"Anforderungen an eine systemübergreifende Lösung für die Synchronisation von strukturierten Inhalten im mobilen Umfeld"

STUDIENARBEIT

von

Gino Fehr

Matrikel-Nummer 4627994

Betreut von Dr. Ralf Klischewski

Arbeitsbereich Softwaretechnik

Fachbereich Informatik

Universität Hamburg

21. September 2003

Gino Fehr
Ruckteschellweg 6
22089 Hamburg
E-Mail: gino@familiefehr.de

Vielen Dank an meine liebe Frau Pam, die mir immer wieder Kraft gibt.

INHALTSVERZEICHNIS

1 Einleitung

1.1 Motivation

Schon heute besteht ein Bedarf, strukturierte Inhalte mobil zu verarbeiten. Jeder Hersteller von Taschencomputern (sog. Handhelds) ermöglicht es, z.B. unterwegs Adressen und Termine zu erfassen oder bestehende zu verändern. Zurück am heimischen Arbeitsplatz können diese Inhalte mit dem Stand des Arbeitplatzes abgeglichen werden.

Es darf angenommen werden, dass der Bedarf dieser mobilen Verarbeitung von Inhalten noch weiter zunimmt. So werden zum einen mit dem UMTS-Netz Bedingungen geschaffen, die eine neue Generation von Funktelefonen ermöglicht, die dann auch klassische Aufgaben der Handhelds abdecken werden. Zum anderen werden die Handhelds immer leistungsfähiger und liegen in der Anwendungsperformance nur wenige Jahre hinter der der Arbeitsplatzrechner zurück.

Die Motivation dieser Arbeit entstand dabei aus der Tatsache, dass der Vorgang, mit dem die Inhalte zwischen Arbeitsplatz und mobilem Gerät abgeglichen werden, von jedem Hersteller unterschiedlich beschrieben wird. Die Firma Palm setzt unter deren Handheld Betriebssystem PalmOS z. Zt. auf das Protokoll HotSync, während z.B. Microsoft für deren Windows CE das hauseigene ActiveSync favorisiert.

Erst seit kurzem gibt es Bemühungen, einen Standard für das Synchronisations-Protokoll zu definieren. Das SyncML Konsortium besteht u.a. aus so namenhaften Firmen wie Ericsson, IBM, Matsushita, Motorola, Nokia, Palm, Psion, Siemens, AOL, Software AG und Vodafone. Mit dem offenen SyncML Protokoll wird es erstmals möglich sein, Inhalte herstellerübergreifend abzugleichen. SyncML beschreibt dabei nur das Protokoll, der Prozess der Synchronisation werden sog. Synchronisations-Agenten überlassen. In dieser Arbeit soll auch der Prozess untersucht und die Anforderungen an einen Agenten definiert werden.

1.2 Fragestellung

Die Fragestellung lautet:

"Wie können strukturierte Inhalte zwischen mobilen und stationären Systemen synchronisiert werden ?"

Etwas abstrakter geht es in der Frage darum, wie Datenstrukturen zwischen unterschiedlichen Systemen übertragen werden können und was bei Änderungen der Daten in einem System passieren muss, damit die anderen Systeme die Änderungen verfolgen können bzw. entsprechend reagieren können. Für diese Arbeit soll es genügen, eine Antwort der Fragestellung auf Protokollebene zu finden. Andere Ebenen werden nicht betrachtet oder nur am Rande erwähnt.

Als Beispiel kann sich eine Termindatenbank vorgestellt werden, die auf einem Arbeitsplatzrechner installiert ist. Die Termineinträge lassen sich auf einen Handheld übertragen, sodass unterwegs darauf zurückgegriffen werden kann. Wird auf diesem System ein neuer Termin erfasst oder bestehende geändert, so muss die Termindatenbank auf dem Arbeitsplatzrechner abgeglichen werden.

1.3 Vorgehensweise

Die Fragestellung soll in einem realistischen Umfeld beantwortet werden. Das Szenario besteht aus einem Content Management System (CMS) und einer Anwendung auf einem Handheld.

Das CMS steht dabei stellvertretend als Datenspeicher, der die strukturierten Daten speichern und notwendige Meta-Informationen bereitstellen kann. Die Handheld-Anwendung soll Content aus dem CMS anzeigen bzw. verändern können (Content-Editor). Eine Kombination dieser beiden Komponenten ist die generische Sicht auf eine Vielzahl von speziellen Kombinationen. So könnte z.B. der Inhalt einer Termindatenbank auf ein CMS abgebildet werden. Mit dem Content-Editor könnten dann Termine eingesehen bzw. verändert werden.

Wird also eine Antwort der Fragestellung für dieses Szenario gefunden, so ist eine Übertragung auf andere Szenarien denkbar.

1.4 Aufbau

Das nächste Kapitel Synchronisation geht auf den eigentlichen Vorgang des Abgleichens von Inhalten auf unterschiedlichen Systemen ein. Zunächst wird ein allgemeiner Einblick in die Problematik gegeben und nach ähnlichen

Problemklassen gesucht. Danach wird der Lösungsansatz beschrieben, der mit dem Industriestandard SyncML gegangen wird. Bis auf propäritäre Alternativen, sind mir keine weiteren bekannt, die einen Vergleich mit SyncML zuließen.

Im dritten Kapitel werden zunächst zentrale Begriffe des CMS/Handheld Szenarios definiert, um danach den Anwendungsfall beschreiben zu können. Anschließend werden die Systemkomponenten identifiziert und deren Anforderungen, die der Beantwortung der Fragestellung dienen, festgehalten.

Das vierte Kapitel zeigt einen Ansatz für eine Umsetzung der Lösung an dem Content Management System der Firma CoreMedia AG und einer Windows-CE Applikation.

Abschließend wird in einem kurzen Kapitel eine Zusammenfassung dieser Arbeit gegeben, sowie ein Ausblick in die nahe Zukunft zu diesem Thema gewagt.

Textübergreifend möchte ich noch auf den Anhang im sechsten Kapitel verweisen. Dort steht das Glossar, auf das im Laufe des Textes mit dem Symbol „**ä**“ verwiesen wird. Im Abschnitt Literatur finden sich nicht nur klassische Verweise, sondern auch Web-Ressourcen, auf die mit einem [@...] verwiesen wird.

2 Was ist Synchronisation von Content?

Der erste Teil dieses Kapitels soll klären, welcher Prozess sich hinter einer Synchronisation verbirgt. Im zweiten Abschnitt wird der Industrie-Standard SyncML vorgestellt, der versucht, diesen Prozess allgemein zu beschreiben. Für die Definition des Begriffs Content möchte ich auf den Abschnitt 3.1 verweisen.

Der Begriff Synchronisation wurde in [Val99] mit „Abstimmung von Handlungen in zeitlicher Hinsicht“ oder mit „Einschränkung von Nebenläufigkeit“ erklärt.

Aus dem Gebiet der verteilten Systeme wissen wir, dass relevante Daten auf verschiedene Geräte (Rechner) verteilt sein können und die beteiligten Prozesse ihre Entscheidungen aufgrund lokaler Information treffen. Dabei ist das Ziel der Synchronisation der logische Einbenutzerbetrieb, d.h. die Vermeidung aller Mehrbenutzeranomalien.

Bei der Synchronisation von Content müssen also die lokalen Handlungen (wie z.B. das Anlegen, Löschen oder Ändern von Inhalten) auf einem entfernten System **zeitversetzt** ausgeführt werden können. Diese Anforderung trifft wohl auch auf verteilte Datenbanken zu, wobei die Algorithmen für eine Umsetzung nicht trivial sind (siehe z.B. @DIS, Kapitel 8).

Für das Szenario in dieser Arbeit soll die „optimistische Nebenläufigkeitskontrolle“ [@KuRo] genügen. Die Handlungen auf eine Menge an Content werden dabei gesammelt und zu einem späteren Zeitpunkt übertragen und auf dem entfernten System ausgeführt. Dabei wird optimistisch angenommen, dass die Elemente der Menge an Content auf dem Zielsystem unverändert sind. Ist dies nicht der Fall, kommt es zu einem Konflikt bei der Synchronisation (mehr dazu siehe 3.2).

Die Synchronisation von „mobilem“ Content ist demnach auch nicht als eine einzige transaktionale Operation zu verstehen, sondern vielmehr als eine Reihe von transaktionalen Einzelaktionen. Durch die Konfliktlösung zum Zeitpunkt der Synchronisation können „aufgezeichnete“ Handlungen verworfen und neue hinzugefügt werden. Als Ergebnis einer Synchronisation kann jedoch festgehalten werden, dass alle beteiligten Content Elemente auf den Systemen gleich sind. Genauer definiert steht es in [@SyncML, Representation Protocol, Version 1.1]:

“In SyncML, a data collection **A** has been successfully synchronized with a data collection **B** if and only if:

There exists a bijective mapping **M**, such that for all identifiers **I**:

- if **A(I)** exists, then **B(M(I))** exists, and **B(M(I))** is equivalent to **A(I)**.
- if **A(I)** does not exist, then **B(M(I))** does not exist.
- If (**A** synchronized with **B**) is true, this does not imply that (**B** synchronized with **A**) is also true. In other words, data synchronization using SyncML does not have to be reflexive.“

2.1 Der Prozess

Der Synchronisations-Prozess soll in Abb. 2-1 visualisiert werden und gliedert sich grob in drei Bereiche:

- Sitzung eröffnen
- Zeitversetztes Nachvollziehen der Handlungen auf den Content
- Status-Meldungen auswerten

