innerhalb der Prozesskette beziehen (es ist also mehr eine abstrakte Sichtweise auf das ganze Geschehen).

Die möglichen Werte von Lesepunkt und Geschäfts-Lokation werden in anwenderspezifischen Wortschätzen definiert.

3.3.2.6. Felder für den Geschäftskontext

Auch für den Geschäftskontext werden aufgrund der unterschiedlichen Sichtweisen zwei Felder eingeführt:

Prozessschritt	Dieses Feld beschreibt die retrospektive Sicht auf den Geschäftskontext und wird mit *Business Step* bezeichnet.
Dispositionsschritt	Das Gegenstück dazu bildet das Feld des Dispositionsschrittes (englische Bezeichnung: *Disposition*), der den prospektiven Aspekt vertritt.

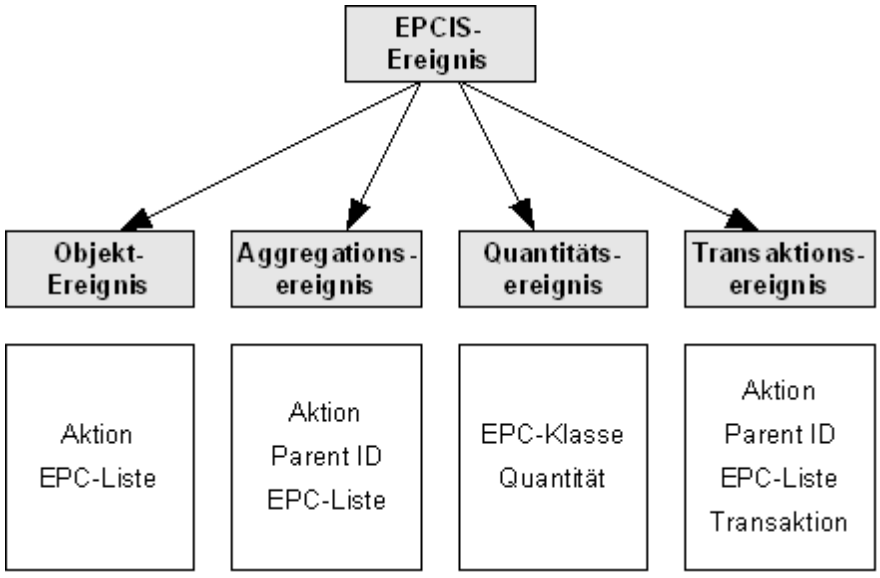
Die Werte beider Datentypen werden in allgemeingültigen Wortschätzen definiert, die von den entsprechenden Gremien entwickelt werden.

3.3.3. Das abstrakte EPCIS-Ereignis

Wie bereits erwähnt definiert die EPCIS-Spezifikation ein abstraktes Ereignis, von dem alle anderen Ereignisse erben müssen. Man kann also davon ausgehen, dass alle Ereignisse, die durch ein EPCIS-System erstellt und gespeichert worden sind, auch ein EPCIS-Ereignis repräsentieren.

Dies bringt enorme Vorteile mit sich, gerade auch während einer weiteren Implementierung solcher Komponenten. Es ist also auf jeden Fall ratsam, sollte man neue Ereignisse entwickeln wollen – was natürlich grundsätzlich möglich ist – dieses dann auch vom abstrakten EPCIS-Ereignis erben lassen.

Abbildung 3.3. Übersicht über die EPCIS-Ereignisarten



Hinweis:

Die Felder der abgebildeten Ereignisse sind nicht vollständig. Es wurde lediglich versucht, die unterschiedlichen Ereignisarten mit denen für sie typischen Eigenschaften darzustellen.

Das EPCIS-Ereignis selbst besitzt nur zwei Felder:

- Das erste enthält eine Angabe für den Zeitpunkt, an dem das Ereignis eingetreten ist. Gesetzt wird dieses Feld durch die Client-Applikation, die das Ereignis generiert. Man bezeichnet es als *Ereigniszeit* bzw. als *Event Time*.

- Das zweite Feld beschreibt den Zeitpunkt des Speichervorgangs durch den EPCIS-Server. Diese Eigenschaft wird als *Registrierungszeit* (oder englisch als *Record Time*) bezeichnet. Gesetzt werden sollte sie natürlich erst serverseitig nach dem Erfassungsvorgang, was aber auch nicht zwingend geschehen muss, da dieser Wert optional ist.

Da jedes abgeleitete Ereignis durch die Vererbung automatisch einen Zeitpunkt für sein Eintreten enthält, ist die zeitliche Schlüsseldimension bereits in allen Ereignissen integriert.

Zusätzlich enthält jedes durch die Spezifikation definierte Ereignis vier Felder, welche die Dimensionen für die Lokalität und den Geschäftskontext beschreiben. Dies ermöglicht eine Einordnung eines jeden Ereignisses in die bereits beschriebenen Sichtweisen der entsprechenden Dimensionen. Allerdings werden diese Felder nicht im abstrakten EPCIS-Ereignis definiert, sondern erst in den abgeleiteten Ereignissen. Dies hat vermutlich den Grund, dass man die Möglichkeit offen lassen will, die beiden Dimensionen in späteren Ereignis-Definitionen auf einem anderen Weg zu beschreiben.

Zusammenfassend heißt dies, dass alle durch die EPCIS-Spezifikation definierten Ereignisse nach dem gleichen Schema beschreiben, *wann*, *wo* und *warum* ein Ereignis eingetreten ist. Individuell definiert werden muss somit nur der Gegenstand des jeweiligen Ereignisses, also *was* eigentlich geschehen ist.

3.3.4. Objekt-Ereignis

Das Objekt-Ereignis tritt ein, wenn Informationen zu einem oder mehreren EPCs gespeichert werden müssen. Dies kommt normalerweise sehr häufig vor, wenn einzelne Objekte einen bestimmten Lesepunkt innerhalb der Prozesskette passieren, also beispielsweise „Ein bestimmtes Objekt passiert den Eingang

zum Lager durch Tor 4.“. Zusätzlich kann ein Objekt-Ereignis dazu genutzt werden, um neue EPCs zu registrieren und die Registrierungen wieder aufzuheben.

Um dies beschreiben zu können, definiert die EPCIS-Spezifikation neben den bereits genannten Feldern weitere Eigenschaften:

- Eine *EPC-Liste*, welche die einzelnen EPCs der betreffenden Objekte enthält.
- Ein *Aktionsfeld*, das die Beziehung dieser EPCs zum Ereignis beschreibt: `ADD` bedeutet, dass die Objekte als neu registriert werden, also zum Beispiel direkt nach der Herstellung. Mit `DELETE` werden die Objekte innerhalb der Prozesskette als *bearbeitet* definiert. Sie stehen dann für spätere Ereignisse nicht mehr zur Verfügung. Zu guter Letzt bedeutet `OBSERVE` eine ausschließliche Beobachtung der Objekte. Dieser häufige Fall würde also höchst wahrscheinlich während des oben genannten Beispieles Verwendung finden.

Zusätzlich enthält das Objekt-Ereignis noch eine *Liste für Transaktionen*, um die Möglichkeit zu geben, dieses Ereignis direkt mit einer bestehenden Transaktion – wie zum Beispiel einer offenen Bestellung – in Verbindung zu bringen.

Logisch betrachtet bezieht sich natürlich jedes Objekt-Ereignis genau auf einen EPC, da ja durch die eindeutige Identfikation jedes Objekt für sich alleine steht. Durch den Einsatz einer Liste soll somit also keine logische Verknüpfung zwischen den einzelnen EPCs hergestellt werden, sondern es ist mehr als eine Kurzform zu verstehen (um nicht zu viele nahezu gleiche Ereignisse erstellen zu müssen), die dann für die Speicherung im EPCIS-Server auch wieder aufgelöst werden kann, wenn dies Vorteile bringen sollte.

3.3.5. Aggregationsereignis

Das Aggregationsereignis wird eingesetzt, um eine bestimmte Menge von Objekten einem anderen unterzuordnen. Sie werden dann im weiteren Pro-

zessverlauf als eine Einheit betrachtet. Dies ist oft der Fall, wenn zum Beispiel einzelne Produkte auf einer Palette zusammengefasst werden.

Zur Darstellung eines solchen Prozesses werden neben den bekannten Feldern wieder zusätzliche definiert:

- Ein Feld für die *übergeordnete Einheit* (*Parent ID*). Diese wird in der Regel durch einen EPC gekennzeichnet, es wäre aber zum Beispiel auch eine URL denkbar, die weitere Informationen enthält. In unserem Beispiel entspräche dieses Feld der Palette.

- Hinzu kommt eine *EPC-Liste*, welche die untergeordneten Objekte enthält. Diese werden daher auch als *Child EPCs* bezeichnet. Sie entsprechen im Beispiel den Produkten, die der Palette hinzugefügt werden.

- Auch innerhalb dieses Ereignisses beschreibt ein *Aktionsfeld*, wie das Ereignis zu verstehen ist: Dabei wird `ADD` eingesetzt, um die enthaltenen EPCs zur Verknüpfung hinzuzufügen, durch `DELETE` können sie dann wieder entfernt werden, wobei der Spezielfall einer leeren EPC-Liste bedeutet, dass alle EPCs von der Assoziation gelöst werden sollen. Der seltene Fall `OBSERVE` wird dann eingesetzt, wenn zum Beispiel die ID der Palette nicht bekannt ist.

Auch für dieses Ereignis existiert ein *Transaktionsfeld*, dass analog zum Objekt-Ereignis dazu verwendet werden kann, das Ereignis mit einer bestehenden Transaktion zu verknüpfen.

Das Aggregationsereignis sollte nicht für sogenannte „schwache Elemente“ eingesetzt werden. Das heißt genauer, dass nur möglichst durch einen EPC gekennzeichnete, reale Objekte aggregiert werden sollten, im Gegensatz zu beispielsweise einer Aggregation eines Produktes mit einer Transaktion.

Die EPCIS-Spezifikation erwähnt eine Fehleranfälligkeit, welche dieses Ereignis leider mit sich führt. Dazu gehört zum Beispiel die Möglichkeit einer Rekursion, wenn für die über- und die untergeordneten Felder der gleiche EPC ver-

wendet wird. Diese Ausnahmen werden allerdings durch die Spezifikation nicht weiter behandelt, so dass sich die entsprechenden Implementierungen mit ihnen auseinander setzen müssen.

3.3.6. Quantitätsereignis

Quantitätsereignisse treten ein, wenn eine Mengenangabe zu einem bestimmten Artikel gemacht werden muss. Dabei wird schnell klar, dass an dieser Stelle die Fähigkeit, jedes Objekt eindeutig identifizieren zu können, nicht weiter benötigt wird. Vielmehr müssen Angaben wie „Es befinden sich noch 15 Pakete Kaffee im Lager.“ gemacht werden. Das Quantitätsereignis wird also für Inventuren und zur Speicherung von Bestandsinformationen genutzt.

Aus diesem Grund gestalten sich auch die Eigenschaften, die neben den bereits bekannten verwendet werden, relativ übersichtlich:

- Jedes Quantitätsereignis besitzt ein Feld, welches die *EPC-Klasse* der entsprechenden Objekte beschreibt. Dies wäre in unserem Beispiel die Klassenbezeichnung für „Kaffee“.
- Als zweites Feld enthält das Ereignis die *Quantität* zu der spezifizierten EPC-Klasse, also einen einfachen Zahlenwert, der die Mengenangabe beschreibt.

Ein Aktionsfeld wird für dieses Ereignis nicht benötigt, da es eigentlich keinen Prozess beschreibt, sondern mehr einen zwischenzeitlichen Status meldet.

3.3.7. Transaktionsereignis

Das vierte und letzte Ereignis, das durch die EPCIS-Spezifikation definiert wird, ist das Transaktionsereignis. Es ermöglicht eine Verknüpfung von einzelnen Objekten mit bestehenden Geschäftstransaktionen, wie zum Beispiel einer offenen Bestellung.

Eine solche Verknüpfung ist zwar grundsätzlich auch durch Angabe der gewünschten Transaktion in den meisten anderen Ereignissen möglich, allerdings kann dies auch explizit durch das Transaktions-Ereignis geschehen,

wodurch die eigentliche Verknüfung in den Vordergrund rückt. Analog kann auf diesem Weg auch eine entsprechende Trennung von einer Transaktion geschehen.

Somit werden auch an dieser Stelle noch weitere Felder definiert, die eine solche Assoziation möglich machen:

- Die wesentliche Komponente stellt eine *Liste von Transaktionen* dar. Sie enthält alle Geschäftstransaktionen, die für die Verknüfung berücksichtigt werden sollen.

- Die zu verknüpfenden Objekte werden neben einer *EPC-Liste* auch durch ein *Parent-ID-Feld* repräsentiert. So wird eine Möglichkeit offeriert, auch den Aggregationsstand der Elemente hervorzuheben.

- Zu guter Letzt ist natürlich auch in diesem Ereignis ein *Aktionsfeld* definiert. Nimmt es den Wert `ADD` an, so werden die angegebenen Objekte den Transaktionen zugeordnet, bei `DELETE` entsprechend wieder von ihnen getrennt. Auch hier gibt es den speziellen Fall einer leeren EPC-Liste, in dem alle Objekte von einer Transaktionen entfernt werden. Zusätzlich kann noch der seltene `OBSERVE`-Fall, der lediglich bekräftigen soll, dass sich die Verknüpfungen nicht ändern.

3.4. Erfassungsschnittstelle

Die dritte Schicht der EPCIS-Spezifikation definiert den Service-Aufbau für die Verwaltung von EPCIS-Daten, insbesondere der soeben beschriebenen Ereignisse. Im Wesentlichen werden innerhalb dieser Schicht zwei Schnittstellen definiert, über die ein Zugriff auf den EPCIS-Server möglich wird. Die erste ist die Erfassungsschnittstelle, welche in diesem Abschnitt vorgestellt werden soll.

Die Erfassungsschnittstelle des EPCIS-Servers bietet den Client-Applikationen einen Dienst an, um fertig generierte Ereignisse im servereigenen Datenverzeichnis zu speichern. Dies wird in der Theorie hauptsächlich durch das Netz-

werkmitglied geschehen, das den jeweiligen Informationsserver pflegt. Grundsätzlich ist es aber auch denkbar, dass ein Außenstehender mit den erforderlichen Rechten ausgestattet wird, um Ereignisse mittels entferntem Zugriff auf dem EPCIS-Server zu speichern.

Eine Speicherung von Stammdaten, die ja auch vom EPCIS-System angeboten werden, ist über diese Schnittstelle nicht vorgesehen. Diese Art von Daten ist insgesamt ohnehin mehr von statischer Natur, und eine Service-Schnittstelle für die Erfassung somit nicht unbedingt notwendig.

3.4.1. Aufbau der Schnittstelle

Die Schnittstelle selbst beschränkt sich auf eine einzige Operation: Sie wird mit `capture()` bezeichnet und erhält als einzigen Parameter eine Liste von EPCIS-Ereignissen (durch die Erbung vom abstrakten EPCIS-Ereignis ist hier natürlich jedes der definierten Ereignisse möglich), die dann vom Server gespeichert werden.

Während einer Implementierung des EPCIS-Servers sollte darauf geachtet werden, dass jedes Ereignis, welches an dieser Schnittstelle ankommt, einen Wert für die Registrierungszeit erhält, sollte dieser noch nicht gesetzt sein.

Um die Sicherheit für diese Service-Operation zu garantieren, werden durch die EPCIS-Spezifikation keine speziellen Vorkehrungen getroffen. Der Server kann einfach über die authentifizierte Identität des EPCIS-Clients entscheiden, ob er die Ausführung der Operation erlaubt oder nicht. In der Theorie sollte ein solcher Zugriff ohnehin mehr das interne Netz eines Unternehmens betreffen.

Mehr muss diese sehr einfache Schnittstelle tatsächlich nicht leisten können.

3.4.2. Technische Umsetzungen

Natürlich gibt es viele verschiedene Möglichkeiten, eine solche Schnittstelle zu implementieren. Die Entscheidung für einen bestimmten Weg kann einerseits aufgrund der technischen Voraussetzungen des Systems, durch die Möglich-

keiten der Client-Applikation, aber auch aufgrund von Vorteilen innerhalb des Anwendungsbereiches getroffen werden.

Wenn man zum Beispiel davon ausgeht, dass sehr viele Daten auf einmal gespeichert werden müssen, wie dies während der Pulk-Erfassung einer kompletten Lieferung geschehen kann, so läge der Schwerpunkt möglicherweise auf einem zeitlichen Aspekt. In einem solchen Fall wäre eine schnelle Übertragungungstechnik von Vorteil, die bestenfalls noch unabhängig von Antworten des Servers arbeiten kann.

Die Entscheidung, welche Technologie den meisten Nutzen bringt, bleibt also vollständig den implementierenden Unternehmen überlassen. Die EPCIS-Spezifikation selbst macht lediglich zwei Vorschläge:

- Implementierung in Form einer *Message Queue*: Die auch als *Warteschlange* bezeichnete Datenstruktur kann die einzelnen Nachrichten so zwischenspeichern, dass sie erst später durch den EPCIS-Server abgearbeitet werden müssen. Dabei werden zuerst diejenigen Elemente aus der Schlange genommen, die am längsten auf eine Bearbeitung warten mussten. Auf diesem Weg kann der EPCIS-Server sehr große Mengen von Daten sicher auf einmal annehmen, ohne sie sofort auswerten zu müssen.

- Implementierung durch einen einfachen *HTTP-Datenaustausch*: Der zweite Vorschlag behandelt eine äußerst einfache Implementierung auf Basis von HTTP. Hierbei werden die Ereignisse durch eine XML-Darstellung repräsentiert (eine solche Darstellung existiert bereits im Zusammenhang mit der später vorgestellten Abfrage-Schnittstelle), die über HTTP als POST-Request an den EPCIS-Server geschickt werden. Dieser kann dann die enthaltenen Daten auswerten und speichern.

Auch wenn die Spezifikation nur diese beiden Möglichkeiten einer Implementierung beschreibt, so ist natürlich trotzdem auch der Einsatz anderer Wege – wie zum Beispiel als Web Service – für diese Schnittstelle denkbar.

3.5. Abfrage-Schnittstelle

Die zweite Schnittstelle eines EPCIS-Servers ist um einiges aufwendiger als die gerade beschriebene Erfassungsschnittstelle. In erster Linie ist sie für Client-Applikationen gedacht, die EPCIS-Daten benötigen, welche bereits auf dem Server gespeichert wurden. Sie bietet daher Operationen an, über die sich solche Daten abfragen lassen. Aus diesem Grund wird dieser Teil der Service-Schicht als *Abfrage-Schnittstelle* bezeichnet. Neben dieser Funktionalität werden aber auch weitere Operationen angeboten, unter anderem für die Beschaffung von Informationen zum verwendeten EPCIS-Server. Auch sie gehören zu dieser Schnittstelle.

Sollen EPCIS-Daten abgefragt werden, so muss zunächst eine spezielle Abfrage definiert werden. Ohne eine solche Definition ist kein Zugriff auf EPCIS-Daten möglich. Der Zugriff selbst kann dann mit Hilfe dieser Abfragen sehr genau eingegrenzt und anschließend synchron, aber auch asynchron ausgeführt werden.

Die Einzelheiten zu diesen Zugriffen werden in den folgenden Abschnitten geneuer beschrieben.

3.5.1. Allgemeiner Aufbau von Abfragen

Um auf EPCIS-Daten innerhalb des Servers zugreifen zu können, wird eine spezielle Abfrage benötigt. Da solche Abfragen unterschiedliche Typen haben können, müssen sie sowohl auf der Client- als auch der Server-Applikation bekannt sein.

Jede dieser Abfragen besitzt zwei wesentliche Bestandteile:

- Einen *eindeutigen Namen*: Dieser Name wird als Schlüssel verwendet, um unterscheiden zu können, was für eine Art von Abfrage es sich handelt. Die Bezeichnung *Name* (englische Bezeichnung: *Query Name*) ist daher unter

Umständen etwas missverständlich: Eigentlich wird durch diesen Namen eher der Typ der Abfrage beschrieben.

- Eine *Liste von Parametern*: Diese Liste enthält verschiedene Schlüssel-Wert-Paare, über die später das Ergebnis der Abfrage eingegrenzt wird. Die Werte wiederum können unterschiedlichste Typen haben. Jeder Abfragetyp besitzt einen fest definierten Katalog solcher Parametern, der bei einer Formulierung herangezogen wird.

Diese Parameter werden dann während eines Zugriffes vom EPCIS-Server genutzt, um die jeweilige Ergebnis-Menge zusammenzustellen. Dabei enthalten die Parameter keinerlei Informationen zu ihrer eigentlichen Bedeutung, wodurch schnell klar wird, dass der Server die Parameter kennen muss, um ein Ergebnis zu erzeugen. Dies erkärt auch, warum es auf jeden Fall notwendig sein muss, eine Abfrage zunächst zu definieren.

Die EPCIS-Spezifikation sieht eine grundsätzliche Möglichkeit zur Definition von Abfragetypen über die Schnittstelle vor. Dies wird allerdings in Version 1.0 noch nicht weiter verfolgt und braucht somit auch noch nicht implementiert zu werden. Stattdessen existieren bereits vordefinierte Typen für Abfragen, die jeder EPCIS-Server verstehen muss und die von Client-Applikationen genutzt werden können, um Daten vom Server zu bekommen.

3.5.2. Synchroner und asynchroner Zugriff

Um bei der Nutzung möglichst flexibel zu sein, sieht die EPCIS-Spezifikation zwei Wege vor, eine Abfrage auszuführen:

- *Synchroner Zugriff:* Die Möglichkeit, eine Abfrage synchron zu starten, funktioniert ähnlich einer klassischen RPC-Ausführung. Hierbei werden die nötigen Parameter der Abfrage von der Client-Anwendung gesetzt und anschließend an den Server geschickt. Dann wartet sie so lange, bis die entsprechende Antwort zurückgeliefert wird.

- *Asychroner Zugriff:* Diese Möglichkeit ist ein deutliches Stück komplexer als der synchrone Zugriff. Hierbei wird zusätzlich zur Abfrage ein sogenanntes Abonnement erstellt, welches neben zeitlichen Daten für eine regelmäßige Ausführung auch ein Ziel definiert. Dieses Abonnement wird neben den Abfragedaten beim EPCIS-Server registriert. Auf diesen Weg kann der Server die Abfrage periodisch ausführenen und das Ergebnis anschließend an das vordefinierte Ziel schicken, ohne dass der Client die ganze Zeit darauf warten muss.

Auf die verschiedenen Einstellungsmöglichkeiten eines solchen Abonnements soll an dieser Stelle nicht weiter eingegangen werden. Es sei nur erwähnt, dass es die Möglichkeiten gibt, sich lediglich neu hinzugekommene Daten schicken zu lassen, auf das Versenden zu verzichten, sollte das Ergebnis leer sein und dass für die Definition der periodische Ausführung eine äußerst komplexe Darstellungsform mit speziellen Parametern und sogar einer eigenen Grammatik existieren, um sehr differenziert Zeitintervalle darstellen zu können.

Die Spezifikation macht keinerlei Angaben, was unter dem Begriff *Ziel* genau zu verstehen ist. Denkbar wäre hier im einfachsten Fall ein Verschicken über SMTP, aber auch zum Beispiel ein Service-Aufruf einer anderen Schnittstelle.

3.5.3. Aufbau der Schnittstelle

Mit Hilfe dieser theoretischen Grundlagen einer EPCIS-Abfrage lassen sich nun einzelne Operationen definieren, die solche Zugriffe möglich machen sollen. Man kann diese grundsätzlich in drei Sparten aufteilen: Eine, die rein informative Daten zum verwendeten Server liefert, eine weitere, über die sich Abonnements verwalten lassen und zu guter Letzt eine Operation, über welche sich Abfragen ausführen lassen.

Informative Operationen der Schnittstelle sind:

`getStandardVersion()` Diese Operation liefert die Version der verwendeten EPCIS-Spezifikation zurück. Dies

ist normalerweise einfach `1.0`, sollte diese Spezifikationsversion verwendet worden sein.

`getVendorVersion()` Im Gegensatz zu `getStandardVersion()` liefert diese Operation die herstellerspezifische Version zurück. Hier wird Bezug auf die Möglichkeit genommen, dass sich fast alle Teile eines EPCIS-Systems von Herstellerseite aus erweitern lassen. Die Spezifikation empfiehlt als Rückgabewert eine URN-Bezeichnung, welche diese erweiterte Version beschreibt oder aber auch eine URL, unter der sich weitere Informationen zur Version abrufen lassen. Ist das Ergebnis leer, so kann davon ausgegangen werden, dass es keine herstellerspezifischen Erweiterungen gibt.

`getQueryNames()` Diese Operation liefert alle Typbezeichnungen für Abfragen zurück, die dem Server bekannt sind. Dazu gehören natürlich auch die vordefinierten Standard-Abfragen, die ohnehin jeder Server anbieten sollte.

Weiter werden Operationen angeboten, über die sich Abonnements organisieren lassen:

`subscribe()` Diese Operation ermöglicht die Registrierung eines Abonnements. Hierfür müssen Steuerparameter und die betreffende Abfrage übergeben werden. Zusätzlich wird ein eindeutiger Name für die Registrierung festgelegt, über den später auf das Abonnement zugegriffen werden kann.

`unsubscribe()` Über diese Operation kann ein bereits registriertes Abonnement wieder gelöscht werden.

`getSubscriptionIDs()` Diese Funktion liefert alle Bezeichner der zur Zeit registrierten Abonnements zurück.

Zu guter Letzt fehlt nur noch die Operation für den synchronen Aufruf:

`poll()` Über diese Funktion können Abfragen direkt ausgeführt werden. Übergeben werden Parameter, die der Server nutzt, um das Ergebnis einzugrenzen. Das zurückgelieferte Ergebnis besteht normalerweise aus einer bestimmten Menge von EPCIS-Daten.

3.5.4. Vordefinierte Abfragen

Wie bereits erwähnt ist es noch nicht möglich, über Schnittstellen eigene Abfragen zu definieren: Diese zwar bereits vorgesehenen Operationen sind erst für zukünftige EPCIS-Versionen vorgesehen. Stattdessen müssen Abfragen mehr oder weniger statisch während der Implementierung des Servers integriert werden. Diese werden als *vordefinierte Abfragen* (*Predefined Queries*) bezeichnet, obwohl auch hier eigentlich wieder der Typ der Abfrage gemeint ist.

Die EPCIS-Spezifikation in der Version 1.0 definiert genau zwei solche Abfragen, die jede Server-Implementierung unterstützen muss. Diese sind einfach aufgebaut und haben den Sinn, möglichst allgemein eine bestimmte Menge von Elementen einer der beiden EPCIS-Datentypen abrufen: Eine für gespeicherte Ereignisse, eine für Stammdaten.

3.5.4.1. Einfache Ereignisabfrage

Die einfache Ereignisabfrage[3] stellt ein umfangreiches Sortiment an Parametern zur Verfügung, um Ereignisse auf dem EPCIS-Server abzufragen und die Zusammenstellung im Ergebnis zu steuern. Der größte Teil dieser Parameter

[3]Die Bezeichnung in der EPCIS-Spezifiaktion lautet *SimpleEventQuery*.

bezieht sich auf Felder der bereits vorgestellten EPCIS-Ereignisse. Werden Parameterwerte gesetzt, so kann auf diesem Weg das Ergebnis beeinflusst werden. Das Ergebnis selbst ist immer eine (evtl. leere) Liste von EPCIS-Ereignissen. Darüber hinaus existieren noch weitere Parameter, zum Beispiel für Sortierung und Limitierung des Ergebnisses.

Das Prinzip, nach dem der EPCIS-Server das Ergebnis der Abfrage zusammenstellt ist recht einfach: Es werden alle Ereignisse der Ergebnismenge hinzugefügt, die bestimmten Kriterien entsprechen. Diese Kriterien werden über die Parameter konfiguriert. Die EPCIS-Spezifikation definiert daher in erster Linie Parameter, die sich wiederum den einzelnen Schlüsseldimensionen der Ereignisse zuordnen lassen, um so eine Filterung nach diesen Werten zu ermöglichen:

- *Zeitliche Dimension:* Für den zeitlichen Aspekt eines Ereignisses stehen Parameter zur Filterung nach Ereignis- und Registrierungszeit zur Verfügung.

- *Ortsspezifische Dimension:* Auch die Lokalitätsfelder der Ereignisse können über Parameter abgefragt werden. Zusätzlich gibt es weitere Parameter, welche die vordefinierten Hierarchien aus dem zugehörigen Stammdatenbestand berücksichtigen.

- *Dimension Geschäftkontext:* Natürlich ist es auch möglich, Ereignisse nach ihrem Geschäftskontext einzugrenzen.

- *Objektspezifische Dimension:* Zu guter Letzt existieren Parameter, welche die individuellen Eigenschaften eines Ereignisses betreffen. Gerade für diesen Bereich existieren extrem viele Parameter, die aber insgesamt betrachtet wiederum nur die einzelnen, speziellen Felder der unterschiedlichen Ereignisarten abdecken. Hier kann zum Beispiel nach Werten von EPCs, übergeordneten Objekten, Quantität und Transaktionen gefiltert werden.

Weitere Parameter bieten eine grundsätzliche Filterung nach Ereignisart und Aktionstyp, der ja auch bei fast allen Ereignissen vorhanden ist. Natürlich ist

neben dem Transaktionsereignis auch bei den anderen EPCIS-Ereignissen eine Abfrage mit Berücksichtigung der zugehörigen Transaktionen zulässig. Zusätzlich existieren spezielle Parameter für Erweiterungsfelder und Attribute, die mit entsprechenden Ereignisfeldern verknüpft sein könnten:

- *Erweiterungen:* Da jedes Ereignis nach den Grundprinzipien der EPCIS-Spezifikation erweiterbar ist und es grundsätzlich auch möglich wäre, neue Ereignisse zu definieren, werden neben den oben vorgestellten Parametern auch solche angeboten, über die sich Felder ansprechen lassen, deren Namen während der Definition noch nicht bekannt waren. Auch ein Parameter für eine grundsätzliche Existensprüfung eines Erweiterungsfeldes ist vorhanden.

- *Attribute:* Da viele der Ereignisfelder Werte enthalten, die innerhalb der Stammdaten definiert sind und wiederum mit Attributen verknüpft sein können, bietet die Spezifikation zusätzlich spezielle Parameter an, über die auch eine Filterung nach Attributen möglich wird.

Zu guter Letzt gibt es noch eine Menge von Parametern, durch die sich der Aufbau der Ergebnismenge beeinflussen lässt. Dazu gehören:

- Die Angabe eines Feldnamens, nach dem das Ergebnis sortiert wird,

- eine Festlegung für die Sortierrichtung, also ob das Ergebnis aufsteigend oder absteigend sortiert wird

- und die Angabe eines Wertes für die maximale Anzahl von Ereignissen, die das Ergebnis enthalten darf.

Hinweis:

Eine genauere Beschreibung aller Abfrage-Parameter, die dann auch die exakten Namen und die mit ihnen verknüpften Datentypen berücksichtigt, findet sich im Anhang dieser Arbeit.

Zur Ergebnismenge fügt der EPCIS-Server nur diejenigen Ereignisse hinzu, für die alle angegebenen Parameter eine wahre Aussage ergeben. Sind mehrere Parameter gesetzt, so müssen sie alle auf das entsprechende Ergebnis zutreffen. Man kann hier also von einer `AND`-Verknüpfung sprechen. Im Gegensatz dazu handelt es sich um eine `OR`-Verknüpfung, wenn einem Parameter mehrere Werte zugordnet sind, was häufig zulässig ist. Dies liegt daran, dass in einem solchen Fall nur einer dieser Werte auf das entsprechende Ereignis zutreffen muss.

Dieser nicht mehr ganz triviale Zusammenhang soll an Hand eines kleinen Beispiels verdeutlicht werden:

Folgende Werte seien gesetzt:

1. `eventType := ObjectEvent, TransactionEvent`

2. `EQ_action := ADD, DELETE`

Das Ergebnis enthält also alle Ereignisse, die entweder vom Typ `ObjectEvent` oder vom Typ `TransactionEvent` sind und deren Aktionsfelder entweder den Wert `ADD` oder den Wert `DELETE` beinhalten. Technisch ausgedrückt bedeutet dies:

```
(eventType == 'ObjectEvent' OR eventType == 'TransactionEvent') AND (action == 'ADD' OR action == 'DELETE')
```

Alle Parameter der einfachen Ereignisabfrage sind optional. Eine Abfrage ganz ohne Konfigurationen entspricht allen Ereignissen, die auf dem EPCIS-Server gespeichert sind. Der Server selbst ist allerdings immer dazu berechtigt, eine Auswertung der Anfrage zu verweigern, wenn die Abfrage zu komplex ist bzw. das daraus resultierende Ergebnis zu lang werden sollte.

