Freie Universität Berlin
Institut für Informatik

Seminar XML Technologien
Wintersemester 2002/03

„SyncML“

Moritz Blöcker
bloecker@inf.fu-berlin.de

Andreas Wendt
andreas@wendtnet.de

Inhaltsverzeichnis:

1. Allgemeine Einführung:

Die Entwicklung unserer modernen Gesellschaft verlangt mehr und mehr Mobilität von uns, was als deutlicher Trend auch in der Entwicklung immer mobilerer Endgeräte zu erkennen ist.
Die Aufgaben unserer Arbeit werden zunehmend von Mobilität geprägt, die reine Arbeit am Arbeitsplatz-PC beginnt zu verschwinden.

Aus diesem Grund entsteht das Bedürfnis die persönlichen Daten offline wie online jederzeit mit sich führen zu können. Modere Handys, Notebooks und PDA's machen diesen Trend sichtbar.

Ein typisches Szenario sind beispielsweise Handelsreisende, die ihre Arbeit fast ausschließlich im Auto,Flugzeug oder der Bahn verrichten. Dennoch sollten sie in der Lage sein ihr Büro morgens zu verlassen und alle für den Tag relevanten Daten sollten sich im Speicher des PDA's befinden, sodaß eMails unterwegs gelesen und beantwortet, Termine verwaltet werden können und aktuelle Telefonnummern im Speicher der Handys zur Verfügung stehen.
Zu diesem Zwecke sollte das Netzwerk im Büro die nötigen Daten auf die Endgeräte übertragen, wobei beide Komponenten in der Lage sein müssen zu erfahren was einerseits an Daten zur Verfügung steht und was andererseits an Daten gespeichert werden kann.

Mit der Vielzahl von mobilen Geräten tritt als wichtiges Problem die Aktualität der Daten auf den einzelnen Geräten zu Tage, was die Lösung in einer möglichst standardisierten Synchronisationsmöglichkeit sucht, welche idealerweise ohne viel Konvertierungsaufwand sowohl von großen Firmennetzwerken, als auch von kleinsten Handys unterstützt werden sollte.

Aus diesen Überlegungen heraus haben sich führende Hersteller solcher Geräte, darunter renomierte Unternehmen wie etwa IBM, Nokia, Ericsson, Palm und Motorola, zusammengeschlossen, um einen gemeinsamen Standard zum Austausch von Daten zwischen ihren Endgeräten zu entwickeln.

Vorläufiges Ergebnis dieser Entwicklung stellt **„SyncML“** dar, welches auf den aus dem Internetbereich bekannten XML-Standards basiert und somit einen hohen Grad an Kompatibilität gewährleisten soll.

Im Folgenden werden wir dem interessieren Leser einen Einblick in die Struktur der Protokolle geben, die SyncML bilden und anhand von einfachen Abläufen und Beispielen den Abläuf der Synchronisation darstellen.

2. Das Syncml-Sync-Protocol:

Allgemein:

In dem SyncML Protokoll gibt es den *SyncML Client*, der meisstens auf einem mobilen Gerät installiert ist, und den *SyncML Server*. Der Klient startet für gewöhnlich eine SyncML Synchronisation, allerdings kann dies durchaus auch der Server initiieren.
Es gibt verschiedene Arten der Synchronisation, die in den folgenden Kapiteln genauer erläutert, und hier nur in kürze vorgestellt werden.