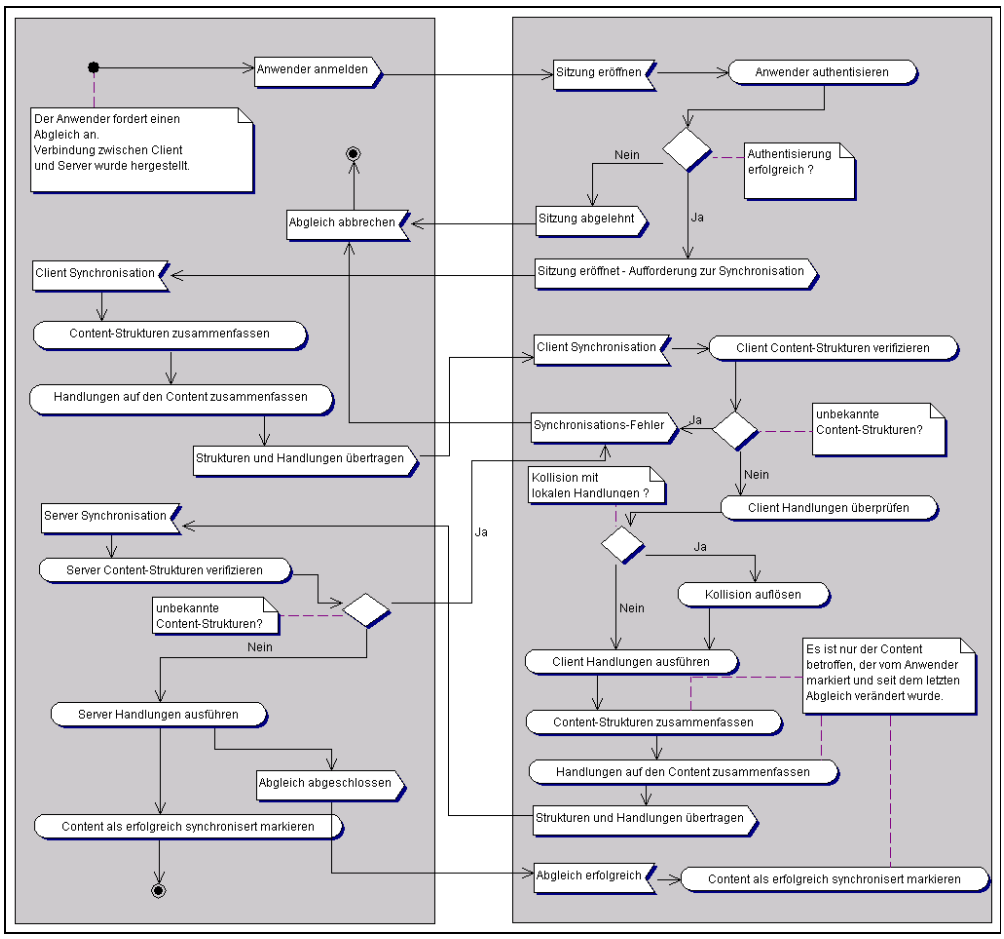

Abbildung 2-1: Prozess der Synchronisation von Content

Eine Sitzung ist genau dann erfolgreich eröffnet, wenn eine Verbindung zwischen einem Client und dem Server besteht und der Server den Anwender authentisieren konnte (Username/Passwort). Die Sitzung bildet dann den Rahmen für weitere Nachrichten zwischen Client und Server. Wird die Verbindung unterbrochen, wird auch die Sitzung ungültig.

Um den Content auf beiden Systemen eindeutig bestimmen zu können, muss es eine bijektive Abbildung geben, die eine Beziehung der Content-Identifikation zwischen den Systemen herstellt. Hier ein kleines Beispiel, wie so eine Beziehung aussehen könnte:

Client ID	Content	Server ID	Content
1	Kalendereintrag A	1234	Kalendereintrag A
2	Text	3001	Text
3	Kalendereintrag B	1249	Kalendereintrag B

Tabelle 2-1: Content IDs auf Server und Client

Client ID	Server ID
1	1234
2	3001
3	1249

Tabelle 2-2: Mapping der IDs

Die Tabelle 3-1 zeigt die IDs, die der Client bzw. der Server dem Content zugewiesen hat. Diese Zuweisung geschieht unabhängig vom anderen System. Erst die Tabelle 3-2 stellt eine Beziehung zwischen den beiden Identifikationen her, sodass bei einer Synchronisation der adressierte Content eindeutig wird.

Wie in Abschnitt 2.1 beschrieben, verfügt jeder Content über eine Struktur (seine Klasse). Sprechen der Client und der Server über die gleichen Klassen von Content, kann dieser abgeglichen werden. Es ist also erforderlich, die Beschreibung der Klassen bei einem Abgleich vorweg bekannt zu geben.

Nachdem die Handlungen auf beiden Systemen erfolgreich nachvollzogen wurden, muss der betroffene Content auf dem Client und Server für den Anwender als synchronisiert markiert werden.

Der SyncML Standard soll nun vorgestellt werden, der eine allgemeine Lösung für diesen Prozess vorsieht.

2.2 SyncML

Es wurde bereits erwähnt, dass das SyncML Konsortium aus namenhaften Herstellern besteht. Sie definieren die Ziele für diesen Protokoll-Standard für folgende Gruppen [@SYNCML]:

- Den Endanwendern soll die Möglichkeit gegeben werden, Inhalte aus unterschiedlichen Anwendungen und Geräten auszutauschen. Heutzutage müssen in der Regel die Daten „per Hand“ zwischen den Anwendungen unterschiedlicher Hersteller übertragen werden.

- Die Gerätehersteller können bei einem Standard wie SyncML ihre Produkte besser an die Bedürfnisse anpassen und optimieren, als wenn die unterschiedlichsten Anforderungen von propäritären Protokollen berücksichtig werden müssen.

- Die Service Provider sind mit SyncML in Lage, ihren Kunden eine größere Auswahl von Applikationen zur Verfügung zu stellen.

- Die Anwendungsentwickler der Applikationen müssen nur noch gegen die SyncML Schnittstellen entwickeln. Redundante Entwicklungen für mehrere Protokolle werden entfallen.

Als übergreifendes Ziel wird noch festgehalten:

„The goal of a common synchronization protocol is symmetric. It would connect any to any, over any network. That is, it would:

- Synchronize networked data with any mobile device

- Synchronize a mobile device with any networked data”

Als Lösung wurde die Spezifikation für ein SyncML Protokoll vorgestellt [@SYNCML – Technology - SyncML Spec, Version 1.1.1]. Das Protokoll definiert einen Satz von wohlgeformten Nachrichten, die zwischen den beteiligten Systemen einer Datensynchronisation ausgetauscht werden. Die Nachrichten werden mit Hilfe von äXML dargestellt.

In SyncML werden Nachrichten konzeptuell in sog. SyncML *Packages* zusammengefasst. Jedes *Package* besteht aus einer oder mehreren SyncML *Messages*. Mit dieser Aufteilung wird der begrenzten Speicherkapazität der mobilen Geräte Rechnung getragen.

Ein typisches two-way-sync Szenario wird in Abbildung 2-2 gezeigt. Hier initiiert der Anwender über den Client den Vorgang. Die ersten beiden *Packages* sind nicht abgebildet, dienen jedoch der Authentisierung und der Bekanntmachung der Content-Strukturen (Initialisierung), die für diesen Vorgang relevant sind. Im *Package#3* versendet der Client seine Operationen, worauf der Server entsprechend mit Status-Meldungen reagiert und seinerseits die Operationen übermittelt. Dies geschieht gesammelt im *Package#4*. Im letzten *Package* wird vom Server die Beziehung zwischen Client und Server Content IDs hergestellt (map acknowledgement). Wie im vorherigen Abschnitt beschrieben, ist sonst keine eindeutige Adressierung des Contents möglich.

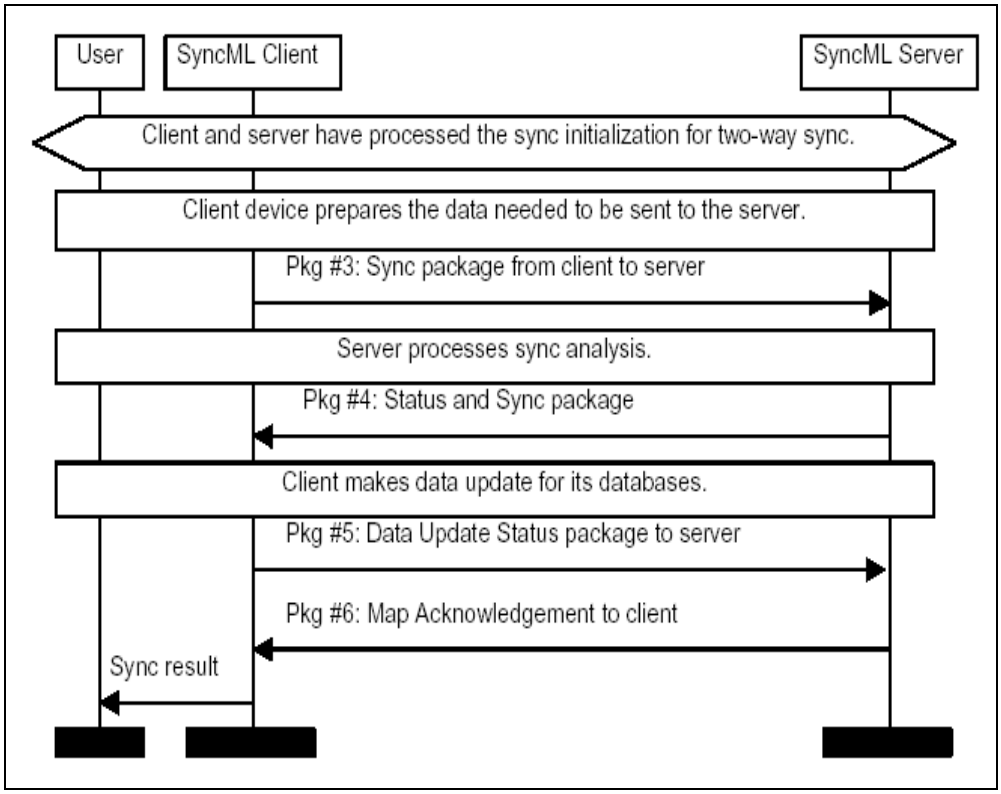

Abbildung 2-2: two-way-sync Szenario

2.2.1 Framework

SyncML bietet nicht nur ein Protokoll für die Datensynchronisation, sondern stellt ein Framework zur Verfügung, welches für Lösungen von realen Anwendungen gedacht ist (siehe Abbildung 2-3).

Die Daten der Anwendungen A und B (App A/B) werden über die lokalen Agenten ausgetauscht. Der Sync Server Agent und der Sync Client Agent übernehmen dabei die in Abschnitt 3.4 beschriebenen Aufgaben. Diese nutzen das SyncML Protokoll, um miteinander zu kommunizieren (in der Abbildung als ‚SyncML I/F' gekennzeichnet). Der Transport der SyncML Packages kann auf verschiedenen Standards basieren. Implementierungshilfen werden für HTTP, WSP und äOBEX zur Verfügung gestellt.