3.5.4.2. Einfache Stammdatenabfrage

Im Gegensatz zur einfachen Ereignisabfrage liefert die einfache Stammdatenabfrage[4] keine Ereignisse zurück, sondern Wortschatz-Elemente, die auf dem EPCIS-Server hinterlegt wurden.

Auch hier lässt sich die Auswahl mittels vordefinierter Parameter konfigurieren, wobei dafür längst nicht so viele unterschiedliche nötig sind wie bei der eben vorgestellten einfachen Ereignisabfrage. Sie lassen sich in folgende Sparten einteilen:

- *Wortschätze betreffend:* Es stehen Parameter zur Verfügung, die bezwecken, dass nur Elemente aus bestimmten Wortschätzen eingefügt werden.

- *Wortschatz-Elemente betreffend:* Hier werden Möglichkeiten angeboten, dass nur bestimmte Elemente eingefügt werden. Auch hier ist eine Differenzierung nach der Hierarchie der Stammdaten möglich.

- *Attribute betreffend:* Zu guter Letzt wird eine Filterung über die enthaltenen Attribute angeboten. Dies wird über den Namen, aber auch über den Wert der Attribute ermöglicht. Zusätzlich kann ausgewählt werden, ob Attribute überhaupt zum Ergebnis hinzugefügt werden sollen (was nicht immer notwendig ist und in einem solchen Fall das Ergebnis nur unnötig vergrößern würde).

Auch die einfache Stammdatenabfrage enthält einen Parameter, über den sich die Ergebnismenge limitieren lässt. An dieser Stelle ist ebenso zu beachten, dass trotzdem der EPCIS-Server auch während einer solchen Abfrage immer dazu berechtigt ist, nach eigenem Ermessen die Ausführung zu verweigern, sollte das Ergebnis zu lang werden.

[4]Bezeichnung innerhalb der EPCIS-Spezifikation: *SimpleMasterDataQuery*.

Hinweis:

Auch für die einfache Stammdatenabfrage existiert eine exakte Auflistung der einzelnen Parameter im Anhang der Arbeit.

Die Verknüpfung der einzelnen Parameter – sollten mehrere auf einmal gesetzt sein – und ihrer Werte (sind mehr als einer angegeben) verhält sich analog zur Verknüpfung der einfachen Ereignisabfrage.

Da Stammdaten insgesamt mehr von statischer Natur sind, untersagt es die EPCIS-Spezifikation ausdrücklich, eine einfache Stammdatenabfrage asynchron auszuführen. Dies wäre zwar technisch problemlos möglich, würde aber dem Grundgedanken einer solchen Ausführung widersprechen, die sich mehr auf dynamische, ständig ändernde Daten bezieht. Die Spezifikation geht sogar so weit, dass sie von einer Server-Implementierung verlangt, die Registrierung eines Abonnements für eine einfache Stammdatenabfrage grundsätzlich abzulehnen.

3.5.5. Umsetzung der Schnittstelle als Web Service

Für die technische Umsetzung dieser Schnittstelle kann der modulare Aufbau der EPCIS-Spezifikation intensiv genutzt werden: Dieser bietet neben der geschichteten Struktur, durch die sogar ein vollständiges Austauschen der kompletten Service-Schicht denkbar wäre, die Voraussetzungen, einzelne Teile unterschiedlich zu implementieren.

Dabei präsentiert die EPCIS-Spezifikation selbst genau zwei Wege, mit denen eine solche Implementierung gemacht werden kann, was natürlich nicht heißen muss, dass keine anderen Techniken eingesetzt werden können. Die beiden Vorschläge für eine Implementierung der Abfrage-Schnittstelle sind:

- *XML-Datenaustausch über Applicability Statement 2 (AS2):* Die meist für EDI-Zwecke genutzte Technologie kann eingesetzt werden, um die in einer

XML-Struktur modellierten Daten zwischen den Systemen sicher auszutauschen.

- *XML-Datenaustausch über Web Services:* Auch diese Technolgie modelliert die Daten durch eine XML-Struktur. Diese werden dann über HTTP mit SOAP übertragen, die Schnittstelle selbst wird durch eine WSDL-Beschreibung definiert.

Beide Wege nutzen eine Darstellung der Daten in XML-Form. Die EPCIS-Spezifikation liefert eine sehr genaue, schematische Beschreibung für alle von der Schnittstelle verwendeten Objekte, so dass sich eine umfangreiche Bibliothek mit allen für den Datenaustausch notwendigen Klassen erstellen lässt.

Außerdem ist für die eigentliche Schnittstelle eine komplette WSDL-Beschreibung inklusive Ausnahme-Definitionen der einzelnen Operationen vorhanden, so dass in Kombination mit den gerade erwähnten Objekten eine sehr genaue Beschreibung der Implementierungsmöglichkeit als Web Service existiert. Diese Technologie wird für eine Server-Implementierung dringend empfohlen.

Es existiert also im Gegensatz zur Erfassungsschnittstelle eine sehr genaue Vorstellung seitens der EPCIS-Spezifikation, wie eine Implementierung aussehen sollte. Neben den weitreichenden technischen Möglichkeiten, die eine solche Technologie bietet, ist dies einer der Hauptgründe, warum dieser Weg während der Erstellung dieser Arbeit zum Einsatz kommt. Auch wenn die EPCIS-Spezifikation zusätzlich Definitionen für den Gebrauch von AS2 bereitstellt, soll dieser Weg nicht weiter behnandelt werden.

3.6. Sicherheitskonzept

Ein Netzwerk, über das umfangreiche, möglicherweise auch vertrauliche Daten ausgetauscht werden sollen, verlangt zwangsläufig nach Sicherheitsmechanismen, um einen solchen Transfer vor unbefugten Zugriffen schützen zu können.

Die Beschreibung des EPC-Netzwerkes sieht – wie im vorherigen Kapitel bereits angedeutet – eine eigene Service-Struktur innerhalb des Netzwerkes vor, über die verschiedene Sicherheitsdienste angeboten werden sollen. Eine Spezifikation dieser Dienste existiert während der Erstellung dieser Arbeit leider noch nicht. Vorhanden ist lediglich eine Spezifikation[5], welche die technischen Inhalte von Zertifikate enthält, die wohl im Bezug auf diese Dienste zum Einsatz kommen sollen.

Die EPCIS-Spezifikation selbst macht zum Thema Sicherheit wenig Angaben. Es werden lediglich zwei Forderungen für die Service-Schnittstellen gestellt:

- *Authentifizierung:* Jeder der beiden Kommunikationspartner muss die Identität des anderen sicher kennen. Das heißt, dass jeder weiß, mit wem er kommuniziert und dass es auch wirklich derjenige ist, für den er sich ausgibt.
- *Authorisierung:* Der Kommunikationspartner muss zur Ausführung der Operation, die er anstoßen will, auch berechtigt sein. Dies lässt sich über Regeln in Verbindung mit seiner Identität klarstellen.

Weitere Angaben zu den Sicherheitsmechanismen macht die EPCIS-Spezifikation in der vorliegenden Version nicht. Um trotzdem eine möglichst realitätsnahe Präsentation eines EPCIS-Systems anbieten zu können, wird im Laufe dieser Arbeit ein eigener Ansatz für eine auf Zertifikaten basierende Sicherheitslösung entwickelt werden. Diese Technologie wird grundsätzlich auch von SOAP gut unterstützt und ist für einen sicheren Datenaustausch innerhalb des Internets durchaus üblich. Genaueres ist dazu im Konzept-Teil dieser Arbeit zu finden.

[5]EPCglobal Certificate Profile. EPCglobal, März 2006.

4. Konzept für eine EPCIS-Implementierung

Überlegungen und konzeptioneller Aufbau einer leichtgewichtigen Referenz-Implementierung

Innerhalb dieses Kapitels soll das eigentliche Projekt dieser Arbeit vorgestellt werden. Dazu gehört eine Beschreibung der Aufgabe, Überlegungen für den Aufbau des dazu benötigten Systems und die Entwicklung der dazugehörigen Komponenten. Ergänzt werden soll das ganze durch beispielhafte Szenarien, mit deren Hilfe das System getestet werden kann.

4.1. Ziele des Projektes

Was erreicht werden soll

Die Idee für die Entwicklung eines EPCIS-Systems ist noch nicht sonderlich alt: Die Spezifikation wurde zum Zeitpunkt der Erstellung dieser Arbeit noch nicht offiziell verabschiedet und aus diesem Grund sind nahezu keine Versuche bekannt, ein solches System aufzusetzen. Wenn überhaupt, wurden bisher meist nur Komponenten aus dem RFID-Lesebereich umgesetzt, deren Spezifikationen deutlich früher verabschiedet wurden. Eine Verknüpfung dieser Komponenten zu den Informationsdiensten des EPC-Netzwerkes ist oft nur angedeutet und nicht immer vollständig. Die meisten dieser Implementierungen basieren auch nicht auf der EPCIS-Spezifikation selbst, sondern auf derjenigen, welche die Struktur und die Bedeutung der einzelnen Bestandteile des EPC-Netzwerkes beschreibt[1].

[1]The EPCglobal Architecture Framework Version 1.0. EPCglobal, Juli 2005.

Aus diesen Gründen ist es zur Zeit nahezu unmöglich, die Funktionsweisen der verschiedenen Komponenten eines EPCIS-Systems, insbesondere der Abfrage-Schnittstelle, zu testen und zu demonstrieren. Gerade das Abfrage-Modul aus der EPCIS-Spezifikation ist aufgrund der Menge der enthaltenen Parameter so vielfältig konfigurierbar, dass durchaus eine Notwendigkeit ersichtlich ist, einzelne Anwendungsfälle an einem beispielhaften System demonstrieren zu können.

Dieser Wunsch festigt auch langsam den Grundgedanken, der dem Projekt dieser Arbeit zugrunde liegt: Es soll eine Client-Anwendung implementiert werden, die es dem Benutzer ermöglicht, individuelle Abfragen für die Service-Schnittstelle eines EPCIS-Systems zu formulieren und anschließend diese an den entsprechenden Server schicken zu können. Das daraus resultierende Ergebnis soll dann so verfügbar sein, dass es eingesehen, ausgewertet und entsprechend weiter verarbeitet werden kann.

Die Umsetzung dieser Idee bringt einige Schwierigkeiten mit sich, die zwangsläufig während der Entwicklung und der Implementierung dieses Projektes berücksichtigt werden müssen:

- Da während der Entwicklungszeit dieses Client-Programms keine reale EPC-Netzwerk-Infrastruktur zur Verfügung steht, an welche die Applikation ihre Anfragen stellen kann, müssen große Bereiche des Systems simuliert werden. Dazu gehört vor allem die verschiedenen EPCIS-Server, welche die Anfragen annehmen sollen.

- Neben den EPCIS-Servern sind natürlich auch alle anderen Dienste des EPC-Netzwerkes nicht vorhanden. Dazu gehören vor allem der Objektnamensdienst und die Ermittlungsdienste. Auch für diese müssen simulierende Lösungen, eventuell auch Kompromisse gefunden werden.

- Besonders wichtig ist der Sicherheitsdienst des Netzwerkes, der die Authentifizierung und die Authorisierung der einzelnen Benutzer sicherstellen soll.

Gerade für diesen muss eine entsprechende Ersatzlösung gefunden werden, welche die Gegebenheiten möglichst realitätsnah darstellen kann. Gerade die Authentifizierung des Anfragenden ist für einen Datenaustausch mit einem EPCIS-Server unverzichtbar, da dieser die Identität des Anfragenden kennen muss.

Somit können also folgende Aufgaben zusammengefasst werden:

1. Entwicklung einer Server-Anwendung, die grundsätzlich in der Lage ist, der Client-Applikation die nötigen Schnittstellen anzubieten, ohne dass zwingend ein vollständiges EPC-Netzwerk existieren muss.

2. Implementierung des durch die Spezifikation definierten Web Services für EPCIS-Abfragen inklusive der für den Datenaustausch notwendigen Objekte innerhalb dieser Server-Anwendung.

3. Entwicklung einer Möglichkeit für die Server-Applikation, simulierte EPCIS-Daten zur Verfügung stellen zu können, die dann bei einer Abfrage zurückgeliefert werden.

4. Zu guter Letzt die Erstellung der eigentlich geforderten Client-Anwendung, in der sich Abfragen formulieren lassen und welche zusätzlich in der Lage ist, diese Daten auch an die Schnittstelle des Servers zu schicken.

4.2. Aufbau eines simulierten EPCIS-Systems

Simulation der notwendigen Komponenten des EPC-Netzwerkes

Das Hauptproblem für eine EPCIS-Client-Implementierung stellt somit die Notwendigkeit einer Netzwerkstruktur dar, die so leider nicht zur Verfügung steht. Aus diesem Grund wird versucht, innerhalb dieses Abschnitts ein eigenes System aufzubauen, welches zwar nicht komplett dem eigentlichen EPC-Netzwerk entspricht, den Benutzer aber trotzdem in der Lage versetzt, möglichst realitätsnah Abfragen durch einen EPCIS-Server auswerten zu lassen.

Dabei wird noch einmal darauf hingewiesen, dass es nicht das Ziel dieses Projektes ist, eine Referenz-Implementierung des Netzwerkes zu schaffen, sondern vielmehr eine möglichst exakte Darstellung einer Abfrage an den EPCIS-Server zu simulieren.

Aus diesen Gründen müssen einige Entwicklungsschritte gemacht werden, die in den folgenden Abschnitten vorgestellt werden sollen.

4.2.1. Beschränkung auf einen EPCIS-Server

Die Netzwerk-Infrastruktur sieht vor, dass jeder Handelspartner, der etwas mit dem Netzwerk zu tun hat, über mindestens einen lokalen EPCIS-Server verfügt, auf dem er die entsprechenden Daten speichern kann, welche in seinem Unternehmen anfallen. Das EPC-Netzwerk selbst kümmert sich dabei in erster Linie darum, dass der Server auch für andere Mitglieder zu finden und zu erreichen ist, bzw. natürlich auch, dass die Server der anderen ebenso genutzt werden können. Für dieses recht komplexe Szenario werden nahezu alle Dienste in Anspruch genommen, die das EPC-Netzwerk anbieten kann.

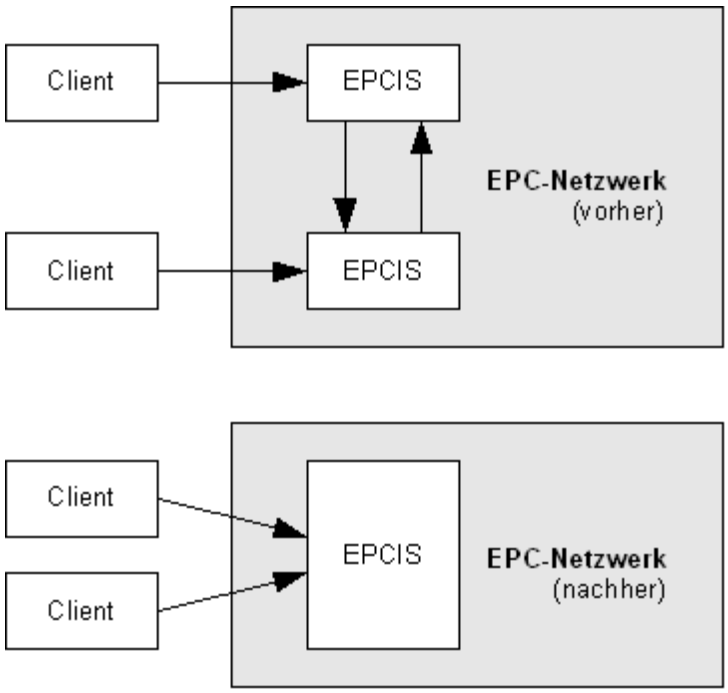
Dies alles ist trotzdem nicht unbedingt der entscheidende Teil eines Systems, das in erster Linie zeigen soll, wie Informationsabfragen aufgebaut und versendet werden können und auf welche Weise die entsprechenden Antworten des Servers anzusehen und zu verstehen sind. Der eigentliche Schwerpunkt dieses Projektes liegt also eindeutig auf der Client-Applikation und nicht auf einer Darstellung des dazugehörigen Netzwerkes. In der Theorie wäre zum Beispiel eine Implementierung des Zuordnungsdienstes *ONS* natürlich möglich, da für diesen bereits eine offizielle Spezifikation existiert, dies wäre allerdings sehr aufwendig im Bezug auf die Tatsache, dass dieser gar nicht den wesentlichen Teil des Projektes darstellt.

Es macht an dieser Stelle durchaus Sinn, von Anfang an festzulegen, dass aus Sicht der Client-Applikation nur ein einziger EPCIS-Server existiert. Diese sehr weitreichende Forderung brächte den enormen Vorteil mit sich, dass auf

alle Zuordnungs- und Suchdienste innerhalb des Netzwerkes verzichtet werden könnte, die zwar ein wesentlicher Bestandteil der Visionen von EPCglobal, allerdings für die Ziele dieses Projektes eher zweitrangig und damit unter allen Umständen zu aufwendig sind.

Ein weiterer Vorteil wäre, dass dieser EPCIS-Server nahezu alle das EPC-Netzwerk betreffenden Funktionen kapseln würde. Es wäre somit dann nicht mehr notwendig, diese innerhalb der Client-Applikation zu integrieren, sondern es befänden sich – bis auf wenige Ausnahmen – fast alle nur noch hinter der Fassade des Servers. Man kann an dieser Stelle noch einen Schritt weitergehen und behaupten, dass sich über diesen Weg nahezu das komplette EPC-Netzwerk über einen einzigen EPCIS-Server abbilden lässt. Für das Projekt dieser Arbeit wäre das auf jeden Fall ausreichend.

Abbildung 4.1. Vereinfachung des Systems auf einen EPCIS-Server



Zusätzlich ist hier noch die Tatsache zu erwähnen, dass ein solcher Zugriff aus Sicht eines einzelnen Clients gar nicht so unrealistisch ist, da für ihn der lokale EPCIS-Server ohnehin die Schnittstelle zum EPC-Netzwerk darstellt. Die

Simulation passiert dabei innerhalb des Servers, der somit auch in der Lage sein muss, Zugriffe von einzelnen Client-Applikationen zu unterscheiden und getrennt voneinander zu verwalten.

4.2.2. Integration von simulierten EPCIS-Daten

Um ein Client-Programm nutzen zu können, das Daten von einem EPCIS-Server abfragen kann, müssen natürlich auch solche Daten vorhanden sein. Da der zu implementierende Server die Gegebenheiten nur simuliert und somit keinen Zugriff auf reale Daten hat, müssen andere Wege gefunden werden, die ihn in die Lage versetzen, repräsentative Daten anbieten zu können.

Die Fähigkeit, auch die EPCIS-Daten zu simulieren, eröffnet natürlich auch neue Möglichkeiten für die Client-Applikation: Theoretisch könnten Daten derart generiert werden, dass sich durch Ereignisse dargestellte Geschäftsprozesse abfragen lassen. Die Unabhängigkeit von bereits vorhandenen Daten muss also ganz und gar nicht als Nachteil ausgelegt werden.

Die weniger aufwendige Form von EPCIS-Daten stellen sicherlich die Stammdaten dar. Wie bereits in den zurückliegenden Teilen dieser Arbeit beschrieben, wird diese Form der EPCIS-Daten mehr statisch eingesetzt, da sie sich selten ändern und durch Client-Applikationen fast ausschließlich für informative Zwecke abgefragt werden. Dies wird allein schon durch die Tatsache deutlich, dass die EPCIS-Spezifikation keinerlei Schnittstellen definiert, über die sich solche Daten erfassen lassen, sondern ausschließlich die Auslieferung solcher Daten behandelt.

Aus diesem Grund kann auch die Datenhaltung für Stammdaten durchaus auf statische Weise geschehen. Es macht keinen Sinn, umfangreiche Wege für eine flexible Organisation dieser Daten zu entwickeln, wenn diese doch nur selten in Anspruch genommen würde. Vielmehr bietet sich eine vollständig statische Lösung für die Datenhaltung an, mit deren Hilfe sich während Abfragen die ent-

sprechenden Stammdaten vom Server auslesen und in gefilterter Form zurückliefern lassen.

Etwas schwieriger gestaltet sich die Integration der Ereignisdaten. Diese werden im Gegensatz zu den Stammdaten keineswegs statisch genutzt, sondern ständig ergänzt. Der Sinn dieser Form von Daten besteht nicht darin, sich zu bestimmten Gegebenheiten weiter zu informieren, so wie dies bei den Stammdaten der Fall ist, sondern es geht vielmehr darum, durch die Kombination und die Veränderung zwischen den einzelnen gespeicherten Ereignissen unterschiedliche Schlussfolgerungen zu machen.

Für die Bereitstellung repräsentativer Ereignisdaten zur Nutzung mit speziell generierten Abfragen gibt es somit verschiedene Möglichkeiten:

- Eine statische Lösung innerhalb des Servers ähnlich des gerade vorgeschlagenen Weges für die Stammdaten. So etwas wäre bei einem simulierten EPCIS-Server natürlich möglich. Der Server hätte also statisch Ereignisdaten für unterschiedliche Szenarien hinterlegt, die sich serverseitig verwalten ließen. Diese würde er entsprechend den Parametern von Abfragen zurückliefern. Diese Lösung ist mit Sicherheit der einfachste, aber auch der am wenigsten flexible Weg.

- Um den statischen Faktor für diese Ereignisart zu vermeiden, könnte der Server die Daten selbst dynamisch generieren, wenn diese gebraucht würden. Man hätte somit jederzeit unterschiedliche Daten, die der Server anbieten könnte. Eine dynamische Erzeugung solcher Daten, die dann auch in Kombination möglichst sinnvoll erscheinen sollten, ist allerdings ein äußerst komplexes Unterfangen. Es setzt eine umfangreiche Integration der Geschäftslogik voraus, über die der EPCIS-Server verfügen müsste.

- Die dritte Möglichkeit ist diejenige, die sich am meisten an der Realität orientiert: Die Daten werden außerhalb des EPCIS-Servers generiert und mittels einer beliebigen Datenhaltung innerhalb des Servers gespeichert. Dieser wäre

dann in der Lage, bei Anfragen wirklich diejenigen Daten zurückzuliefern, die zuvor das System erreicht haben. Die Geschäftslogik ließe sich also mehr oder weniger vollständig auf die Client-Applikationen auslagern, so wie es ja eigentlich auch durch das EPCIS-Konzept vorgesehen ist.

Jede dieser Wege bringt Vor- und Nachteile mit sich: Einerseits wäre es gut, eine möglichst dynamische, flexible Lösung zu haben, auf der anderen Seite darf die dafür notwendige Implementierung nicht zu aufwendig werden. Aus diesen Gründen spricht am meisten für den dritten Weg: Durch ihn muss innerhalb des Servers kaum etwas von der zu den Daten gehörigen Geschäftlogik bekannt sein, die Daten selbst könnten – je nach Bedarf statisch oder dynamisch – einfach an den Server geschickt werden, der diese dann nur zu speichern braucht. Er wäre also für Szenarien unterschiedlichster Art seitens des Clients gerüstet.

Das Problem an dieser Stelle ist das Fehlen einer Möglichkeit für die Datenübermittlung an den Server. Bisher sollte unsere Client-Applikation nur Abfragen an den Server schicken, ihn allerdings nicht zwingend mit neuen Daten versorgen. An dieser Stelle kann aber wieder ein Vorschlag aus der EPCIS-Spezifikation herangezogen werden, durch den dieses Problem schnell zu lösen ist: Sie definiert schließlich eine eigene Erfassungsschnittstelle, mit deren Hilfe Ereignisdaten an den Server übergeben werden können. Es wäre dabei zwar eine weitere Schnittstelle zu implementieren, was durch deren geringen Umfang (sie besteht nur aus einer einzigen Operation) aber nicht weiter ins Gewicht fallen würde.

Man kann diesen Weg durchaus als äußerst elegant bezeichnen, da tatsächlich alle von der Spezifikation geforderten Schnittstellen implementiert wären und sich einzelne Geschäftsfälle sehr flexiblel und wirklichkeitsnah übertragen und wieder abfragen ließen.

4.2.3. Entwicklung einer Erfassungsschnittstelle

Da die Spezifikation für die Erfassungsschnittstelle keine genaueren Vorgaben macht, sondern vielmehr nur das beschreibt, was sie leisten muss, soll nun an

dieser Stelle ein Versuch unternommen werden, eine passende Schnittstelle für die Testumgebung zu entwickeln.

Die Erfassungsschnittstelle selbst hat für das Projekt dieser Arbeit sicherlich einen niedrigeren Stellenwert als die Abfrage-Schnittstelle. Sie ist äußerst simpel aufgebaut und ist nach der Spezifikation darauf ausgelegt, in möglichst kurzer Zeit äußerst effizient Ereignisse innerhalb des Servers wegzuschreiben. Aus diesem Grund macht die Spezifikation lediglich den Vorschlag eines extrem einfachen HTTP-Zugriffs für eine Implementierung, bzw. empfiehlt sie eine Message-Queue, sollten sehr viele Daten zu erwarten sein.

Dies sind alles Punkte, die im Simulationssystem kaum eine Rolle spielen: Der Zeitfaktor ist zu vernachlässigen, da das System in erster Linie zeigen soll, durch welche Logik die Daten abgefragt werden und nicht, wie man sie möglichst effizient von einem Ort zum anderen schickt. Auch kann beim Nutzen von Testszenarien davon ausgegangen werden, dass nicht so extrem viele Ereignisse auf einmal beim Server ankommen, dass dieser dadurch ein Problem mit der Verarbeitung bekommen könnte. Auch mit Stockungen während der Ereignis-Annahme, die durch die Message-Queue-Lösung aufgefangen werden sollen, muss wohl nicht gerechnet werden.

Für das Simulationsprojekt ist mit Sicherheit der geeignetste Weg zu einer solchen Erfassungsschnittstelle derjenige, der sich am einfachsten implementieren lässt: Eine Implementierung völlig analog zur Abfrage-Schnittstelle als Web Service. Dies ist aus oben genannten Gründen sicherlich nicht die Lösung, die man in einem realen System wählen würde, hat aber für das Simulationssystem enorme Vorteile:

- Eine Definition der (sehr kleinen) Schnittstelle in Form einer WSDL-Datei stellt kein großes Problem dar.

- Klassen der Datentypen für den Austausch mit der Abfrage-Schnittstelle können problemlos wiederverwendet werden.

- Auch das noch zu entwickelnde Sicherheitskonzept der Abfrage-Schnittstelle sollte mehr oder weniger auch für diese Schnittstelle eingesetzt werden können.

- Während einer späteren Implementierung ist die Umgebung für einen weiteren Web Service bereits vorhanden. Es muss also kein weiteres System, wie zum Beispiel für eine Message-Queue, eingerichtet werden.

Während der Entwicklung selbst ist darauf zu achten, dass die neue Schnittstelle möglichst ähnlich der Abfrage-Schnittstelle aufgebaut ist, um später Schwierigkeiten vermeiden zu können. Dies betrifft insbesondere die Ausnahmebehandlung und die Erweiterbarkeit. Somit besteht sie spezifikationskonform nur aus einer einzigen Operation `capture()`, die als Paramter eine Liste von Ereignissen erhält.

Hinweis:

Weiter soll an dieser Stelle nicht auf die Entwicklung der Schnittstelle eingegangen werden. Im Implementierungsteil dieser Arbeit wird noch einmal die Erstellung der Schnittstelle aufgegriffen und im Anhang befindet sich die vollständige, schematische Definition.

4.2.4. Funktionen der Client-Applikation

Mit diesen Voraussetzungen kann man sich nun überlegen, über welche Funktionen der EPCIS-Client eigentlich verfügen soll.

Da nun beide Schnittstellen aus der EPCIS-Spezifikation seitens des Servers vorhanden sind, ist es an dieser Stelle sicherlich sinnvoll, die modulare Unterteilung dieser beiden Komponenten beizubehalten und sie wieder als einzelne Module in das Client-Programm zu integrieren.

Weil der EPCIS-Client jetzt nahezu vollständig Zugriff auf die Geschäftslogik hat, ist es an dieser Stelle durchaus reizvoll, eine Möglichkeit zu schaffen, so

allgemein wie möglich beliebige Ereignisse erzeugen und diese testweise wieder abzufragen zu können. Dabei sind Eingabeformulare für alle EPCIS-Ereignisse denkbar, durch die sich mit beliebigen Werten Datensätze erstellen lassen, die dann anschließend auf dem Server gespeichert werden können. An diesen speziellen Daten ließe sich dann auf ideale Weise die Funktionalität einzelner Abfrage-Parameter testen.

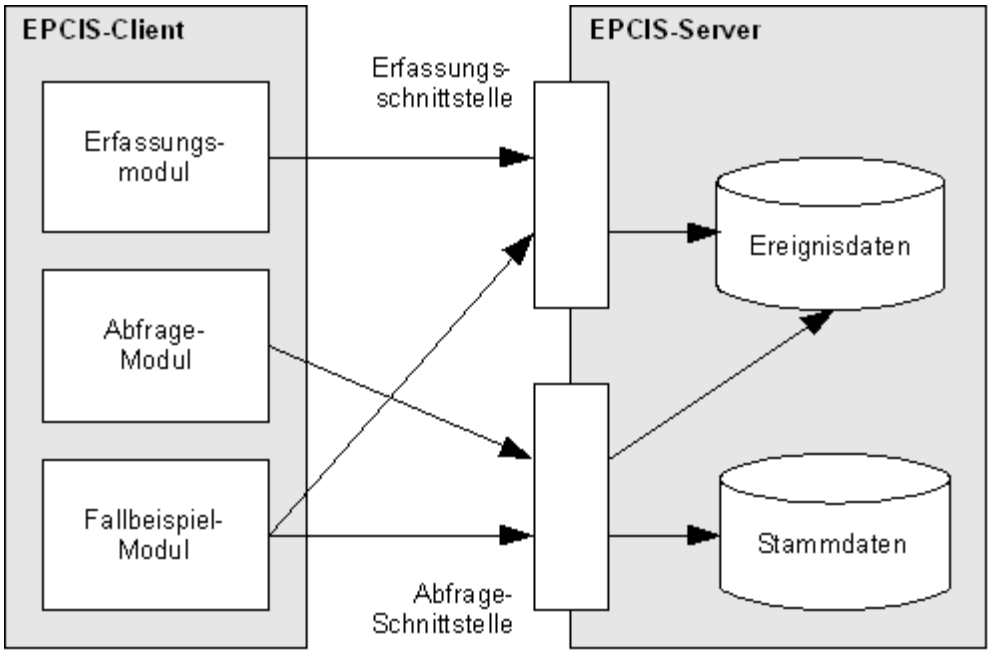
Da dies ein sehr technisch orientierter Fall ist (die gesamte Geschäftslogik wird in diesem Moment in die Hände des Anwenders gelegt), wäre zusätzlich noch ein weiteres Modul wünschenswert, dass ein spezielles Fallbeispiel simuliert und somit einen Einsatz von EPCIS in der Realität demonstrieren kann. Für dieses müsste dann kein genaues Wissen über die Struktur und die Bedeutung der einzelnen Ereignisse vorausgesetzt werden.

Ein solches Beispiel, dass natürlich noch zu entwickeln ist, könnte dann auch wieder statisch innerhalb des Clients integriert werden. Es würden dem Fall entsprechend Ereignisse erzeugt und zeitversetzt an den EPCIS-Server geschickt, so wie sie auch in Wirklichkeit eingetreten wären. Aus diesen Daten könnten dann mittels vorgefertigten Abfragen, bzw. auch allgemein über das Abfrage-Modul, Schlussfolgerungen zu der aktuellen Situation gezogen werden. Da die Funktionalitäten für Erfassung und Abfrage bereits in den beiden anderen Modulen vorhanden sind, könnten diese einfach wieder aufgegriffen werden.

4.2.5. Vorläufige Zusammenfassung

An dieser Stelle sollen nun die bisher erarbeiteten Bestandteile des zu erstellenden EPCIS-Systems gesammelt und zu einem ersten Zwischenstand vereint werden.

Abbildung 4.2. Erster Überblick über das simulierte System



Das Simulations-System besteht also aus zwei Hauptkomponenten, dem EPCIS-Server und dem EPCIS-Client.

Bestandteile des Servers sind:

- Ein statisches Repository, das die Aufgabe hat, die Stammdaten zu sammeln.
- Ein dynamisches Repository für die Organisation der Ereignisdaten.
- Eine Schnittstelle, über die Ereignis-Daten erfasst werden können, die der EPCIS-Server dann dem Ereignis-Repository hinzufügt.
- Eine weitere Schnittstelle, die einerseits Abfragen entgegen nimmt, aber auch Zugriff auf beide Repositories hat, um Ereignis- und Stammdaten entsprechend der Anfrage zurückliefern zu können.

Der Server kümmert sich also im Wesentlichen darum, die EPCIS-Daten zu verwalten und zu speichern und sie anderen Applikationen über eine Schnittstellen-Struktur wieder zur Verfügung zu stellen. Eine solche Applikation stellt der EPCIS-Client dar, der sich schematisch aus folgenden Komponenten zusammensetzt:

- Ein Modul, um EPCIS-Ereignisse zu erstellen und welches zusätzlich anbietet, diese auch an die Erfassungsschnittstelle des EPCIS-Servers zu schicken.