Two-way sync:
Das ist die Art der Synchronisation die für gewöhnlich vorgenommen wird.
Hierbei werden zuerst die Daten von dem Client zu dem Server geschickt, und dann in umgekehrter Richtung.
Slow sync:
Hierbei handelt es sich um eine Form der zwei Wege Synchronisation, bei der die Daten Feld für Feld verglichen wird. In der Praxis heisst das, dass der Klient erst alle Daten an den Server schickt, dort dann Feld für Feld verglichen werden und dann die jeweils aktuellen Daten zurück an den Klienten geschickt werden.
One-way sync from Client only:
Bei dieser Synchronisationsart teilt nur der Klient dem Server die Änderung an seinen Daten mit.
Refresh sync from Client only:
Hier sendet der Klient alle seine Daten an den Server. Dieser soll seine Daten dann mit denen des Klienten ersetzen.
One-way sync from Server only:
Bei dieser Synchronisationsart teilt nur der Server dem Klient die Änderung an seinen Daten mit.
Refresh sync from Server only:
Hier sendet der Server alle seine Daten an den Klienten. Dieser soll seine Daten dann mit denen des Servers ersetzen.
Server Alertet sync:
Bei dieser Form der Synchronisation wird der Klient mit Hilfe eines *Alert* Kommandos vom Server informiert, das er eine Synchronisation starten soll. Die Form der Synchronisation wird in dem *Alert* angegeben.

Allgemeiner Protokollablauf:

Austausch von Log Informationen:

Dies Protokoll erfordert es, dass sowohl Klient als auch Server fähig sind alle Änderungen an ihren Daten seit der letzten Synchronisation mitzuprotokollieren. Die Arten der Modifikationen können zum Beispiel löschen, hinzufügern oder ersetzen sein. SyncML sagt nichts darüber aus, wie der Klient diese Modifikationen protokolliert. Wenn die Synchronisation anfängt muss das Gerät (sowohl Klient als auch Server) dem Gegenüber mitteilen welche Daten wie verändert wurden. Um die Daten zu identifizieren gibt es eindeutige Bezeichner (siehe auch *Map* oder *Mapping table*), un um die Art der Modifikation anzuzeigen werden unterschiedliche Kommandos benutzt (z.B. *Add*, *Delete* oder *Replace*).

Wenn sich ein Gerät mit mehreren anderen Geräten synchronisiert, muss es in der Lage sein, für jedes einzelne dieser Geräte die Änderungen seit der letzten Synchronisation mit dem jeweiligen Gerät mitzuprotokollieren.

Synchronisationsanker

Um zu überprüfen, dass die keine Synchronisationspunkte durch Datenverlust vergessen wurden, werdenm am Anfang jeder Synchronisation sogennant Synchronisationsanker oder auch *sync anchor* verwendet. Es gibt für jeden Synchronisationspartner zwei Anker. Den *Last* und den *Next* Anker. Der *Last* Anker gibt den Zeitpunkt der letzten Synchronisation aus der Sicht des sendenen Gerätes an und der *Next* Anker gibt den Zeitpunkt der aktuellen Synchronisation an.

Beide Geräte senden also ihre zwei Anker mit den *Meta* Element des *Alert* Kommandos zueinander. Das empfangende Gerät muss jeweils den *Next* Anker in dem *Status* des *Alert* Kommandos zurücksenden.
Der genau Umgang mit den Synchronisationsankern ist Implementationabhängig, aber um die Anker nutzen zu können müssen die *Next* Synchronisationsanker bis zur nächsten Synchronisation gespeichert werden (wo sie dann als *Last* Synchronisationsanker verwendet werden).
Wenn das Gerät den Next Synchronisationsanker speichert, kann es dann während der nächsten Synchronisation anhand dieser Anker überprüfen ob es zu Fehlern zwischen den Synchronisationen gekommen ist (bei Gleichheit der Anker kam es zu keinen Fehlern). Sollte ein Fehler festgestellt werden, kann das Gerät eine spezielle Form der Synchronisation von dem anderen Gerät verlangen (den *slow sync*).
Die gespeicherten Synchronisationsanker dürfen nicht aktualisiert werden, bis die Synchronisation abgeschlossen wurde.
Die Synchronisation erst abgeschlossen wenn ein Gerät keine SyncML Nachrichten mehr senden möchte, keine Nachrichten mehr erwartet, und die Synchronisation auf Kommandoebende erfolgreich verlief (nur 200er *Status* von den SyncML Kommandos zurückgegeben wurde).
Auch auf Transportebene (eine Schicht unter SyncML) muss die Kommunikation ordnugsgemäß abgebaut sein, bevor die Synchronisation als beendet angesehen werden kann. Sollte die Kommunikation im Sinne der Transportebene nicht ordnugsgemäss abgebaut worden sein, darf die Synchronisation nicht als erfolgreich angesehen werden, und die Synchronisationsanker dürfen nicht aktualisiert werden.