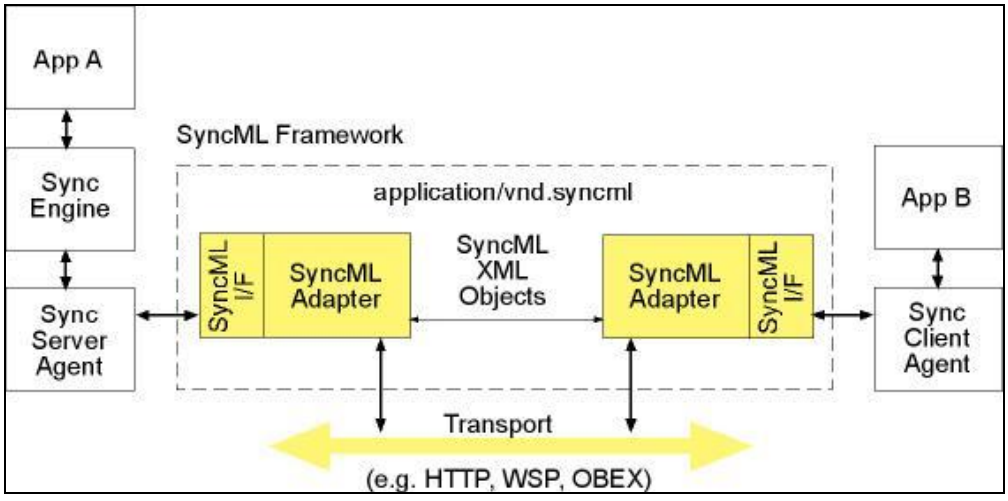

Abbildung 2-3: SyncML Framework

In dem Framework ist nur auf der Seite des Servers eine Sync Engine vorgesehen, jedoch wird in der Spezifikation ein Einsatz auf der Seite des Clients nicht ausgeschlossen. Die Sync Engine wird bei SyncML als „black-box" gesehen und wird nicht weiter spezifiziert, dient aber, wie in Abschnitt 3.4 beschrieben, als Schnittstelle zwischen Anwendung und Agent. Die Funktionalität der Engine geht aus den Anforderungen der SyncML Befehle hervor, die im Abschnitt 2.2.3 kurz vorgestellt werden.

Bevor im weiteren Verlauf auf die Protokollbestandteile eingegangen wird, soll in diesem Abschnitt der äußere Rahmen einer Synchronisationsnachricht

beschrieben werden. Es gibt eine strikte Trennung in einen Header (eingeleitet durch den Tag <SyncHdr>), der allgemeine Informationen zur verwendeten Protokollversion, zur Client- und Serveradresse und zur Authentifizierung enthält. Auf diese Informationen folgen im Datenkörper eines Nachrichtenpakets (markiert durch <SyncBody>) Angaben über die zu synchronisierenden Daten. Innerhalb dieser einzelnen Gliederungs-Tags werden weitere Tags verwendet, welche die jeweilige Funktion der übermittelten Zeichenfolge markieren. Insgesamt ergibt sich folgendes Gerüst für eine SyncML-Nachricht:

```
<SyncML>
 <SyncHdr>
              <!-- Hier stehen Informationen über die Client- und
      Serveradresse, sowie Informationen zur Protokollversion und
                                                  Authentifizierung -->
 </SyncHdr>
 <SyncBody>
    <!-- Hier stehen Informationen zu den veränderten Datensätzen
                                          beim Client und Server -->
 </SyncBody>
</SyncML>
```

2.2.2 Konzepte

In diesem Abschnitt sollen die grundlegenden Konzepte von SyncML vorgestellt werden, die für die Synchronisation von Inhalten relevant sind. Die Spezifikation kann unter [@SYNCML – Technology – SyncML Sync Protocol] heruntergeladen werden.

2.2.2.1 Anker

Um Handlungen auf Daten zwischen zwei Systemen nachvollziehen zu können, ist es notwendig zu wissen, wann auf beiden Systemen zuletzt der gleiche Zustand festgehalten wurde. Handlungen nach diesem Zeitpunkt können dann später bei einer Synchronisation zeitversetzt erfolgen. SyncML sieht dafür die Übermittlung einer eindeutigen ID (Anker), bestehen aus dem aktuellen Datum und der Uhrzeit, vor. Sendet nun ein Client dem Server diese ID, kann der Server überprüfen, ob auf ihm für diesen Anker Handlungen nachvollzogen werden können. Ist dem

Server der Anker gänzlich unbekannt, so müssen z.B. alle Inhalte vom Client zum Server übertragen werden.

In der Abbildung 2-4 wird in der ersten Sitzung der Anker *LAST* (mit dem Datum 9.9.2002, Uhrzeit 09:09:09) vom Client zum Server übertragen. Zusätzlich wird auch der nächste Anker *NEXT* übertragen, der nach der aktuellen Sitzung gültig wird und vom Server gespeichert wird.

Vor der zweiten Sitzung wurde der gesamte Speicher des Clients gelöscht, sodass der Anker *LAST* nicht mit dem Anker des Servers übereinstimmt. Der Server meldet dies dem Client und alle Inhalte müssen komplett übertragen werden.

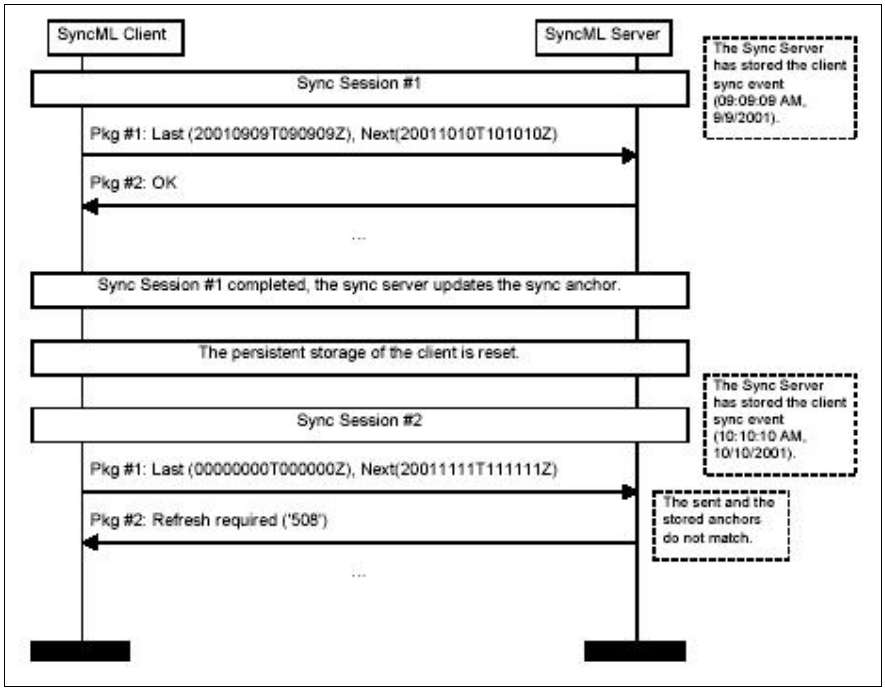

Abbildung 2-4: Anker

2.2.2.2 ID Mapping

Wie auch schon im Abschnitt 2.1 gefordert, ermöglicht SyncML jedem SyncML Knoten die getrennte Zuordnung von IDs auf Inhalte. Durch ein Mapping der IDs aller Knoten wird der Bezug zum Content hergestellt (Map Acknowledgement).

2.2.2.3 Konflikte auflösen

Treten während der Synchronisation Konflikte auf, sieht SyncML Mechanismen vor, wie entsprechende Status-Meldungen weitergereicht werden können und wie diese aufzulösen sind.

2.2.2.4 Sicherheit

SyncML unterstützt Basic und äDigest Access Authentifizierung. Dabei berücksichtigt das Protokoll innerhalb einer Sitzung mehrere Ebenen der Authentifizierung, abhängig von den jeweiligen Daten.

2.2.2.5 Adressierung

Um die Teilnehmer einer Synchronisation anzusprechen, wird innerhalb des `<SyncHdr>`-Tags ein definiertes URI-Schema des Gerätes angegeben [@SYNCML – Technology – SyncML Representation Spec]. Geräte, die permanent mit dem Internet verbunden sind, können über die bekannte URI adressiert werden, ebenso ist es denkbar, dass von einigen Geräten (z.B. Mobiltelefonen) eigene Identifikationskennzeichnungen benutzt werden (IMEI = International Mobile Equipment Identity). Ein Server, der über HTTP im Internet verfügbar ist, kann über eine URL adressiert werden und würde in einem SyncML Dokument als `Source` wie folgt angesprochen werden:

```
<Source>
 <LocURI>http://www.syncml.org/sync-server</LocURI>
</Source>
<Target>
 <LocURI>IMEI:491724343616</LocURI>
</Target>
```

Um Inhalte auf diesem Server zu adressieren, werden entsprechende URIs in den Dokumenten angegeben. So würde das folgende SyncML Fragment...

```
<Sync>...
<Target>
```

```
 <LocURI>./calendar/james_bond</LocURI>
</Target>
<Item>
 <Target>
  <LocURI>42</LocURI>
 </Target>
</Item>
...</Sync>
```

...den 42. Eintrag der Kalender-Datenbank „james_bond“ ansprechen.

2.2.2.6 Austausch der Geräteinformationen

SyncML Mandanten müssen über eventuelle Hardware Beschränkungen Auskunft geben können und ggfs. auch mit Einschränkungen anderer Mandanten umgehen können. Wenn ein Handheld z.B. nur Nachrichten mit einer maximalen Länge von 64 k Bytes verarbeiten kann, muss der Server größere SyncML *Packages* in mehrere *Messages* aufteilen können.

Zu dem Austausch der Geräteinformation, zählen bei SyncML auch Angaben zu den bekannten Content Klassen. Wenn ein Handy z.B. die Klasse „Kalendereintrag“ nicht verarbeiten kann (diese also in der Anwendung nicht definiert wurde), darf der Server Inhalte dieses Typs nicht übermitteln.

2.2.2.7 Arten der Synchronisation

SyncML unterscheidet einige Arten der Synchronisation, die sich vom Ablauf gänzlich unterscheiden können. Für diesen Text reicht die Erklärung des Begriffs „two-way sync“ aus, der schon benutzt wurde: Bei der Verwendung dieser Methode übermitteln zwei SyncML Knoten die lokalen Änderungen an ihren Inhalten, wobei der Client beginnt. Bei dieser Methode wird von dem Normalfall gesprochen, denn werden andere Wege des Datenabgleichs benötigt, liegen auch besondere Umstände vor, wie es z.B. der Fall ist, wenn die Anker vom Client und Server nicht übereinstimmen.