- Ein weiteres Modul, um Abfragen zu formulieren, die dann an die Abfrage-Schnittstelle des Servers gerichtet werden können, der dann je nach Abfragetyp Stamm- oder Ereignisdaten zurückliefert.

- Und zu guter Letzt ein Modul, dass bereits vordefinierte, statische Ereignisse systematisch an die Erfassungsschnittstelle schickt und anschließend über die Abfrage-Schnittstelle des EPCIS-Servers wieder abrufen kann, um gewisse Gegebenheiten zu verdeutlichen.

4.3. Erweiterung um Sicherheitsmechanismen

Integration von Authentifizierung und Authorisierung

Die vorangehenden Abschnitte haben gezeigt, dass die weitreichenden Folgen der Reduzierung des EPC-Netzwerkes auf einen einzigen EPCIS-Server die Client-Applikation kaum betreffen: Die Fassade des EPCIS-Servers, die den eigentlichen Netzwerkzugriff abschirmt, hat sich aus Sicht des Clients kaum geändert. Noch immer betreffen ihn eigentlich nur die beiden Schnittstellen des Servers, die sich nach wie vor an die EPCIS-Spezifikation halten. Der simulierende Teil befindet sich sicherlich auf der Seite des Servers.

Das Hauptproblem, welches durch diesen Ansatz entsteht, ist die Tatsache, dass der Server in der Lage sein muss, Daten unterschiedlicher Herkunft zu verwalten. Dies betrifft in erster Linie die Erfassungsschnittstelle. Der Grundgedanke hinter einem EPCIS-System sieht eigentlich vor, dass jedes Unternehmen seine eigenen EPCIS-Server unterhält, auf die ausschließlich sie selbst Ereignisse speichern dürfen. Die Spezifikation lässt zwar die Möglichkeit eines Zugriffs für die Erfassung von EPCIS-Daten anderer Handelspartner offen, doch dieser Fall sollte wahrscheinlich selten auftreten. Die grundsätzliche Idee ist mit

Sicherheit, dass sich nur Daten des zugehörigen Unternehmen auf einem Server befinden, die dann aber auch von Handelspartnern abgerufen werden können.

Da der geplante EPCIS-Client durch seine Modulstruktur auch in unterschiedlichen Rollen gegenüber dem Server auftreten soll, muss dieser die unterschiedlichen Identitäten des Clients natürlich auch unterscheiden können. Dies entspricht serverseitig dem Zugriff vieler, unterschiedlicher Client-Applikationen, also gerade der seltene Fall, der oben beschrieben wurde.

Die EPCIS-Spezifikation definiert dafür ein grundsätzliches Vorgehen: Werden Ereignisse an die Erfassungsschnittstelle geschickt, so müssen sich Client und Server zunächst untereinander authentifizieren. Anschließend muss der Server während der Speicherung der neuen Ereignisse zusätzlich zu jedem einzelnen die dazugehörige Identität des Erzeugers sichern. Auf diesem Weg lassen sich später die einzelnen Ereignisse wieder einem Benutzer zuordnen, bzw. kann eine Entscheidung getroffen werden, ob eine Anfrage beantwortet werden darf oder nicht.

Diese Definition der EPCIS-Spezifikation scheint wie geschaffen für die Server-Simulation, da über eine solche Lösung mehrere Client-Applikationen getrennt voneinander bedient werden können. Natürlich setzt sie eine Integration von Sicherheitmechanismen voraus, insbesondere eine Möglichkeit zur Authentifizierung des Anfragenden. Gerade schon aus diesem Grund wird sehr schnell deutlich, dass das Simulationssystem im Prinzip ohne Sicherheitsfunktionalitäten kaum funktionieren kann.

4.3.1. Authentifizierung über digitale Zertifikate

Die EPCIS-Spezifikation fordert zwar, dass Service-Zugriffe auf den EPCIS-Server über eine Authentifizierung und eine anschließende Authorisierung geschützt werden sollten, allerdings nicht, auf welche Weise. Immer wieder wird von Sicherheitsdiensten innerhalb des EPC-Netzwerkes gesprochen, die bisher aber noch nicht weiter spezifiziert wurden. Es wurde bislang nur eine

Spezifikation verabschiedet, die den Aufbau von Zertifikaten beschreibt, die wohl einmal für eine Authentifizierung genutzt werden sollen. Aus diesem Grund liegt es nahe, auch für die Entwicklung eines eigenen Sicherheitskonzeptes auf solche Zertifikate zu setzen.

Da für einen solchen Einsatz theoretisches Hintergrundwissen vorausgesetzt werden muss, soll an dieser Stelle ein kurzer Exkurs eingefügt werden, der die nötigen Gegebenheiten erläutern soll.

4.3.1.1. Nichtabstreitbarkeit von Dokumenten

Ein sicherer Datenaustausch zwischen zwei Kommunikationspartnern ist normalerweise fest mit einem Begriff verbunden: „Nichtabstreitbarkeit“. Dies bedeutet, dass sichergestellt wird, dass der Absender bekannt ist, dass es sich wirklich um denjenigen handelt, für den er sich ausgibt und dass die Nachricht unterwegs nicht verändert worden ist.

Für sichere Zugriffe im Internet wird normalerweise *SSL* eingesetzt, ein Verschlüsselungsprotokoll, das diese Tatsachen sicherstellen kann und somit natürlich weit verbreitet ist. Allerdings werden dabei die Sicherheitsmechanismen nahezu vollständig auf die Protokoll-Ebene ausgelagert, was der Idee einer serviceorientierten Architektur widerspricht, die eigentlich unabhängig von einzelnen Protokollen arbeiten soll.

Aus diesem Grund wird ein neuer Ansatz gemacht, der es ermöglicht, die sicherheitsrelevanten Daten in das XML-Dokument selbst zu integrieren. SOAP bietet einen Kopfbereich an, der dafür genutzt werden kann. Mit Hilfe von digitalen Signaturen (elektronische Unterschriften) kann die Nichtabstreitbarkeit des Dokumentes gewährleistet werden. Diese bestehen aus einem fest definierten XML-Element, welches die dafür nötigen Informationen enthält. Der eigentliche Datenteil des betreffenden Dokumentes bleibt dabei vollständig unberührt. Man spricht in einem solchen Fall von einem *signierten Dokument*.

4.3.1.2. Einsatz von digitalen Zertifikaten

Für die Erstellung einer digitalen Signatur werden oft Zertifikate eingesetzt. Durch ein Zertifikat wird die Zugehörigkeit einer einzelnen Identität (also eine Person, eine Firma oder aber auch eine Maschine) zu einem öffentlichen Schlüssel bestätigt.

Diese Schlüssel gehören zu einem System, dass sich *Public Key Infrastructur* nennt: Für jede Identität wird über einen mathematischen Algorithmus ein Schlüsselpaar generiert, mit denen sich beliebige Daten verschlüsseln lassen. Dabei sind die beiden Schlüssel dieses Paares so gewählt, dass immer derjenige für die Entschlüsselung benötigt wird, mit dem nicht verschlüsselt wurde. Einer dieser Schlüssel ist öffentlich, der andere ist nur seinem Besitzer bekannt.

Mit diesen Schlüsseln können nun verschiedene Szenarien realisiert werden:

- *Vertraulichkeit:* Werden die Daten vom Sender mit dem öffentlichen Schlüssel des Empfängers verschlüsselt, so ist nur dieser in der Lage, mittels seinen privaten Schlüssels das Dokument wieder lesbar zu machen. Es ist also sichergestellt, dass nur derjenige die Nachricht lesen kann, für den sie auch bestimmt war.

- *Integrität:* Um sicherzustellen, dass die Daten unterwegs nicht geändert wurden, können einzelne Daten vom Sender mit seinem privatem Schlüssel verschlüsselt werden. Diese kann dann jeder Empfänger mit dem zugehörigen öffentlichen Schlüssel wieder entschlüsseln und aufgrund der erhaltenen Daten feststellen, ob Änderungen vorhanden sind (dies funktioniert, da nur der Besitzer des privaten Schlüssels die Nachricht so verschlüsseln konnte).

- *Authentizität:* In Verbindung mit einem Zertifikat, das – wie bereit erwähnt – die Identität zu einem öffentlichen Schlüssels bestätigt, kann also auch ein Nutzer authentifiziert werden, da durch das Zertifikat seine Identität ja bekannt ist.

Mit Hilfe eines Zertifikates können also nun digitale Signaturen erstellt werden, mit denen sich Dokumente signieren lassen und dadurch deren Nichtabstreitbarkeit gewährleistet werden kann.

Die mehrfach erwähnte Zertifikat-Spezifikation sieht dafür X.509-Zertifikate vor, die weit verbreitet sind und zur Zeit als wichtigster Standard für digitale Zertifizierung gelten. Sie enthalten auch Informationen über den Besitzer des Schlüssels, so dass sie gut für eine Authentifizierung eingesetzt werden können.

4.3.2. Konzept zur Integration in die Testumgebung

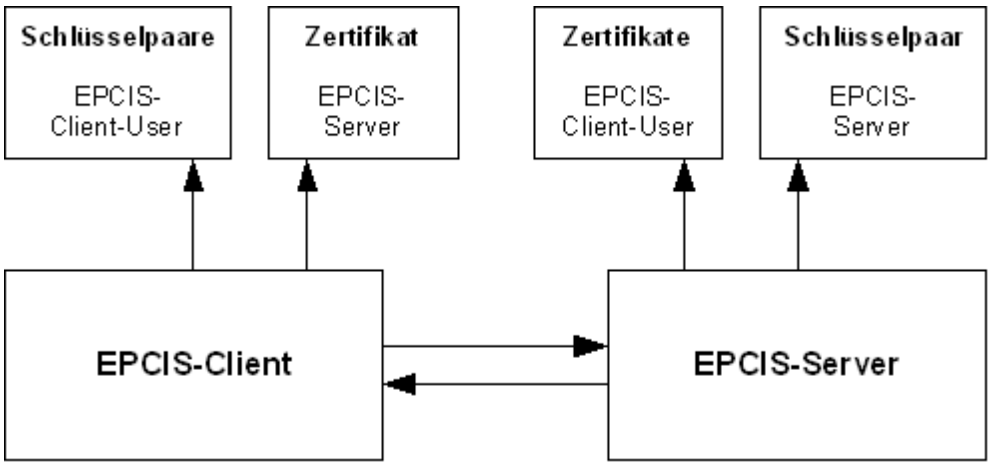
Um eine solche Zertifikatlösung in das Testsystem zu integrieren, müssen zunächst einzelne Identitäten festgelegt werden, die jeweils ein Schlüsselpaar erhalten:

1. Eine Identität für den *EPCIS-Server*, damit dieser seine Antworten signieren kann und somit jede Client-Applikation sicher weiß, dass sie auch wirklich mit dem Server kommuniziert.

2. Eine Identität für jede Rolle innerhalb des Client-Programms, also zum Beispiel ein *normaler Anwender*, ein *Administrator* und ein *Benutzer für das Fallbeispiel-Szenario*. Diese Identitäten werden dann dafür verwendet, um den Server in die Lage zu versetzen, einzelnen Benutzer unterscheiden zu können. Es wäre auch ein zusätzlicher Benutzer denkbar, der als nicht vertrauenswürdig eingestuft wird, um auch einen solchen Fall zu testen.

Das Szenario baut sich also folgendermaßen auf:

Abbildung 4.3. Erweiterung um digitale Zertifikate



- Die Client-Applikation hat Zugriff auf das Zertifikat des Servers, das dessen öffentlichen Schlüssel enthält. Zusätzlich besitzt sie eine Liste mit Schlüsselpaaren aller Identitäten, die innerhalb des Programmes verfügbar sein sollen.

- Der EPCIS-Server hat dagegen neben seinem eigenen Schlüsselpaar Zugriff auf alle Zertifikate der Client-Benutzer, die er bedienen darf.

Auf diese Weise wird es möglich, eine einfache, sichere Infrastruktur aufzubauen, in der sich die einzelnen Kommunikationspartner untereinander authentifizieren können. Je nach Benutzer kann der Server dann selbstständig entscheiden, ob er Daten ausliefern darf und wenn ja, welche.

4.4. Entwicklung eines geeigneten Szenarios

Ein Fallbeispiel

Um die doch recht abstrakte Einsatzweise der EPCIS-Ereignisse etwas anschaulicher darstellen zu können, soll ein passendes Fallbeispiel entwickelt werden, welches dann später in das Szenario-Modul integriert werden soll. Dafür wird ein Paket Kaffee von seiner Herstellung bis zu seinem Verkauf im Supermarkt begleitet.

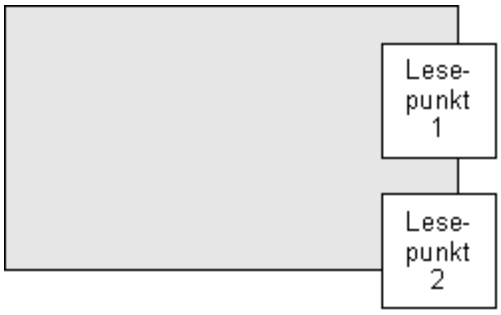
An jedem Schritt der Prozesskette werden entsprechende Ereignisse an den Server geschickt, die dieser dann speichert. Vordefinierte Abfragen sollen dann zeigen, auf welche Weise diese Ereignisse interpretiert werden können. Da die Bewegung innerhalb der Lieferungskette schrittweise verläuft, sollen auch Zwischenstände abgerufen werden können.

4.4.1. Orte innerhalb der Versorgungskette

Um ein solches Fallbeispiel aufbauen zu können, sollen zunächst einige Lokalitäten definiert werden, die innerhalb des Szenarios eine Rolle spielen werden. Dazu gehören ein Hersteller, ein Distributionszentrum und eine Filiale.

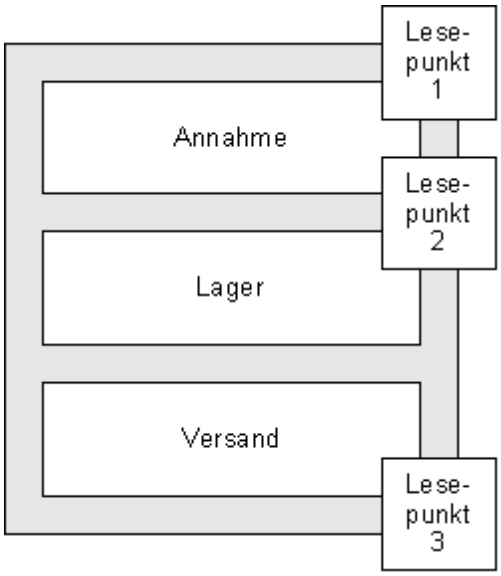
- Der *Hersteller* produziert den Kaffee. Er ist derjenige, der jedes Paket mit einem eindeutigen EPC bestückt. Innerhalb der Fabrik existieren keine besonderen Orte, allerdings sind Lesegeräte am Produktionsort und am Ausgang des Gebäudes vorhanden, so dass Ereignisse registriert werden können. Der Hersteller selbst wartet auf Bestellungen des Distributionszentrums und schickt entsprechend Waren dorthin.

Abbildung 4.4. Schematischer Aufbau der Hersteller-Lokalität



- Das *Distributionszentrum* ist dafür verantwortlich, Waren unterschiedlicher Hersteller zu sammeln, einzulagern und für den Versand an Filialen neu zusammenzustellen. Für diesen Zweck ist dieser Ort in drei logische Einheiten unterteilt: Einem Empfangsbereich, in dem die Waren angenommen werden, einem Lagerraum für die Zwischenlagerung des Kaffees und einem Bereich für den Versand.

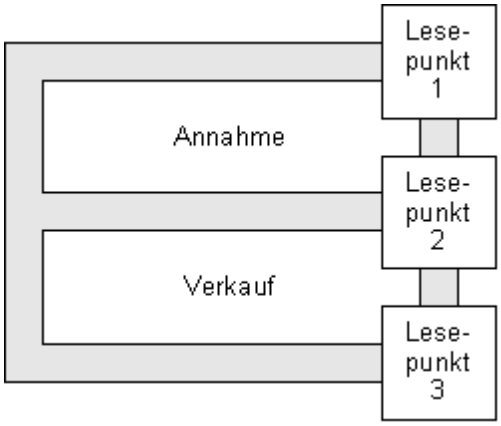
Abbildung 4.5. Schematischer Aufbau der Distributionszentrumslokalität



Lesepunkte werden am Ein- und am Ausgang sowie am Übergang zwischen Empfang und Lager bereitgestellt. Das Distributionszentrum wartet auf Bestellungen der Filiale und ordert daraufhin selbstständig beim Hersteller die geforderten Waren, um diese anschließend an die Filiale liefern zu können.

- Die *Filiale* könnte in diesem Szenario zum Beispiel einem Supermarkt entsprechen. Auch diese verfügt über einen Empfangsbereich, von wo aus die Waren in den Verkaufsraum gebracht werden können. Der letzte Schritt der Prozesskette besteht dann darin, dass ein Kunde den Kaffee kauft. Auch in der Filiale sind Lesepunkte an allen Übergängen aufgestellt, so dass sich Ereignisse zu den einzelnen Prozessschritten speichern lassen.

Abbildung 4.6. Schematischer Aufbau der Filialen-Lokalität



An dieser Stelle ist noch zu erwähnen, dass dieses Beispiel beliebig komplex erweitert werden kann. Gerade bei der Wahl der einzelnen Standort-Unterteilungen und der Lesepunkte ist ein möglichst einfacher Weg gewählt worden. So hätte man den Hersteller auch einen eigenen Versandbereich unterhalten lassen können, dies wäre allerdings technisch gesehen als Wiederholung zu betrachten und soll somit, um weiterhin eine gute Übersicht behalten zu können, nicht eingeführt werden.

Auch wird während der Umsetzung nicht jedes Mitglied eine eigene Benutzer-Identität erhalten (unser ganzes Beispiel arbeitet – wie bereits angedeutet – mit einem einzigen Szenario-Benutzer), was in der Realität mit Sicherheit anders gelöst wäre. Auf der anderen Seite ist es ja trotzdem möglich, zum Beispiel Lesepunkte einer einzelnen Personen zuzuordnen, was für die Simulation des Szenarios auf jeden Fall ausreichen sollte.

4.4.2. Entwicklung der Prozessschritte

Der Ablauf des Fallbeispiels soll nun derart funktionieren, dass der Anwender Schritt für Schritt die einzelnen Operationen entlang der Prozesskette ausführen kann. Dies bedeutet genauer, dass während jedem Schritt fest definierte Ereignisse an den EPCIS-Server geschickt werden, welche die definierten Vorkommnisse modellieren.

Um dies zu erreichen, sollen an dieser Stelle nun einzelne Fälle definiert werden, die als Glieder der Prozesskette anzusehen sind und sich natürlich durch EPCIS-Ereignisse beschreiben lassen müssen. Letztere können dann in chronologischer Reihenfolge durch das Client-Programm aufgerufen und auf dem Server gespeichert werden.

Dabei ist zusätzlich darauf zu achten, dass dieses Szenario über eine eigene Zeitrechnung verfügen soll, um auch zeitliche Zusammenhänge simulieren zu können. Für jeden Schritt ist eine bestimmte Dauer definiert, die der simulierten Uhrzeit hinzugerechnet und dann in den zugehörigen Ereignissen gespeichert wird. Außerdem soll eine Arbeitszeit von morgens um 8:00 Uhr bis abends um 18:00 Uhr eingehalten werden. Kann ein Schritt in diesem Zeitraum nicht mehr ausgeführt werden, so wird er erst am nächsten Morgen begonnen.

Hinweis:

Die folgende Beschreibung der Ereignisse zu den einzelnen Situationen legt keinen Wert auf Vollständigkeit, sondern versucht eher grundsätzlich zu zeigen, wann und wie spezielle Ereignisse gespeichert werden. Im Anhang dieser Arbeit ist allerdings eine genaue Auflistung der entsprechenden Ereignisse zu jeder einzelnen Operation enthalten.

Die einzelnen Prozessschritte des Szenarios setzen sich also folgendermaßen zusammen:

1. Der Hersteller produziert zwanzig Pakete Kaffee. Dabei erhält jedes einzelne Paket einen neuen EPC, der beim EPCIS-Server registriert wird.
   - Die Registrierung von neuen Objekten geschieht mit Hilfe von Objekt-Ereignissen, deren Aktion auf `ADD` gesetzt wurde.
2. Im zweiten Schritt werden die Pakete für eine Bestellung des Distributionszentrums auf einer Palette zusammengefasst.
   - Auch die Palette wird zunächst registriert.

- Die Zusammenfassung selbst geschieht über ein Aggregationsereignis, in dem als übergeordnetes Objekt die Palette angegeben wird.

- Die Zuordnung der Produkte zur Bestellung wird an dieser Stelle über Objekt-Ereignisse realisiert.

3. Anschließend wird diese Palette an das Distributionszentrum geschickt.

   - Sowohl die Palette als auch die einzelnen Produkte werden beim Verlassen des Herstellers und beim Empfang im Distributionszentrum durch Objekt-Ereignisse registriert.

   - Zusätzlich wird durch ein Transaktionsereignis mit der Aktion `DELETE` die Bestellung für beendet erklärt.

4. Die Waren werden zusammen mit der Palette ins Warenlager des Distributionszentrums gebracht.

   - Auch während dieses Schrittes werden alle Einheiten durch Objekt-Ereignisse erfasst.

5. Fünfzehn Pakete des eingelagerten Kaffees werden aufgrund einer Bestellung der Filiale auf einer neuen Palette zusammengestellt.

   - Auch für diesen Vorgang muss über ein Objekt-Ereignis zunächst eine neue Palette registriert werden.

   - Das Umpacken der Waren geschieht über zwei Aggragtionsereignisse: Eines mit der Aktion `DELETE` für das Entnehmen der Waren von der alten, eines mit der Aktion `ADD` für das Hinzufügen der Waren zu der neuen Palette.

   - In diesem Fall werden die Waren nun über ein Transaktionsereignis mit der neuen Bestellung assoziiert.

6. Dann wird die neue Palette an die Filiale verschickt.

   - Der Vorgang verläuft nahezu analog zum Versenden der Waren vom Hersteller zum Distributionszentrum.

   - Um einen zusätzlichen Sonderfall zu simulieren, verschwinden unterwegs vier Pakete Kaffee spurlos.

7. Im nächsten Schritt werden die verbleibenden Produkte in den Verkaufsraum gebracht.

   - Nach dem Lösen der Beziehung zwischen Produkten und Palette durch ein Aggregationsereignis wird diese durch ein Objekt-Ereignis mit der Aktion `DELETE` wieder freigegeben.

   - Jedes der Produkte erhält mit einem Objekt-Ereignis den Status „Bereit zum Verkauf“.

   - Die Filiale selbst verfügt über einen Mechanismus, der mit Hilfe von Quantitätsereignissen immer den aktuellen Bestand von Waren im Verkaufsraum dokumentiert.

8. Als letzte Operation kann noch ein Paket Kaffee gekauft werden.

   - In diesem Schritt wird das Produkt durch ein Objekt-Ereignis mit der Aktion `DELETE` gelöscht. Zusätzlich wird der Warenbestand der Filiale durch ein Quantitätsereignis aktualisiert.

Diese Fälle sind von rein beispielhafter Natur: Man kann natürlich jede Operation auch durch andere Ereignisse ausdrücken. Auch wäre denkbar, zusätzliche Punkte einzufügen, welche das Geschehen noch genauer beschreiben, bzw. könnte auch auf den ein oder anderen Punkt verzichtet werden. In den gewählten Fällen wurde insgesamt mehr darauf geachtet, die vielfältigen Möglichkeiten innerhalb der Lieferkette durch unterschiedliche Verhaltensweisen der ein-

zelnen Handelspartner auszudrücken. Dies soll mittels zwei Beispielen aus dem gerade definierten Szenario verdeutlicht werden:

- Der Hersteller generiert für jeden Objekt-Zugriff ein eigenes Objekt-Ereignis. Im Gegensatz dazu steht das Distributionszentrum, welches alle Produkte in einem Objekt-Ereignis zusammenfasst.

- Die Verknüpfung mit einer Bestellung realisiert der Hersteller mit Objekt-Ereignissen, wobei das Distributionszentrum und die Filiale ein Transaktionsereignis einsetzen.

Dies zeigt bereits deutlich, dass es sicherlich keinen eindeutigen, richtigen Weg gibt, um bestimmte Situationen zu beschreiben, sondern dass vielmehr die Kommunikationspartner sich vorher absprechen müssen, auf welche Weise sie ihre Geschäftssituationen modellieren.

4.4.3. Definition von Abfragen

Neben dem systematische Speichern von Ereignissen zu bestimmten Fällen soll das Fallbeispiel-Modul außerdem die Möglichkeit bieten, Abfragen zu stellen. Natürlich ist es möglich, diese auch mit dem Abfrage-Modul des EPCIS-Clients zusammenzustellen und an den Server zu schicken, allerdings ist es an dieser Stelle sinnvoll, bereits vorgefertigte Abfragen anzubieten, die bestimmte Fragen zum Status der Lieferkette beantworten können.

Innerhalb dieses Abschnitts sollen eine ganze Reihe solcher Abfragen definiert und beschrieben werden.

Hinweis:

Auch dieser Abschnitt ist nicht dafür gedacht, eine exakte Darstellung der Abfragen anzubieten, sondern vielmehr dafür, ihre Funktionsweise zu erläutern. Auch für die Abfragen befindet sich eine genaue, technische Beschreibung im Anhang dieser Arbeit.

1. *Welche Produkte wurden vom Hersteller bislang produziert?*

   Dies entspricht allen Produkte, die direkt nach der Herstellung registriert wurden.

   Das Ergebnis enthält also alle EPCs innerhalb von denjenigen Objekt-Ereignissen, die vom Produktionslesepunkt des Herstellers aufgezeichnet wurden und die Aktion ADD besitzen.

2. *Welche Paletten stehen mit der Bestellung des Distributionszentrums in Beziehung?*

   Dies entspricht allen Paletten, die mit Produkten dieser Bestellung verknüpft wurden.

   Dies sind also alle EPCs in Aggregationsereignissen, die eine Aktion ADD besitzen und die entsprechende Bestellung als Transaktion gespeichert haben.

3. *Wann hat die Palette des Herstellers das Distributionszentrum erreicht?*

   Dies ist die Uhrzeit, die während der Registrierung der Palette aufgezeichnet wurde.

   Das Ergebnis entspricht also dem Ereigniszeitpunkt des Objekt-Ereignisses, das vom Lesepunkt am Empfang des Distributionszentrums aufgezeichnet wurde und den EPC der Palette enthält.

4. *Welche Objekte befinden sich zur Zeit im Warenlager des Distributionszentrums?*

   Dies sind alle Produkte, für die gespeichert wurde, dass sie sich im Warenlager befinden und die das Lager noch nicht verlassen haben.

   Dies entspricht also allen EPCs in Objekt-Ereignissen, welche die Geschäftslokalität *Warenlager* enthalten, abzüglich derer, die durch ein Aggregati-

onsereignis von Ihrer Palette entfernt wurden (woraus man schließen kann, dass sie das Lager verlassen haben).

5. *Welche Objekte haben aufgrund der Bestellung der Filiale das Distributionszentrum verlassen?*

   Dies sind alle mit der Bestellung verknüpften Produkte, die durch den Versand des Distributionszentrums gekommen sind.

   Das Ergebnis enthält also alle EPCs aus Objekt-Ereignissen, die eine Transaktion mit der entsprechenden Bestellung und als Lesepunkt denjenigen am Versand des Distributionszentrums besitzen.

6. *Welche Produkte haben die Filiale aufgrund der Bestellung erreicht?*

   Dieser Fall ist dem vorherigen sehr ähnlich; nur der Lesepunkt muss ausgetauscht werden.

   Das Ergebnis enthält also diesmal alle EPCs aus den Objekt-Ereignissen, die eine Transaktion mit der Bestellung und als Lesepunkt denjenigen des Wareneingangs der Filiale haben.

7. *Welche Produkte sind während der Lieferung verlorengegangen?*

   Dies entspricht der Differenz zwischen den EPCs, welche die Filiale erreicht und denen, die das Distributionszentrum verlassen haben.

   Hierfür sind alle EPCs zu ermitteln, die sich in Objekt-Ereignissen befinden, die als Transaktion die Bestellung haben und einen Lesepunkt des Warenausgangs des Distributionszentrums oder den des Wareneingangs der Filiale besitzen. Die EPCs, die nicht doppelt vorkommen, entsprechen der Ergebnismenge.

8. *Wie viele Pakete Kaffee befinden sich im Verkaufsraum der Filiale?*

   Dies enspricht dem Wert der letzten Bestandsaktualisierung für den Kaffee.

Das Ergebnis entspricht also der Mengenangabe des Quantitätsereignisses für die EPC-Klasse *Kaffee*, das zuletzt gespeichert wurde.

9. *Wie viele Produkte wurden heute verkauft?*

   Dies sind alle Produkte, die am heutigen Tag durch die Kasse gekommen sind.

   Also alle EPCs aus den Objekt-Ereignissen, die als Lesepunkt die Kasse der Filiale haben, deren Aktion `DELETE` ist und deren Ereigniszeit größer oder gleich heute 8:00 Uhr und kleiner morgen 8:00 Uhr ist.

An dieser Stelle ist gut zu beobachten, dass sich trotz der allgemein gehaltenen Parameterauswahl doch sehr unterschiedliche Abfragen entwickeln lassen. So haben wir Beispiele zur Verfolgung der Herkunft über das Suchen von Produkten bis zu Inventureinsätzen.

Trotzdem ist an dieser Stelle noch einmal ausdrücklich zu bemerken, dass die Formulierung solcher Abfragen nicht allgemein gehalten werden kann, da eine sehr starke Abhängigkeit von den durch die Handelspartner gespeicherten Daten vorliegt. Es sollte also nur sehr schwer möglich sein, in die Ergebnis-Ereignisse etwas zu interpretieren, ohne die logischen Zusammenhänge innerhalb des anderen Unternehmens zu kennen.

Trotzdem versetzen uns die im vorherigen Abschnitt definierten Szenarien zusammen mit diesen Abfragen in die Lage, mit Hilfe der Client-Applikation eine repräsentative Darstellung des Einsatzes eines EPCIS-Systems zu machen.

5. Implementierung der einzelnen Komponenten

Ausprogrammierung des entwickelten EPCIS-Simulationssystems

Dieser Teil der Arbeit soll zeigen, auf welche Weise die Ideen des vorherigen Kapitels umgesetzt werden. Dazu gehört eine Vorstellung der eingesetzten Technologien, Sprachen und Werkzeuge, aber auch eine Präsentation des implementierungsabhängigen Aufbaus der einzelnen Komponenten.

Um das im vorherigen Kapitel entwickelte Syste-Konzept umsetzen zu können, müssen einige Einzelkomponenten entwickelt werden, die unterschiedliche technische Ansprüche mit sich bringen. Diese lassen sich gut in drei einzelne Projekte aufteilen, die aufeinander aufbauen, bzw. auch ineinander verschachtelt sind:

1. Die Erstellung einer *Bibliothek*, welche eine Implementierung der in der EPCIS-Spezifikation definierten Typen enthält, zuzüglich der allgemein benötigten Klassen der Web-Service-Schnittstellen. Diese Bibliothek soll dann später sowohl vom Server als auch vom Client verwendet werden können.

2. Die Implementierung des *Simulationsserver*, welcher die EPCIS-Daten aufnehmen und wieder zurückliefern soll. Dieser nutzt die zuvor entwickelte Bibliothek mit den Grunddatentypen.

3. Als letzter Punkt die Implementierung der *Client-Applikation*, die dem Benutzer die Generierung und das anschließende Verschicken der EPCIS-Daten anbieten soll. Auch diese benötigt natürlich die bereits erwähnte Bibliothek.

Diese Aufzählung entspricht auch ganz grob der chronologischen Ausführung des Projektes.

5.1. Technische Grundlagen für eine Implementierung

Eingesetzte Technologien, Sprachen und Werkzeuge

Zunächst einmal müssen einige technische Entscheidungen getroffen werden, um eine solche Implementierung beginnen zu können. So ist eine geeignete Programmiersprache zu finden, ein Server, auf dem die einzelnen Anwendungen laufen können und eine passende Web-Service-Engine, welche in der Lage ist, die geforderten Sicherheitsmechanismen zu integrieren.[1]

5.1.1. Wahl einer Programmiersprache

Um eine Entscheidung für eine passende Umgebung treffen zu können, muss natürlich zunächst eine Programmiersprache gefunden werden, in der das Projekt realisiert werden kann.

Da die Möglichkeit offen gelassen werden soll, das Projekt selbst, oder auch nur einzelne Teile davon, in späteren Arbeiten weiterzuverwenden, ist für die Programmierung eine Sprache zu wählen, die auf der einen Seite weit verbreitet, auf der anderen aber auch frei verfügbar ist. Zusätzlich sollte sie natürlich auch der Komplexität des Projektes, insbesondere der Struktur der Grunddatentypen, gewachsen sein.