Beispiel: Synchronisationsanker für Datenelemente

Das Protokoll bietet nicht die Funktionalität um Synchronisationsanker die sich auf einzelne Datenelemente beziehen zu übertragen. Wenn dies gewünscht wird, muss sie von den Datenelementen selbst unterstützt werden. Ein Beispiel hierfür ist die *Sequence Number* Eigenschaft von dem Typ *vCalendar*.

Bezeichner und Abbildungen zwischen Bezeichnern

Diese Protokoll basiert auf der Idee, das Klient und Server nicht die selben Bezeichner für einen Datensatz haben. Das heisst, dass jeder seinen eigenen Bezeichner für ein und denselben Datensatz hat. Dieses kommt zur Anwendung, wenn der Klient aufgrund seiner Architektur nicht die Bezeichnerlänge bereitstellen kann, die der Server verwendet. In diesem Fall ist es die Aufgabe des Servers sich die Abbildung zwischen den Bezeichnern zu merken. Dies geschieht mit Hilfe des *Mapping tables.* Hier werden die beiden Bezeichner paarweise gespeichert. Einmal der Bezeichner des Servers (*GUID*) und daneben die des Klienten (*LUID*). Die *LUID* wird immer vom Klienten zugewiesen, selbst wenn der Server einen Datensatz zu der Datenbank des Klienten hinzufügt. In diesem Fall teilt der Klient danach dem Server die neue *LUID* mittels des *Map* Kommandos mit. Wenn der Server nun einen neuen Datensatz bei dem Klienten hinzufügen will, sendet er seine *GUID* mit. Sollte diese *GUID* grösser sein als die maximale zugelassene Bezeichnergrösse des Klienten wiest dieser dem Datensatz einen neuen Bezeichner zu (die *LUID*). Dieser wird dann an den Server zurückgesendet (mit *Map*), worauf hin dieser einen neuen Eintrag in seiner *Mapping table* macht. Sollte der Server diesen Datensatz irgendwann nochmal verändern wollen, muss er die *LUID* des Klienten angeben, nicht etwa seine *GUID*. Wenn der Klient den Server über Änderungen an diesem Datensatz informaieren möchte, geschieht dies auch mittels der *LUID*, da nur der Server sich die Abildungen *GUID* zu *LUID* merkt.

Puffern von Map Operationen

Nachdem der SyncML Server mindestens eine Hinzufügungen bei dem SyncML Klienten beantragt hat, und dieser dies auf seiner Datenbank auch ausgeführt hat, stallt der Klient die *LUID* für die
neuen Datensätze bereit. Nun hat er die Möglichkeit die *Map* Kommandos zu Puffern. Der Klient speichert diese Kommandos zwischen, wenn der Server angegeben hat, das er keine Antwort auf seine Sync Nachricht (in der die Hinzufügungen angefragt wurden) braucht. Der Klient könnte aber, obwohl der Server keine Antworten wünscht, trozdem sofort die neuen *LUID* an den Server senden.

Bei Zwischenspeicherung der *Map* Zuweisungen, werden diese dann am Anfang der folgenden Synchronisationssitzung an den Server gesannt (im dritten Packet das der Klient zum Server sendet). Das heisst, dass der Server diese *Map* Operationen empfängt bevor er überhaupt irgendwelche Aktuallisierungen, die sich auf die hinzugefügten Daten beziehen, zu dem Klienten senden kann.
Sollte der Server Kontrolle über das Transportprotokoll haben, muss er immer auf einer Antwort besetehen. Also darf der Server auch nicht die Verbindung beenden, bevor er nicht alle Antworten von dem Klienten bekommen hat.