Für eine komplette Übersicht sei auf das Kapitel 1.3 (Sync Types) in der SyncML Protokoll Spezifikation verwiesen.

2.2.3 Protokoll-Übersicht

Der Ablauf einer Datensynchronisation besteht bei SyncML nach Abbildung 2-2 aus einem Austausch von Paketen. Diese ***Packages*** bestehen wiederum aus einer Reihe von XML Dokumenten, die durch entsprechende Document Type Definitions (DTD) definiert sind [@SYNCML – download]. Das Konsortium hat das SyncML Protokoll im wesentlichen in zwei Klassen aufgeteilt, auf die jetzt kurz eingegangen werden soll. Der volle Funktionsumfang des Protokolls kann hier nicht erklärt werden, es soll lediglich ein Gefühl vermittelt werden, welche Dokumente zwischen den Endgeräten ausgetauscht werden.

2.2.3.1 Device und Meta Daten

Eine Anforderung an SyncML war es, Datenstrukturen mit den verschiedensten Endgeräten zu synchronisieren. Die Device Information DTD definiert die Sprache für den Austausch von Geräte- und Metadaten. Angaben zum Gerät sind z.B.:

`<DevTyp>`

Spezifiziert den Typ des Geräts, dies könnte z.B. ein Mobiltelefon oder eine Workstation sein.

```
<DevTyp>workstation</DevTyp>
```

`<MaxMsgSize>`

Gibt die maximale Größe einer SyncML Message an. In unserem Beispiel setzt der Sender diese Grenze auf 64 k Bytes.

```
<MaxMsgSize>
 65536
<MaxMsgSize>
```

`<CTCap>`

Hier werden die unterstützen Content Klassen aufgeführt und beschrieben. Die Device Information DTD ermöglicht eine genaue Beschreibung der Datenstrukturen.

Es folgt ein Beispiel einer Gerätebeschreibung. Die Semantik der hier nicht beschriebenen Tags kann in der SyncML Device Information Spezifikation nachgelesen werden.

```
<DevInf xmlns='syncml:devinf'>
<VerDTD>1.1</VerDTD>
<Man>Big Factory, Ltd.</Man>
<Mod>4119</Mod>
<OEM>Jane's phones</OEM>
<FwV>2.0e</FwV>
<SwV>2.0</SwV>
<HwV>1.22I</HwV>
<DevID>1218182THD000001-2</DevID>
<DevTyp>phone</DevTyp>
<UTC/>
<SupportLargeObjs/>
<SupportNumberOfChanges/>
<DataStore>
<SourceRef>./contacts</SourceRef>
<DisplayName>Phonebook</DisplayName>
<MaxGUIDSize>32</MaxGUIDSize>
<Rx-Pref>
<CTType>text/vcard</CTType>
<VerCT>3.0</VerCT>
</Rx-Pref>
<Tx-Pref>
<CTType>text/vcard</CTType>
<VerCT>3.0</VerCT>
</Tx-Pref>
<Tx>
<CTType>text/x-vcard</CTType>
<VerCT>2.1</VerCT>
</Tx>
<DSMem>
<MaxMem>32650</MaxMem>
<MaxID>250</MaxID>
</DSMem>
<SyncCap>
<SyncType>1</SyncType>
<SyncType>7</SyncType>
</SyncCap>
</DataStore>
<CTCap>
<CTType>text/x-vcard</CTType>
<PropName>BEGIN</PropName>
<ValEnum>VCARD</ValEnum>
<PropName>END</PropName>
```

```
<ValEnum>VCARD</ValEnum>

<PropName>VERSION</PropName>
<ValEnum>2.1</ValEnum>
<PropName>N</PropName>
<PropName>TEL</PropName>
<ParamName>VOICE</ParamName>
<ParamName>FAX</ParamName>
<ParamName>CELL</ParamName>
<CTType>text/vcard</CTType>
<PropName>BEGIN</PropName>
<ValEnum>VCARD</ValEnum>
<PropName>END</PropName>
<ValEnum>VCARD</ValEnum>
<PropName>VERSION</PropName>
<ValEnum>3.0</ValEnum>
<PropName>N</PropName>
<PropName>TEL</PropName>
<ParamName>VOICE</ParamName>
<ParamName>FAX</ParamName>
<ParamName>CELL</ParamName>
</CTCap>
<Ext>
<XNam>srtmsg</XNam>
<XVal>Hello World</XVal>
</Ext>
<Ext>
<XNam>endmsg</XNam>
<XVal>Goodbye</XVal>
</Ext>
</DevInf>
```

2.2.3.2 Synchronisations-Protokoll

Der wesentliche Teil der SyncML beschreibt den eigentlichen Prozess der Daten-Synchronisation. Nach einer Verbindungsaufnahme zweier SyncML Knoten (z.B. einer Workstation und einem Handheld) wird in einem ***Package*** der Prozess initialisiert. Dazu gehört nicht nur der Austausch der o.g. Geräte- und Metadaten, sondern auch die Angabe, welche Inhalte während des Prozesses referenziert werden.

Wieder soll ein Beispiel helfen, zu zeigen, wie dies mit SyncML umgesetzt wurde:

```
<SyncML xmlns='syncml:syncml'>
<SyncHdr>
```

```
<VerDTD>1.1</VerDTD>
<VerProto>SyncML/1.1</VerProto>
<SessionID>1</SessionID>
<MsgID>1</MsgID>
<Target><LocURI>http://www.syncml.org/syncserver</LocURI></Target>
<Source><LocURI>IMEI:493005100592800</LocURI></Source>
<Cred> <!--The authentication is optional.-->
<Meta><Type xmlns='syncml:metinf'>syncml:auth-basic</Type></Meta>
<Data>QnJ1Y2UyOk9oQmVoYXZl</Data>
<!--base64 formatting of "userid:password"-->
</Cred>
<Meta> <!--The Meta is now used to indicate the maximum SyncML
message size, which client can receive.-->
<MaxMsgSize xmlns='syncml:metinf'>5000</MaxMsgSize>
</Meta>
</SyncHdr>

<SyncBody>
 <Alert>
 <CmdID>1</CmdID>
 <Data>200</Data> <!-- 200 = TWO_WAY_ALERT -->
 <Item>
  <Target><LocURI>./contacts/james_bond</LocURI></Target>
  <Source><LocURI>./dev-contacts</LocURI></Source>
  <Meta>
   <Anchor xmlns='syncml:metinf'>
    <Last>20020909T090909Z</Last>
    <Next>20020910T080000Z</Next>
   </Anchor>
  </Meta>

 </Item>
 </Alert>
</SyncBody>

</SyncML>
```

In der SyncML Nachricht leitet ein Mobiltelefon (`Source`) den Abgleich seiner lokalen Datenbank „`dev-contacts`“ mit einem Server (`Target`) ein. Mit den Angaben aus dem Anker Tag kann der Server nun entscheiden, ob die vorgeschlagene Art der Synchronisation („two-way“) zulässig ist.

Während der Synchronisation sind mit SyncML typische Befehle für das Verarbeiten von Datensätzen verfügbar. Die folgende Tabelle soll einen Überblick geben (aus Kapitel 5.5 der SyncML Representation Spec [@SYNCML – Technology – SyncML Representation Protocol]):

Befehl	„Parent“	Kurzbeschreibung
`add`	`atomic, sequence`	Einen Eintrag hinzufügen.
`atomic`	`sequence`	Alle folgenden Befehle transaktional ausführen.
`copy`	`atomic`	Kopien innerhalb einer Datenbank anfertigen.
`delete`	`atomic, sequence`	Einen Eintrag löschen.
`get`		Einen Eintrag vom Empfänger anfordern.
`map`	`atomic, sequence`	Mapping (siehe 2.2.2.2)
`put`		Einen Eintrag zum Empfänger schicken.
`replace`	`atomic, sequence`	Einen Eintrag mit einem anderen ersetzen.
`search`		Einen Eintrag beim Empfänger suchen.
`results`		Ergebnis der Anfrage nach einem search oder get
`sequence`	`atomic`	Alle folgenden Befehle in der angegebenen Reihenfolge ausführen.

Ein komplettes Bespiel für den Austausch entsprechender SyncML Dokumente vom Client zum Server und zurück findet sich im Anhang 6.1.

3 Szenario: CMS und Handheld

Das Szenario aus Abbildung 3-1 besteht aus einer Reihe von komplexen Einzelsystemen. Wichtige Begrifflichkeiten sollen nun erläutert werden, dabei wird jedoch nicht versucht, diese genau zu definieren. Tiefergehende Erklärungen sind der jeweils angegeben Literatur zu entnehmen.

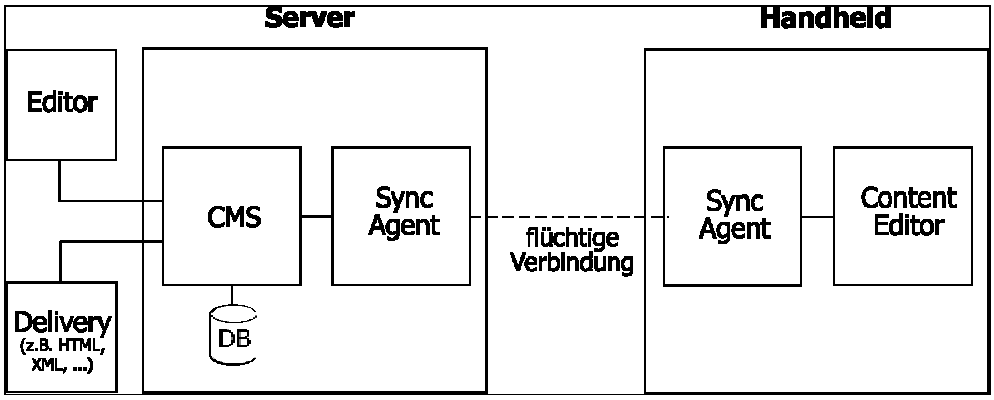

Abbildung 3-1: Ein mobiles Endgerät und ein CMS Server bilden das Szenario

Das CMS stellt in diesem Szenario die Grundlage für ein stationäres System und nimmt dabei die klassische Rolle eines Servers ein. Ein Synchronisations-Agent soll aus der Sicht des CMS Informationen bereitstellen können, welche Inhalte in dem CMS für einen bestimmten Client seit der letzten Synchronisation geändert wurden. Dieser Agent ergänzt also das CMS und stellt eine Schnittstelle nach außen zur Verfügung.