Gerade im Hinblick auf eine webbasierte Anwendung kommen mit den gerade genannten Forderungen eigentlich nur zwei Sprachen in eine engere Betrachtung:

1. Eine webbasierte Lösung auf der Basis von *Java*.

[1]Eine genaue Liste der eingesetzten Programme, inklusive aller Versionsnummern und Internetquellen, findet sich im Anhang dieser Arbeit.

2. Oder eine Implementierung auf der Basis des *.NET-Frameworks*, in einem solchen Falle am zweckmäßigsten durch die Sprache *C#*.

Für eine Wahl der .NET-Lösung würde an dieser Stelle das *Mono-Projekt*[2] zum Einsatz kommen, da es eine Plattformunabhängigkeit garantieren kann, die für diese Arbeit unverzichtbar ist. Dieses bietet unterschiedliche Möglichkeiten für eine webbasierte Lösung auf Basis des .NET-Frameworks an und hat dadurch, natürlich gerade im Bereich Web Service, Vorteile gegenüber anderen Technologien. Ein großer Nachteil ist allerdings, dass dieses Projekt noch nicht sonderlich alt ist und sich immer noch in der Entwicklung befindet. Eine größere Verbreitung und eine gewisse Reife fehlt dadurch natürlich.

Dies ist bei der Sprache Java nicht der Fall: Diese konnte sich über die letzten Jahre deutlich weiterentwickeln, ist nach wie vor sehr beliebt und somit natürlich auch weit verbreitet. Gerade der Webbereich hat viele, teils sehr ausgereifte Technologien zu bieten, die mit vielen Beispielen im Internet dem Projekt dieser Arbeit von Vorteil werden könnten.

Aus diesen Gründen soll die Entscheidung für eine Programmiersprache bei den Techniken aus der Java-Welt liegen.

5.1.2. Suche eines passenden Web-Servers

Mit der Entscheidung für die Programmiersprache Java liegt auch der passende Webserver relativ nahe: Da für die Umsetzung des Projektes nicht zwingend die von Java unterstützten Enterprise-Technologien notwendig sind, kann auf den Einsatz eines Applikationsservers wie *JBoss*[3] verzichtet werden. Es soll im Rahmen dieses Projektes eine Umsetzung in Verbindung mit dem Servlet-Container *Tomcat* der Apache Software Foundation ausreichen (der auch innerhalb des JBoss-Applikationsservers genutzt wird), der neben den gängigen Servlet-

[2]Open-Source-Portierung des .NET-Frameworks von Microsoft, das auf verschiedenen Plattformen lauffähig ist. Internet: http://www.mono-project.com.

[3]JBoss Application Server: http://labs.jboss.com/portal/jbossas.

und JSP-Standards auch einen eigenen Webserver mitbringt, über den sich die einzelnen Komponenten aufrufen lassen.

Da der Tomcat selbst in Java geschrieben ist, lassen sich sehr einfach weitere Bibliotheken einbinden, die dann innerhalb der Programme genutzt werden können, was für das Projekt dieser Arbeit, das selbst auf viele andere Bibliotheken angewiesen sein wird, sicherlich von enorm großem Nutzen ist.

Ein weiterer Vorteil ist auf jeden Fall die einfache Installation der verschiedenen, noch zu implementierenden Programmteile mit Hilfe des Tomcats: Für jede Komponente, die während der Implementierung erstellt wird, muss nur ein sogenanntes *WAR-File* generiert werden, das bereits die vollständige Applikation enthält und einfach durch eines Neustarts des Servers installiert werden kann. Diesen Archiven können auch die erforderlichen Bibliotheken beigelegt werden, so dass das Programm mehr oder weniger unabhängig von bestehenden Installationen laufen kann.

Der Nachteil der Langsamkeit des Webservers spielt für das zu erstellende Simulationssystem keine nennenswerte Rolle.

5.1.3. Die Web-Service-Engine

Um Objekte in XML-Code übersetzen zu können, der dann mit Hilfe von SOAP ausgetauscht werden kann, ist eine geeignete Web-Service-Engine von großem Nutzen: Diese bestehen für Java-Umgebungen normalerweise aus einem Servlet, das in den Server integriert wird und anderen Applikationen die erforderlichen Schnittstellen anbietet. Die dafür nötigen Informationen werden dabei oft aus WSDL-Dateien gezogen. Sehr oft bringen diese Projekte auch eigene Werkzeuge mit, durch die sich über diese Definitionsdateien eigener Programmcode generieren lässt.

Für die EPCIS-Simulation kommen auch an dieser Stelle wieder zwei Systeme in Frage:

1. *Axis*, das Web-Service-Projekt der Apache Software Foundation.

2. Die *Web Service API* von Sun.

Auch wenn das Axis-Projekt[4] weit verbreitet ist, wurde für die Implementierung dieser Arbeit doch die Engine von Sun gewählt. Sie ist in einer sehr aktuellen, neuen Version verfügbar, die zukünftig auch in den Standard-Funktionen von Java enthalten sein soll. Diese Engine hat den großen Vorteil gegenüber von Axis, dass sie wenig und vor allem einfachen Code erzeugt, der sich später auch gut anpassen lässt. Auch die noch zu erstellenden Sicherheitslösungen lassen sich darin gut integrieren.

5.2. Erstellung einer Bibliothek von Grunddatentypen

Grunddatentypen für Server und Client

Nach der Klärung der einzusetzenden Technologien kann nun mit dem ersten Schritt der Implementierung begonnen werden: Dieser besteht aus der Erstellung einer Klassensammlung, die sowohl vom EPCIS-Server, als auch von der Client-Applikation benötigt wird. Dabei handelt es sich um die Umsetzung der Schnittstellen-Definitionen, die der Server anbieten soll und die einzelnen Klassen der für den Datenaustausch benötigten Elemente.

Die Überlegung, welche Datentypen das genau sind, gestaltet sich relativ einfach: Wie bereits erwähnt, ist einer der wesentlichen Standardisierungsziele der EPCIS-Spezifikation ja gerade eine Vereinheitlichung der Schnittstellen und der zu dieser gehörenden Datentypen. Die Spezifikation enthält somit sehr genaue Schema-Definitionen aller Typen und eine WSDL-Beschreibung der Abfrage-Schnittstelle. Da während der Entwicklung dieser Arbeit zusätzlich eine Schnittstelle für die Erfassung von Ereignissen erstellt wurde, sind bereits alle nötigen Elemente vorhanden.

[4]Internet: http://ws.apache.org/axis/, bzw. http://ws.apache.org/axis2/.

5.2.1. Generierung der Klassen

Die Web-Service-Umgebung von Sun bringt ein Werkzeug mit, durch das sich mit Hilfe von Schema-Beschreibungen sehr gut die benötigten Klassen generieren lassen.

Die Übersetzung der XML-Definitionen in Java-Typen und umgekehrt geschieht dabei über *JAXB*[5], einer Architektur, die Bestandteil der Web-Service-Engine ist und solch eine Bindung herstellen kann. Dafür werden nach einem festen Schema zu jedem Typ Klassen erstellt, die über Annotationen mit den XML-Typen verknüpft werden können.

Annotationen sind in Java erst seit Version 1.5 verfügbar und stellen eine Möglichkeit dar, Zusatzinformationen in den Quellcode einzubetten, die auch während einer Übersetzung erhalten bleiben und somit später wieder ausgelesen werden können. Eine Bindung über diesen Weg soll an folgendem Beispiel gezeigt werden:

Beispiel 5.1. Binding eines Datentypes

```
<xsd:complexType name="BusinessTransactionType">
  <xsd:simpleContent>
    <xsd:extension base="epcis:BusinessTransactionIDType">
      <xsd:attribute name="type"
        type="epcis:BusinessTransactionTypeIDType" use="optional" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="BusinessTransactionIDType">
  <xsd:restriction base="xsd:anyURI" />
</xsd:simpleType>

<xsd:simpleType name="BusinessTransactionTypeIDType">
  <xsd:restriction base="xsd:anyURI" />
</xsd:simpleType>
```

```
package net.eisprinz.epcis.core.event;

import javax.xml.bind.annotation.*;
```

[5]Java Architecture for XML Binding. Internet: http://java.sun.com/webservices/jaxb/index.jsp

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "BusinessTransactionType", propOrder = {
  "value"
})
public class BusinessTransaction
{
  @XmlValue
  protected String value;
  @XmlAttribute
  protected String type;

  public String getValue() {
    return value;
  }

  public void setValue(String value) {
    this.value = value;
  }

  public String getType() {
    return type;
  }

  public void setType(String value) {
    this.type = value;
  }
}
```

An diesem Bespiel wird ersichtlich, dass für jeden primitiven Schema-Datentyp eine Umwandlung in einen Java-Typ existiert (hier wird zum Beispiel `xsd:anyURI` zu `java.lang.String` konvertiert). Auch ist an dem Beispiel schön zu sehen, wie durch die Annotationen die einzelnen Elemente der Klasse dem XML-Schema zugeordent werden.

Die Schnittstellen selbst werden durch ein Interface und eine Port-Klasse dargestellt. Auch an dieser Stelle werden Annotationen für die Bindung verwendet, so dass eine Abbildung in beide Richtungen möglich wird. Der Server sollte dann später über eine Klasse verfügen, die das Interface implementiert und auf diesem Weg die Service-Operationen anbieten. Diese Klasse wird neben einigen anderen Eigenschaften in der Konfiguartionsdatei des Service-Servlets registriert. Client-Applikationen können dann über die Port-Klasse auf die einzelenen Funktionalitäten zugreifen.

Gerade an obigem Beispiel wird deutlich, dass diese Web-Service-Engine mit sehr wenig Code auskommt, der noch dazu gut lesbar bleibt. Dies ist bei anderen Systemen, wie beispielsweise der Axis-Engine leider nicht immer der Fall.

Auch lassen sich die Klassen problemlos um weitere Methoden erweitern, so dass gerade bei der hohen Zahl von Typen der EPCIS-Schnittstellen sehr viel für dieses System spricht.

5.2.2. Anpassung und Veröffentlichung

Die Generierung des Java-Codes lässt sich über eine Konfigurationsdatei steuern: In dieser lassen sich Ziel-Packages für die entstehenden Klassen zuordnen und es ist auch möglich, Bezeichnungen innerhalb des Codes manuell zu setzen (was im Beispiel auch gemacht wird: die Klasse heißt `BusinessTransaction`, nicht wie in der Definition `BusinessTransactionType`). Über den Mittel dieses Konzepts kann bereits während der Generierung eine Verteilung auf verschiedene Packages realisiert werden, die grob betrachtet dem modularen Aufbau der EPCIS-Spezifikation entsprechen.

Für die Klassensammlung selbst soll ein JAR-Archiv erstellt werden, auf das sowohl der Server als auch der Client Zugriff haben. Während der Implementierung dieser beiden Komponenten können dann die einzelnen Klassen der Bibliothek um zusätzliche Methoden erweitert werden, um einen optimalen Zugriff zu garantieren. Im obigen Beispiel wäre zum Beispiel ein Konstruktor sinnvoll, der sofort den Typ und den Wert des Objektes initialisiert.

Leider stellt an manchen Stellen die hohe Komplexität des Schemas ein Problem während der Generierung der einzelnen Klassen dar: Der Code lässt sich zwar grundsätzlich generieren, allerdings funktioniert er nicht sofort an allen Stellen. Gerade bei Definitionen, die Vererbung einsetzen (wie zum Beispiel beim abstrakten `EPCISEvent`), müssen manuell viele Änderungen vorgenommen werden, damit die Klassen verwendet werden können.

Solche Probleme sind bei Tests anderer Systeme mit dem EPCIS-Schema allerdings auch zu beobachten. Im Auswertungsteil dieser Arbeit werden diese Probleme noch einmal aufgegriffen werden.

5.3. Implementierung des EPCIS-Servers

Ausprogrammierung der Simulationsfassade

Nachdem der Zugriff auf die allgemeinen Datentypen geregelt ist, kann mit der Programmierung des Servers begonnen werden. An dieser Stelle ist noch einmal zu erwähnen, dass es sich dabei nicht um eine Referenz-Implementierung eines EPCIS-Servers handelt, sondern lediglich um ein System, das versucht, die EPC-Netzwerkumgebung zu simulieren.

Dieser Abschnitt soll nun zeigen, auf welche Weise die konzeptionellen Überlegungen aus dem vorherigen Kapitel umgesetzt werden können.

5.3.1. Technischer Aufbau des Servers

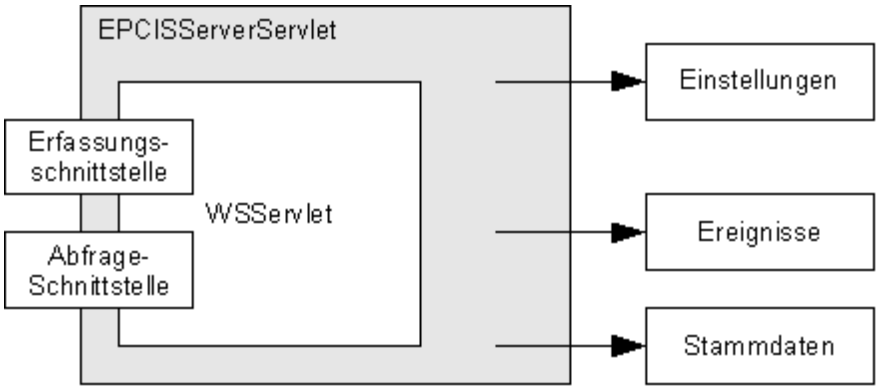
Zunächst einmal soll der allgemeine, technische Aufbau der Applikation beschrieben werden. Dabei wird auch die grundlegende Funktionsweise des Servers und die Integration der nötigen Komponenten erläutert.

Die Anforderung an den EPCIS-Server sind relativ gering: Er muss die Schnittstellen in Kombination mit den Sicherheitsmechanismen implementieren und benötigt außerdem Module für die Datenhaltung. Da die Konfiguration des Server-Systems ohnehin auf Administratorebene liegt und außerdem keine weiteren Funktionen – sieht man von den Service-Zugriffen ab – über eine Weboberfläche angeboten werden müssen, kann auf eine eigene Benutzer-Schnittstelle verzichtet werden.

Mit diesem Hintergrund bietet es sich durchaus an, den Server selbst als einfaches Servlet zu implementieren. Da die Web-Service-Engine auch durch ein Servlet realisiert ist, das natürlich auch verfügbar sein muss, lässt sich eine Beziehung zwischen diesen beiden Systemen sehr gut durch einen Vererbungsansatz herstellen: Definiert man das EPCIS-Server-Servlet als Unterklasse des Engine-Servlets, so kann es durch die „Ist-eine-Beziehung“ die Anfragen für den Web Service bearbeiten, allerdings ist auch zusätzlich möglich, weite-

re Komponenten zu integrieren. Diese Vorgehensweise soll in nachfolgender Abbildung dargestellt werden:

Abbildung 5.1. Erweiterung des Engine-Servlets



Diese Abbildung zeigt, dass das EPCIS-Servlet nicht nur Zugriff auf die einzelnen Komponenten hat, sondern durch die Vererbung auch die Dienst-Schnittstellen anbieten kann.

Der Server-Applikation selbst soll über spezielle Parameter konfigurierbar sein, die innerhalb einer XML-Datei gespeichert werden: Diese enthält Informationen über Fehlerbedingungen, den Speicherort der lokalen WSDL-Dateien, aber auch Klassennamen, die dynamisch zum Programm hinzugeladen werden müssen. Diese speziellen Einstellungen werden beim Start des Servlets einmalig geladen und sind ab diesem Zeitpunkt über ein statisches Objekt jederzeit verfügbar. Mit Hilfe der Parameter wird ein Administrator somit in die Lage versetzt, das Verhalten des Servers beeinflussen zu können.

Auch die anderen gezeigten Komponenten werden während des Ladevorgangs des Servlets erzeugt und initialisiert.

5.3.2. Implementierung der Service-Schnittstellen

Um eine konkrete Implementierung für die bereits durch das Web-Service-System generierten Interfaces anbieten zu können, müssen Klassen erstellt werden, welche diese Interfaces nach den entwickelten Vorgaben umsetzen. Die-

se Klassen müssen dann innerhalb der Engine-Einstellungen der Web-Service-Umgebung als Operationsklassen registriert werden und werden dann ab diesem Zeitpunkt bei jeder Anfrage genutzt.

Die Operationen innerhalb der Abfrage-Schnittstelle lassen sich an dieser Stelle grob in zwei Gruppen unterteilen (die Erfassungsschnittstelle besteht nur aus einer Operation):

1. Operationen, die ausschließlich Informationen zum Server und den verwendeten EPCIS-Funktionalitäten zurückliefern.

2. Operationen, die für die EPCIS-Daten selbst verantwortlich sind.

Der erste Typ ist nicht schwer zu realisieren: Für jede Information wird statisch ein Eigenschaftswert erstellt, der beim Start des Servlets über die angesprochene XML-Datei automatisch geladen wird. Während einer entsprechenden Anfrage können diese Werte dann durch Zugriff auf das allgemeine Eigenschaftsobjekt zurückgeliefert werden.

Deutlich komplexer gestaltet sich die Implementierung der anderen Operationen. Diese benötigen Zugriff auf unterschiedliche Komponenten der Server-Applikation. Um die Implementierungsklasse nicht mit den verschiedenen Funktionalitäten zu überladen, sondern vielmehr eine logische Trennung zu erhalten, sollen innerhalb dieser Klasse die Anfragen nur aufbereitet an die zuständige Komponente weitergereicht werden. Diese muss sich dann selbstständig um das weitere Vorgehen kümmern. Das gilt natürlich auch für die Operation der Erfassungsschnittstelle.

5.3.3. Realisierung der Datenhaltung

Als nächstes müssen die Komponenten für die Datenhaltung umgesetzt werden. Insbesondere sind dies zwei Systeme, eines für die Organisation der Stammdaten, eines für die Ereignisdaten.

5.3.3.1. Stammdaten

Wie bereits erwähnt, ist die Haltung der Stammdaten grundsätzlich eher statisch anzusehen: Sie ändern sich selten und das größere Gewicht liegt bei einer effizienten Auslieferung (welche innerhalb dieses Projektes aber auch keine große Rolle spielt). Aus diesem Grund soll innerhalb des Servers auch kein aufwendiges Datenbanksystem für die Stammdaten integriert werden. Die Speicherung innerhalb einer XML-Datei reicht vollkommen aus.

Dieses Vorgehen hat einen zwei große Vorteile:

1. Eine XML-Repräsentation der Stammdaten ist durch die Definitionen aus der EPCIS-Spezifikation bereits vorhanden.

2. Zusätzlich wird in der Spezifikation auch ein Dokumenten-Element für Stammdaten definiert, welches die einzelnen Wortschätze sammeln und erweitern kann. Dieses Dokument ist ideal für eine Speicherung der Stammdaten geeignet und es müssen somit keine zusätzlichen Definitionen entwickelt werden.

Für die Umsetzung wird eine Klasse erstellt, die von der bereits existierenden Dokumentenklasse erbt und diese um die Funktionalität erweitert, den Inhalt dieses zugehörigen Dokumentes aus einer Datei auszulesen. Der Inhalt selbst kann dann statisch in ein solches XML-Dokument geschrieben werden. Zusätzlich wird eine Filterklasse erstellt, über die sich das Ergebnis während einer Abfrage je nach Anfrage-Parametern anpassen lässt.

5.3.3.2. Ereignisdaten

Etwas komplexer zeigt sich die Datenhaltung für die Ereignisse, da diese im Gegensatz zu den Stammdaten dynamisch genutzt werden sollen. Eine Datenhaltung mit Hilfe einer XML-Datei ließe sich natürlich auch in diesem Fall realisieren (auch für die Ereignisdaten existiert bereits eine XML-Darstellung und die Definition für ein dazu passendes Dokument), allerdings ist dies beim

Gedanken an die größere Masse von Daten und die komplexeren Abfragen eher unzweckmäßig.

Ein weiterer Grund, der gegen eine Dateilösung spricht, ist die Tatsache, dass viele der Abfrage-Parameter die Werte innerhalb der Ereignisse vergleichen müssen, was auf dem beschriebenen Weg unter Umständen sehr kompliziert ist. Für einen solchen Zweck würde sich eine richtige Datenbank deutlich besser eignen, über die sich mit zu den Abfrage-Parametern passenden SQL-Queries sehr komfortabel die für das Ergebnis relevanten Daten ermitteln ließen.

Um die Entscheidung nicht entgültig treffen zu müssen und um auch grundsätzlich andere Lösungen zuzulassen, soll das Repository abstrakt erstellt werden. Für diesen Zweck wird eine abstrakte Basisklasse implementiert, die alle nötigen Funktionen anbietet, die der Web Service benötigt. Nun können beliebig Implementierungen dieser Klasse erstellt werden, von denen die aktuell gültige in den Einstellungen des Servers registriert wird. Das Servlet versucht dann beim Start dynamisch, die entsprechende Repository-Klasse zu laden und zu initalisieren. Auch diese Klasse soll mit einem Filterobjekt abgerundet werden, welches die Behandlung der übergebenen Abfrage-Parameter realisiert. Dessen Struktur ist so allgemein gehalten, dass es für alle Repository-Implementierungen theoretisch einsetzbar ist.

Mit diesem Konzept ist eine vielfältige Implementierung und eine theoretische Anpassung an laufende Systeme möglich. Trotzdem soll innerhalb dieses Projektes nur eine einzige davon entwickelt werden: Eine Lösung basierend auf einer extrem einfachen Datenbank, welche keinen größeren Aufbauarbeiten erfordert, sich aber trotzdem mit den üblichen SQL-Abfragen ansprechen lässt.

Ein System, das diese Anforderungen sehr gut erfüllt, ist das Datenbank-System *HSQLDB*, das darum auch für die Server-Implementierung zum Einsatz kommen soll. Es bringt einige, enorme Vorteile mit sich:

- Die Datenbank ist selbst in Java geschrieben und kann mit dem Einfügen eines einzigen JAR-Archivs in eine bestehende Software integriert werden. Der Zugriff wird dann über die Java-Standardklassen für Datenbanken realisiert.
- Sie kann unabhängig eines Servers laufen und unterstützt somit also einen sogenannten „Standalone-Modus“.
- Die Daten selbst werden dabei einfach in eine Datei geschrieben. Zusätzlich ist es aber auch möglich, unabhängig von einer solchen Datei die Daten auch nur im Speicher zu halten.
- Die Datenbank ist als quelloffenes Projekt frei verfügbar.

Die Ansätze dieser Datenbank gestaltet sich für das Projekt dieser Arbeit also äußerst zweckmäßig: Der Server wird in die Lage versetzt, ankommende Ereignisse in den Arbeitsspeicher zu schreiben und von dort über SQL-Queries wieder abzufragen. Die Daten selbst bleiben im Speicher so lange erhalten, bis der Server (in diesem Fall also das entsprechende Servlet) neu gestartet wird. Diese Konzept ist äußerst schnell, einfach und für die Nutzung dieses Projektes auf jeden Fall ausreichend.

Hinweis:

Zusätzlich wird dieses abstrakte Repository auch dafür eingesetzt, Abonnements zu speichern, die eine Client-Applikation registrieren möchte.

5.3.4. Integration der Sicherheitsmechanismen

Ein wichtiger Schritt stellt die Integration des Sicherheitskonzeptes für den Server dar: Er muss nach dem entwickelten Konzept schließlich in der Lage sein, die anfragenden Client-Applikationen voneinander zu unterscheiden. Die Lösung dieses Problems mit Hilfe des entwickelten Zertifikat-Konzepts auf der Serverseite soll in diesem Abschnitt der Arbeit beschrieben werden.

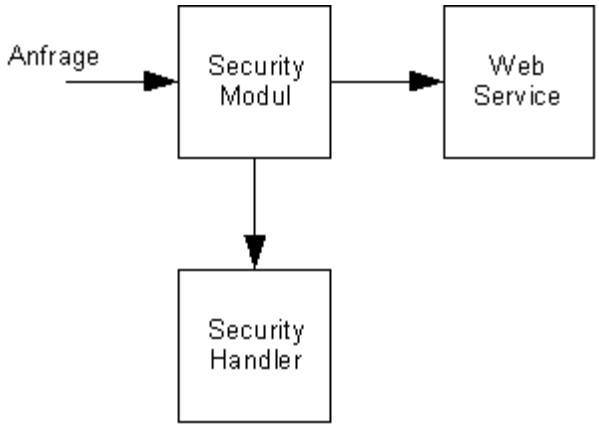
Sowohl der Client- als auch die Server-Applikation sollen für die Sicherheitsfragen das Security-System *XWSS* von Sun einsetzen, das sehr gut auf die Web-Service-Engine zugeschnitten ist.

Die Einbindung der Sicherheitsmechanismen gestaltet sich zunächst recht übersichtlich: Es werden weitere Bibliotheken eingebunden und über eine Konfigurationsdatei der Schutz des Web Service aktiviert. Ab diesem Zeitpunkt ist ein Modul verfügbar, das als eine Art Proxy jeden Zugriff auf den Web Service kontrolliert und erst nach einer Prüfung an die eigentliche Funktion weiterleitet.

In der Konfigurationsdatei selbst werden Angaben gemacht, was für Sicherheitsrichtlinien (Signierung, Verschlüsselung, usw.) bei Anfragen eingehalten werden müssen, aber auch in welcher Form eine entsprechende Antwort verschickt wird. Zusätzlich muss ein Handler geschrieben werden, der über eine spezielle Callback-Funktion verfügt, die wiederum dem Sicherheitsmodul Elemente wie Schlüssel und andere Informationen zur Verfügung stellt und Entscheidungen darüber trifft, ob ein Zugriff gültig ist oder nicht.

Der grundsätzliche Aufbau dieses Konzeptes wird in nachfolgender Abbildung illustriert:

Abbildung 5.2. Funktionsweise des Sicherheitskonzeptes

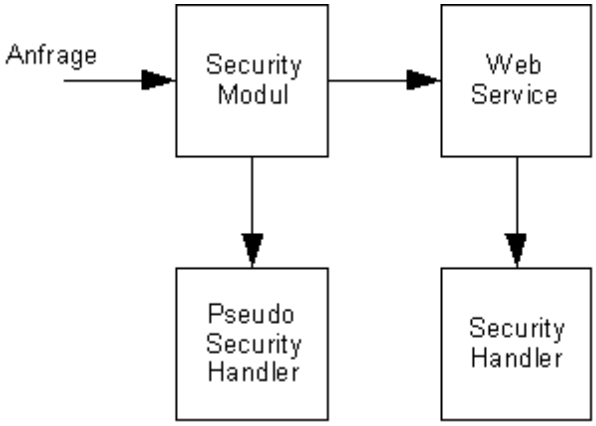


Dabei ist sehr wichtig zu beachten, dass im Fehlerfall die Anfrage den Web Service nie erreicht: Das Sicherheitsmodul generiert selbstständig eine Ausnahme, die dann an den Client zurückgeschickt wird.

Dieses Vorgehen nimmt dem Programmierer zwar einerseits viel Arbeit ab, stellt für das Projekt aber ein Problem dar: Die EPCIS-Spezifikation definiert eine eigene Sicherheitsausnahme, wenn eine Operation nicht ausgeführt werden darf. Da die Implementierung der Operationen im Fehlerfall allerdings nie erreicht wird, ist es leider nicht möglich, diese EPCIS-Ausnahme zu versenden. Darüber hinaus wird auch eine Definition von unterschiedlichen Sicherheits-Anforderungen für jede einzelne Funktion sehr schwierig.

Um dieses Problem lösen zu können, muss das eben vorgestellte Sicherheitskonzept um eine weitere Komponente ergänzt werden:

Abbildung 5.3. Erweiterung des Sicherheitskonzeptes



Dafür wird zusätzlich eine spezielle Handler-Klasse geschrieben, die alle Anfragen zunächst als gültig abzeichnet. Verknüpft man nun das Sicherheitsmodul mit dieser neuen Klasse, wird natürlich auch jede Anfrage an den Web Service weitergeleitet. Innerhalb der dortigen Implementierung kann nun der richtige Handler aufgerufen werden, um diesmal eine gültige Sicherheitsentscheidung treffen zu können. Auf diesem Weg kann jede Funktion unterschiedlich behandelt werden und es ist möglich, mit denen durch die EPCIS-Spezifikation definierten Ausnahmen auf Fehler zu reagieren.

Der Server ist dabei so konfiguriert, dass er grundsätzlich Anfragen ohne Signatur zulässt, allerdings nur bei den Informationsoperationen. Alle anderen setzen einen gültigen Benutzer voraus. Dessen Name wird beispielsweise dann wäh-

rend einer Erfassung von Ereignissen zu jedem einzelnen Datensatz gespeichert, um später auch wieder genau die Ereignisse zurückliefern zu können, die zu einem bestimmten Benutzer gehören. Zusätzlich ist es möglich, über die Einstellungsparameter einen administrativen Benutzer festzulegen, der alle Ereignisse sehen darf. Die Antworten des Servers werden immer signiert.

Die einzelnen Identitäten werden mittels Zertifikaten gespeichert und ausgewertet. Für diesen Zweck wird innerhalb des Servers ein *KeyStore*[6] integriert. Innerhalb dieser Datei können dann die einzelnen Zertifikate des Clients und das Schlüsselpaar des Servers abgelegt werden. Java bringt von Hause aus Werkzeuge für die Erstellung von Schlüsseln und deren Organisation (also auch den genannten KeyStore) mit.

Mit der Integration der Sicherheitskonzeptes sind die Arbeiten am Server abgeschlossen. Es ist nun möglich, gesicherte Anfragen an die beiden Schnittstellen zu stellen. Der Server liefert dann die Antworten entsprechend des anfragenden Benutzers zurück.

Anmerkung:

Worauf während der Implementierung vollständig verzichtet wurde, ist eine Benutzerunterscheidung für die Auslieferung der Stammdaten. Da es ohnehin nicht möglich ist, diese Daten über Service-Zugriffe zu erstellen oder zu bearbeiten, ist es für dieses Projekt auch wenig sinnvoll, eine Benutzerunterscheidung für dieses Daten zu implementieren.

Die Stammdaten, die bereits beispielhaft innerhalb des EPCIS-Servers gespeichert sind, entsprechen im Großen und Ganzen denen des im letzten Kapitel entwickelten Fallbeispieles.

[6]Spezielle, verschlüsselte Datei, in der Zertifikate und Schlüssel gespeichert werden können. Der Zugriff wird durch ein Passwort geschützt.

5.4. Implementierung des EPCIS-Clients

Zugriffe auf den Server-Stub

Der vorerst letzte Programmierabschnitt dieses Projektes stellt die Client-Applikation innerhalb des Simulationssystems dar. Sie repräsentiert einen der wesentlichen Teile des Projektes, allein schon deswegen, weil sie dem Anwender die Möglichkeit gibt, aktiv ins Geschehen einzugreifen.

Wie während der Entwicklung bereits geplant, soll diese Anwendung in drei einzelne Module unterteilt werden:

1. Ein Modul, welches den Zugriff auf die Erfassungsschnittstelle des EPCIS-Servers ermöglicht.

2. Ein weiteres Modul für den Zugriff auf die Abfrage-Schnittstelle.

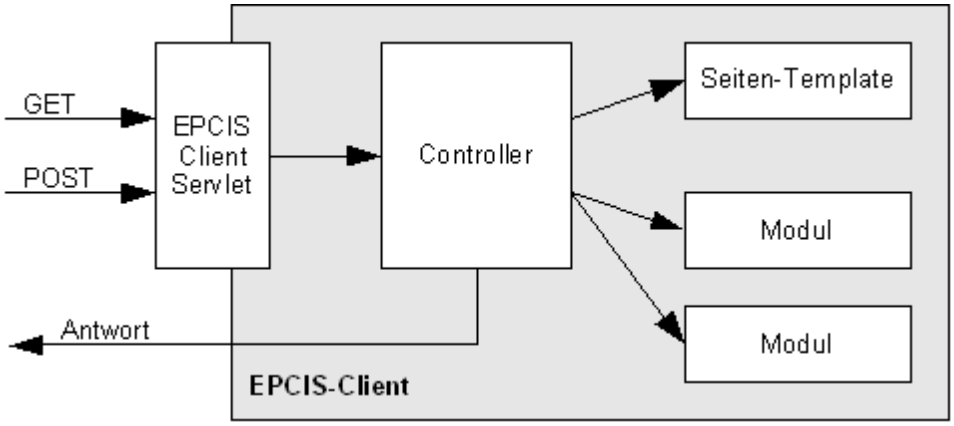
3. Das Szenario-Modul, welches diese beiden Schnittstellen beispielhaft nutzt.

Diese Module entsprechen somit den logischen Einheiten des Client-Systems, die auch für die Realisierung eingehalten werden sollen. Innerhalb der nächsten Abschnitte wird somit beschrieben, auf welche Weise das bereits entwickelte Konzept für die Client-Applikation im Zusammenhang mit dieser logischen Struktur umgesetzt werden kann.

5.4.1. Bestandteile der Client-Applikation

Obwohl die Client-Anwendung selbst über einen webbasierten Aufbau verfügen soll, so wird sie trotzdem als Herzstück einen Controller erhalten, der den Knotenpunkt innerhalb der Anwendung bildet. Dadurch wird ein deutlich strukturierter Aufbau möglich und die einzelnen Funktionalitäten können besser gekapselt werden. Folgende Abbildung soll verdeutlichen, auf welche Weise dieser Controller integriert ist:

Abbildung 5.4. Vereinfachte Darstellung des Client-Aufbaus



Der dargestellte Controller wird einmalig bereits beim Start des Servlets erzeugt und ist ab diesem Zeitpunkt verfügbar. Die eigentliche Web-Applikation ist dabei so konfiguriert, dass alle Anfragen, die das Servlet erreichen, direkt und ungefiltert an den Controller weitergeleitet werden. Dieser kümmert sich dann selbstständig darum, dass die Anfrage ausgewertet, eine Antwort erzeugt und diese an den Anwender zurückgeschickt wird.

Für diesen Zweck läd der Controller ein Template, das den allgemeinen Seitenaufbau (Kopfbereich der Seite, Navigation, usw.) enthält. Zusätzlich wird je nach angefragtem Dateinamen ein Modul erzeugt, welches in der Lage ist, mit Hilfe der Anfrage den eigentlichen Inhalt der Seite zu erzeugen. Diesen fügt der Controller dann in das Template ein. Der daraus entstehende Code wird dann an den Anwender zurückgeschickt.

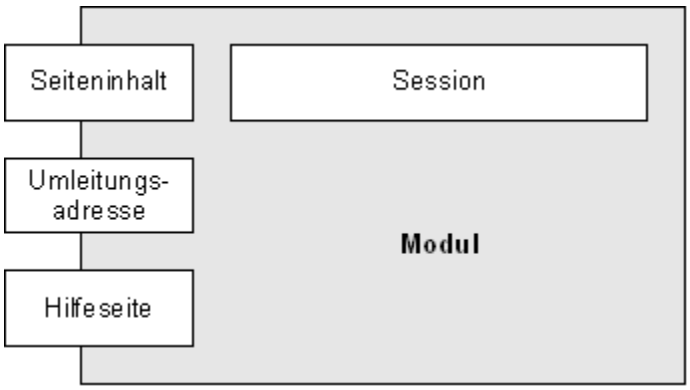
Eine Ausnahme bilden dabei nur die Seiten, die unabhängig von einem Modul funktionieren. Dies ist zum Beispiel bei der Startseite der Fall, die kein Modul benötigt. Für solche Seiten hält die Template-Klasse Methoden bereit, die dann realtiv statisch den entsprechenden Code erzeugen.

5.4.1.1. Allgemeiner Aufbau eines Moduls

Jedes Modul ist als eigenständige Einheit innerhalb der Applikation gedacht. Es erhält bereits während seiner Erzeugung alle Anfragedaten und verfügt somit auch über einen entsprechend großen Aktionsradius.

Damit die Funktionsweise eines Moduls allerdings einigermaßen gleich bleibt und der Controller sie auch unabhängig verwenden kann, müssen sie über eine gemeinsame Schnittstelle verfügen, die jedes Modul implementiert. So wäre theoretisch der Client auch in der Lage, später noch weitere Module zu verwenden. Der Aufbau der betreffenden Schnittstelle soll in nachfolgender Abbildung gezeigt werden:

Abbildung 5.5. Modul-Schnittstelle



Die Abbildung zeigt, dass jedes Modul über drei grundlegende Methoden verfügen muss, die der Controller aufrufen kann:

- Eine für den *Seiteninhalt*. Diese liefert den eigentlichen Inhalt der zu erzeugenden Seite als Zeichenkette zurück.

- Eine für die *Umleitungsadresse*. Wird eine Umleitung auf eine andere Seite benötigt, kann über diese Methode eine Adresse zurückgegeben werden.

- Zuletzt eine für die Adresse der *Hilfeseite*. Bietet das Modul eine Hilfe an, so kann über diese Methode die Adresse ermittelt werden, die dann in der Navigation angezeigt wird.

Da das Modul bereits während seiner Erstellung die vollständige Anfrage erhält, ist es auch in der Lage, sich selbst um eine Sessionverwaltung zu kümmern. Es ist also davon auszugehen, dass jedes Modul, welches Daten speichern muss, sich auch eine Session anlegt.

Auch wenn es nicht durch die Schnittstelle zwingend gefordert ist, arbeiten die Module nach einem festen Schema:

1. Zuerst wird überprüft, ob die Anfrage den Parameter `exec` enthält. Ist dieser gesetzt, so bedeutet dies, dass ein ausführender Zugriff gemacht wird, also zum Beispiel das verschicken an den Web Service.

   In einem solchen Fall führt das Modul den Befehl aus, aktualisiert dann in der Regel die Session und fordert anschließend vom Controller eine Umleitung auf eine andere Seite. Dadurch kann ein mehrfaches Ausführen durch Aktualisieren der Seite vermieden werden.

2. Ist dies nicht der Fall, so wird der Parameter `page` gesucht. Ist dieser gesetzt, so wird ein vom Wert abhängiges Objekt erzeugt, dass den eigentlichen Seiteninhalt zurückliefert. Auch diese speziellen Seiteninhaltsklassen haben ein feste Schnittstelle.

   Über diesen Mechanismus ist weitestgehend eine Trennung von Logik und Inhalt möglich. Wird keine Seite explizit verlangt, so liefert das Modul den Inhalt einer Standardseite.

Sollte an irgendeiner Stelle dieses Weges ein Fehler auftreten, so kann das Modul jederzeit eine Ausnahme werfen, woraufhin der Controller statt der gewünschten Seite eine Fehlerseite erzeugt und zurückliefert. Diese enthält neben einem Beschreibungstext des aufgetretenen Fehlers auch eine Adresse, die als Link zum Fortfahren angezeigt wird.

5.4.1.2. Integration eines Erweiterungsansatzes

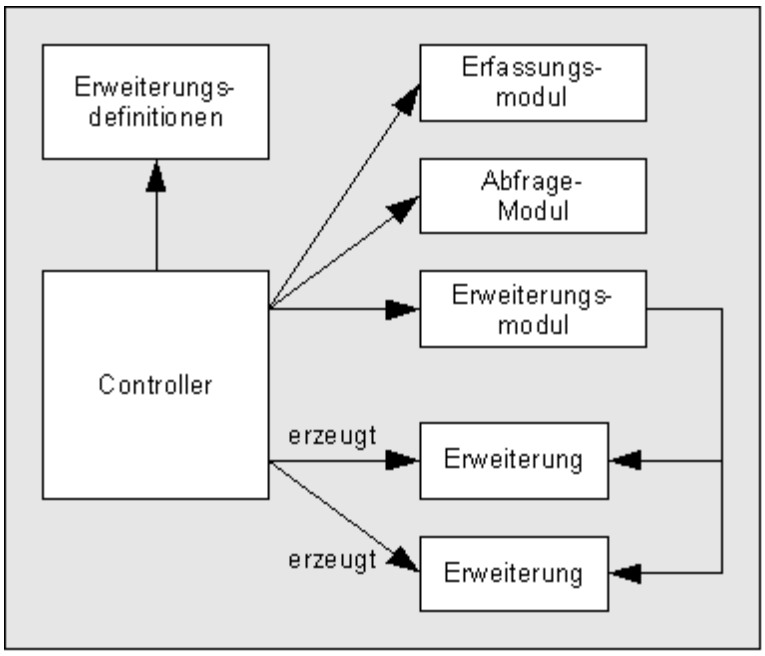
Auch wenn die Module einen allgemeinen Aufbau besitzt, müssen diese trotzdem auf irgend eine Weise beim Controllers registriert werden. Um dabei eine gewisse Dynamik zu gewährleisten, wäre es somit wünschenswert, einen Erweiterungsmechanismus zu integrieren, der es Entwicklern ermöglichen soll,

weitere Module zu schreiben, ohne den eigentlichen Quellcode der Anwendung ändern zu müssen.

Um diese Idee realisieren zu können, soll das vorbereitete Fallbeispiel nicht, wie bisher geplant, als eigenständiges Modul implementiert werden, sondern in Form einer solchen Erweiterung. Das Szenario bietet sich für einen derartigen Fall hervorragend an, da es im Prinzip keine neuen Zugriffe implementiert, sondern vielmehr die der anderen Module nutzt. Zusätzlich kann es auf diesem Weg auch als Beispiel für spätere Erweiterungen dienen.

Um innerhalb der Client-Applikation solche Erweiterungen einbinden zu können, müssen natürlich einige kleine Änderungen am bisherigen Konzept vorgenommen werden. Der prinzipielle Aufbau aus der Entwicklung des letzten Kapitels (insbesondere die Unterteilung in einzelne nach Funktion unterteilten Module), bleibt dabei allerdings weitstgehend erhalten. Nachfolgende Abbildung stellt schematisch dar, auf welche Weise die Erweiterungen eingebunden und genutzt werden:

Abbildung 5.6. Schematische Darstellung des Erweiterungsmechanismus



Neben den bekannten Modulen für Erfassung und Abfragen wird jetzt ein drittes für die Erweiterungen definiert. Auch dieses implementiert selbstverständlich die allgemeine Modul-Schnittstelle. Seine Aufgabe besteht darin, die Anfragen, welche Erweiterungen betreffen, an die zuständigen Klassen weiterzuleiten.

Diese Klassen müssen alle von einer abstrakten Erweiterungsklasse erben, welche die entsprechenden Methoden bereithält, die das Modul benötigt. Neben den bereits bekannten Methoden für Seiteninhalt und Hilfeseite bietet diese Klasse eine Funktion für die Ausführung von Befehlen an. Diese kommt zur Anwendung, wenn das Erweiterungsmodul den Parameter `exec` findet. Nach der Ausführung des Befehles leitet das Erweiterungsmodul selbstständig eine Umleitung ein. Auf diesem Weg wird die Entscheidung für eine Ausführung und der anschließenden Umleitung nicht in die Hände der Erweiterung gelegt und somit eine der wesentlichen Fehlerquellen ausgeschlossen.

Die Erweiterungsklassen selbst müssen in einer XML-Datei zusammen mit einer eindeutigen Bezeichnung definiert werden. Wird durch eine entsprechende Anfrage bedingt eine Klasse vom Erweiterungsmodul benötigt, so versucht der Controller mit Hilfe der XML-Definitionen innerhalb der Datei die dort registrierte Klasse dynamisch zu laden und eine entsprechende Instanz zu erzeugen. Diese kann er dann an das Erweiterungsmodul ausliefern.

Beispiel 5.2. Ausschnitt der Erweiterungsdefinition

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

  [...]

  <extensions>
    <extension id="scenario">
      <title>EPCIS Scenario Extension</title>
      <class>
        net.eisprinz.epcis.client.extension.scenario.ScenarioExtension
      </class>
      <stylesheet>css/extensions/scenario/scenario.css</stylesheet>
    </extension>
  </extensions>

</epcis-client-settings>
```