Lösen von Konflikten

Konflikte entstehen dadurch, dass zwischen zwei Synchronisationssitzung verschiedene Änderungen an dem selben Datensatz sowohl auf dem Klient wie auch auf dem Server vorgenommen wurden. Diese Konflikte werden im allgemeinen von der *Sync Engine* auf dem Server gelöst. Das Protokoll bietet die Möglichkeit, den Klienten über die Form der Konfliktlösung zu unterrichten.

Obwohl der Server im allgemeinen die Konflikte auflöst, ist nicht ausgeschlossen, das dem Klienten diese Aufgabe übertragen wird. In diesem Fall würde der Server eine Benachrichtigung schicken, dass ein Konflikt stattgefunden hat, den der Klient dann auflöst. Eine andere Möglichkeit wäre es den Benutzer des Klienten mit Hilfe des *Exec* Kommandos auzufordern den Konflikt zu lösen.
Es gibt mehrere Taktiken um Konflikte zu lösen. Um zwischen diesen Taktiken zu wählen, stellt das *SyncML Representation Protocol* Statusmeldung zur Verfügung um zwischen den gebräuchlichsten wählen.
Wie diese Aussehen, steht leider nicht in der Beschreibung des Protokolls.

Sicherheit

Diese Protokoll erfordert die Bereitstellung der grundlegenden Authentikation und der *MD5 digest acces* Authentifikation auf der Serverebene (hier: in dem *SyncHdr*). Sowohl der Server als auch der Klient können eine Authentifikation fordern, und das Gerät das diese Forderung erhält, muss in der Lage sein mit den erforderlichen Autorisationsdaten zu antworten. Die genaue Vorgehensweise wird später erläutert.

Adressierung

Um Geräte oder Dienste im SyncML *SyncHdr* zu adressieren, wird das URI Schema benutzt das im Repräsentationsprotokoll definiert ist. Geräte die an das Internet angeschlossen sind, könnten wie wie folgt aussehen:

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

Geräte, die nur über einen kurzen Zeitraum verbunden sind wie beispielsweise Mobiltelefone könnten folgende Addressierung benutzen:

```
<Source>
        <LocURI>IMEI:493005100592800</LocURI>
</Source>
```

Das Adressierungsschema auf der Transportebene passt nicht mit diesem Adressierungsschema zusammen.

Verwendung von RespURI und Umleitungs Statusmeldungen

Das Protokoll fordert, dass das Gerät ein Empfangen des RespURI Elementes unterstützt (wie im *SyncML Representation Protocol* spezifiziert), aber es ist nicht notwendig das die Statusmeldungen die eine Umleitung einleiten (*3xx*) unterstützt werden.

Datenbankadressierung

Die Datenbankadressierung erfolgt mit dem URI Schema das im Repräsentationsprotokoll definiert ist. Absolute oder relative URIs können für die Server und Klient Datenbanken benutzt werden.

Hier ist ein Beispiel das beide Fälle erläutert:
absolute Adressierung:

```
<Sync>
        ...
        <Target>
        <LocURI>http://www.syncml.org/sync-server/calendar/james_bond</LocURI>
        </Target>
        ...
</Sync>
```

relative Adressierung:

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

Adressierung von Datensätzen

Die Datensatzadressierung erfolgt mit dem URI Schema das im Repräsentationsprotokoll definiert ist. Die Adressierung erfolgt mit Hilfe der *LUID* oder *GUID.* Ein Beispiel für eine relative Adressierung wäre:

```
<Item>
        ...
        <Source>
                <LocURI>101</LocURI>
        </Source>
        ...
</Item>
```

Austausch von Geräteeigenschaften

Der Austausch über die Eigenschaften der Geräte erfolgt während der initialisaion. Dieser Austausch kann sowohl vom Server als auch vom Klient gefordert werden.

Der Klient muss die Informationen über seine Eigenschaften bei der ersten Synchronisation mit dem Server senden, oder wenn sich seine statischen Eigenschaften verändert haben. Der Klient muss auch fähig sein jederzeit auf entsprechende Anfragen des Servers zu reagiern. Er sollte desweiteren fähig sein die Informationen über die Eigenschaften zu Empfangen (und verarbeiten).