Der Content-Editor auf dem Handheld ist das mobile System und springt in die Rolle des Clients. Auch hier muss ein Synchronisations-Agent über Änderungen am Zustand des Contents Auskunft geben können. In diesem Szenario wird davon ausgegangen, dass der Client-Agent eine Verbindung mit dem Server-Agent aufnimmt und einen Abgleich initiiert.

In diesem Kapitel soll untersucht werden, welche Eigenschaften die Server- und Client-Agenten erfüllen müssen, um strukturierte Inhalte miteinander abgleichen zu können.

3.1 CMS - Wichtige Begriffe

Content kann als von Menschen erzeugte und in medienspezifischer Form aufbereitete Daten in unterschiedlichster Art beschrieben werden. Durch die Immaterialität von Content kann eine Distribution auf beliebigen Medien erfolgen. Werden elektronische Medien genutzt, vereinfacht sich die Verbreitung der Inhalte. Die Formulierung „**strukturierter Inhalt**“ ist äquivalent zu dem Begriff Content.

In der Literatur finden sich weitere Definitionen:

- „Content sind Informationen, die im Internet in unterschiedlichster Form angeboten werden, z. B. Texte, Bilder, Audio- und Videosequenzen, Grafiken sowie Fotografien.“ [@KlSe]

- „Der Begriff Content beschreibt jede Art von Informationen, die zur Ansicht, Be- und Verarbeitung zur Verfügung gestellt werden sollen.“ [ZTZ01]

- „Content bezeichnet alle dargebotenen Informationen. Ein Text im ASCII-Format gehört dazu. Genauso können digitale Bilder und Töne zum Content zählen und auch Daten in allen erdenklichen Formaten, wie z.B. PDF-Dateien und gepackte Daten wie tar- und zip-Dateien. [...] Alle seriösen Content Management Systeme bieten die Möglichkeit, Content zu strukturieren.“ [KrKo02]

Ein Content-Objekt ist also aus objektorientierter Sicht eine Instanz einer Content Klasse. Die Klasse beschreibt die Struktur des Contents und besteht aus typisierten Feldern.

So kann z.B. die Klasse „Kalendereintrag“ aus folgenden Attributen bestehen:

- Uhrzeit und Datum (Date Feld)

- Ereigniseintrag (String Feld)

- Erinnern (Boolean Feld)

- Wieviel Minuten vor dem Ereignis erinnern? (Integer Feld)

Wird bei der OO-Sicht geblieben, könnten Nachrichten an ein Content-Objekt z.B. folgende Ausgaben liefern:

- Kalendereintrag.alsXML() **à** XML-Repräsentation
- Kalendereintrag.alsHTML() **à** Layout im HTML-Format

Der Begriff **Content Management** (CM) kann wie folgt definiert werden:

CM ist…

"... eine Reihe von Prozessen und Technologien, mit denen die Erstellung und Verpackung von Inhalten ... als Teil einer dynamischen, integrierten und auf das Internet ausgerichteten Umgebung möglich ist" [Int98] oder

"... the collection of applications that help organize this information so that its owners can benefit from the flexibility of digital information without getting lost in the virtuality of this new information process" [Léc98].

Demnach kann Content Management als Prozess der Erstellung, Verwaltung und Verfügbarmachung von Inhalten in vernetzten Umgebungen definiert werden.

Unter dem Begriff **Content Management System** (CMS) werden im Allgemeinen die systemtechnischen Grundlagen für CM verstanden. In dieser Arbeit wird ein CMS als computergestütztes Archivierungs- und Verwaltungssystem für Inhaltselemente (Contents) jeglicher Art definiert, welches berechtigten Nutzern in vernetzten Umgebungen jederzeit einen einfachen und simultanen Zugriff auf abgelegte Inhalte zur anwenderspezifischen Weiterverwendung ermöglicht.

Ein CMS stellt also Meta-Informationen zu den Inhalten zur Verfügung, wobei folgende typisch und für diese Arbeit relevant sind:

Zugriffsberechtigungen: Da Zugriffe auf ein CMS eine Authentisierung voraussetzen, ist es möglich, dass nur begrenzte Aktionen auf ein Content möglich sind. So kann z.B. für einen bestimmten Personenkreis der Schreibzugriff fehlen.

Versionisierung: Wird ein Content verändert und abgespeichert erstellt das CMS eine neue Version, sodass eine Historie zu einem Content aufgebaut wird.

Konsistenzüberwachung: Steht ein Content in Verbindung mit einem Content, so dürfen diese z.B. nur zusammen gelöscht werden, um inkonsistente Zustände zu vermeiden.

Da ein CMS auch Aktionen auf seinen Content zulässt, sollen hier auch noch einige Begriffe aus diesem Bereich beschrieben werden:

Ausleihen (zurückgeben): Für ein CMS ist ein Multi-User Betrieb die Regel. Viele User greifen auf ein Repository zurück, wobei es schnell zu Überschneidungen bei den schreibenden Zugriffen kommen kann. Daher ist es notwendig, vor dem Ändern von Inhalten diese „auszuleihen“. Das Ausleihen ist dabei als eine Anfrage zum Schreiben zu verstehen. Es wird eine neue Version erstellt und zum Ändern freigegeben. Niemand kann den gleichen Inhalt noch mal ausleihen, und somit werden Überschneidungen vermieden. Wird ein ausgeliehener Inhalt zurückgegeben, kann dieser wieder von anderen Usern ausgeliehen werden. Ein Abbruch des Ausleihens verwirft die neue Version und gibt den Inhalt wieder frei.

Archivieren: Sollen einige Bereiche eines CMS aus dem Repository entfernt, die Inhalte jedoch nicht gelöscht werden, können diese als archiviert markiert werden. Dabei werden die betroffenen Daten auf Archivierungssysteme ausgelagert, die große Datenmengen aufnehmen können. Das Repository des CMS wird somit entlastet.

Speichern: Wurde ein ausgeliehener Inhalt von einem User abgespeichert, ist der geänderte Inhalt für andere User im lesenden Zugriff sichtbar. Es ist also nicht unbedingt notwendig Inhalte zurückzugeben, es wird beim Speichern nur keine neue Version angelegt.

Zerstören (löschen): Dabei sollen Inhalte gelöscht werden. Die Ausführung kann problematisch werden, wenn z.B. Referenzen von anderen Inhalten auf den zu löschenden Inhalt existieren. Manche CMS bieten daher diese Operation gar nicht erst an, sondern bieten den Umweg über einen „Papierkorb“ an.

3.2 Darstellung eines Anwendungsfalls

Um einen typischen Ablauf innerhalb des gewählten Szenarios zu zeigen, wird nun ein konkreter Anwendungsfall beschrieben werden, der in Abbildung 3-2 dargestellt wird.

Alle ä*Use Cases* beziehen sich auf den Content, der somit eindeutig im Mittelpunkt steht. Die Inhalte sollen nur von einem Akteur erstellt werden können, der direkt an dem CMS angebunden ist. Dies könnte z.B. ein Innendienst-Mitarbeiter sein, der mit einem Client des CMS arbeitet. Denkbar wäre aber auch ein automatischer Import von Content aus einem anderen Prozess.

Ein weiterer Akteur, z.B. ein Außendienst-Mitarbeiter, hat nun die Möglichkeit aus dem gesamten Repository des CMS bestimmte Inhalte zu markieren, die bei einem Abgleich auf das mobile Gerät des Außendienst-Mitarbeiters übertragen werden sollen. Solange diese Markierung aktiv ist, werden Änderungen an den Inhalten zwischen dem Server und dem Client synchronisiert.

Nach dem ersten Abgleich hat der Außendienst-Mitarbeiter also lokale Kopien der markierten Inhalte auf seinem Gerät. Diese können dann mobil bearbeitet werden, und durch einen weiteren Abgleich können die Änderungen in das CMS übernommen werden. Die markierten Inhalte sollen zwischen Handheld und CMS nach diesem Vorgang übereinstimmen.

Auf eine eventuelle Eskalation beim Abgleich wurde bis jetzt noch nicht eingegangen. Diese tritt immer dann auf, wenn der Content von beiden Systemen (CMS und Handheld) verändert wurde. Die Lösung dieses Konflikts kann unterschiedlich aussehen, so könnte z.B. beim Abgleichen ein Dialog eröffnet werden, der diese Entscheidung dem Anwender überlässt. Es sind aber auch feste Regeln denkbar, die solche Konflikte auflösen.

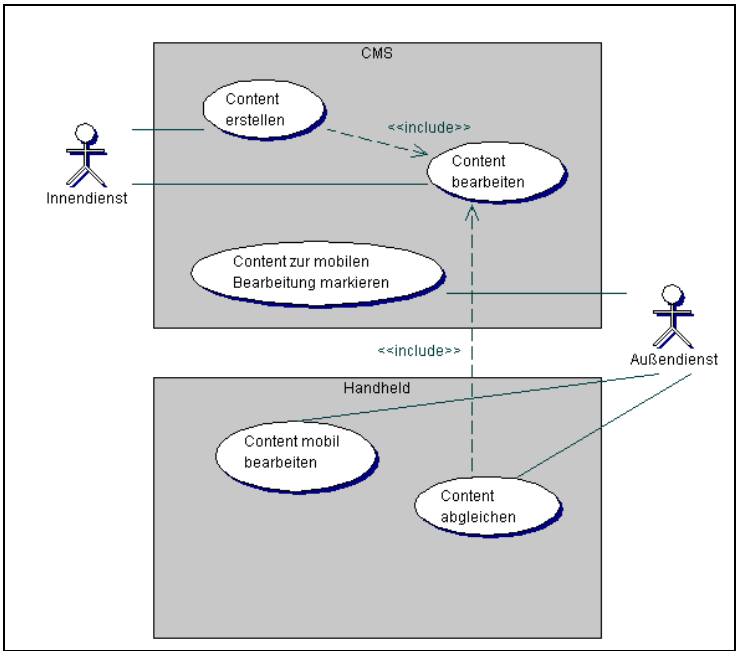

Abbildung 3-2: Content soll „offline“ bearbeitet werden

3.3 Zustände des Contents

Die Abbildung 3-3 zeigt die unterschiedlichen Zustände, die eine Content-Instanz in dem CMS annehmen kann:

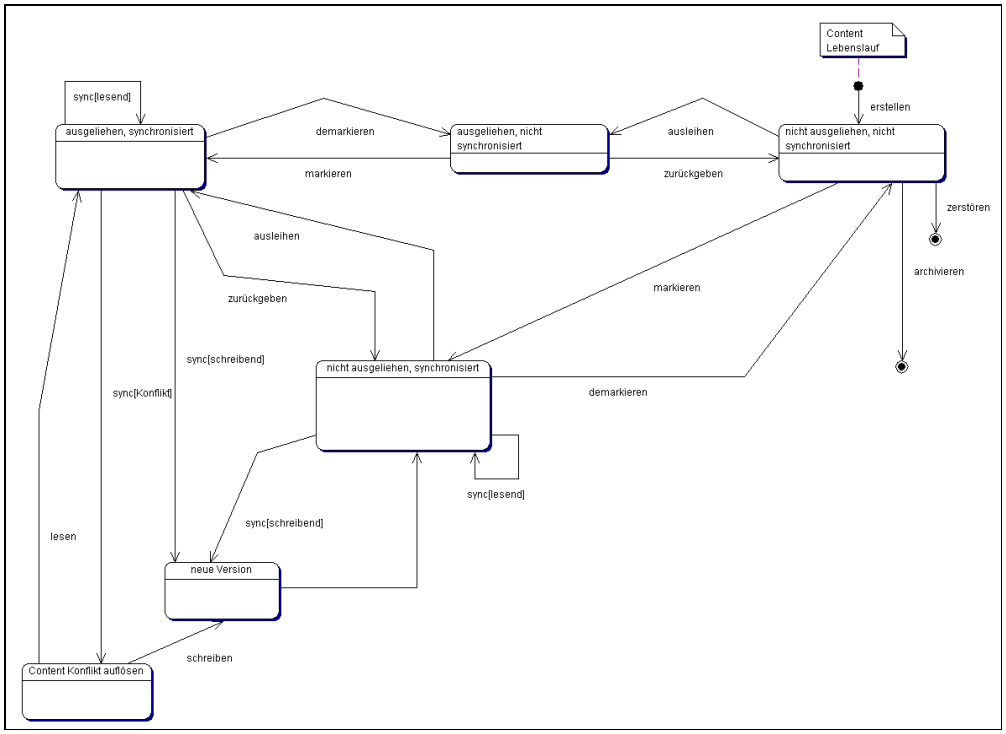

Abbildung 3-3: Content Zustände im CMS

In dem ä*Zustandsdiagramm* wurden einige Vereinbarungen eingearbeitet:

Als erstes gilt ein Content nach dem Erstellen als „nicht ausgeliehen“ und „nicht synchronisiert“. Die zweite Eigenschaft soll kennzeichnen, dass bis zu diesem Zustand ein Content noch nicht zum Abgleich markiert wurde. Außerdem darf der Content nur in diesem Zustand zerstört oder archiviert werden. Alle `sync[]` Transitionen sollen den Abgleichvorgang berücksichtigen. Die Bedingung „lesend“ soll dabei bedeuten, dass der Content vom CMS zum Handheld übertragen wird. Dies geschieht immer dann, wenn sich ein markierter Inhalt im CMS verändert hat oder bei einer möglichen Eskalation entschieden wurde, die Handheld Instanz zu verwerfen. Im Gegenteil dazu meint „schreibend“ ein Übernehmen des mobilen Contents.

3.4 Systemkomponenten

Die beiden Hauptsysteme zu dem Szenario sind auf Abbildung 3-4 dargestellt.

Der stationäre Sync-Server enthält in der obersten Schicht einen Sync-Server-Agenten, der einen sync Dienst für den Sync-Client-Agenten zur Verfügung stellt. Dieser Dienst nimmt eine Anforderung zu einem Abgleich der Inhalte an und steuert diesen Prozess. Der genaue Ablauf des Prozesses wird im Kapitel „Synchronisation“ beschrieben.

In der nächsten Schicht bedienen die Sync-Engines Anfragen der Agenten. Die Engines haben im Server und im Client die gleichen Aufgaben, müssen nur individuell implementiert werden. Folgende Anfragen der Agenten müssen bedient werden können:

Eröffnung einer Sitzung

Der Client stellt eine Verbindung zum Server über ein geeignetes Transportprotokoll (z.B. HTTP) her, um einen Abgleich einzuleiten. Dieser erste Schritt wird wiederum vom Anwender manuell eingeleitet (siehe Abbildung 2-1 „Content abgleichen“). Die Agenten teilen eine erfolgreiche Verbindungsaufnahme an die Engines mit und eröffnen damit eine Sitzung.

Authentisierung

Der Client muss sich gegenüber dem Server authentisieren. Die Engines müssen dafür entsprechende Funktionen zur Verfügung stellen.

Abgleich der Inhalte

Zunächst teilt der Client dem Server mit, welche Inhalte geändert wurden und überträgt diese an den Server. Danach genau umgekehrt. Die Engines werden dabei befragt, welche Änderungen sich an den lokalen Inhalten seit dem letzten Abgleich ergeben haben. Die Server Engine muss dabei zwischen verschiedenen Clients unterscheiden können.

Die Server-Sync-Engine greift auf das CMS zurück, um ihre Aufgaben erfüllen zu können. Die Authentisierung wird an das CMS weitergereicht und notwendige Meta-Informationen zum Content abgerufen:

- Wann wurde ein Content zuletzt geändert?
- Wann wurde ein Content mit einem bestimmten Client abgeglichen?

- Welcher Content soll für einen bestimmten Client abgeglichen werden? Und wie sieht die Struktur für diesen Content aus?

Die Client-Sync-Engine kann für die Erfüllung ihrer Aufgaben nicht auf ein CMS zurückgreifen, sondern wird z.B. auf das lokale Filesystem zurückgreifen müssen. Dabei müssen ähnliche Meta-Informationen zu Inhalten verfügbar sein:

- Wurde ein Inhalt seit dem letzten Abgleich geändert?
- Wie sieht die Struktur des Inhalts aus?

In dem Sync-Client ist noch ein Content Editor integriert, mit dem die lokalen Inhalte angesehen bzw. bearbeitet werden können.

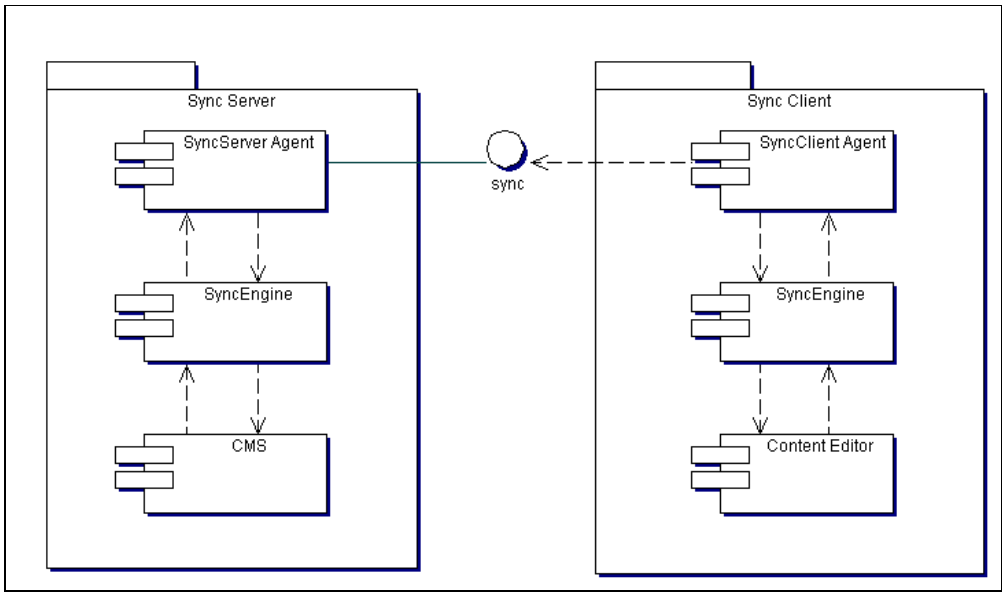

Abbildung 3-4: Der Client leitet die Aktion „sync“ ein.

3.5 Anforderungen an die Systeme

Um den Content jederzeit zwischen dem Server und den mobilen Clients abgleichen zu können, muss eine Verbindungsaufnahme über unterschiedliche Transportwege möglich sein. Die WAP Protokoll Suite sieht dafür das **ä**Wireless Session Protocol (WSP) für Mobiltelefone vor, während Handhelds durchaus HTTP unterstützen. Unabhängig vom Transport der Daten, muss der `sync` Dienst vom Server bereitgestellt werden. Mit dem Dienst wird eine Sitzung eröffnet,

protokolliert und die Content Daten übertragen. Ein Client muss sich zu Beginn einer Sitzung gegenüber dem Server authentisieren können, um einen Missbrauch zu vermeiden.

Beide Systeme müssen die Struktur des Contents beschreiben können, der für einen Abgleich vorgesehen ist. Dabei kann es vorkommen, dass ein System eine Content-Struktur nicht kennt und einen Abgleich nicht akzeptiert.

Da beide Systeme Änderungen am Content-Repository zulassen, müssen bei Bedarf diese Status-Informationen abrufbar sein:

- Wurde Content xy verändert?
- Wann wurde der Content xy verändert (Datum/Zeit)?

Der Server muss sich zusätzlich „merken“, wann ein Client den Content xy zuletzt abgeglichen hat und kann dadurch bestimmen, welcher Content sich für welchen Client verändert hat.

Ein Bespiel:

Auf dem Server wurde um 09:00 Uhr ein Text angelegt. Dieser Text wird für zwei Clients A und B zum Abgleich markiert. Um 09:15 Uhr leitet der Client A einen Abgleich ein und erhält eine lokale Kopie des Textes. Der Text wird um 09:30 Uhr auf dem Server verändert. Kurz danach leitet der Client B einen Abgleich ein.

Leiten jetzt beide Clients einen weiteren Abgleich ein, so erhält nur Client A die neue Version des Textes. Für Client B ist dies nicht notwendig, da der Zeitpunkt des letzten Abgleichs nach der Änderung des Textes liegt.