Hinweis:

Innerhalb dieser Definitionen können noch weitere Angaben zu den Erweiterungen gemacht werden. Dazu gehören ein Titel und auch eine Referenz auf ein Stylesheet, das der Controller dann zusätzlich in die Seite integriert.

Über diesen Erweiterungsmechanismus ist ein Programmierer also jetzt in der Lage, die Client-Applikation um zusätzliche Funktionen zu erweitern: Er muss nur eine Klasse schreiben, die von der Erweiterungsklasse erbt und diese dann in der Datei registrieren. Nach einer Veröffentlichung der Klassen und einem Neustart des Servers kann diese Erweiterung bereits genutzt werden. Natürlich kann diese auf alle Bestandteile der Client-Applikation problemlos zugreifen, insbesondere auch auf die Schnittstellen des EPCIS-Servers.

5.4.1.3. Erweiterung um eine Benutzerverwaltung

Ein weiteres Element, das global über die ganze Anwendung verfügbar sein muss, ist eine Benutzerverwaltung. Sie soll dem Anwender die Möglichkeit geben, sich beim Programm anzumelden zu können, um dann in einer bestimmten Rolle Server-Zugriffe machen zu können.

Für diesen Zweck werden folgende Benutzerrollen für Testzwecke definiert:

1. Ein Benutzer mit dem Namen *user*: Dieser entspricht einem ganz allgemeinen Standard-Benutzer, der für Testzwecke genutzt werden kann.

2. Ein weiterer Benutzer namens *administrator*: Wie der Name schon sagt repräsentiert dieser eine Administrator-Rolle. Die Besonderheit ist, dass er auch innerhalb des Servers als Administrator registriert ist und bei Abfragen Zugriff auf die Daten aller Benutzer hat.

3. Zusätzlich ein Benutzer namens *scenario*: Dieser entspricht dem Benutzer *user*, wird aber zwingend von der Fallbeispiel-Erweiterung verlangt.

4. Zuletzt ein Benutzer mit dem Namen *scamp*: Dieser Benutzer repräsentiert einen Schurken, der innerhalb des Servers kein gültiges Zertifikat besitzt. Er kann für Sicherheitstests verwendet werden.

Ähnlich wie während der Server-Implementierung werden auch innerhalb der Client-Anwendung für jeden dieser Benutzer ein Schlüsselpaar generiert, die auch an dieser Stelle innerhalb eines KeyStores gespeichert werden. Zusätzlich verwaltet dieser das Zertifikat des EPCIS-Servers, damit dessen Identität überprüft werden kann.

Um eine Anmeldung zu realisieren, bietet der Controller jetzt eine weitere, allgemeine Seite an, auf der sich ein Anwender einloggen kann. Auf dieser werden die gerade definierten Benutzer angeboten. Als Passwort wird jenes verwendet, das vom KeyStore für die Herausgabe des privaten Schlüssels verlangt wird.

Die Einbindung einer Signatur für die Server-Zugriffe gestaltet sich diesmal etwas einfacher: Vor einem Zugriff muss der Sicherheitsmechanismus einfach an den Service-Port gebunden werden. Auch hier arbeitet das Sicherheitspaket wieder mit einer Handler-Klasse, in der die Sicherheitseinstellungen implementiert werden können.

Der EPCIS-Client selbst arbeitet dabei mit einer festen Strategie: Ist gerade ein Benutzer angemeldet, so wird eine Anfrage mit seiner Identität signiert, ist der Benutzer nicht bekannt, so wird versucht, den Zugriff ohne Signatur zu machen (was bei manchen Operationen ja auch erlaubt ist).

5.4.2. Erstellung des Erfassungsmoduls

Innerhalb dieses Moduls sollen EPCIS-Ereignisse erstellt und an den Server verschickt werden können.

Für diesen Zweck bietet das Modul für jede Ereignisart eine einzelne Seite an, die ein Formular mit den dazugehörigen Feldern enthält. Diese können dann

beliebig ausgefüllt und anschließend gespeichert werden. Für diesen Zweck nutzt das Modul die Möglichkeit, Daten innerhalb einer Session abzulegen.

Da viele der Felder als Wert ein vordefiniertes Wortschatz-Element erwarten, sollte es eigentlich nahe liegen, bereits innerhalb der Formulare solche Werte anzubieten. Zweckmäßigerweise würde man diese Stammdaten beim Server abfragen und sie dann als Option der einzelnen Felder innerhalb des Formulars auflisten.

Auch wenn dies mit Sicherheit der Weg ist, der die Logik hinter den Ereignissen am besten treffen würde, soll dieser für das Erfassungsmodul des EPCIS-Clients nicht gewählt werden. Der Anwender soll selbst die Möglichkeit bekommen, zu entscheiden, mit was für Werten die einzelnen Felder belegt werden sollen. Dafür gibt es zwei Gründe:

1. Das Modul soll einen möglichst allgemeinen Anwendungsbereich zulassen. Ein Verknüpfung mit bestimmten Werten ist in der Realität auf jeden Fall sinnvoll, schränkt aber auch das Modul in seinen Möglichkeiten ein.

2. Für die Beschaffung der benötigten Stammdaten wäre ein Abfrage-Zugriff nötig, da die benötigten Wortschätze auf dem Server liegen. Die beiden Hauptmodule des EPCIS-Clients zeichnen sich aber ja gerade dadurch aus, dass sie getrennt voneinander die Zugriffe auf die beiden Schnittstellen des Servers implementieren. Aus diesem Grund sollen innerhalb des Erfassungsmoduls keine Abfragen gemacht werden.

Sind die gewünschten Ereignisse erstellt, so können sie an den Server geschickt werden. Da für jeden Server-Zugriff die WSDL-Beschreibung des zugehörigen Dienstes benötigt wird[7], bietet das Modul die Möglichkeit an, eine URL zu einer solchen Datei anzugeben, deren enthaltene Service-Adresse dann für den eigentlichen Zugriff verwendet wird. Wird keine solche Beschreibung definiert,

[7]Die Beschreibung wird nicht nur für die Lokalisierung des Service-Ports gebraucht, sondern auch, um die einzelnen Java-Klassen in XML-Form darzustellen. Sie muss also bei jedem Zugriff vorhanden sein.

versucht das Modul, lokal eine zu finden (deren Standardort ist innerhalb der Einstellungsdatei des EPCIS-Clients definiert). Eine lokale Standard-Beschreibung hat natürlich den Vorteil, dass nicht für jeden Zugriff die nötigen Daten geladen werden müssen.

Wurden die vordefinierten Ereignisse erfolgreich an den Server übertragen, so werden sie automatisch innerhalb der Session wieder gelöscht. Natürlich ist es trotzdem möglich, auch nach dem Versenden wieder neue Ereignisse zu erstellen und zum Server zu schicken.

Mit diesen Funktionalitäten sind die Arbeiten am Erfassungsmodul bereits abgeschlossen.

5.4.3. Erstellung des Abfrage-Moduls

Ein deutliches Stück umfangreicher präsentiert sich das Abfrage-Modul: Es ist durch die zahlreichen Abfrage-Parameter nicht nur um einiges komplexer als das Erfassungsmodul, sondern allein die größere Menge an Web-Service-Operationen erfordert diese Komponente eine erweiterte Zuwendung.

Aus diesem Grund sollen an dieser Stelle zunächst einmal die Grundfunktionalitäten dargestellt werden, die das Modul anbieten soll:

1. Auch dieses Modul soll es dem Anwender ermöglichen, eine vom Standard abweichende Service-URL anzugeben.

   Da dies völlig analog zur Erfassungsschnittstelle geschieht, soll innerhalb dieses Abschnitts nicht weiter darauf eingegangen werden.

2. Als zweite Möglichkeit soll das Modul Zugriff auf die informativen Operationen des Abfrage-Dienstes machen können. Dazu gehört die Abfrage der EPCIS-Version, die anwenderspezifische Version und die vom Server unterstützen Abfragetypen.

Diese Werte sollen auf einfachste Weise beim Server abgefragt werden, um sie daraufhin innerhalb der Modul-Session speichern zu können. Selbstverständlich soll auch eine Aktualisierung möglich sein.

3. Der dritte Block soll sich dann mit der Formulierung von Abfragen beschäftigen.

   Da dies eines der wesentlichen Elemente der gesamten Client-Applikation ist, soll dieser Punkt auf den nächsten Seiten ausführlicher behandelt werden.

4. Zu guter Letzt fehlt natürlich noch der Punkt innerhalb des Moduls, über den sich die gerade formulierten Abfragen auch ausführen lassen. Auch dieser Punkt wird genauer behandelt werden.

5.4.3.1. Formulierung einzelner Abfragen

Ähnlich den Ereignissen des Erfassungsmoduls sollen die einzelnen Abfragen zunächst innerhalb einer Session erstellt werden, bevor sie an den EPCIS-Server geschickt werden können. Dies bringt dem Anwender den großen Vorteil, dass er die Abfragen mehrfach hintereinander ausführen kann, ohne sie jedes Mal wieder neu erstellen zu müssen. Auch leichte Abwandlungen zu Testzwecken werden so vereinfacht.

Um dabei zu verhindern, das Abfragetypen genutzt werden, die der EPCIS-Server nicht unterstützt, sollen nur diejenigen angeboten werden, die der Server auf die entsprechende Informationsanfrage zurückgeliefert hat. Aus diesem Grund müssen auf jeden Fall zuerst die Ereignistypen abgefragt werden, bevor eine entsprechende Abfrage erstellt werden kann. Sollte der Server einen Typ zurückliefern, den der Client nicht unterstützt, so wird dies entsprechend angezeigt.

Der Client selbst soll nach den Minimalanforderungen der EPCIS-Spezifikation die beiden allgemeinen Abfragearten unterstützen: Die *einfache Ereignisabfrage* und die *einfache Stammdatenabfrage*. Sind diese Typen bekannt, so kann

eine entsprechende Abfrage erstellt werden. Jede erhält dabei einen eindeutigen Namen (nicht zu verwechseln mit dem *QueryName*), über den die Abfrage innerhalb der Client-Anwendung zu identifizieren ist.

Ist die Abfrage erstellt, so kann sie konfiguriert werden. Dies heißt genauer, dass einzelne Parameter für die Abfrage gesetzt werden können. Dies gestaltet sich in technischer Sicht aus zwei Gründen äußerst kompliziert:

1. Der Wert eines Parameters hat innerhalb der Java-Klasse den Datentyp `Object`. Das heißt, dass bei Zugriffen auf den Parameterwert sehr oft eine Überprüfung auf den richtigen Typ gemacht werden muss, da dieser manchmal auch eine komplexe Klasse (wie zum Beispiel die spezifizierte String-Liste) sein kann.

2. Wie im theoretische Teil beschrieben haben viele der Parameternamen eine generische Form, die natürlich auch gesondert behandelt werden muss.

Um die unterschiedlichen Parameter berücksichtigen zu können, wird für jeden Abfragetyp eine eigene Klasse erzeugt, die wiederum von einer abstrakten Abfrageklasse erbt. Die Klassen enthalten eine statische Liste von Parameter-Informationen, die durch den jeweiligen Abfragetyp unterstützt werden. Aus diesen Information lassen sich dann die Struktur und der Inhaltstyp des Parameters ermitteln.

Beispiel 5.3. Darstellung von Parametern (Ausschnitte)

```
package net.eisprinz.epcis.client.query;

public class ParameterInfo
{
  [...]

  public static final boolean GENERIC_YES = true;
  public static final boolean GENERIC_NO = false;

  private String parameterName;
  private String valueClass;
  private boolean required;
  private boolean extendable;
  private boolean generic;
```

```
  public ParameterInfo(String parameterName, String valueClass,
    boolean required, boolean extendable)
  {
    [...]
  }

  [...]
}
```

```
package net.eisprinz.epcis.client.query;

public class SimpleEventQuery extends Query
{
  protected static Map<String, ParameterInfo> PARAM_INFO =
    new HashMap<String, ParameterInfo>();

  static
  {
    PARAM_INFO.put("eventType",
      new ParameterInfo(
        "eventType",
        "ArrayOfStrings",
        ParameterInfo.REQUIRED_NO,
        ParameterInfo.EXTENDABLE_NO
      )
    );

    PARAM_INFO.put("GE_eventTime",
      new ParameterInfo(
        "GE_eventTime",
        "Time",
        ParameterInfo.REQUIRED_NO,
        ParameterInfo.EXTENDABLE_NO
      )
    );

    [...]
  }

  [...]
}
```

Über diesen Mechanismus können dann problemlos Parameter erstellt werden. Das Speichern selbst geschieht wieder durch eine Kombination aus `String` (für den Namen des Parameters) und aus `Object` (für den Wert). Da der eigentliche Datentyp des Wertes durch die Informationsliste bekannt ist, bringt diese Art der Darstellung auch keine größeren Probleme mehr mit sich.

5.4.3.2. Ausführung der Abfragen

Nachdem Abfragen formuliert werden können, soll es natürlich auch möglich sein, diese an den EPCIS-Server zu schicken.

Wie bereits im theoretische Teil dieser Arbeit beschrieben, definiert die EPCIS-Spezifikation zwei Mechanismen für eine solche Abfrage:

1. Den asynchronen *Weg über ein Abonnement*: Hierbei wird ein regelmäßiges Ausführen einer Abfrage beim EPCIS-Server registriert. Dieser schickt das Ergebnis dann selbständig an ein festgelegtes Ziel.

2. Eine synchrone *Ausführung im RPC-Stil*: Dies ist der Weg, der für den EPCIS-Client entscheidend ist.

Der Mechanismus einer Ausführung über eine Abonnement dürfte in der Realität durchaus sinnvoll sein, da sich auf diesem Weg verschiedene Prozesse gut automatisieren lassen. Für einen Test-Client, der zu demonstrativen Zwecken Abfragen verschickt und auswertet, ist er aber eher zweitrangig.

Aus diesem Grund soll das Abfrage-Modul auch keine größere Implementierung für diese Funktionalität erhalten. Es ist lediglich ein Dialog vorgesehen, in der sich solche Abonnements formulieren und konfigurieren lassen, um deren Bestandteile zu demonstrieren. Diese können dann beim Server registriert und auch wieder abgerufen werden[8]. Natürlich werden dabei die in der Session vorbereiteten Abfragen verwendet.

Deutlich wichtiger ist für dieses Projekt die Möglichkeit einer direkten Ausführung: Dabei wird die Abfrage zum Server geschickt und man erhält sofort ein Ergebnis.

Die Implementierung dieses Teiles gestaltet sich relativ einfach: Es wird für jede bereits formulierte Abfrage angeboten, diese auf dem Server auszuführen. Das Ergebnis wird dann in der Session zuzüglich eines Zeitstempels gespeichert. Dabei wird immer das vorherige Ergebnis überschrieben, so dass der Anwender immer Zugriff auf das aktuellste hat.

[8]Der Server dieses Projektes speichert nur die einzelnen Bestandteile des Abonnements. Ein Ausführung der Abfrage und eine Auslieferung des Ergebnisses findet auf diesem Weg nicht statt, da sich dies zu weit von den Zielen des Simulationssystems entfernen würde.

Zuletzt fehlt innerhalb dieses Moduls noch ein Mechanismus, um die erhaltenen Ergebnisse darzustellen. Da an dieser Stelle eher eine technische Darstellung der EPCIS-Daten im Vordergrund steht, kann auch das Ergebnis dieser Sichtweise folgen: So ist es auf jeden Fall ausreichend, einfach die einzelnen EPCIS-Daten (also entweder Ereignisse oder Wortschatz-Elemente) mit ihren jeweiligen Bestandteilen in Listenform anzuzeigen.

5.4.4. Implementierung der Szenario-Erweiterung

Der letzte Baustein des Simulationssystemes dieser Arbeit ist die Erweiterung um das Fallbeispiel. Wie bereits beschrieben bringt es im Gegensatz zu den anderen Modulen keine technischen Neuerungen mit, sondern versucht vielmehr, die Ereignisse und die Abfragen nicht aus technischer Sicht, sondern aus einem Anwendungskontext heraus zu betrachten. Für diesen Zweck wurde bereits während der Planungsphase dieser Arbeit ein entsprechendes Szenario entwickelt.

Der Ablauf des Szenarios ist so gedacht, dass der Anwender einen Operationsschritt nach dem anderen in einer vorgeschriebenen Reihenfolge ausführen kann. Dazu erhält er die Möglichkeit, über ein festes Sortiment von Abfragen jederzeit unterschiedliche Informationen vom Server zu abzurufen. Um die Möglichkeit eines Vergleiches zu bekommen, soll innerhalb einer Session die aktuelle Situation des Szenarios parallel zu den Ereignissen auf dem Server protokolliert werden. Damit erhält die Erweiterung folgende Bestandteile:

1. Für jedes Mitglied des Fallbeispieles wird eine Informationsseite angeboten. Auf dieser wird der Aufbau der Lokation des Teilnehmers beschrieben und was sich zum gegebenen Zeitpunkt an Paletten und Waren an diesem Ort befindet. Diese Daten sollen aus der Session genommen werden, um keine Abfragen machen zu müssen.

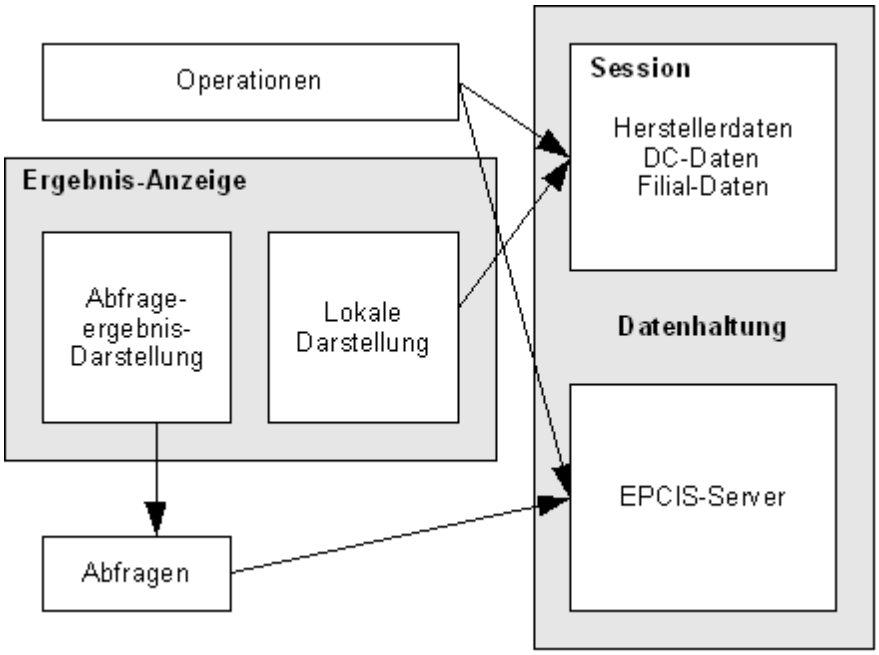
2. Zusätzlich hat jeder Szenario-Teilnehmer einen Katalog an vordefinierten Abfragen, die bereits im theoretischen Teil dieser Arbeit entwickelt wurden.

Zu jeder dieser Abfrage sollen auch die technischen Informationen abrufbar sein.

3. Weiter gibt es ein Feld, in dem der nächste Operationsschritt angezeigt wird und natürlich auch ausgelöst werden kann. Auch für die Operationen sind technische Details verfügbar.

Daraus lassen sich folgende Zugriffe schematisch darstellen:

Abbildung 5.7. Zugriffe der Szenario-Komponenten



- Beim Ausführen einer Operation wird die Darstellung innerhalb der Session aktualisiert und die erforderlichen Ereignisse an den EPCIS-Server geschickt.

- Eine Detaildarstellung der aktuellen Mitglieder-Situation wird ausschließlich aus den Daten innerhalb der Session erzeugt.

- Während der Ausführung einer Abfrage werden die entsprechenden Daten über eine Abfrage vom Server geholt. Das Ergebnis entspricht dann der jeweiligen Interpretation der erhaltenen Ereignisse. Die Daten innerhalb der Session werden dafür nicht genutzt.

Um dies realisieren zu können, muss diesmal eine relativ aufwendige Session-Struktur erstellt werden: Sie enthält einzelne Objekte, welche die Netzwerk-Mitglieder und natürlich ihren aktuellen Warenbestand darstellen. Zusätzlich werden weitere Werte wie die simulierte Zeit über diese Session verwaltet.

Die Operationen werden über eine Aufzählung organisiert, so dass ein sehr eleganter Weg für eine chronologische Ausführung möglich wird. Die Operationen selbst werden über eine abstrakte Operationsklasse erzeugt, von der sie gleichzeitig erben und enthalten neben Informationstexten für die Anzeige auch die erforderlichen Ereignisse, die dann an den EPCIS-Server geschickt werden sollen.

Fast analog werden die Abfragen organisiert: Auch ihre Erstellung wird über eine abstrakte Klasse realisiert und sie enthalten natürlich die jeweiligen Parameter, über die die betreffende Abfrage definiert ist. Die Implementierung soll an einem Beispiel gezeigt werden:

Beispiel 5.4. Ausschnitt einer Abfrage

```
package net.eisprinz.epcis.client.extension.scenario.query;

public class PalletArrivedDCQuery extends Query
{
  [...]

  protected QueryParams getQueryParams()
  {
    QueryParams queryParams = new QueryParams();

    ArrayOfStrings eventType = new ArrayOfStrings();
    eventType.add("ObjectEvent");
    queryParams.getParams().add(new QueryParam(
      "eventType", eventType
    ));

    [...]

    return queryParams;
  }

  [...]

  public String getResult()
  {
    if(this.result == null)
    {
      return "<ul><li>No results available.</li></ul>";
    }
```

```
      else if(
        this.result.getResultsBody().getEventList().getEvents().isEmpty()
      )
      {
        return "<ul><li>Pallet has not arrived yet.</li></ul>";
      }
      else
      {
        ObjectEvent event =
          (ObjectEvent)this.result.getResultsBody()
          .getEventList().getEvents().get(0);

        return "<ul><li>Pallet arrived at "
          + event.getEventTime()
          + " the distribution center.</li></ul>";
      }
    }
  }
```

An diesem Beispiel ist gut zu sehen, dass die Abfrage ihre Parameter selbst definiert und auch in der Lage ist, die Ergebnisse zu interpretieren. Dies sind diejenigen Punkte, welche für verschiedene Abfragen auch unterschiedlich behandelt werden müssen. Das Ergebnis, welches innerhalb dieses Szenario immer durch eine Liste von Ereignissen dargestellt wird, kann dann in der Basisklasse `Query` gespeichert werden.