Der Server muss die Daten an den Klienten senden können, wenn dieser es verlangt. Der Server muss auch die Fähigkeit haben, die Informationen über den Klient zu empfangen und zu verarbeiten wenn dieser sie sendet.

Speicher Managment der Geräte

Das Protokoll bietet mit den Meta Informationen die Möglichkeit, die dynamischen Speicherresourcen für Datenbanken auf einem Gerät, die Grösse des Festpeichers eines Gerätes an den Synchronisationspartner und die Grösse des freien Speichers zu übertragen. Diese dynamischen Kapazitäten können zu jeder Zeit während der Synchronisation übertragen werden.
Obwohl es optional ist die Grösse des Festspeichers zu übertragen, sollte der Klient dies tun, während es dem Server freigestellt ist.
Die Benutzung von unterschiedlichen Darstellungsformen der Speicherresourcen hängt von dem Modell des Festspeichers auf dem Gerät ab. Hier ist eine Beispiel, wie die dynamischen Speicherresourcen einer Datenbank dargestellt werden:

```
<Sync>
        <CmdID>1</CmdID>
        <Target><LocURI>./calendar/james_bond</LocURI></Target>
        <Source><LocURI>./dev-calendar</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>
        ...
</Sync>
```

Mehrere Nachrichten pro Packet

Wenn ein Packet zu gross ist, um es in einer SyncML Nachricht zu übertragen, ist es möglich, es aufzuteilen und in mehreren Nachrichten zu übertragen. Dies könnte passieren, weil das Transportprotokoll eventuell keine grösseren Übertragungseinheiten zulässt.
Wenn ein Packet in mehreren Nachrichten übertragen wird, muss das letzte das *Final* Element beinhalten. Alle anderen Nachrichten dieses Packetes dürfen das *Final* Element nicht enthalten. Das *Final* Element darf nur gesendet werden, wenn alle nötigen Kommandos die zu diesem Packet gehören gesendet wurden.Das *Final* Element darf nicht gesendet werden, wenn der Kommunikationspartner sein vorheriges Packet noch nicht geschlossen hat (da er in dem aktuellen Packet eventuell noch Kommandos aus dem vorherigen Packet des Partners bestätigen muss).

Behandlung großer Nachrichten

Sollte eine Datenobjekt zu groß sein, um innnerhalb einer Nachricht geschickt zu werden, kann es in mehrere Protionen aufgeteilt werden. Dieses geschieht, indem man in das *Data* Element ein *</MoreData>* Element hinzufügt. Damit wird dem Empfänger angezeigt, das weitere Teile des Datenobjekts folgen. Dies muß der Empfänger dem Sender mittels einer (*213*) *Chunked item accepted and buffered* Statusmeldung quittieren. Desweiteren muss er mittels eines *Alert* Kommandos nach weiteren Nachrichten fragen.
Ein Kommando, das ein auf mehrere Nachrichten verteiltes Objekt enthält muss als atomar angesehen werden. Desweiteren dürfen keine neuen Datenobjekte in die Datenbank hinzugefügt werde, solange das aktuelle Datenobjekt nicht abgeschlossen ist.
Innerhalb des *Data* Elementes muss die Größe des Datenobjektes angegeben sein (mit *<Size>*). Sollte die angegebene größe nicht mit der tatsächlichen übereinstimmen, darf das Datenobjekt nicht in die Datenbank hinzugefügt werden.

Synchronisation ohne eigene Initialisierungsphase

Die Synchronisation kann ohne extra Initialisierungsphase gestartet werden. Hierbei werden einfach die Packete 1-3 in einem Packet untergebracht. Dies hat zur folge, das sowohl der Klient als auch der Server bereit seinen müssen, alle (zu synchronisierenden) Daten des ersten Packetes nochmal zu senden, sollte der Gegenüber einen *SlowSync* fordern.