Aus diesem Beispiel wird noch eine weitere Anforderung an den Server deutlich: Was passiert, wenn der Text auf dem Server ***und*** einem Client verändert wird? Die Antwort kann sicherlich unterschiedlich ausfallen, jedoch müssen solche Eskalationen auf dem Server lösbar sein. In der Regel möchte man diese Entscheidung dem Anwender überlassen und wird somit in einem Dialog geklärt.

4 Umsetzung

Während diese Studienarbeit entstand, wurde bei der CoreMedia AG [@cm] mit der Entwicklung eines SyncML Servers begonnen. Ein Prototyp sollte das Content Management Kernsystem erweitern, um den Austausch von CoreMedia Dokumenten über SyncML mit entsprechenden Agenten zu ermöglichen.

Bei der Umsetzung des Szenarios wurde nicht nur der SyncML Server entwickelt, sondern auch ein SyncML Client, da zu dieser Zeit noch keine freie Umsetzung zur Verfügung stand. Die folgende Abbildung bietet eine Übersicht über die Architektur der beteiligten Komponenten:

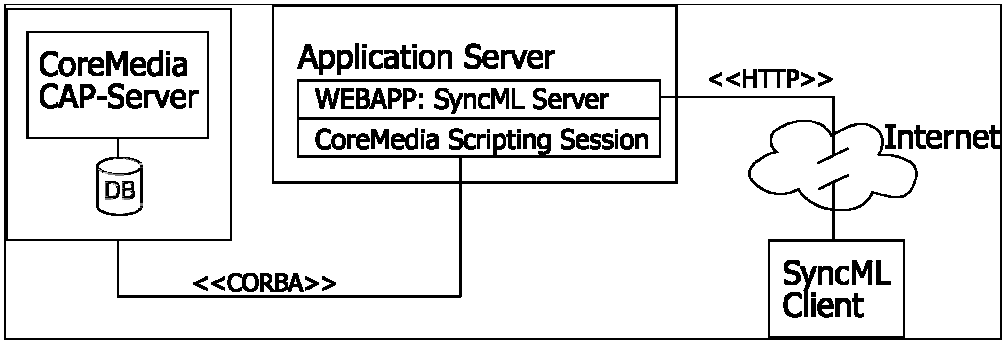

Abbildung 4-1: Architektur der Umsetzung

Der CoreMedia CAP Server musste für die Umsetzung nicht erweitert werden. Lediglich ein paar Content Klassen wurden dem Server hinzugefügt, die im Abschnitt 4.1 beschrieben werden.

Der eigentliche SyncML Server ist eine Web-Applikation, die in jedem Java Applikation Server betrieben werden kann. Der SyncML Server nutzt die CoreMedia Scripting API (eine offene Java Schnittstelle zum CoreMedia System), die über CORBA mit dem CoreMedia Server kommuniziert.

SyncML Server und Client tauschen die Daten über HTTP aus, sodass das Internet als Netzwerk dient.

Die weiteren Abschnitte gehen nun auf die einzelnen Komponenten ein und beschreiben, was für die Umsetzung zu beachten war.

4.1 CoreMedia Dokumente

Das Dokumenten-Modell aus Abbildung 4-2 zeigt u.a. die beiden Content Klassen „`Kontakt`“ und „`Termin`“. In dem CoreMedia Server können somit Kontakte und Termine eines CoreMedia Users angelegt werden, die über eine Aggregation eines `SyncList` Dokuments für den Abgleich bereitgestellt werden können.

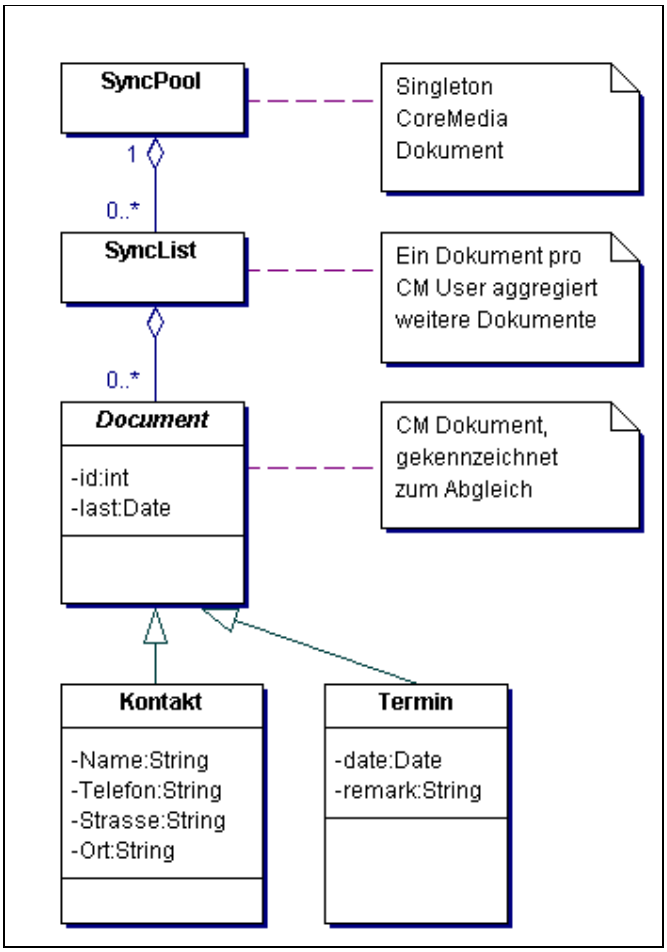

Abbildung 4-2: CoreMedia Content Modell

Beide Content Klassen erben von der Klasse `Document`, die allgemeine Daten, wie z.B. die `ID` und den Zeitpunkt der letzten Synchronisation (siehe Abschnitt Anker 2.2.2.1), bereitstellt.

4.2 SyncML Agent

Die klassische Aufteilung der Aufgaben von Server und Client treffen bei dem Modell von SyncML nur bedingt zu. Beide Seiten müssen den vollen SyncML-Funktionsumfang zur Verfügung stellen und die Befehle entsprechend verarbeiten können. In den meisten Fällen wird der Server allerdings eine komplexere persistente Schicht als der Client aufweisen. In unserem Szenario speichert der Server die Daten in einem CM System und der Client in das lokale Filesystem.

Beim Entwurf stellte sich damit der SyncML Agent als eine allgemeine Komponente für Server und Client heraus. Diese erledigt den Datentransport der SyncML Dokumente, sowie deren Erzeugung und Interpretation.

4.3 Anbindung an den CoreMedia Server

Der SyncML Server erweitert den Agenten, damit Daten in dem CMS verarbeitet werden können. In diesem Fall dient ein Java Servlet als Grundlage, da der Client über HTTP eine Verbindung aufbauen soll. Die ausgewerteten SyncML Befehle des Agenten delegiert der SyncML Server an den CoreMedia Server. Somit fungiert der SyncML Server als eine Schnittstelle zwischen den beiden Schichten.

Bei den ersten Entwurfsüberlegungen fiel schon auf, dass der SyncML Befehl „`Atomic`“ nicht umgesetzt werden kann, da das CMS keine transaktionalen Operationen bietet. Nach der SyncML Spezifikation ist der Befehl jedoch optional, sodass ein Client vorher „um Erlaubnis“ fragen muss, will er diesen Befehl nutzen.

4.4 SyncML Client

Der Client basiert auf der Personal Java Umgebung, die für diverse Handhelds verfügbar ist [@PJ]. Im Wesentlichen basiert sie auf dem Java Development Kit (JDK) in der Version 1.1. Damit die Client Anwendung auch auf dem Agenten aufsetzen konnte, war es also notwendig, dass alle Java Klassen des Agenten unter Java 1.1 liefen.

Der Prototyp kann SyncML Dokumente über HTTP verschicken und empfangen und legt diese im Filesystem ab.

5 Zusammenfassung und Ausblick

Schon während diese Arbeit entstand, hat sich viel in dem Bereich „mobiler Content“ getan. Mit der Eingliederung von SyncML in die Open Mobile Alliance (OMA, http://www.openmobilealliance.org) wurde dem Bedarf nach einem offenen Standard für die Synchronisierung Rechnung getragen.

Parallel zu dieser Entwicklung kamen auch schon erste Handys und Organizer mit SyncML-Unterstützung auf den deutschen Markt. Für eigene Software Projekte, sowohl Server- als auch Client-seitig, muss das „SyncML-Rad“ inzwischen auch nicht mehr neu erfunden werden. Inzwischen kann sich jeder bei dem Open-Source Projekt Sync4J (SyncML auf Basis von Java) engagieren und von den Erfahrungen profitieren.

Um die zentrale Fragestellung dieser Arbeit lösen zu können, müssen geeignete Dienste zur Synchronisation bereitgestellt werden. Die SyncML Initiative versucht nur das Protokoll, also die Sprache, die in dem Dienst gesprochen wird, zu vereinheitlichen. SyncML überläßt das Bereitstellen von notwendigen Meta-Informationen und das Umsetzen einer Operation der Blackbox ‚SyncEngine’ (siehe Abschnitt 2.2.1). In dem Szenario für diese Arbeit wurde die Blackbox mit einem CMS ausgefüllt.

Mit der Blackbox ‚SyncAgent’ wird die Kommunikations-Schnittstelle des eigentlichen Dienstes beschrieben und wurde in diesem Fall mit HTTP über das Internet ausgefüllt. Mobile Endgeräte erfordern aber sicherlich noch andere Schnittstellen, wie z.B. äOBEX über das GSM Funknetz.

Für eine dauerhafte Lösung sind offene Standards wie SyncML sehr hilfreich, da in diesen heterogenen Landschaften viele Schnittstellen von den unterschiedlichsten Herstellern „überwunden“ werden müssen. Daher ist es auch kritisch zu bewerten, dass die Firma Microsoft dem SyncML Gremium noch nicht beigetreten ist und die eigene Lösung ActiveSync propagiert. Der Endanwender muss sich also noch für eine Variante entscheiden. Inhalte aus „der Welt der anderen Variante“ bleiben für den mobilen Einsatz außen vor oder müssen umständlich in die neue Welt transformiert werden, sofern dies möglich ist.

In der Zukunft wird es eine Herausforderung sein, Strukturen von Inhalten allgemein zu beschreiben. In SyncML gibt es zwar schon etablierte Strukturen, wie z.B. die Visitenkarte oder den Kalendereintrag, es muss jedoch auch möglich sein, komplexere Strukturen, wie z.B. diese Studienarbeit, beschreiben zu können. Content Editoren, die auf Basis von SyncML arbeiten, könnten dann eine generische Sicht auf diesen Content haben und geeignete Werkzeuge zum Bearbeiten zur Verfügung stellen.