Über die beschriebenen Mechanismen können nun Operationen ausgeführt werden und es sind Abfragen möglich. Allerdings gilt es noch zwei weitere, etwas kleinere Probleme zu lösen:

- Um die Ergebnisse nicht zwingend durch Erfassungen anderer Benutzer zu verfälschen, wird bei jedem Aufruf dieser Erweiterung geprüft, ob sich der Anwender mit dem Benutzer *scenario* angemeldet hat. Auf diesem Weg wird verhindert, dass Operationen und Abfragen unter einem anderen Benutzer ausgeführt werden: Die Operationskette selbst kann also nicht mehr unterbrochen werden.

- Zu Beginn des Fallbeispieles wird außerdem überprüft, ob sich noch alte Szenario-Daten auf dem EPCIS-Server befinden. Dies wird auch über eine Abfrage gelöst, die vom Server das zuletzt gespeicherte Ereignis anfordert. Existiert ein solches, wird der Benutzer zu einer Neuinitialisierung des Ser-

vers angehalten. Über diesen Mechanismus werden doppelte Ereignisse in den Ergebnissen vermieden.

Mit dieser Erweiterung ist die Implementierung der Client-Anwendung und somit auch des zu erstellenden EPCIS-Simulationssystems abgeschlossen. Es ist nun möglich, auf anschauliche Weise Ereignisse an den Server zu versenden und über spezielle Abfragen wieder abzuholen. Natürlich sind durch die Erweiterungsschnittstelle des EPCIS-Clients weitere Funktionalitäten denkbar, wobei während der Erstellungszeit dieser Arbeit vorerst keine weiteren Implementierungen gemacht werden sollen.

6. Diskussion der Ergebnisse

Bewertender Rückblick auf den Verlauf der Arbeit

Innerhalb dieses Kapitels soll nun zum Abschluss noch ein Blick auf die einzelnen Phasen der Arbeit und die erreichten Ergebnisse geworfen werden. Hinzu kommt eine Bewertung des EPC-Netzwerk-Konzeptes von EPCglobal, natürlich mit einem besonderen Augenmerk auf die Informationsdienste.

Betrachtet man die einzelnen Aufgabenteile, welche innerhalb des Einleitungskapitels definiert wurden, so ist festzustellen, dass die gewünschten Ziele im Großen und Ganzen erreicht werden konnten: Es ist gelungen, im Laufe dieser Arbeit ein funktionierendes Simulationssystem für EPCIS-Abfragen zu entwickeln und zu implementieren.

Zwar musste an einigen Stellen das ursprüngliche Konzept mehr oder weniger stark angepasst werden, um zu einer funktionierenden Lösung zu kommen, was aber nicht immer ein Nachteil sein musste: In vielen Fällen haben diese Nachbesserungen das Projekt sogar noch bereichern können.

Innerhalb dieses Abschnitts sollen nun gerade diese Punkte noch einmal im Zusammenhang mit der im ersten Teil vorgestellten Aufgabenstellung reflektiert und bewertet werden. Dabei wird versucht zu zeigen, wann und auch warum es zu Problemen gekommen ist und wie diese gelöst werden konnten. Ergänzt werden soll dies mit einer subjektiven Auseinandersetzung mit den Eindrücken und Erfahrungen bezüglich der EPC-Informationsdienste und anderer Komponenten des Netzwerkes.

6.1. Sichtweisen auf das Simulationssystem

Wie bereits erwähnt, ist es während der Implementierungsphase tatsächlich gelungen, ein lauffähiges Testsystem für EPCIS-Abfragen ins Leben zu rufen. Durch dieses wird dem Benutzer ein Werkzeug zur Verfügung gestellt, über dessen webbasierte Dialoge er unterschiedliche Abfragen erstellen, konfigurieren und an einen EPCIS-Server schicken kann.

Der umfangreiche Weg zu dieser Infrastruktur konnte dabei leider nicht immer ganz so bestritten werden, wie er ursprünglich vorgesehen war: Immer wieder ergaben sich – zum Teil auch überraschend – verschiedene, neue Probleme, mit denen zu Beginn der Arbeit noch nicht gerechnet werden konnte. Weitere Schwierigkeiten wurden auch durch die Tatsache ausgelöst, dass einige der benötigten Technologien noch nicht vollständig spezifiziert waren. So mussten an vielen Stellen Lösungen gefunden werden, um diese Teile zu umgehen, bzw. oft auch zu simulieren. Trotzdem ist dies in der Regel gelungen, wodurch man die meisten Probleme grundsätzlich als gelöst betrachten kann.

6.1.1. Reflektierung der gesteckten Ziele

Um feststellen zu können, was während dieses Projektes eigentlich wirklich erreicht wurde, soll zunächst ein Blick auf die Aufgabenstellung aus dem ersten Kapitel dieser Arbeit geworfen werden. Diese definiert unterschiedliche Ziele, die an dieser Stelle für eine erste Bilanz herangezogen werden können.

Danach kann man als Hauptziel dieses Projektes eine Client-Applikation sehen, die dem Anwender Werkzeuge offeriert, mit deren Hilfe er EPCIS-Abfragen formulieren und an ein festgelegtes Server-System versenden kann. Dies konnte in Form des Abfrage-Moduls des EPCIS-Clients mit Sicherheit realisiert werden. Darüber hinaus gliedert sich an dieser Stelle auf jeden Fall auch die Szenario-Erweiterung in das Ergebnis ein, welche zusätzlich diese Thematik noch einmal aufgreift.

Ein weiteres, daraus resultierendes Ziel stellt die Simulation des EPC-Netzwerkes dar: Für Tests der zu implementierenden Client-Applikation muss mindestens ein EPCIS-Server vorhanden sein, der in der Lage ist, die Abfragen des Clients mit repräsentativen Daten zu beantworten. Dieser muss zusätzlich auch Fähigkeiten besitzen, andere Dienste des Netzwerkes simulieren zu können, da für diese Arbeit leider noch keine Testumgebung zur Verfügung stand. Auch diese Aufgabe konnte durch die Entwicklung des Server-Konzeptes und die dazugehörige Implementierung grundlegend gelöst werden.

Das letzte Ziel dieser Arbeit stellt die Authentifizierung eines Benutzers dar. Diese Aufgabe war nur mit Schwierigkeiten zu bewältigen, da leider noch keine Spezifikation für einen grundsätzlichen Aufbau einer Sicherheitslösung innerhalb des Netzwerkes verabschiedet wurde. Aus diesem Grund musste ein eigenes Konzept für eine Lösung entwickelt und in das System integriert werden, deren Aufbau einer zukünftigen Lösung aber wohl nicht allzu fern sein dürfte. Innerhalb eines realen Informationssystems ist diese Umsetzung natürlich trotzdem mehr als eine Art Vorschlag anzusehen.

In den folgenden Abschnitten sollen die einzelnen, entwickelten Komponenten noch einmal genauer betrachtet und bewertet werden, um so einen umfassenden Eindruck über den Verlauf dieses Projektes zu bekommen.

6.1.2. Bewertung der Client-Applikation

Begonnen werden soll an dieser Stelle mit einer Behandlung der Client-Applikation, die – wie gerade schon erwähnt – auf jeden Fall als eine der Schlüsselkomponenten des Projektes anzusehen ist: Durch dieses Programm erhält der Anwender eine visuelle Darstellung der Funktionsweisen des EPCIS-Konzepts.

Neben der Minimal-Anforderung eines Programmteiles, der das Generieren und Auswerten von Abfragen anbieten kann, sind innerhalb der Client-Applikation noch weitere Komponenten realisiert worden: So ist auf jeden Fall die Erfassungsschnittstelle zu nennen, die durch die Aufgabenstellung zwar nicht zwin-

gend gefordert war, allerdings das gesamte Client-Konzept weiter abrundet. So wird der Benutzer nicht nur in die Lage versetzt, Abfragen zu bestimmten Ereignissen zu stellen, sondern darüber hinaus mittels dieser Schnittstelle auch solche registrieren zu können. Diese Tatsache eröffnet natürlich neue Möglichkeiten für weitere Testszenarien, da diese einzelnen Schritte auch aufeinander abgestimmt werden können.

Vergleicht man nun die einzelnen Bausteine des EPCIS-Clients mit dem Aufbau der dazugehörigen Spezifikation, so fällt auf, dass weitestgehend die geforderte Architektur eingehalten werden konnte:

- Durch die Trennung in eine Bibliothek von Grunddatentypen, einzelnen Programm-Komponenten und in die Dienst-Funktionalitäten konnte das spezifizierte *Schichtenmodell* weitestgehend eingehalten werden. Auch die chronologische Vorgehensweise während der Implementierung richtete sich in der Regel nach diesem Ansatz.

- Auch die geforderte *Modularität* wurde nahezu vollständig eingehalten: Der gesamte Aufbau der Client-Applikation – sowohl der sichtbare als auch der unsichtbare – richtet sich durch die Unterteilung in Erfassungs- und Abfrage-Modul und die davon unabhängigen Typ-Komponenten nach der Struktur aus der EPCIS-Spezifikation. Zusätzlich wurde auch die Basis für die Erweiterungen, auch wenn diese nicht explizit durch die Spezifikation gefordert wird, in den modularen Aufbau des Programmes integriert und greift somit den theoretischen Kontext wieder auf.

- Auch die grundsätzliche Forderung nach *Erweiterbarkeit* konnte weitestgehend aufgegriffen werden: Die vordefinierten Erweiterungspunkte innerhalb der Grunddatentypen wurden implementiert (auch während der Entwicklung der Erfassungsschnittstelle konnten diese Punkte, mit der Schnittstelle für Abfragen als Vorbild, berücksichtigt werden), auch wenn sie innerhalb der Client-Applikation bislang nicht genutzt werden. Darüber hinaus verfügt die Client-Applikation ja zusätzlich über eine eigene Erweiterungsschnittstelle,

die durch die Implementierung des Fallbeispieles auch eingesetzt wird. Diese entspricht zwar nicht exakt der Erweiterungsidee der EPCIS-Spezifikation, greift aber durchaus den Grundgedanken einer durchgängigen Erweiterbarkeit wieder auf.

Fasst man nun diese ganzen Punkte zusammen, so kann man – eine nicht ganz so strenge Sichtweise vorausgesetzt – durchaus von einer kleinen *Referenzimplementierung* der EPCIS-Spezifikation seitens des Clients sprechen: Insgesamt wurden sowohl die vorgesehene Architektur, als auch alle wesentlichen Funktionalitäten realisiert.

6.1.3. Bewertung des Simulationsservers

Im Gegensatz zur gerade besprochenen Client-Applikation kann man den EPCIS-Server natürlich nicht mit einer Referenzimplementierung gleichsetzen. Dies ist allerdings allein schon durch die Tatsache bedingt, dass nahezu alle Gegebenheiten, die simuliert werden mussten, auf Seiten des Servers verlagert wurden, um gerade dadurch den EPCIS-Client so realitätsnah wie nur möglich gestalten zu können. Der für dieses Projekt implementierte Server ist somit wirklich mehr als eine Fassade anzusehen, die für Testzwecke mit dem Client genutzt werden kann.

Dieser Ansatz brachte während der Entwicklung natürlich einige Vorteile mit sich:

- Durch die Server-Schnittstelle findet eine vollständige Kapselung des EPC-Netzwerkes statt. In der Theorie sollten Abfragen des Clients auf ähnliche Weise auch an ein reales System gestellt werden können.

- Durch die Trennung der Serverfunktionalitäten und seine Unabhängigkeit konnten Teile des Servers den Gegebenheiten eines Simulationssystems angepasst werden: So verfügt der implementierte EPCIS-Server zum Beispiel über eine Möglichkeit, den Datenbestand zurückzusetzen, was für Test-

zwecke des Clients natürlich sinnvoll ist (denken wir nur an das Fallbeispiel), in der Realität wäre eine solche Funktion allerdings nicht vorstellbar.

Man kann an dieser Stelle also vorerst zusammenfassen, dass der Server im Gegensatz zu der Client-Applikation nie als möglichst realitätsnahe Lösung konzipiert war: Lediglich die Schnittstelle für Abfragen musste zwingend den spezifizierten Gegebenheiten entsprechen, so dass konforme Abfragen möglich werden. Selbstverständlich sollten auch die entsprechenden Antwortdaten eine wohlgeformte Struktur besitzen.

Mit diesem Hintergrund konnte für die meisten Komponenten innerhalb des Servers der einfachste und zweckmäßigste Weg gewählt werden. Da ohnehin die meisten der Netzwerkvorgaben simuliert werden mussten, konnten auch viele der andere Teile entsprechend vereinfacht angepasst werden.

So wurde zum Beispiel für die Entwicklung der Erfassungsschnittstelle nicht der in der EPCIS-Spezifikation empfohlene Weg eines reinen HTTP-Zugriffs gewählt, sondern eine Schnittstelle auf der Basis eines Web Services erstellt. Diese würde wahrscheinlich in der Realität keine Verwendung finden, da sie für die Verarbeitung der Daten nicht schnell genug ist, für die Komponenten dieses Projektes brachte sie aber den enormen Vorteil, dass an vielen Stellen das System der Abfrage-Schnittstelle wiederverwendet werden konnte, dass ja ohnehin bereits vorhanden war.

Ein weiteres, repräsentatives Beispiel für diese Tatsache ist die Wahl der verwendeten Datenhaltung: Der Einsatz einer Datenbank ohne verwaltenden Server – im gewählten Fall sogar ausschließlich im Arbeitspeicher und dadurch mit nur sehr eingeschränkter Persistenz – ist für ein reales Server-System undenkbar. Ein funktionierender EPCIS-Server muss normalerweise in der Lage sein, extrem zuverlässig und schnell unter Umständen auch sehr große Mengen an Daten auf einmal behandeln zu können, was natürlich eine darauf abgestimmte Datenhaltung voraussetzt. Bei den Testfällen im Rahmen dieser Diplomarbeit ist von solchen Szenarien allerdings nicht auszugehen, was den Einsatz einer

deutlich einfacheren Struktur durchaus rechtfertigt. Um trotzdem die Entscheidung für eine passende Datenbank nicht vollständig vorwegzunehmen, wurde die zugehörige Schnittstelle abstrakt gehalten und die eigentliche Datenbank erst während des Serverstartes dynamisch geladen. Auf diesem Weg wird trotzdem ein einfaches Austauschen des verwendeteten Systems ermöglicht.

Eine vordefinierte Erweiterungsschnittstelle, ähnlich der des Client-Programms, steht dem Server nicht zur Verfügung. Trotzdem lässt seine allgemeine, recht einfach gehaltene Struktur durchaus einen weiteren Ausbau zu. Zur Steuerung möglicher Ergänzungen können auch wieder Parameter verwendet werden, analog zum Einsatz bei den Standard-Funktionalitäten. Der Mechanismus, welcher diese Daten aus einer XML-Datei extrahiert, berücksichtigt bereits jetzt auch später definierte Parameter. Somit kann auch für den Server durchaus von einer grundsätzlichen Erweiterbarkeit gesprochen werden.

Insgesamt ist sowohl zur Entwicklung, als auch zur Implementierung des EPCIS-Servers zu sagen, dass trotz dieser Ansätze das eigentliche Gewicht immer bei der Client-Anwendung, insbesondere bei der dortigen Generierung der Abfragen lag. Aus diesem Grund wurde hier eher auf manche Funktion verzichtet, wenn sich diese zu weit von der eigentlichen Aufgabenstellung entfernt hatte.

Dazu gehört zum Beispiel eine benutzerfreundliche Erfassung von Stammdaten und die Interpretation einiger Parameter innerhalb von Abfragen: Letztere beziehen sich auf Hierarchien, die zwischen gespeicherten Ereignissen bestehen und durch Stammdaten definiert werden. Da die Berücksichtitigung dieser Parameter nur serverseitig liegende Definitionen betrifft (Stammdaten lassen sich von Client-Seite aus nicht definieren), wurde hier auf eine Interpretation verzichtet[1].

[1]Der Server antwortet in einem solchen Fall mit einer Ausnahme, die besagt, dass er den Parameter nicht interpretieren kann.

Trotz solcher Einschränkungen kann der Server durchaus als gut einsetzbare Komponente innerhalb des Simulationssystems angesehen werden, da er die Aufgabe, bei Anfragen repräsentative EPCIS-Daten zurückzuliefern, auf jeden Fall erfüllen kann.

6.1.4. Einordnung des Sicherheitskonzeptes

Weit mehr problematisch zeigte sich die Integration der Sicherheitsmechanismen. Immer wieder wird innerhalb der Spezifikationen von eigenen Sicherheitsdiensten gesprochen, durch die eine Authentifizierung und eine Authorisierung möglich gemacht werden soll. Eine eigene Spezifikation, welche dieses Gegebenheiten verdeutlicht und beschreibt, ist bislang leider noch nicht verabschiedet worden.

Der Einsatz von Diensten innerhalb des EPC-Netzwerkes für eine Benutzer-Authentifizierung hat natürlich zur Folge, dass auch eine Client-Applikation direkte Zugriffe auf die Netzwerk-Infrastruktur machen muss. Dies ist mit dem Simulationskonzept, nach dem ein Zugriff auf das EPC-Netzwerk allein über den Server zur Anwendung kommt, so nicht vereinbar: Es wäre neben den beiden Schnittstellen des EPCIS-Server, die normalerweise die alleinige Verbindung für einen Client zum Netzwerk darstellen sollten, noch eine weitere nötig, die sich ohne Kenntnis eines Aufbaues so allerdings nur schwer implementieren lässt.

Da trotzdem für das Projekt dieser Arbeit eine Authentifizierung unverzichtbar ist (der Simulationsserver ist darauf angewiesen, dass er die Identität seines Kommunikationspartners kennt, um die Ereignisse entsprechend zuordnen zu können), musste eine eigene Lösung gefunden werden, die eine Authentifizierung des Benutzers ermöglicht, obwohl gerade dieses, wie soeben beschrieben, mit Sicherheit einer der Bereiche ist, der eigentlich nicht ohne eine Standardisierung auskommen kann.

Für einen solchen Einsatz wurde somit ein Konzept entwickelt, das auf Zertifikaten basierend mit Sicherheit einer späteren Lösung sehr nahe kommt. Trotzdem ist dabei natürlich die Wahrscheinlichkeit äußerst hoch, dass eine vollständige Kompatibilität zu einem späteren Zeitpunkt nicht gegeben sein wird.

Die Folgen einer eigenen Lösung liegen somit nicht innerhalb des Simulationssystems dieser Arbeit, das natürlich weiter in der Lage ist, entsprechend auf die entwickelte Lösung zu reagieren, sondern vielmehr beim Einsatz einzelner Komponenten innerhalb einer realen Umgebung, was selbstverständlich wünschenswert wäre:

- Versucht man, Abfragen mit dem entwickelten EPCIS-Client an ein reales Server-System zu stellen, so ist leider nicht sichergestellt, das dieses den angemeldeten Benutzer auch authentifizieren kann.

- Auf der anderen Seite würde auch der für dieses Projekt entwickelte EPCIS-Server möglicherweise einen anderen, anfragenden Client nicht authentifizieren können.

Aus diesem Grund wurde versucht, die Lösung so einfach wie möglich zu halten, um später einmal die Integration einer standardisierten Fassung schnell realisieren zu können.

Die Lösung in ihrer jetzigen Form ist insgesamt mehr als ein Ansatz zu sehen, welcher demonstriert, wie allgemein Dienste über Zertifikate geschützt werden können und sowohl den Client als auch den Server in die Lage versetzen, sich vorerst gegenseitig authentifizieren zu können. Eine genauere Lösung ist zu diesem Zeitpunkt leider noch nicht möglich, was wahrscheinlich zur Folge haben wird, dass systemübergreifend zunächst nur die wenigen anonymen Zugriffe möglich sind.

6.1.5. Probleme während der Implementierung

Auch wenn die eigentliche Implementierungsphase dieser Arbeit im Großen und Ganzen weitestgehend reibungslos verlaufen ist, kam es doch an einigen Stellen zu Problemen, die sich nicht auf Anhieb haben lösen lassen.

Das größte Problem tat sich gleich zu Beginn des Projektes auf: Die Generierung der Grunddatentypsklassen aus den gegebenen Schema-Definitionen der Spezifikation. Es konnte einfach keine geeignete Web-Service-Engine gefunden werden, mit deren Hilfe diese Definitionen ohne Probleme umgesetzt werden konnten.

Alle größeren Web-Service-Engines verfügen über eigene Werkzeuge, mit deren Hilfe sich in Kombination mit einem bestehenden Schema entsprechende Klassen generieren lassen. Leider wurden aufgrund des enormen Umfangs und der Komplexität der gegebenen Definitionen alle Tools, die für einen Einsatz getestet wurden, an ihre Grenzen gebracht.

Genau genommen war allein die Web-Service-Engine von Sun in der Lage, durch ihre Programme eine Generierung zu Ende bringen, ohne vorher mit das Schema betreffenden Fehlern abzubrechen, wodurch bedingt sie später dann auch wirklich zum Einsatz kommen sollte. Eine Erstellung der erforderlichen Klassen mit den Werkzeugen der weit verbreiteten Axis-Engine war zum Beispiel nicht möglich. Trotzdem erzeugten auch die Tools von Sun teilweise fehlerhaften Code, so dass an vielen Stellen manuelle Nachbesserungen gemacht werden mussten, was ohne genauere Kenntnis der Binding-Strukturen alles andere als einfach war.

Dazu kam, dass sich die über ein Servlet realisierte Engine nur sehr schwer in das laufende System integrieren ließ: Die Konfiguration eines solchen Systems funktioniert mit einer speziellen Datei, in der verschiedene Werte angegeben werden müssen. Leider waren Fehlermeldungen, die gerade bei missverständlichen Namensräumen innerhalb dieser Datei auftauchten, kaum zu interpretie-

ren. Hinzu kam, dass dieses System eine große Anzahl an Bibliotheken voraussetzt, wobei selten dabei klar ist, wann welche eigentlich wirklich benötigt werden. Insgesamt hat der Aufbau des Web-Service-Systems mit der Integration der generierten Klassen einen enormen Zeitraum in Anspruch genommen, was ja eigentlich durch ein Konzept, spezielle Werkzeuge für eine Generierung zur Verfügung zu stellen, gerade verhindert werden soll.

Weitere Probleme entstanden während der Integration der Sicherheitskomponenten: Da dabei die erforderlichen Elemente direkt mit der Web-Service-Engine zu verknüpfen sind, müssen diese auch dementsprechend eingebunden werden: Somit werden einzelne Bibliotheken zur Anwendung hinzugefügt und anschließend mittels einer passenden Konfigurationsdatei im Web-Verzeichnis die gewünschten Sicherheitsmechanismen aktiviert. Das Problem bei der Integration selbst war, dass durch die vermeindliche Einfachkeit kaum Informationen zur Einrichtung des Systems (was natürlich nicht auf Anhieb funktionierte) vorhanden waren. Außerdem existieren durch die aktuellen Umstellungen der Architektur noch sehr wenig Dokumentationen, die sich zum Teil auch mit denen früherer Versionen mischen.

Hat man die Sicherheits-Engine zum Laufen gebracht, so versucht diese, so viele Funktionalitäten wie möglich innerhalb des dazugehörigen Systems zu kapseln. Dies macht einen Einsatz für einen Programmierer natürlich sehr übersichtlich und einfach, da er sich dabei kaum selbst um etwas kümmern muss, allerdings gibt es auch wenig Möglichkeiten, ein System an verschiedene Anforderungen anzupassen. So war es – wie bereits während dieser Arbeit beschrieben – nicht möglich, im Fehlerfall spezielle Ausnahmen zu senden, was dann schließlich eine weitere Anpassung des Servers notwendig machte.

Interessanterweise kann man an dieser Stelle zusammenfassen, dass Probleme während der Implementierung nicht etwa durch komplexe Ideen hinter der EPCIS-Spezifikation oder durch deren Umsetzung entstanden sind, sondern bereits bei der Einrichtung der benötigten Umgebungen, während der eigentlich

keine größeren Schwierigkeiten zu erwarten waren. Die Probleme, die während des Aufbaus dieses Systems aufgetreten sind, zeigen somit deutlich, dass viele der Web-Service-Engines bei weitem noch nicht ausgreift sind.

6.2. Subjektive Beurteilung des EPCIS-Konzeptes

Nachdem die Ergebnisse der Implementierungsphase ausführlich behandelt worden sind, soll nun auch an dieser Stelle noch ein allgemeinerer Blick auf den dabei eingesetzten EPCIS-Hintergrund geworfen werden. Dabei sollen Erfahrungen und Einschätzungen behandelt werden, die sich während der Einarbeitungszeit und der Entwicklung dieser Arbeit ergeben haben. Daher ist es sicherlich auch verständlich, dass die gleich beschriebenen Aussagen nicht vollständig objektiv sein können.

Die Eindrücke, die innerhalb dieses Abschnittes beschrieben werden sollen, beziehen sich insgesamt gesehen mehr auf die technischen Aspekte innerhalb des EPC-Netzwerkes. Auch wenn es zur Zeit viel Kritik – gerade beim Einsatz der RFID-Technologie – gibt (zum Teil mit Sicherheit auch berechtigt, gerade was Themen wie Datenschutz und Überwachnung angeht), sollen diese hier nicht weiter besprochen werden, da sich diese Bereiche zu weit von der eigentlichen Aufgabenstellung dieser Arbeit entfernen würden. Das Thema der Arbeit beschäftigt sich hauptsächlich mit dem Einsatz der Informationsdienste innerhalb des EPC-Netzwerkes, so dass diese auch Grundlage einer Diskussion sein sollten.

6.2.1. Positive Eindrücke

Das Ziel bei der Einführung eines gemeinsamen Informationsnetzwerkes ist klar gesetzt und wurde im theoretischen Teil dieser Arbeit bereits ausführlich behandelt: Die Schaffung einer gemeinsamen Netzwerk-Infrastruktur, die allen daran beteiligten Mitgliedern ermöglichen soll, über die gesamten Produktionsketten hinweg Information zu einzelnen Objekten erfragen zu können.

Sehr positiv fällt bei diesem Konzept auf, dass grundsätzlich kein Unternehmen ausgeschlossen wird: Durch den offenen Standard und den dazugehörigen, neutralen Standardisierungsprozess werden prinzipiell alle Unternehmen angesprochen, wobei dazu natürlich zu bemerken ist, dass die aufzubauende Struktur auch nur funktionieren kann, wenn möglichst viele daran teilnehmen.

Gerade beim Gedanken an eine große Anzahl von Netzwerkmitgliedern fällt ein weiterer, sicherlich auch an der zukünftigen Situationen orientierter Punkt ins Auge: Die dezentrale Struktur des EPC-Netzwerkes. Durch die Tatsache bedingt, dass jedes Netzwerkmitglied grundsätzlich nur seine eigenen Informationsdaten innerhalb eines lokalen Systems pflegt und das EPC-Netzwerk selbst sich ausschließlich darum kümmert, dass die Daten von den anderen Kommunikationspartnern gefunden und abgerufen werden können, kann ein großer Teil der Netzwerklast auf die einzelnen Mitglieder verlagert werden. Die einzelnen Informationssysteme arbeiten dabei weitestgehend unabhängig voneinander und können den technischen Anforderungen und natürlich auch der zu erwartenden Datenmenge der einzelnen Unternehmen individuell angepasst werden. Zusätzlich ist auch eine Auslagerung an unabhängige Dienstleister denkbar. Somit dürfte das EPC-Netzwerk bereits jetzt allein durch die zugrundeliegende Struktur auch auf größere Transfers vorbereitet sein.

Die Informationsdienste des EPC-Netzwerkes selbst machen aus technischer Sicht gesehen einen durchaus soliden Eindruck: Ein gutes Beispiel dafür ist die vollständig beschriebene Schnittstelle für Abfragen und die dafür verwendeten Typdefinitionen, die zwar durch die große Anzahl von verschiedenen Objekten relativ umfangreich ausfallen, trotzdem aber durchgehend einem festen Schema folgen. Auch die verwendeten Operationen der Schnittstelle sind zweckmäßig gewählt und behalten somit eine übersichtliche Struktur. Die im vorherigen Abschnitt beschriebenen Probleme während der Generierung der dazugehörigen Klassen sind somit nicht unbedingt auf das EPCIS-Schema selbst zurückzuführen, sondern eher auf die Verarbeitung durch die eingesetzten Werkzeuge.

Besonders positiv fällt innerhalb des Schemas das gleichbleibende Konzept für den Aufbau von Datentypen auf. Dabei ist besonders der Erweiterungsmechanismus hervorzuheben, der in allen wesentlichen Datentypen immer auf gleiche Weise vertreten ist. Durch die verwendeten Erweiterungspunkte wird einem Unternehmen ein mächtiges Werkzeug in die Hände gelegt, das es ihm ermöglicht, nicht nur die Standard-Funktionalitäten innerhalb der Objekte zu nutzen, sondern diese auch um eigene Felder zu ergänzen. Trotzdem bleibt dabei jeder andere Handelspartner in der Lage, wenigstens die Standard-Felder lesen zu können.

Ein weiterer Datentyp, der einen durchdachten Erweiterungsmechanismus mit sich bringt, ist zugleich auch das Kernelement der EPCIS-Spezifikation: Das abstrakte EPCIS-Ereignis. Auch wenn viele der Anforderungen an das Informationsnetzwerk bereits durch die vier existierenden Objekte behandelt werden, die ja von dieser Klasse abgeleitet sind, so ist es trotzdem möglich, entweder, wie eben beschrieben, die Ereignisse entsprechend zu erweitern, oder aber auch ganz neue Ereignisse zu entwickeln, die in der Theorie durch eine Ableitung vom allgemeinen EPCIS-Ereignis sofort durch das Netzwerk unterstützt werden sollten.

Insgesamt zeigen diese Beispiele, dass während der Entwicklung des EPCIS-Konzeptes viele gute Ideen integriert werden konnten, auf die mit Sicherheit auch in zukünftigen Versionen aufgebaut werden kann. Ob dieses Konzeptes dabei den erwünschten Erfolg bringt, wird sich allerdings wohl erst nach einem weitläufigeren Einsatz in der Praxis zeigen können.

6.2.2. Bewertung des Abfrage-Konzeptes

Ein momentan größeres Problem innerhalb dieses Konzeptes wird durch ein anderes Element geschaffen, das zugleich auch eines der Schlüsselpunkte dieser Diplomarbeit darstellt: Die Formulierung von Abfragen.

Der konzeptionelle Aufbau einer Abfrage ist äußerst einfach und so allgemein wie möglich gehalten: Sie besteht im Wesentlichen aus einer Menge von vordefinierten Parametern, die wiederum aus einem eindeutigen Namen und einem Wert zusammengesetzt sind. Da dessen Datentyp nicht festgelegt ist, könnten theoretisch eine beliebige Anzahl von Abfragen für die unterschiedlichste Zwecke definiert werden (mit *Definierung* ist hier etwas technischer gesehen die Entwicklung eines Katalogs von möglichen Parametern und deren Bedeutung zu verstehen).

Das große Problem dabei ist, dass durch die äußerst einfache Struktur kaum genauere Konfigurationen gemacht werden können. Betrachtet man sich den allgemeinen Ansatz für den Aufbau eines einzelnen Parameters, so besteht die komplexeste Struktur eines Wertes, der einem Parameter zugeordnet werden kann, wohl aus einer Liste von Zeichenketten. Dabei ist die Listenstruktur selbst als `OR`-Verknüpfung anzusehen, die Parameter untereinander werden `AND`-verknüpft. Mehr kann allein durch den Aufbau einer Abfrage nicht ausgedrückt werden: Alle anderen Gegebenheiten, die für eine Abfrage nötig sind, müssen in die Semantik innerhalb der Parameter verlagert werden.

Es macht Sinn, an dieser Stelle einen Vergleich mit einer anderen, sehr populären Abfrageform zu machen: Die Datenbank-Abfragesprache *SQL*. Stellt man die beiden Ansätze gegenüber, so sind sie sich (gerade beim Einsatz) gar nicht so unähnlich:

- Die EPCIS-Abfragen werden eingesetzt, um auf EPCIS-Objekte, wie Ereignisse und Wortschatz-Elemente, zugreifen zu können, die zuvor aufgrund von Vergleichen der enthaltenden Felder mit festgelegten Werten selektiert wurden.

- SQL kann eingesetzt werden, um auf einzelne Datensätze innerhalb einer Datenbank zugreifen zu können, die zuvor aufgrund von Vergleichen ihrer enthaltenen Zellen mit festgelegten Werten selektiert wurden.