Busy Signal

Der *Busy* Status gibt dem Server die Möglichkeit, dem Klienten mitzuteilen, daß er nicht in einem angemessenen Zeitraum auf die Anfragen/Modifikationen des Klienten antworten kann.
In diesem Fall kann der Klient entweder später nach den Ergebnissen der Kommados fragen (mittels eines *Result Alert*s), oder aber die Synchronisation erneut starten. In letzterem Fall der der *Last* Sync Anker nicht aktualisiert (mit *Next* überschrieben) werden.

Authentifizierung

Es gibt zwei Formen der Authentifikation, die Basis Authentifikation oder die MD5 Authentifikation. Die Authentifikation kann in beide Richtungen erfolgen, d.h. sowohl Klient als auch Server können eine Authentifizierung des andern verlangen. Bei der Basis Authentifikation handelt es sich um das encodete (meist Base64) Wertepaar „Benutzername:Passwort“ während es bei der MD5 Authentifikation das Tripple „Benutzername:Passwort:Nonce“ auf den MD5 Algorithmus angewand wird, und das Ergebnis dann encoded wird.

Aufforderung zur Authentifizierung

Wenn in einer Antwort auf eine Anfrage die (*401*) *Unauthorized* oder (*407*) *Authentication required* Statusmeldung erscheint, bedeuted dies, das man für diese Operation eine Authentifikation (mit den nötigen Rechten) benötigt. In diesem Fall wird die Antwort ein *Chal* Element enthalten.
Das *Chal* Element enthält eine Authentifikationsaufforderunng die der angefragten Resource entpricht. Das Gerät, das die Anfrage gestellt hat kann mit einem entsprechendem *Cred* Element (Authentifikationsdaten) antworten. Sollte das Gerät es bereits getan haben un daraufhin die *401* Statusmeldung erhalten haben, bedeutet dies, daß die Authentifikation fehlgeschlagen ist.
Bei einer erfolgreichen Authentifikation wird die Statusmeldung (*212*) *Authentication accepted*.
Sollte es sich hierbei um eine MD5 Authentifikation gehandelt haben, wird jetzt das nächte *Nonce* übertragen. Dieses wird dann bei der nächsten Authentifikation benutzt.
Authentifikation kann auf Snychronisationsebene (im *SyncHdr* Element), auf Datenbankebene (im *Sync* Element).

Beispiel (Basis Authentifikation):

Pkg #1 from Client

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

Pkg #2 from Server

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

```
    <MsgRef>1</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
    <TargetRef>http://www.syncml.org/sync-server</TargetRef>
    <SourceRef>IMEI:493005100592800</SourceRef>
    <Chal>
     <Meta>
      <Type xmlns='  syncml:metinf'  >syncml:auth-basic</Type>
      <Format xmlns='  syncml:metinf'  >b64</Format>
     </Meta>
    </Chal>
    <Data>407</Data> <!--Credentials missing-->
   </Status>
   ...
  </SyncBody>
 </SyncML>
```

Pkg #1 (with credentials) from Client

```
<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>
  <Cred>
   <Meta><Type xmlns='  syncml:metinf'  >syncml:auth-basic</Type></Meta>
   <Data>QnJ1Y2UyOk9oQmVoYXZl</Data> <!—base64 formatting of "userid:password"-->
  </Cred>
 </SyncHdr>
 <SyncBody>
 ...
 </SyncBody>
</SyncML>
```

Pkg #2 from Server

```
<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>1</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
   <TargetRef>http://www.syncml.org/sync-server</TargetRef>
   <SourceRef>IMEI:493005100592800</SourceRef>
   <Data>212</Data> <!--Authenticated for session-->
  </Status>
 ...
 </SyncBody>
</SyncML>
```

Initialisierungsphase

Die Initailisierungsphase dient zum festlegen der Synchronisationart, der zu synchroniserenden Datenbanken, der Synchronisationsanker und um sich eventuell gleich zu initialisieren.