Eine weitere Ergänzung wird meiner Ansicht nach die Aufnahme der möglichen Operationen auf Inhalte sein. Bleiben wir bei dem Beispiel „Studienarbeit“, könnte eine Operation „erstelle vor der Synchronisation auf dem Server eine Sicherungskopie“ lauten. Genauso wäre es denkbar, ganze Workflows durch die Synchronisation auszulösen (z.B. „Studienarbeit ins Englische übersetzen“). Wie weit dies gehen kann und soll, hängt sicherlich davon ab, was mit Veränderung von Inhalten und deren Synchronisation genau zusammenhängen soll.

5.1 Allgemeingültigkeit des gewählten Szenarios

Um die Lösung aus diesem Szenario auf andere zu übertragen, müssen die Randbedingungen vergleichbar sein. Für die Synchronisation von Content mit SyncML müssen auf jeden Fall die genannten Meta-Informationen aus Abschnitt 3.4 und 3.5 erstellt und gepflegt werden. So würde z.B. das einfache Bereitstellen der zu synchronisierenden Daten auf einem Fileserver nicht reichen. Solche Systeme lassen in der Regel keine personalisierte Sicht auf die Daten zu. Fragen wie: „Haben sich (File-) Server Inhalte nach dem letzten Filetransfer zu einem bestimmten Client geändert?“, könnten ohne die Meta-Informationen nicht beantwortet werden. Der reine Fileserver als Datenträger kann auch keine Informationen zur Struktur der gespeicherten Daten zur Verfügung stellen.

Sobald eine Applikationsschicht diese Anforderung jedoch auffängt und, vergleichbar zu dem CMS Szenario, diese Informationen bereitstellt, sind die Randbedingungen vergleichbar.

Selbst wenn nicht mit SyncML synchronisiert werden soll, werden für die Dienste wohl auch bei anderen (zukünftigen) Protokollen ähnliche Meta-Informationen bereitgestellt werden müssen.

6 Anhang

6.1 SyncML : Synchronisation

Paket vom Client zum Server :

```
<SyncML>
      <SyncHdr>
            <VerDTD>1.1</VerDTD>
            <VerProto>SyncML/1.1</VerProto>
            <SessionID>1</SessionID>
            <MsgID>2</MsgID>
            <Target><LocURI>http://www.syncml.org/sync-
server</LocURI></Target>
            <Source><LocURI>IMEI:493005100592800</LocURI></Source>
      </SyncHdr>
      <SyncBody>
            <Status>
                  <CmdID>1</CmdID>

      <MsgRef>1</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
                  <TargetRef>IMEI:493005100592800</TargetRef>
                  <SourceRef> http://www.syncml.org/sync-server
</SourceRef>
                  <Data>212</Data> <!--Statuscode for OK,
authenticated for session-->
            </Status>
            <Status>
                  <CmdID>2</CmdID>

      <MsgRef>1</MsgRef><CmdRef>5</CmdRef><Cmd>Alert</Cmd>
                  <TargetRef>./dev-contacts</TargetRef>
                  <SourceRef>./contacts/james_bond</SourceRef>
                  <Data>200</Data> <!--Statuscode for Success-->
                  <Item>
                        <Data>
                              <Anchor
xmlns='syncml:metinf'><Next>200005022T093223Z </Next></Anchor>
                        </Data>
                  </Item>
            </Status>
            <Sync>
                  <CmdID>3</CmdID>

      <Target><LocURI>./contacts/james_bond</LocURI></Target>
                  <Source><LocURI>./dev-contacts</LocURI></Source>
                  <Meta>
                        <Mem xmlns='syncml:metinf'>
                              <FreeMem>8100</FreeMem>
                              <!--Free memory (bytes) in Calendar
database on a device -->
```

```
                              <FreeId>81</FreeId>
                              <!--Number of free records in
Calendar database-->
                        </Mem>
                  </Meta>
                  <Replace>
                        <CmdID>4</CmdID>
                        <Meta><Type xmlns='syncml:metinf'>text/x-
vcard</Type></Meta>
                        <Item>

      <Source><LocURI>1012</LocURI></Source>
                              <Data><!--The vCard data would be
placed here.--></Data>
                        </Item>
                  </Replace>
            </Sync>
            <Final/>
      </SyncBody>
</SyncML>
```

Paket vom Server zum Client:

```
<SyncML>
      <SyncHdr>
            <VerDTD>1.1</VerDTD>
            <VerProto>SyncML/1.1</VerProto>
            <SessionID>1</SessionID>
            <MsgID>2</MsgID>
            <Target><LocURI>IMEI:493005100592800</LocURI></Target>
            <Source><LocURI>http://www.syncml.org/sync-
server</LocURI></Source>
      </SyncHdr>
      <SyncBody>
            <Status>
                  <CmdID>1</CmdID>

      <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
                  <TargetRef>http://www.syncml.org/sync-
server</TargetRef>
                  <SourceRef>IMEI:493005100592800</SourceRef>
                  <Data>200</Data>
            </Status>
            <Status><!--This is a status for the client
modifications to the server.-->
                  <CmdID>2</CmdID>

      <MsgRef>2</MsgRef><CmdRef>3</CmdRef><Cmd>Sync</Cmd>
                  <TargetRef>./contacts/james_bond</TargetRef>
                  <SourceRef>./dev-contacts</SourceRef>
                  <Data>200</Data> <!--Statuscode for Success-->
            </Status>
```

```
            <Status>
                  <CmdID>3</CmdID>

      <MsgRef>2</MsgRef><CmdRef>4</CmdRef><Cmd>Replace</Cmd>
                  <SourceRef>1012</SourceRef>
                  <Data>200</Data> <!--Statuscode for Success-->
            </Status>
            <Sync>
                  <CmdID>4</CmdID>
                  <Target><LocURI>./dev-contacts</LocURI></Target>

      <Source><LocURI>./contacts/james_bond</LocURI></Source>
                  <Replace>
                        <CmdID>5</CmdID>
                        <Meta><type xmlns='syncml:metinf'>text/x-
vcard</type></Meta>
                        <Item>

      <Target><LocURI>1023</LocURI></Target>
                              <Data><!--The vCard data would be
placed here.--></Data>
                        </Item>
                  </Replace>
                  <Add>
                        <CmdID>6</CmdID>
                        <Meta><type xmlns='syncml:metinf'>text/x-
vcard</type></Meta>
                        <Item>

      <Source><LocURI>10536681</LocURI></Source>
                              <Data><!--The vCard data would be
placed here.--></Data>
                        </Item>
                  </Add>
            </Sync>
            <Final/>
      </SyncBody>
</SyncML>
```

6.2 Glossar

Use Case

Ein Anwendungsfall ist die Beschreibung einer typischen Interaktion zwischen dem Anwender und dem System, d.h. es stellt das externe Systemverhalten in einer begrenzten Arbeitssituation aus der Sicht des Anwenders dar [Oes97].

Digest Access

Diese Erweiterung von HTTP ermöglicht eine sichere Authentifizierung.

Link zur RFC 2069: http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2069.html

OBEX

„Provides object exchange services similar to HTTP, but is more flexible, compact and efficient for embedded devices. It supports the basic need for devices to send arbitrary data objects to each other in a logically straight forward manner. OBEX consists of a model for representing objects and a protocol to support communication between devices.” - http://www.irda.org/

WSP

“The Session layer protocol family in the WAP architecture is called the Wireless Session Protocol, WSP. WSP provides the upper-level application layer of WAP with a consistent interface for two session services. The first is a connection-mode service that operates above a transaction layer protocol WTP, and the second is a connectionless service that operates above a secure or nonsecure datagram transport service.” - http://www.wapforum.org/

XML

Extensible Markup Language, siehe http://www.w3c.org/XML/

Zustandsdiagramm

Ein Zustandsdiagramm zeigt eine Folge von Zuständen, die ein Objekt im Laufe seines Lebens einnehmen kann und aufgrund welcher Stimuli Zustandsänderungen stattfinden [Oes97].

6.3 Abbildungsverzeichnis

6.4 Literatur

6.4.1 Bücher und Artikel

[BaWe99] Gabriele Bannert und Martin Weitzel, Objektorientierter Softwareentwurf mit UML, Addison-Wesley, 1999

[FeZu01] Yu Feng, Dr. Jun Zhu, Wireless Java Programming with J2ME, SAMS Publishing, 2001

[GHJV95] Erich Gamma, Richard Helm, Ralh Johnson, John Vlissides; Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

[Int98] Interleaf, XML in der Praxis: Unternehmensübergreifende Vorteile durch Enterprise Content Management. White Paper, Eschborn 1998

[KrKo02] Krüger, Kopp; Web Content managen, Markt & Technik Verlag, Februar 2002

[Léc98] Lécluse, C. SGML Databases & Content Management for the Web. In: Graphic Communications Association (Hrsg.), Conference Proceedings SGML/XML Europe '98. sofiac, Paris 1998, S. 367-372

[Oes97] Bernd Oestereich, Objektorientierte Softwareentwicklung mit der UML, Oldenbourg Verlag, 3. Auflage 1997

[Se92] Robert Sedgewick, Algorithmen in C, Addison-Wesley, 1992

[Val99] Rüdiger Valk, Modelle für Rechensysteme, Universität Hamburg, FB Informatik, August 1999

[ZTZ01] Zschau, Traub, Zahradka; Web Content Management, Galileo Business, 2.Auflage November 2001

6.4.2 WWW Seiten

(zuletzt besucht am 13. Juli 2003)

[@SYNCML] Web Site of the SyncML Initiative
http://www.openmobilealliance.org/syncml/

[@DIS] "Datenbanken und Informationssysteme (DIS)"
http://vsis-www.informatik.uni-hamburg.de/teaching/ss-03/dis/

[@KlSe] Klinger, C., Segert, R. Das Online ABC.
http://www.webwunder.de/asp/abc.asp?abfrage=Content

[@KuRo] On Optimistic Methods for Concurrency Control
http://swig.stanford.edu/pub/summaries/database/index.html

[@PJ] PersonalJava Application Environment
http://java.sun.com/products/personaljava/index.html

[@HPJ] HP Jornada Homepage,
http://www.hewlett-packard.de/mobile/jornada/

[@J] Java Homepage,
http://java.sun.com/

[@CM] CoreMedia AG,
http://www.coremedia.com