Zusätzlich verfügen beide Formen über Mechanismen, um die erzeugten Ergebnismengen zu sortieren und zu limitieren (wobei im EPCIS-Fall dafür wieder eigene Parameter definiert werden müssen, individuell für jede einzelne Abfrage). Ein Vergeich dieser beisen Systeme ist also gar nicht so abwegig.

Der wesentliche Unterschied zwischen diesen beiden Ansätzen liegt bei der Verteilung der Ausführungssemantik. Damit ist gemeint, dass beide Formen auf unterschiedliche Weise definieren, was eigentlich wirklich getan werden soll.

Die Sprache SQL verfügt dabei über eigene Vergleichsoperatoren. Mit deren Hilfe werden auf immer gleichbleibender, fest definierter Weise die gewünschten Vergleiche realisiert: Ein (Tabellen-)Feld wird mit der darauf anzuwendenden Operation verknüpft. Man kann also vorsichtig sagen, dass die Semantik einer Ausführung in einem solchen Fall bereits innerhalb der Abfrage durch technische Mittel festgelegt ist.

Im Vergleich dazu muss während einer EPCIS-Abfrage, die ja erst einmal nicht über eigene Vergleichsoperatoren verfügt, das ausführende System selbst entscheiden, was für eine Art von Operator im Bezug auf jeden einzelnen Parameter anzuwenden ist. Es wird also vorausgesetzt, dass dieses abfragende System nicht nur den übergebenen Parameter kennt, sondern zusätzlich auch die Operation, mit der er verknüpft wurde. Die Semantik der Ausführung ist also sehr weit auf die Seite des ausführenden Systems verlagert und mit Sicherheit nicht direkt durch die Abfrage selbst gegeben.

Dies bringt einen enormen Nachteil mit sich: In der Theorie muss für jedes einzelne Feld, welches durch Vergleiche abgefragt werden soll, nicht nur ein einzelner Parameter definiert werden, sondern gleich fünf: Für jede einzelne Vergleichsform ein eigener[2]. Darüber hinaus wäre noch ein sechster für die

[2]Vergleichsoperatoren: Echt kleiner (<), kleiner oder gleich (<=), gleich (=), echt größer (>), größer oder gleich (>=).

Ungleich-Operation und ein siebter für `NOT` denkbar, sollten diese zusätzlich benötigt werden.

Die Folgen dieses Problems werden bereits an der eigentlich äußerst einfachen, vordefinierten Abfrage für Ereignisse deutlich, welche durch die EPCIS-Spezifikation definiert wird: Obwohl sich jedes einzelne Ereignis eigentlich nur aus sehr wenigen Feldern zusammensetzt, die sich oft auch wiederholen, definiert die Spezifikation tatsächlich über 30 Parameter für die Abfrage dieser Felder. Dabei wird jeder Parameter, der einen Vergleich realisieren soll, mit einer Vorsilbe (zum Beispiel `EQ` für „gleich“) versehen, um die einzelnen Operationen darzustellen. Da diese Vorsilben allerdings nicht spezifiziert wurden, kann eine Server-Implementierung in der Realität leider nicht einfach logisch von der Vorsilbe auf die Operation schließen, sondern muss wirklich jeden einzelnen Parameter und seine Bedeutung kennen, auch wenn der Aufbau sich immer wiederholt.

Dieses Beispiel zeigt sehr deutlich, dass das Abfrage-Konzept durch seine enorme Einfachheit bereits bei einfachen Aktionen an seine Grenzen gebracht wird. Die Einordnung durch Vorsilben ist insgesamt wohl mehr als Notlösung zu werten, was leider nicht gerade für das Parameter-Konzept spricht. Die Schwächen dieses Aufbaus lassen sich an einem weiteren Beispiel noch um einiges deutlicher demonstrieren: Ab dem Moment, wenn sich ein Vergleichswert aus mehreren einzelnen Werten zusammensetzt (zum Beispiel eine Transaktion, die durch einen Typ und einen dazu verknüpften Wert beschrieben wird), kann dieses durch einen einzelnen Parameter schon nicht mehr dargestellt werden. Um trotzdem derartige Felder abfragen zu können, werden einfach einzelne Werte an den Namen des Parameters angehängt, was insgesamt auch mehr wie eine Kompromisslösung wirkt.

An diesem Punkt stellt sich sicherlich die Frage, ob der Preis für eine möglichst einfache, allgemeine Abfrageform, was sicherlich einer der ausschlaggebenden Gründe für dieses Konzept war, nicht zu hoch ist. Es sollten auf jeden Fall inner-

halb dieses Bereiches weitere Lösungen gefunden werden, um die Abfragen verbessern zu können, da allein das Beispiel der einfachen Ereignisabfrage die Frage aufwirft, auf welchem Weg sich noch deutlich komplexere Abfragen realisieren lassen sollen.

Zusätzlich ist innerhalb der EPCIS-Spezifikation bereits angedeutet, dass in einer zukünftigen Version eine eigene Abfragesprache vorgesehen ist[3], mit der sich individuell definierte Abfragen sogar über die Schnittstellen beim EPCIS-Server registrieren lassens sollen. Wie dabei der Server an die eigentliche Bedeutung der Parameter kommen soll, die er für die Auswertung ja schließlich braucht, wird leider nicht gesagt. Auch diese Idee zeigt deutlich, dass es mit den bisherigen Mitteln schwierig werden dürfte, ein flexibles Abfrage-System aufzubauen.

6.2.3. Weitere Überlegungen

Neben dem gerade behandelten, doch recht weit reichendem Problem ergeben sich natürlich darüber hinaus weitere Fragen, die möglicherweise erst durch spätere Erfahrungen zu beantworten sind. Trotzdem sollen an dieser Stelle noch ein paar spekulative Überlegungen bezüglich der Idee eines gemeinsamen Netzwerkes gemacht werden.

Eine Frage, die sich an dieser Stelle sehr schnell ergibt, beinhaltet, ob und auf welche Weise sich die großen Mengen an EPCIS-Daten überhaupt verwalten lassen. Da prinzipiell vorgesehen ist, dass der EPC über die nächsten Jahre den bekannten Barcode ersetzen soll, ist auch damit zu rechnen, sollte es tatsächlich zu einer weit verbreiteten Realisierung des Informationsnetzwerkes kommen, dass auch die Speicherung von Ereignissen über die Versorgungsketten nicht nur auf Palettenebene betrieben werden wird. Vielmehr ist in einem solchen Fall eine extrem große Menge von Datensätzen zu jedem einzelnen Produkt zu erwarten. Durch ihre Abhängigkeit untereinander bedingt (wie im Theorie-

[3]EPCIS Specification Version 1.0, Last Call Working Draft, Seite 57.

teil dieser Arbeit bereits erläutert), müssen diese Daten natürlich auch jederzeit zuverlässig zur Verfügung stehen. Es wurde zwar bereits gesagt, dass durch die dezentrale Struktur des EPC-Netzwerkes bereits ein erster Schritt zur Lösung des Problems gemacht wurde, ob sich trotzdem die Organisation der vielen Daten realisieren lässt, bleibt abzuwarten.

Weiter ist durch die Szenario-Erweiterung des Projektes sicherlich deutlich geworden, dass die durch eine Abfrage zurückgelieferten Ereignisse auf unterschiedlichste Weise interpretiert werden können: So können – eine entsprechende Kenntnis des Geschäftsprozesses vorausgesetzt – allein durch eine Menge von Ereignissen Aussagen über Lagerbestand, Lieferungsstatus oder Zugehörigkeit zu einer Bestellung gemacht werden.

Diese Interpretation liegt allerdings fast auschließlich auf der Seite der abfragenden Client-Anwendung, die natürlich, um entsprechende Aussagen machen zu können, auch die komplette Logik der Prozesskette kennen muss. Um die daraus entstehenden Schwierigkeiten besser demonstrieren zu können, wurde während der Entwicklung des Fallbeispiels bewusst darauf geachtet, dass die am Szenario beteiligten Handelspartner ähnliche Situationen mit unterschiedlichen Strategien darstellen (so werden zum Beispiel Produkte auf unterschiedliche Art zu einer Bestellung hinzugefügt). Der Client muss also in der Lage sein, diese unterschiedlichen Wege zu berücksichtigen.

In der Realität kann dies allerdings ein großes Problem darstellen, weil sich mit Sicherheit die Prozessketten nicht derartig standardisieren lassen, dass sie allgemein in ein Client-System integriert werden können, auch wenn bereits spezielle Arbeitsgruppen gebildet wurden, die sich das Ziel gesetzt haben, für unterschiedliche Situationen Stammdaten zu entwickeln, welche für eine Interpretation herangezogen werden können.

Eine unzureichende Standardisierung innerhalb der Prozesse könnte daher eine individuelle, unverzichtbare Anpassung einzelner Systeme an gegebene Szenarien zur Folge haben (ähnlich der Fallbeispiel-Erweiterung, die auch nur funk-

tioniert, weil der Geschäftsprozess in die Implementierung integriert ist), was unter Umständen für viele – gerade auch kleinere – Unternehmen finanziell nicht zu bewältigen wäre. Das Konzept einer Netzwerk-Struktur, die durch offene Standards eine weite Verbreitung mit möglichst vielen teilnehmenden Unternehmen finden soll, könnte so bereits gebremst werden.

Gerade der erwähnte finanzielle Aspekt kann zu einem weiteren Problem führen, da im Gegensatz zum offenen Standard in erster Linie kommerzielle Anwendungen zu erwarten sind, die einem interessierten Unternehmen die Nutzung des EPC-Netzwerkes ermöglichen sollen. Solche Software könnte genauso wie die zusätzlich nötige Hardware (zum Beispiel für die Erfassung von Ereignissen) für viele Unternehmen aus finanzieller Sicht nicht zu realisieren sein. Eine quelloffene Referenzimplementierung durch EPCglobal, wie sie einmal vorgesehen war, wurde eingestellt, so dass es abzuwarten gilt, ob sich andere freie Implementierungen, die sich bereits in der Entwicklung befinden, gegen die kommerzielle Konkurrenz behaupten können und auf diesem Weg auch kleineren Unternehmen den Gebrauch des EPC-Netzwerks ermöglichen.

Alle diese Beispiele zeigen, dass in vielen Bereichen des EPC-Konzeptes die zukünftige Entwicklung noch nicht abzusehen ist. Man darf also gespannt sein, ob und wie sich die Konzepte umsetzen lassen, wie diese sich weiterentwickeln und ob sich diese Entwicklung beeinflussen lässt.

7. Zusammenfassung

Abschluss der Arbeit mit einem Ausblick in die Zukunft

Mit diesem letzten Kapitel werden die Ausführungen dieser Diplomarbeit abgeschlossen. Dabei sollen aber noch ein paar Überlegungen für einen möglichen Ausbau in der Zukunft gemacht werden.

Mit der Diskussion innerhalb des letzten Kapitels soll nun auch die Diplomarbeit zu ihrem Ende gebracht werden. Insgesamt ist es hoffentlich gelungen, innerhalb dieser Arbeit die zum Teil doch recht technischen Gegebenheiten mit Hilfe von Erklärungen der zugrundeliegenden Theorie und mit praktischen Beispielen zu erläutern und zu untermauern.

Natürlich wird ein solches Projekt – gerade bei einem so umfangreichen Thema – nicht mit dem Abschluss der begleitenden Arbeit beendet. Vielmehr ist es zu wünschen, dass die einzelnen Programmteile auch in Zukunft weiter ausgebaut und ergänzt werden und natürlich auch andere Projekte mit einem ähnlichen Thema durch die mit dieser Arbeit gewonnenen Erfahrungen bereichert werden können. Mit diesem Hintergedanken sollen somit an dieser Stelle noch einige Anregungen für die Zukunft gegeben werden, auf welche Weise das Projekt fortgesetzt und ergänzt werden kann.

Ein sehr interessanter Punkt, welcher im Laufe dieser Arbeit aufgrund fehlender Systeme leider nicht realisiert werden konnte, ist der Test der implementierten Client-Applikation an einem realen EPCIS-Server. Grundsätzlich sollten dabei Abfragen über die entsprechende Schnittstelle möglich sein. Probleme während solcher Tests, die zu diesem Zeitpunkt natürlich noch nicht eindeutig zu bestimmen sind, könnten möglicherweise durch folgende Vorausset-

zungen entstehen, deren Anpassung innerhalb des Programms selbstverständlich zu wünschen wäre:

- Aller Wahrscheinlichkeit nach werden sich nur anonyme Zugriffe realisieren lassen, da die Sicherheitslösung innerhalb des EPC-Netzwerkes während der Implementierung noch nicht verfügbar war. Da das Konzept, welches im Moment für die Authentifizierung eines Benutzers eingesetzt wird, im Rahmen dieser Arbeit entstanden ist, wäre es selbstverständlich sinnvoll, dieses durch eine andere Lösung zu ersetzten, sobald diese offiziell verabschiedet ist.

- Die EPCIS-Spezifikation war während der Erstellung dieser Arbeit selbst noch nicht offiziell verabschiedet, sondern es konnte nur eine fortgeschrittene Working-Draft-Version genutzt werden. Auch wenn grundsätzlich davon auszugehen ist, dass wohl bis zur entgültigen Standardisierung keine extrem weitreichenden Änderungen gemacht werden, so kann dies natürlich trotzdem geschehen. In einem solchen Fall sollten die einzelnen Applikationen des Projektes an den entgültigen Stand angepasst werden.

Zu den einzelnen Programm-Komponenten, die während diesem Projekt entstanden sind, ist zu sagen, dass sie immer als eine Art Grundlage konzipiert waren: Alle verschiedenen Teile bieten auf jeden Fall das Potenzial für Ausbau und Erweiterung unterschiedlichster Art. Einige Ideen sollen an dieser Stelle als Anregung aufgeführt werden:

- Die Bibliothek, welche die Klassen der Grunddatentypen sammelt, könnte an vielen Stellen weiter ausgebaut werden. Sie ist sehr umfangreich und gerade einige der komplexeren Zugriffe könnten von der logischen Seite aus betrachtet durchaus in diese Klassen verlagert werden.

- Auch Tests mit anderen Web-Service-Systemen wären vorstellbar. Durch andere Schnittstellen-Implementierungen könnte auf unterschiedliche Weise auf die einzelnen Dienste zugegriffen werden und man hätte so die Möglich-

keit, die unterschiedlichen Realisierungen – gerade auch mit einem zeitlichen Aspekt – zu vergleichen und die Vorteile miteinander zu vereinen.

- Weiter wäre ein Ausbau der Client-Applikation über die Erweiterungsbasis denkbar: Über dieses Modul könnte das Programm neben den bislang mehr grundlegenden Elementen um viele nützliche Funktionalitäten erweitert werden, welche auch die Rolle eines Testprogramms für EPCIS-Systeme weiter unterstreichen könnten.

- Zuletzt ist an dieser Stelle noch eine weitreichendere Integration von Stammdaten in den logischen Prozess der Abfragen zu nennen: Dieser Bereich wurde im Rahmen des Projektes nur am Rand behandelt, da er für die gesteckten Ziele dieser Arbeit nicht so sehr relevant war und somit auch zeitlich zu aufwendig geworden wäre. Trotzdem wären zum Beispiel Funktionen im Programm denkbar, mit denen sich über Stammdaten auch Hierarchien zwischen den einzelnen Ereignissen definieren lassen.

Ansonsten kann das entstandene Simulationssystem durchaus auch als Ganzes beispielhaft dafür eingesetzt werden, um zu demonstrieren, auf welche Weise sich die in der EPCIS-Spezifikation definierten Funktionalitäten umsetzen lassen. Mit den entstandenen Ergebnissen und gewonnenen Erfahrungen dieser Arbeit ist es sicherlich möglich, auch für zukünftige EPCIS-Projekte eine solide Grundlage zu schaffen, Anregungen für die Lösung bestimmter Problemfälle zu bekommen und dass sich dabei vielleicht sogar ganze Programmteile wiederverwenden lassen.

A. Abfrage-Parameter

In diesem Anhang werden die Parameter aufgelistet, über die eine Abfrage konfiguriert werden kann. Dabei werden die vordefinierten Abfragen `SimpleEventQuery` und `SimpleMasterDataQuery` berücksichtigt.[1]

Den Namen der Parameter kann ein Präfix vorangestellt sein, welches die jeweilige Vergleichsform beschreibt, die zur Anwendung kommen soll. `EQ` steht dabei für „gleich“, `LT` für „kleiner“, `LE` für „kleiner oder gleich“, `GT` für „größer“ und `GE` für „größer oder gleich“.

A.1. Parameter der einfachen Ereignisabfrage

Parametername	Datentyp	Beschreibung
`eventType`	Liste von Zeichenketten	Nur die angegebenen Ereignisarten werden gewählt.
`GE_eventTime` `LT_eventTime`	Zeitangabe	Es werden nur Ereignisse gewählt, deren Ereigniszeit größer oder gleich, bzw. kleiner als der angegebene Wert ist.
`GE_recordTime` `LT_recordTime`	Zeitangabe	Es werden nur Ereignisse gewählt, deren Registrierungszeit größer oder gleich, bzw. kleiner als der angegebene Wert ist.
`EQ_action`		

[1] Die Liste hält sich dabei weitestgehend an die Auflistung aus der EPCIS-Spezifikation (EPCglobal: EPC Information Services (EPCIS) Specification Version 1.0, Last Call Working Draft, März 2006, S. 68 – 84).

Parametername	Datentyp	Beschreibung
	Liste von Zeichenketten	Nur Ereignisse mit einem der angegebenen Aktionswerte werden gewählt.
`EQ_bizStep`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die einen der angegebenen Geschäftsschritte besitzen.
`EQ_disposition`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die einen der angegebenen Dispositionsschritte besitzen.
`EQ_readPoint`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die einen der angegebenen Lesepunkte besitzen.
`WD_readPoint`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die einen der angegebenen Lesepunkte bzw. einen seiner Nachfahren besitzen.
`EQ_bizLocation`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die eine der angegebenen Geschäftlokationen besitzen.
`WD_bizLocation`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die eine der angegebenen Geschäftslokationen bzw. einen ihrer Nachfahren besitzen.
`EQ_bizTransaction_`*`type`*	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die einer der angegebenen Transaktionen angehören. Dabei wird `type` im Parameternamen durch den Bezeichner des Transaktionstypes ersetzt.

Parametername	**Datentyp**	**Beschreibung**
`MATCH_epc`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die einen der angegebenen EPCs enthalten.
`MATCH_parentID`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die eines der angegebenen, übergeordneten Objekte besitzen.
`MATCH_childEPC`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die exakt die angegebene, untergeordnete EPC-Liste haben.
`MATCH_epcClass`	Liste von Zeichenketten	Es werden nur Quantitätsereignisse gewählt, die eine der angegebenen EPC-Klassen besitzen.
`MATCH_parentID`	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die eines der angegebenen, übergeordneten Objekte besitzen.

`EQ_quantity`
`GT_quantity`
`GE_quantity`
`LT_quantity`
`LE_quantity`
	Zahlenwert	Es werden nur Quantitätsereignisse gewählt, deren Quantität gleich, größer, größer oder gleich, kleiner oder gleich, bzw. echt kleiner als der angegebene Wert ist.

`EQ_`*`fieldname`*	Eine Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die um ein Feld erweitert sind, das den Namen `fieldname` trägt und dessen Wert einem der angegebenen Werte entspricht.
`GT_`*`fieldname`* `GE_`*`fieldname`*	Zeitangabe Zahlenwert	Es werden nur Ereignisse gewählt, die um ein Feld erweitert sind, das den

Parametername	Datentyp	Beschreibung
`LT_`*`fieldname`* `LE_`*`fieldname`*		Namen `fieldname` trägt und dessen Wert größer, größer oder gleich, kleiner oder gleich, bzw. echt kleiner als der angegebene Wert ist.
`EXISTS_`*`fieldname`*	Kein Typ	Es werden nur Ereignisse gewählt, die ein Feld namens `fieldname` besitzen.
`HASATTR_`*`fieldname`*	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die ein Feld namens `fieldname` besitzen, dessen Wert ein Wortschatz-Element ist, welches ein Attribut besitzt, das durch einen Wert aus der Liste bezeichnet wird.
`HASATTR_`*`fieldname`* *`_attrname`*	Liste von Zeichenketten	Es werden nur Ereignisse gewählt, die ein Feld namens `fieldname` besitzen, dessen Wert ein Wortschatz-Element ist, welches ein Attribut namens `attrname` besitzt, das einen der angegebenen Werte enthält.
`orderBy`	Zeichenkette	Die Ereignisse im Ergebnis werden nach dem angegebenen Wert sortiert. Möglich sind Ereigniszeit, Registrierungszeit und Quantität.
`orderDirection`	Zeichenkette	Das Ergebnis wird in Richtung des angegebenen Wertes sortiert. Möglich sind `ASC` für aufsteigende und `DESC` für absteigende Richtung.

Parametername	Datentyp	Beschreibung
`eventCountLimit`	Zahlenwert	Übersteigt die Ereigniszahl des Ergebnisses den angegebenen Wert, so werden nur die ersten `n` Ereignisse hinzugefügt, wobei `n` dem angegebenen Wert entspricht.
`maxEventCount`	Zahlenwert	Übersteigt die Ereigniszahl des Ergebnisses den angegebenen Wert, so wird ein Fehler ausgelöst.

A.2. Parameter der einfachen Stammdatenabfrage

Parametername	Datentyp	Beschreibung
`vocabularyName`	Liste von Zeichenketten	Es werden nur Elemente gewählt, die zu einem der angegebenen Wortschätzen gehören.
`EQ_name`	Liste von Zeichenketten	Es werden nur Elemente gewählt, die eine der angegebenen Bezeichnungen haben.
`WD_name`	Liste von Zeichenketten	Es werden nur Elemente gewählt, die eine der angegebenen Bezeichnungen haben, bzw. einen deren definierte Nachfahren.
`includeAttributes`	Wahr oder falsch	Dieser Parameter bestimmt, ob dem Ergebnis die zugehörigen Attribute hinzugefügt werden sollen oder ob sie grundsätzlich weggelassen werden.
`attributeNames`		Es werden nur die angegebenen Attribute zu den

Parametername	**Datentyp**	**Beschreibung**
	Liste von Zeichenketten	Ergebnis-Elementen hinzugefügt.
`HASATTR`	Liste von Zeichenketten	Es werden nur die Elemente berücksichtigt, die ein Attribut mit einem der angegebenen Namen besitzt.
`EQATTR_`*`attrname`*	Liste von Zeichenketten	Es werden nur die Elemente berücksichtigt, die ein Attribut namens `attrname` besitzen, welches einen der angegebenen Werte hat.
`maxElementCount`	Zahlenwert	Übersteigt die Elementzahl des Ergebnisses den angegebenen Wert, so wird ein Fehler ausgelöst.

B. Definitionen für das Fallbeispiel

In diesem Anhangsabschnitt sollen die Bestandteile des Fallbeispieles für die Client-Applikation aus technischer Sicht aufgelistet werden. Dazu gehören zum einen die Definition der einzelnen Ereignisse, die während den Operationen gespeichert werden und auf der anderen Seite auch die genauen Parameter der Abfragen.

Hinweis:

Es werden an dieser Stelle keine Erklärungen zu den Zusammenhängen und den Bedeutungen der Elemente gemacht. Dies ist eine rein technische Darstellung. Erläuterungen finden sich innerhalb des Konzeptteiles dieser Arbeit, wo die Szenarien und die Abfragen entwickelt werden.

B.1. Ereignisse für die Darstellung der Operationen

Anmerkung:

Bei der Definition ist darauf zu achten, dass zusätzlich zu jedem einzelnen Ereignis der Zeitpunkt gespeichert wird, zu dem es eingetreten ist. Dies entspricht der simulierten Zeit des Fallbeispieles.

1. *Produktion von Produkten:*

   a. Ein Objekt-Ereignis für jedes neue Produkt:

Aktion	ADD
EPC-Liste	Enthält den EPC des neuen Produktes
Lesepunkt	Lesepunkt bei der Herstellung des Kaffees

2. *Zusammenfassen der Produkte auf einer Palette:*

   a. Ein Objekt-Ereignis für die Registrierung der Palette:

Aktion	OBSERVE
EPC-Liste	Enthält den EPC des neuen Produktes
Transaktionen	Bestellung des Distributionszentrums

   b. Ein Objekt-Ereignis für jedes Produkt, um diese mit der Bestellung zu verknüpfen:

Aktion	OBSERVE
EPC-Liste	Enthält den EPC des Produktes
Transaktionen	Bestellung des Distributionszentrums

   c. Ein Aggregationsereignis, um die Produkte auf der Palette zusammenzufassen:

Aktion	ADD
Überg. Objekt	EPC der Palette
EPC-Liste	Enthält die EPCs der Produkte
Transaktionen	Bestellung des Distributionszentrums

3. *Versenden der Palette zum Distributionszentrum:*

   a. Ein Objekt-Ereignis für das Versenden jeden Produktes:

Aktion	OBSERVE
EPC-Liste	Enthält den EPC des Produktes
Lesepunkt	Ausgang des Herstellers
Geschäftsschritt	Versenden
Transaktionen	Bestellung des Distributionszentrums

b. Ein Objekt-Ereignis für das Versenden der Palette:

Aktion	OBSERVE
EPC-Liste	Enthält den EPC der Palette
Lesepunkt	Ausgang des Herstellers
Geschäftsschritt	Versenden
Transaktionen	Bestellung des Distributionszentrums

c. Ein Objekt-Ereignis für das Empfangen der Produkte:

Aktion	OBSERVE
EPC-Liste	Enthält die EPCs der Produkte
Lesepunkt	Wareneingang des Distributionszentrums
Geschäftsschritt	Empfangen
Geschäftslokation	Distributionszentrum (Empfang)
Transaktionen	Bestellung des Distributionszentrums

d. Ein Objekt-Ereignis für das Empfangen der Palette

Aktion	OBSERVE
EPC-Liste	Enthält den EPC der Palette
Lesepunkt	Wareneingang des Distributionszentrums
Geschäftsschritt	Empfangen
Geschäftslokation	Distributionszentrum (Empfang)
Transaktionen	Bestellung des Distributionszentrums

e. Ein Transaktionsereignis für den Abschluss der Bestellung:

Aktion	DELETE
Überg. Element	EPC der Palette

Lesepunkt	Wareneingang des Distributionszentrums
Transaktionen	Bestellung des Distributionszentrums

4. *Einlagerung der empfangenen Waren:*

   a. Ein Objekt-Ereignis für die Quittierung der Palette:

Aktion	`OBSERVE`
EPC-Liste	Enthält den EPC der Palette
Lesepunkt	Eingang des Warenlagers
Geschäftslokation	Distributionszentrum (Warenlager)

   b. Ein Objekt-Ereignis für die Quittierung der Produkte:

Aktion	`OBSERVE`
EPC-Liste	Enthält die EPCs der Produkte
Lesepunkt	Eingang des Warenlagers
Geschäftslokation	Distributionszentrum (Warenlager)

5. *Waren für die Filialbestellung zusammenstellen:*

   a. Ein Objekt-Ereignis, um eine neue Palette zu registrieren:

Aktion	`ADD`
EPC-Liste	Enthält den EPC der Palette
Geschäftslokation	Distributionszentrum (Versand)
Geschäftsschritt	Versenden
Transaktionen	Bestellung der Filiale

   b. Ein Aggregationsereignis, um die Waren von der Palette zu nehmen:

Aktion	`DELETE`
Überg. Element	EPC der alten Palette

EPC-Liste	Enthält EPCs von 15 Produkten
Geschäftslokation	Distributionszentrum (Warenlager)

c. Ein Aggregationsereignis, um die Waren auf die neue Palette zu legen:

Aktion	ADD
Überg. Element	EPC der neuen Palette
EPC-Liste	Enthält die EPCs der Produkte
Geschäftslokation	Distributionszentrum (Versand)
Transaktionen	Bestellung der Filiale

d. Ein Transaktionsereignis für die Verknüpfung der Produkte mit der Bestellung:

Aktion	ADD
EPC-Liste	Enthält die EPCs der Produkte
Transaktionen	Bestellung der Filiale

6. *Versenden der Waren zur Filiale:*

a. Ein Objekt-Ereignis zum Versenden der Palette:

Aktion	OBSERVE
EPC-Liste	Enthält den EPC der Palette
Lesepunkt	Ausgang des Distributionszentrums
Geschäftschritt	Versenden
Transaktionen	Bestellung der Filiale

b. Ein Objekt-Ereignis zum Versenden der Produkte:

Aktion	OBSERVE
EPC-Liste	Enthält die EPCs der Produkte

Lesepunkt	Ausgang des Distributionszentrums
Geschäftschritt	Versenden
Transaktionen	Bestellung der Filiale

c. Ein Objekt-Ereignis für den Empfang der Palette und der Produkte:

Aktion	`OBSERVE`
EPC-Liste	Enthält die EPCs der Palette und der Produkte
Lesepunkt	Eingang der Filiale
Geschäftschritt	Empfangen
Transaktionen	Bestellung der Filiale

d. Ein Transaktionsereignis, um die Bestellung abzuschließen:

Aktion	`DELETE`
Überg. Element	EPC der Palette
EPC-Liste	Enthält die EPCs der Produkte
Lesepunkt	Eingang der Filiale
Transaktionen	Bestellung der Filiale

7. *Einräumen der Waren im Verkaufsraum:*

a. Ein Aggregationsereignis, um die Produkte von der Palette zu befreien:

Aktion	`DELETE`
Überg. Element	EPC der Palette
EPC-Liste	Enthält die EPCs der Produkte

b. Ein Objekt-Ereignis für die Freigabe der Palette:

Aktion	`DELETE`

EPC-Liste	Enthält den EPC der Palette

c. Ein Objekt-Ereignis für die Statusänderung der Produktes:

Aktion	`OBSERVE`
EPC-Liste	Enthält die EPCs der Produkte
Dispositionsschritt	Bereit zum Verkauf
Geschäftslokation	Filiale (Verkaufsraum)
Lesepunkt	Eingang zum Verkaufsraum

d. Ein Quantitäts-Ereignis, um den neuen Warenbestand zu sichern:

EPC-Klasse	Klasse der Kaffeesorte
Quantität	Neue Anzahl im Verkaufsraum
Geschäftslokation	Filiale (Verkauf)

8. *Kaufen eines Produktes:*

a. Ein Objekt-Ereignis, um das Produkt zu entfernen:

Aktion	`DELETE`
Geschäftslokation	Filiale (Verkauf)
EPC-Liste	Enthält den EPC des Produktes
Lesepunkt	Kasse der Filiale

b. Ein Quantitätsereignis, um den neuen Warenbestand zu sichern:

EPC-Klasse	Klasse der Kaffeesorte
Quantität	Neue Anzahl im Verkaufsraum
Geschäftslokation	Filiale (Verkauf)

B.2. Aufbau der einzelnen Abfragen

Anmerkung:

Beim Erscheingsbild der Abfragen ist darauf zu achten, dass die generische Parameternamen (wie zum Beispiel die der Transaktionen) in dieser Auflistung nicht vollständig ausgeschrieben werden. Sie sind teilweise so lang, dass eine gewisse Übersichtlichkeit nicht mehr gewährleistet wäre.

1. *Welche Produkte wurden vom Hersteller bislang produziert?*

`eventType`	`ObjectEvent`
`EQ_action`	`ADD`
`orderBy`	`eventTime`
`orderDirection`	`ASC`
`EQ_readPoint`	Lesepunkt der Herstellerproduktion

2. *Welche Paletten stehen mit der Bestellung des Distributionszentrums in Beziehung?*

`eventType`	`AggregationEvent`
`EQ_action`	`ADD`
Transaktion	Bestellung des Distributionszentrums

3. *Wann hat die Palette des Herstellers das Distributionszentrum erreicht?*

`eventType`	`ObjectEvent`
`MATCH_epc`	EPC der Palette des Herstellers
`EQ_readPoint`	Lesepunkt am Empfang des Distributionszentrums

4. *Welche Objekte befinden sich zur Zeit im Warenlager des Distributionszentrums?*

`eventType`	`ObjectEvent`, `AggregationEvent`
`EQ_action`	`OBSERVE`, `DELETE`
`EQ_bizLocation`	Distributionszentrum (Warenlager)

5. *Welche Objekte haben aufgrund der Bestellung der Filiale das Distributionszentrum verlassen?*

`eventType`	`ObjectEvent`
`EQ_readPoint`	Lesepunkt am Ausgang des Distributionszentrums
Transaktion	Bestellung der Filiale

6. *Welche Produkte haben die Filiale aufgrund der Bestellung erreicht?*

`eventType`	`ObjectEvent`
`EQ_readPoint`	Lesepunkt am Warenempfang der Filiale
Transaktion	Bestellung der Filiale

7. *Welche Produkte sind während der Lieferung verlorengegangen?*

`eventType`	`ObjectEvent`
`EQ_readPoint`	Lesepunkte am Ausgang des Distributionszentrums und am Eingang der Filiale
Transaktion	Bestelleung der Filiale

8. *Wie viele Pakete Kaffee befinden sich im Verkaufsraum der Filiale?*

`eventType`	`QuantityEvent`
`MATCH_epcClass`	EPC-Klasse der Kaffeesorte
`EQ_bizLocation`	Verkaufsraum der Filiale

`orderBy`	`eventTime`
`orderDirection`	`DESC`
`eventCountLimit`	`1`

9. *Wie viele Produkte wurden heute verkauft?*

`eventType`	`ObjectEvent`
`EQ_action`	`DELETE`
`EQ_readPoint`	Kasse der Filiale
`GE_eventTime`	Heutiges Datum, 8:00 Uhr
`LT_eventTime`	Morgiges Datum, 8:00 Uhr

C. Definition der Erfassungsschnittstelle

Dieser Anhang zeigt die Definition der Erfassungsschnittstelle und die dazu gehörigen Typen. Die Schnittstelle wurde in dieser Form nicht durch die EPCIS-Spezifikation, sondern im Rahmen dieser Arbeit entwickelt.