Synchronisationsarten

Zwei-Wege Synchronisation

Bei der Zwei-Wege Synchronisation tauschen sowohl der Server als auch der Klient Daten aus. Hierbei sendet der Klient erst seine neuen oder veränderten Daten, dann gleicht der Server diese Daten mit seinen Daten ab, löst eventuell auftretende Konflikte auf und sendet seine neuen oder veränderten Daten (und die Resultate der aufgelösten Konflikte) an den Klienten.

Ein-Weg Synchronisation

Die Ein-Weg Synchronisation geht wie der Name schon sagt in nur eine Richtung. Hierbei sendet also nur entweder der Klient (bei einer klientenseitigen ein-Weg Synchronisation) neue oder veränderte Daten an den Server oder aber umgekehrt (bei einer serverseitigen ein-Weg Synchronisation).

Langsame Synchronisation

Bei einer Langsamen Synchronisation werden nicht nur die neuen oder veränderten Daten gesendet, sondern alle Daten des Gerätes. Diese Form der Synchronisation wird angewand, wenn die Synchronisationanker der Geräte nicht übereinstimmten. Ansonsten verläuft eine Langsame Synchronisation wie eine Zwei-Wege Synchronisation.

Auffrischungs Synchronisation

Ist eine Langsame Ein-Weg Synchronisation. Es werden also alle Daten in nur eine Richtung verschickt.

Server Alerted Synchronisation

Die Server Alerted Synchronisation ist die einzige Synchronisation, bei der der Server die Synchonisation auslöst. Sonst tut dies immer der Klient. Das Problem hierbei ist, das der Klient erst einmal eine Verbindung auf Transportebene gestartet haben muss, und auch den SynKlient (das klientseitige SyncML Programm) gestartet haben muss.

3. Syncml-Sync-Represent:

Allgemein

Datenformate:

SyncML stellt nicht nur eine allgemein verfügbare Menge von Befehlen zur Datensynchronisation bereit, sondern auch eine kleine Menge von Datenformaten. Diese Datenformate entsprechen einer allgemein akzeptierten Menge von Medientypen, wie Kontakte, Kalenderinformationen und Nachrichten. Um die Spezifikation einzuhalten ist die Unterstützung dieser Formate ist verbindend. Zusätzlich zu diesen Formaten kann SyncML die Datentypen des MIME Framework benutzen.

Resourcen Austausch:

SyncML unterstüzt den Austausch von Informationen über Geräteresourcen. Mit diesen Informationen können Probleme mit z.B. unterschiedlich lange Identifieren, oder zu wenig Speicher vermieden werden.

Daten Identifizierungs Tabelle (**Data Identifier Mapping**):

SyncML verlangt nicht, dass die synchronisierten Daten homogen auf verschidenen Geräten gespeichert werden. SyncML erlaubt sogar das sowohl Datenformate als auch die Identifier verschidenen seinen können. Um dies zu ermöglichen braucht man eine Zuweisungstabelle, mit deren Hilfe man den Identifier des einen Gerätes in den Identifier des anderen Gerätes übersetzen kann.

Datenauffrischung:

Zusätzlich zu der Datensynchronisation enthält SyncML Kommandos um Daten zu aktualisieren. Dies ist aus praktischen Gründen in einem Synchronisationsprotokoll notwendig (Sekräterin aktualisiert Addresse, die die Cheffin bereits im Palm hat). Desweiteren bietet das Protokoll die Möglichkeit an, den kompletten Datenbestand aufzufrischen. Dies könnte durch einen Hardwarefehler in dem mobilen Gerät erfroderlich werden. Dies wird durch ein „refresh Alert“ Kommando ausgelöst, das der Client an den Server sendet.