Es wird trotzdem versucht, sich streng am konzeptionellen Aufbau der Abfrage-Schnittstelle zu orientieren.

C.1. WSDL-Definition der Schnittstelle

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

<wsdl:definitions
  targetNamespace="http://eisprinz.net/epcis/capture"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:epcisc="http://eisprinz.net/epcis/capture/xsd"
  xmlns:impl="http://eisprinz.net/epcis/capture"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- EPCIS CAPTURE SERVICE TYPES -->

  <wsdl:types>
    <xsd:schema
      targetNamespace="http://eisprinz.net/epcis/capture"
      xmlns:impl="http://eisprinz.net/epcis/capture"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import
        namespace="http://eisprinz.net/epcis/capture/xsd"
        schemaLocation="./epcisc.xsd" />
    </xsd:schema>
  </wsdl:types>

  <!-- EPCIS CAPTURE SERVICE MESSAGES -->

  <wsdl:message name="captureRequest">
    <wsdl:part name="parms" element="epcisc:Capture" />
  </wsdl:message>
  <wsdl:message name="captureResponse">
    <wsdl:part name="captureReturn" element="epcisc:VoidHolder" />
  </wsdl:message>

  <!-- CAPTURE SERVICE FAULT EXCEPTIONS -->
```

```
  <wsdl:message name="SecurityExceptionResponse">
    <wsdl:part name="fault" element="epcisc:SecurityException" />
  </wsdl:message>
  <wsdl:message name="ImplementationExceptionResponse">
    <wsdl:part name="fault" element="epcisc:ImplementationException" />
  </wsdl:message>

  <!-- CAPTURE SERVICE PORTTYPE -->

  <wsdl:portType name="CaptureServicePortType">

    <wsdl:operation name="capture">
      <wsdl:input
        message="impl:captureRequest" name="captureRequest" />
      <wsdl:output
        message="impl:captureResponse" name="captureResponse" />
      <wsdl:fault
        message="impl:SecurityExceptionResponse"
        name="SecurityExceptionFault" />
      <wsdl:fault
        message="impl:ImplementationExceptionResponse"
        name="ImplementationExceptionFault" />
    </wsdl:operation>

  </wsdl:portType>

  <!-- CAPTURE SERVICE BINDING -->

  <wsdl:binding name="CaptureServiceBinding"
    type="impl:CaptureServicePortType">

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

    <wsdl:operation name="capture">
      <wsdlsoap:operation soapAction="" />
      <wsdl:input name="captureRequest">
        <wsdlsoap:body use="literal" />
      </wsdl:input>
      <wsdl:output name="captureResponse">
        <wsdlsoap:body use="literal" />
      </wsdl:output>
      <wsdl:fault name="SecurityExceptionFault">
        <wsdlsoap:fault
          name="SecurityExceptionFault" use="literal" />
      </wsdl:fault>
      <wsdl:fault name="ImplementationExceptionFault">
        <wsdlsoap:fault
          name="ImplementationExceptionFault" use="literal" />
      </wsdl:fault>
    </wsdl:operation>

  </wsdl:binding>

  <!-- EPCIS CAPTURE SERVICE -->

  <wsdl:service name="CoreCaptureService">
    <wsdl:port
      binding="impl:CaptureServiceBinding" name="CaptureServicePort">
      <!-- The address shown below is an example;
        an implementation MAY specify any port it wishes -->
      <wsdlsoap:address
        location="http://localhost:7070/EPCISServer/capture" />
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

C.2. Datentypen der Schnittestelle

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

<xsd:schema
  targetNamespace="http://eisprinz.net/epcis/capture/xsd"
  xmlns:epcis="urn:epcglobal:epcis:xsd:1"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:epcisc="http://eisprinz.net/epcis/capture/xsd"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified"
  version="1.0">

  <!-- EXTERN TYPES -->

  <xsd:import namespace="urn:epcglobal:epcis:xsd:1"
    schemaLocation="./epcis.xsd" />

  <!-- TYPES -->

  <xsd:element name="Capture" type="epcisc:Capture" />

  <xsd:complexType name="Capture">
    <xsd:sequence>
      <xsd:element name="eventList" type="epcis:EventListType" />
      <xsd:element name="extension"
        type="epcisc:CaptureExtensionType" minOccurs="0" />
      <xsd:any namespace="##other"
        processContents="lax" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="CaptureExtensionType">
    <xsd:sequence>
      <xsd:any namespace="##local"
        processContents="lax" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax" />
  </xsd:complexType>

  <xsd:element name="VoidHolder" type="epcisc:VoidHolder" />

  <xsd:complexType name="VoidHolder">
    <xsd:sequence>
    </xsd:sequence>
  </xsd:complexType>

  <!-- EXCEPTIONS -->

  <xsd:element name="EPCISException" type="epcisc:EPCISException" />

  <xsd:complexType name="EPCISException">
    <xsd:sequence>
      <xsd:element name="reason" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="SecurityException"
    type="epcisc:SecurityException" />

  <xsd:complexType name="SecurityException">
    <xsd:complexContent>
      <xsd:extension base="epcisc:EPCISException">
        <xsd:sequence />
      </xsd:extension>
    </xsd:complexContent>
```

```
    </xsd:complexType>

    <xsd:element name="ImplementationException"
      type="epcisc:ImplementationException" />

    <xsd:complexType name="ImplementationException">
      <xsd:complexContent>
        <xsd:extension base="epcisc:EPCISException">
          <xsd:sequence>
            <xsd:element name="severity"
              type="epcisc:ImplementationExceptionSeverity" />
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>

    <xsd:simpleType name="ImplementationExceptionSeverity">
      <xsd:restriction base="xsd:NCName">
        <xsd:enumeration value="ERROR" />
        <xsd:enumeration value="SEVERE" />
      </xsd:restriction>
    </xsd:simpleType>

  </xsd:schema>
```

D. Beispiel eines Datenaustausches

Innerhalb dieses Anhangs soll beispielhaft der SOAP-Code einer Abfrage gezeigt werden, der während eines Transfers verschickt wird. Dazu gehört natürlich der Code der Anfrage und der dazu passenden Antwort.

Für diesen Zweck wurde diejenige Abfrage aus dem Fallbeispiel gewählt, welche als Ergebnis alle Elemente enthält, die das Distributionszentrum aufgrund der Filial-Bestellung verlassen haben.

Hinweis:

Beim Antwort-Code wurde auf die Darstellung des Sicherheitsblocks verzichtet, da dieser in seiner Struktur analog dem der Anfrage aufgebaut ist.

D.1. SOAP-Code der Abfrage

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

  <SOAP-ENV:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-wssecurity-secext-1.0.xsd"
      SOAP-ENV:mustUnderstand="1">
      <wsse:BinarySecurityToken
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-utility-1.0.xsd"
        EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary"
        ValueType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-x509-token-profile-1.0#X509v3"
        wsu:Id="XWSSGID-11627370648621880775443">
        MIIB5DCCAU0CBEUbxZowDQYJKoZIhvcNAQEEBQAwOTEVMBMGA1UEChMMZWlzc
        HJpbnoubmV0MSAwHgYDVQQDExdTY2VuYXJpbyBFeHRlbnNpb24gVXNlcjAeFw
        0wNjA5MjgxMjUyNDJaFw0wNzA5MjgxMjUyNDJaMDkxFTATBgNVBAoTDGVpc3B
        yaW56Lm5ldDEgMB4GA1UEAxMXU2NlbmFyaW8gRXh0ZW5zaW9uIFVzZXIwgZ8w
        DQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIlKOrO9Ll7K+IiM5qik3HbktZj43
        wXsopGu9odYAvM5CI5v7+/+cUdcCf5qeIlPKE5LWVT7r5ZOpPA1k3x31hdf83
        EHiRpcHD9h7Eh3cKYrnkGsCZkBGTQljWIGBsBE49hWtjfeHALuYDEeE3XPyjb
        SKLtzb4BRwoQ2GVRHn+ZdAgMBAAEwDQYJKoZIhvcNAQEEBQADgYEAFiypUhvL
        5T2GicW3OLznWFItzi5pb9ijag8giSBhzcjdpmskT/K8VwUUP+38+yW4+TIOH
        0lA8/nI5d2dW5qFy8GU66BfR5q43vZdzfpci7wQKR1A5JqD1EA11QznfCH+Bh
```

```
          MecqYvi5ZR+mZbpbJbbYcsQvenCkDl8eZ87ayBxQ=
        </wsse:BinarySecurityToken>
        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <ds:SignedInfo>
            <ds:CanonicalizationMethod
              Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
              <InclusiveNamespaces
                xmlns="http://www.w3.org/2001/10/xml-exc-c14n#"
                PrefixList="wsse SOAP-ENV"/>
            </ds:CanonicalizationMethod>
            <ds:SignatureMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <ds:Reference URI="#XWSSGID-1162737065012-85348548">
              <ds:DigestMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>
                F0kYMgAHN9p1I+k1GOTY9E0BFIc=
              </ds:DigestValue>
            </ds:Reference>
            <ds:Reference URI="#XWSSGID-1162737065012-1811668020">
              <ds:DigestMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>
                Ffn7eEaeXoDxhlHoQRK+vXbhy3Y=
              </ds:DigestValue>
            </ds:Reference>
          </ds:SignedInfo>
          <ds:SignatureValue>
            RCwObGF8GdXIYClQR/jA3MgLlLK4RJtv2hkFgc//mpvU+yl49BWosGu9EXQ
            M2C1Qnm1NcnOFCZUxzg/MEqnciWNhXMwNx0Wv9WF407/ud52TCK1iEwb34H
            jYtZLRwbNbI7094KvlAGmeDimuy5E5U+C2JZYZ+XKfRx6dMT507uo=
          </ds:SignatureValue>
          <ds:KeyInfo>
            <wsse:SecurityTokenReference
              xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                oasis-200401-wss-wssecurity-utility-1.0.xsd"
              wsu:Id="XWSSGID-116273706501214229307">
              <wsse:Reference
                URI="#XWSSGID-11627370648621880775443"
                ValueType="http://docs.oasis-open.org/wss/2004/01/
                  oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
            </wsse:SecurityTokenReference>
          </ds:KeyInfo>
        </ds:Signature>
        <wsu:Timestamp
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
            oasis-200401-wss-wssecurity-utility-1.0.xsd"
          wsu:Id="XWSSGID-1162737065012-1811668020">
          <wsu:Created>2006-11-05T14:31:05Z</wsu:Created>
          <wsu:Expires>2006-11-05T14:31:10Z</wsu:Expires>
        </wsu:Timestamp>
      </wsse:Security>
    </SOAP-ENV:Header>

    <SOAP-ENV:Body
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-wssecurity-utility-1.0.xsd"
      wsu:Id="XWSSGID-1162737065012-85348548">
      <ns3:Poll
        xmlns:ns2="http://www.unece.org/cefact/
          namespaces/StandardBusinessDocumentHeader"
        xmlns:ns3="urn:epcglobal:epcis-query:xsd:1"
        xmlns:ns4="urn:epcglobal:epcis:xsd:1"
        xmlns:ns5="urn:epcglobal:epcis-masterdata:xsd:1">
        <queryName>SimpleEventQuery</queryName>
        <params>
          <param>
            <name>eventType</name>
            <value
```

```
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="ns3:ArrayOfStrings">
            <string>ObjectEvent</string>
          </value>
        </param>
        <param>
          <name>EQ_readPoint</name>
          <value
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="ns3:ArrayOfStrings">
            <string>urn:epcglobal:fmcg:loc:1234001123002.RP-3</string>
          </value>
        </param>
        <param>
          <name>EQ_bizTransaction_urn:epcglobal:fmcg:btt:po</name>
          <value
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="ns3:ArrayOfStrings">
            <string>
              http://eisprinz.net/epcis/scenario/po/12345602
            </string>
          </value>
        </param>
      </params>
    </ns3:Poll>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

D.2. SOAP-Code der Antwort (Ausschnitt)

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

  <SOAP-ENV:Header>

    [...]

  </SOAP-ENV:Header>

  <SOAP-ENV:Body
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
      oasis-200401-wss-wssecurity-utility-1.0.xsd"
    wsu:Id="XWSSGID-1162737065182-2029864752">
    <ns3:PollResult
      xmlns:ns2="http://www.unece.org/cefact/
        namespaces/StandardBusinessDocumentHeader"
      xmlns:ns3="urn:epcglobal:epcis-query:xsd:1"
      xmlns:ns4="urn:epcglobal:epcis:xsd:1"
      xmlns:ns5="urn:epcglobal:epcis-masterdata:xsd:1">
      <resultsBody>
        <eventList>
          <ObjectEvent>
            <eventTime>2000-01-02T12:30:00.0</eventTime>
            <recordTime>2006-11-05T15:28:42.868</recordTime>
            <epcList>
              <epc>urn:epc:id:sgtin:1234001.123001.1</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.2</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.3</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.4</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.5</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.6</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.7</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.8</epc>
```

```
              <epc>urn:epc:id:sgtin:1234001.123001.9</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.10</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.11</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.12</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.13</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.14</epc>
              <epc>urn:epc:id:sgtin:1234001.123001.15</epc>
            </epcList>
            <action>OBSERVE</action>
            <bizStep>urn:epcglobal:fmcg:bizstep:shipping</bizStep>
            <readPoint>
              <id>urn:epcglobal:fmcg:loc:1234001123002.RP-3</id>
            </readPoint>
            <bizTransactionList>
              <bizTransaction type="urn:epcglobal:fmcg:btt:po">
                http://eisprinz.net/epcis/scenario/po/12345602
              </bizTransaction>
            </bizTransactionList>
          </ObjectEvent>
          <ObjectEvent>
            <eventTime>2000-01-02T12:30:00.0</eventTime>
            <recordTime>2006-11-05T15:28:42.858</recordTime>
            <epcList>
              <epc>urn:epc:id:sgln:1234002.12301.1</epc>
            </epcList>
            <action>OBSERVE</action>
            <bizStep>urn:epcglobal:fmcg:bizstep:shipping</bizStep>
            <readPoint>
              <id>urn:epcglobal:fmcg:loc:1234001123002.RP-3</id>
            </readPoint>
            <bizTransactionList>
              <bizTransaction
                type="urn:epcglobal:fmcg:btt:po">
                http://eisprinz.net/epcis/scenario/po/12345602
              </bizTransaction>
            </bizTransactionList>
          </ObjectEvent>
        </eventList>
      </resultsBody>
    </ns3:PollResult>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

E. Eingesetzte Software

Die Programmierung und das Projekt selbst basieren auf unterschiedlicher Software, die in diesem Anhang aufgezeigt werden soll. Dabei wurde darauf geachtet, dass die Software frei verfügbar und somit im Normalfall problemlos über das Internet zu beziehen ist.

Hinweis zu den Versionsangaben:

Die angegebenen Versionsnummern beziehen sich auf die für die Entwicklung verwendeten Versionen der Programme. Es ist somit durchaus möglich, dass unter den angegebenen Webadressen in der Zwischenzeit aktuellere Fassungen zu finden sind, die dann aber nicht mit diesem Projekt getestet wurden.

Folgende Software kam für die Implementierung des Projektes zum Einsatz:

Programmiersprache	*Java 2 Standard Edition, Version 1.5.0 Update 6*. Internet: http://java.sun.com/javase/
Webserver	*Apache Tomcat, Version 5.5.17*. Internet: http://tomcat.apache.org
Web-Service-Engine	*Sun Java Web Services Developer Pack (Java WSDP), Version 2.0*. Internet: http://java.sun.com/webservices/jwsdp/index.jsp
Sicherheitsengine	*Sun XML Web Service Security (XWSS), Version 2.0*. Internet: http://java.sun.com/webservices/xwss/index.jsp

Datenbank	*HSQLDB, Version 1.8.0.4*. Internet: http://hsqldb.org
Entwicklungsumgebung	*Eclipse SDK, Version 3.1.2*. Internet: http://www.eclipse.org

Zusätzliche Plugins:

- *Web Standard Tools, Version 1.0.2*. Internet: http://www.eclipse.org/webtools/
- *J2EE Standard Tools, Version 1.0.2*. Internet: http://www.eclipse.org/webtools/

Grafikprogramm	*The GNU Image Manipulation Program (GIMP), Version 2.2.10*. Internet: http://www.gimp.org
Protokoll-Proxy	*TCPMonitor aus dem Apache Axis Projekt, Version 1.4*. Internet: http://ws.apache.org/axis/
Webbrowser	*Mozilla Firefox, Version 1.5.0.7*. Internet: http://www.mozilla-europe.org/de/products/firefox/

F. Inhalt der beiliegenden CD

- `Binaries`
  - `epcis.jar` (JAR-Archiv mit der Bibliothek der Grunddatentypen)
  - `EPCISClient.war` (WAR-File der EPCIS-Client-Applikation)
  - `EPCISServer.war` (WAR-File des EPCIS-Servers)
- `Diplomarbeit`
  - `diplomarbeit.pdf` (dieses Dokument)
  - `dtd` (xinclude-Modul)
  - `images` (verwendete Abbildungen)
  - `src` (Docbook-Quellen der Arbeit)
  - `xsl` (Stylesheets für die Transformation)
- `Dokumentation` (API-Beschreibungen des Quellcodes)
- `Screeshots` (Grafiken mit Abbildungen der Client-Applikation)
- `Sources`
  - `EPCISCore` (Quellcode der Bibliothek der EPCIS-Grunddatentypen)
  - `EPCISClient` (Quellcode der EPCIS-Client-Applikation)
  - `EPCISServer` (Quellcode des EPCIS-Servers)

Glossar

Barcode

Strichcode, über dessen Linien und den dazwischenliegenden Abständen sich ein bestimmter Wert codieren lässt. Er wird häufig im Handel eingesetzt, um eine EAN-Nummer darzustellen.

Siehe auch EAN.

CRM

Customer Relationship Management. Verwaltung von Kundenbeziehungen über eine durch Software realisierte Organisiation von bestimmten, den Kunden betreffenden Daten, insbesondere seinen Transaktionen.

EAN

International Article Number (ehemals *European Article Number*). Weltweit eindeutige Auszeichnungsnummer, die für die Kennzeichnung von Handelsartikeln eingesetzt wird. Sie beschreibt im Gegensatz zum EPC nur die Klasse des Produktes.

Siehe auch EPC.

EDI

Electronic Data Interchange. Automatisierter, elektronischer Austausch von Daten zwischen einzelnen Anwendungssystemen.

EPC

Electronic Product Code. Code, durch den sich in Kombination mit der RFID-Technologie weltweit Objekte eindeutig identifizieren lassen. Im Gegensatz zum EAN-Code verfügt der EPC auch über eine Seriennummer.

Siehe auch RFID, EAN.

EPC-Netzwerk

Eine durch EPCglobal standardisierte Netzwerk-Infrastruktur, über die Informationen zu eindeutig beschriebenen Objekten in Echtzeit abgerufen werden können.

Siehe auch EPCglobal Inc., EPC.

EPCglobal Inc.

Non-for-profit Organisation, die durch *EAN International* und das *Uniform Code Council, Inc.* gegründet wurde und wirtschaftliche und technische Standards für das EPC-Netzwerk entwickelt.

Siehe auch EPC.

EPCIS

EPC Information Services. Informationsdienste innerhalb des EPC-Netzwerkes, über die sich Informationen zu einzelnen Produkten oder auch deren Status speichern und wieder abfragen lassen.

Siehe auch EPC.

EPCIS-Daten

Spezielle Datenstrukturen, die von Client-Anwendungen mit einem EPCIS-Server ausgetauscht werden. Dazu zählen insbesondere sogannte Ereignisse und Stammdaten.

Siehe auch Ereignis, Stammdaten, EPCIS.

Ereignis

Eine Datenstruktur innerhalb von EPCIS-Systemen, die bestimmte Situationen innerhalb der Prozesskette modellieren. Sie gehören zu den EPCIS-Daten.

Siehe auch EPCIS-Daten.

GTIN

Global Trade Item Number. Sammelbegriff für spezielle Nummern, mit denen sich Produkte eindeutig identifizieren lassen.

Siehe auch EPC.

ONS

Object Naming Service. Zählt zu den Ermittlungsdiensten des EPC-Netzwerkes und wird eingesetzt, um die EPCIS-Server zu lokalisieren, auf denen Informationen zu einem bestimmten EPC hinterlegt sind.

Siehe auch EPC, URL.

Pulkerfassung

Verfahren, über das mit Hilfe der RFID-Technologie eine bestimmte Menge von Objekten zeitgleich gescannt werden kann.

Siehe auch RFID.

QName

Qualified Name. Global eindeutiger Bezeichner, bestehend aus einem Namensraum-Präfix und einem diesem Namensraum zugeordnetem, lokalen Namen.

RFID

Radio Frequency Identification. Technologie, durch die eine automatische Identifikation von Objekten und deren Lokalisierung über Funk ermöglicht wird. Dabei wird oft der EPC eingesetzt.

Siehe auch EPC.

Stammdaten

Spezielle Form von EPCIS-Daten, die dafür eingesetzt werden, um einzelne Elemente innerhalb des EPC-Netzwerkes genauer zu beschreiben. Dafür werden spezielle Wortschätze definiert.

Siehe auch EPCIS-Daten.

URI

Uniform Resource Identifier. Zeichenfolge, mit der abstrakte und reale Ressourcen identifiziert werden können. Unterarten einer URI sind URLs und URNs.

Siehe auch URL, URN.

URL

Uniform Resource Locator. Unterart der URI, die zur Adressierung von Ressourcen über das verwendete Protokoll und den Domainnamen eingesetzt wird. Besonders verbreitet ist dies für Adressieurungen im Internet.

Siehe auch URI, URN.

URN

Uniform Resource Name. Unterart der URI, welche die Identifikation einer Ressource ortsunabhängig über ihre gesamte Lebensdauer ermöglicht.

Siehe auch URI, URL.

Literaturverzeichnis

Bücher

[Axis] Wang, Dapeng ; Bayer, Thomas ; Frotscher, Thilo ; Teufel, Marc: *Java Web Services mit Apache Axis*. 1. Auflage. Frankfurt : Software & Support Verlag GmbH, 2004.

[CoreServlets] Hall, Marty ; Brown, Larry: *Core Servlets und Java Server Pages*. 2. Auflage. München : Markt+Technik Verlag, 2004.

[DocBook] Walsh, Norman ; Muellner, Leonard: *DocBook. The Definitive Guide*. 1. Auflage. Sebastopol : O´Reilly & Associates, Inc., 1999.

[DocBookXML] Trieloff, Lars: *DocBook-XML. Einführung und Anwendung*. 1. Auflage. Bonn : Mitp-Verlag, 2005.

[DocBookXSL] Stayton, Bob: *DocBook XSL. The Complete Guide*. 3. Auflage. Santa Cruz CA : Sagehill Enterprises, 2005.

[J2EECodebook] Stark, Thomas ; Samaschke, Karsten: *Das J2EE-Codebook*. 1. Auflage. München : Addison-Wesley Verlag, 2005.

[JavaHandbuch] Krüger, Guido: *Handbuch der Java-Programmierung*. 4. Auflage. München : Addison-Wesley Verlag, 2005.

[JavaWebServices] Chappell, David A. ; Jewell, Tyler: *Java Web Services*. 1. Auflage. Köln : O´Reilly Verlag, 2003.

[RFID2] Finkenzeller, Klaus: *RFID-Handbuch*. 2. Auflage. München : Carl Hanser Verlag München, 2000. http://www.rfid-handbook.de .

[RFID3] Finkenzeller, Klaus: *RFID-Handbuch*. 3. Auflage. München : Carl Hanser Verlag München, 2002. http://www.rfid-handbook.de .

[XMLKompendium] Phillips, Lee Anne: *XML. Modernes Daten- und Dokumentenmanagement*. 1. Auflage. München : Markt+Technik Verlag, 2002.

Zeitschriften

[Computerwoche0706] Füßler, Andreas ; Springob, Katrin: *Gen 2 – die Zukunft der Funktechnik*. In: *Computerwoche* 7 (2006), S. 34-35. http://www.gs1-germany.de/content/e39/e466/e468/datei/ccg/zukunft_funktechnik.pdf .

[Funkschau1998] Finkenzeller, Klaus: *Kontaktlose Chipkarten*. In: *Funkschau* 19 (1998), S. 40-43. http://www.rfid-handbook.de/downloads/fs9819040.pdf .

[Javamagazin0303] Edlich, Stefan: *Fliegengewicht. Open Source-Perlen: Die Sourceforge Datenbank HSQLDB*. In: *Javamagazin* 3 (2003), http://entwickler.com/itr/online_artikel/psecom,id,312,nodeid,11.html .

[Javamagazin0402] Ford, Neal: *Das Traumpaar Java und XML. Workshop: Arbeiten mit XML in Java, Teil 1*. In: *Javamagazin* 4 (2002), http://javamagazin.de/itr/online_artikel/psecom,id,94,nodeid,11.html .

Informationspapiere

[Dinge] Popova, Tania: *Internet der Dinge. Das EPCglobal-Netzwerk*. GS1 Germany GmbH, März 2005. http://www.gs1-germany.de/content/e39/e466/e468/datei/epc_rfid/mip_6_internet_der_dinge_gs1.pdf .

[EPCISInfo] Kuhlmann, Frank ; Amende, Marcel: *EPC-Informationsservice (EPCIS) und Umsetzung im EPC-Showcase. Konzept und Anwendung des EPCIS im EPCglobal-Netzwerk*. GS1 Germany GmbH, August 2006. http://www.gs1-germany.de/content/e39/e466/e468/datei/epc_rfid/epcis_epcshowcase.pdf .

[Etikettenloesung] GS1 Germany: *RFID/EPC-Etikettenlösung für die Gestaltung und den Einsatz eines einheitlichen RFID/EPC-Transportetiketts, inklusive Datensicherungsfunktion*. GS1 Germany GmbH, Mai 2006. http://www.gs1-germany.de/content/e39/e466/e468/datei/epc_rfid/transportetikett.pdf .

[Faelschungssicherheit] Füßler, Andreas: *Fälschungssicherheit per EPC. Über EPC und EPCglobal-Netzwerk Warenechtheit gewährleisten*. GS1 Germany GmbH, März 2006. http://www.gs1-germany.de/content/e39/e466/e468/datei/epc_rfid/mip_faelschungssicherheit.pdf .

[GCIExecutive] Global Commerce Initiative: *Global Commerce Initiative EPC Roadmap Executive Brief*. Deutsche Ausgabe. Global Commerce Initiative, Mai 2004. http://www.gs1-germany.de/content/e39/e466/e468/datei/ccg/gci_epc_roadmap_executive_brief_dt.pdf .

[GCIRoadmap] Global Commerce Initiative: *Global Commerce Initiative EPC Roadmap*. Global Commerce Initiative, November 2003. http://www.gs1-germany.de/content/e39/e466/e468/datei/ccg/gci_epc_roadmap.pdf .

[IFDCapture] GS1 Hong Kong: *EPCnetwork Interface Document. Capture Interface*. GS1 Hong Kong, Dezember 2005.

[IFDQuery] GS1 Hong Kong: *EPCnetwork Interface Document. Query Interface*. GS1 Hong Kong, Dezember 2005.

[Kommunikation] Kuhlmann, Frank ; Popova, Tania: *EPC-Kommunikation. Prozesse und Nachrichten unter Verwendung des EPC-Informationsnetzwerks und EDI*. GS1 Germany GmbH, August 2005. http://www.gs1-germany.de/content/e39/e466/e468/datei/epc_rfid/mip_9_kommunikation.pdf .

[Mavrommatis] Assiotis, Marios ; Mavrommatis, Panayiotis: *The EPC Global Network. A formal specification of EPCIS*. Massachusetts Institute of Technology, Mai 2006. http://mavrommatis.googlepages.com/report.pdf .

[Pilotprojekte] Kuhlmann, Frank ; Popova, Tania: *EPC-Pilotprojekte*. GS1 Germany GmbH, Juni 2006. http://www.gs1-germany.de/content/e39/e466/e468/datei/epc_rfid/pilotprojekte.pdf .

[ValueChain] Efficient Consumer Response: *RFID – Optimierung der Value Chain. Einsatzbereiche, Nutzenpotenziale und Herausforderungen*. Centrale für Coorganisation GmbH, Mai 2003. http://www.gs1-germany.de/content/e39/e466/e468/datei/epc_rfid/mip_user_req_uhf_regulations.pdf .

Technische Spezifikationen

[ALE] EPCglobal: *The Application Level Events (ALE) Specification Version 1.0*. EPCglobal, September 2005. http://www.epcglobalinc.org/standards_technology/EPCglobal_ApplicationALE_Specification_v112-2005.pdf .

[Architecture] EPCglobal: *The EPCglobal Architecture Framework Version 1.0*. EPCglobal, Juli 2005. http://www.epcglobalinc.org/standards_technology/Final-epcglobal-arch-20050701.pdf .

[Certificate] EPCglobal: *EPCglobal Certificate Profile*. EPCglobal, März 2006. http://www.epcglobalinc.org/standards_technology/SAG%20Security%20Cert%20Profile%20March%208%202006.pdf .

[EPCIS] EPCglobal: *EPC Information Services (EPCIS) Version 1.0*. Last Call Working Draft. EPCglobal, März 2006.

[ONS] EPCglobal: *Object Naming Service (ONS) Version 1.0*. EPCglobal, Oktober 2005. http://www.epcglobalinc.org/standards_technology/EPCglobal_Object_Naming_Service_ONS_v112-2005.pdf .

[SBDH] UN/CEFACT: *Standard Business Document Header Technical Specification Version 1.3*. UN/CEFACT, Juni 2004. http://www.disa.org/cefact-groups/atg/downloads/CEFACT_SBDH_TS_version1.3.zip .

[TDS1.27] EPCglobal: *EPC Tag Data Standards Version 1.1 Rev. 1.27*. EPCglobal, Mai 2005. http://www.epcglobalinc.org/standards_technology/EPC_TDS_1%201_Rev_1%2027_Ratification_final%201-2006.pdf .

[TDS1.3] EPCglobal: *EPCglobal Tag Data Standards Version 1.3*. EPCglobal, März 2006. http://www.epcglobalinc.org/standards_technology/Ratified%20Spec%20March%208%202006.pdf .

Dokumente

[JWSDP1.6] Sun Microsystems Inc.: *The Java Web Services Tutorial. For Java Web Services Developer´s Pack, v1.6*. Sun Microsystems Inc., Juni 2005. http://java.sun.com/webservices/docs/1.6/tutorial/doc/JavaWSTutorial.pdf .

[JWSDP2.0] Sun Microsystems Inc.: *The Java Web Services Tutorial. For Java Web Services Developer´s Pack, v2.0*. Sun Microsystems Inc., Februar 2006. http://java.sun.com/webservices/docs/2.0/tutorial/doc/JavaWSTutorial.pdf .

[JWSDPArticle] Ort, Ed ; Mandava, Ramesh: *Java Web Services Developer Pack – Part 2. RPC Calls, Messaging, and the JAX-RPC and JAXM API*. Sun Microsystems Inc., Oktober 2002. http://java.sun.com/developer/technicalArticles/WebServices/WSPack2/wspack2.pdf .

Internet-Quellen

[Accada] Auto-ID Lab ETH Zurich ; University St. Gallen: *Accada*. *EPC Network Prototyping Platform*. Stand: 15. September 2006. http://www.accada.org/index.html .

[EPCForum] Institut, Management + Consulting AG: *EPC-Forum*. Stand: 21. Juni 2006. http://epc-forum.de/index.html .

[EPCglobal] EPCglobal Inc.: *EPCglobal Homepage*. Stand: September 2006. http://www.epcglobalinc.org/home .

[GS1Germany] GS1 Germany GmbH: *Produkte und Dienstleistungen. EPCglobal*. Stand: 9. November 2006. http://www.gs1-germany.de/internet/content/produkte/epcglobal/index_ger.html .

[MITEPCNet] Auto-ID Labs: *MIT EPC Net (MENTOR). The EPC Information Services (EPCIS) community*. Stand: 27. September 2006. http://epcis.mit.edu/CS/Default.aspx .