Weiche und harte Datenlöschung:

Das „Delete“ Kommando gibt SyncML die Möglichkeit, bei einem Empfänger die Löschung von Daten zu beantragen. Hierbei gibt es zwei Formen der Löschung. Normalerweise wird sollen die Daten komplett aus dem Speicher des Gerätes entfernt werden. Die gelöschten Daten sollen dann nicht mehr mit den synchronisierten Daten auf dem anderem Gerät in Verbindung gebracht werden. Diese Form der Löschung wird harte Löschung („Hard Delete“) genannt. Im Unterschied dazu steht die weiche Löschung „soft Delete“ . Hierbei sollen die Daten auch beim Empfänger gelöscht werden, aber nicht aus den synchronisierten Daten. Der Grund für diesen „soft Delete“ liegt in dem oft begrenzten Speicherplatz von mobilen Geräten. Die Daten werden gelöscht um wichtigeren Daten Platz zu machen, sie verbleiben aber in dem Satz der synchronisierten Daten (auf dem Server) vorhanden.

Daten archivieren:

Daten die nicht in einem Archiv sind, können nicht gelöscht werden, wenn eine mobiles Gerät kein Archiv hat, und Daten gelöscht werden sollen kommt es zu einem „Delete without Archive“ Fehler.

Daten ersetzen:

Das „Replace“ Kommando, befähigt den Auslöser Daten auf dem Empfänger zu erstzen. Durch dieses Kommando kann es zu einem „Update Conflict“ kommen.
Zusätzlich kann der Auslöser des Kommandos Metainformationen auf dem Empfänger ändern, z. B. Eine ungelesene E-Mail als gelesen markieren. Diese Fähigkeit wird durch das „Mark“ Element in dem „Replace“ Kommando zur Verfügung gestellt.
Gelegentlich kann es zu einer Ausnahme kommen, und zwar wenn die gleichen Daten auf dem SyncML Clienten und Server verändert oder ersetzt wurde. In diesem Fall kommt es zu einem „Update Conflict“.
Dazu kann es aber nur bei einer „two-way synchronization“ (eine Anpasssung der Daten in beide Richtungen) kommen.

Nach Daten suchen:

Um nach einem bestimmten Datensatz zu suchen, kann der ein Teil der Synchronisation ein „Search“ Kommando geben. Dieses Kommando unterstüzt jede registrierte Suchgrammatik. Diese wird in dem „Type“ Element des „Meta“ Elementes des Suchkommandos angegeben.
Das „Search“ Kommando kann benutzt werden, um Einträge in einer Datenbak zu bestimmen, die dann als Quelle für ein untergeordnetes SyncML „Sync“ Kommando gelten.
Zusätzlich können Suchen oder Filter nur auf einem „Target LocURI“ (eine einzelne Datenbank auf dem Zielgerät) Element in dem „Sync“ Kommando definiert werden. Mit dieser Fähigkeit kann man z.B. die Synchronisation mit einem mobilen Klienten nur über die Termine des heutigen Tages durchführen.

Lokalität:

Mit Hilfe des *xml:lang* Attributes kann der Auslöser die geforderte Sprache für Antworten bestimmen. Insbesondere als Attribute in den *Get* und *Search* Befehlen.
Die Standarteinstellung ist UTF-8.

MIME Nutzung:

Es gibt zwei MIME Typen für SyncML Datensynchronisationsnachrichten. Den Mime Type *application/vnd.syncml+xml* für Klartext XML, und den Typ *application/vnd.syncml+wbxml* der eine binäre Repräsentation (WBXML) der SyncML-Nachrichten darstellt.
Einer dieser beiden Typen muß genutzt werden, um SyncML Nachrichten zu übertragen.

Ziel- und Quelladressierung:

Die *Target* und *Source* Elemente werden in SyncML benutzt um Ziel- und Quellroutingadressen bezüglich des *LocURI* Elementes anzugeben. Die *LocURI* sollte entweder eine Orts URI oder eine Orts URN sein, aber in bestimmten Fällen kann es also ein lokal eindeutiger Index (Identifier) sein. Zusätzlich kann ein optionaler Name (*LocName*) innerhalb des *Target* oder *Source* Elementes angegeben werde, um einen lesbaren Namen für einen *LocURI* Wert zur Verfügung zu stellen.
Die Semantik eines *LocURI* Wertes ist kontextabhängig. Das heisst, der Wert wird in verschieden Befehlen anders interpretiert. In einem *Alert* Kommando beschreibt der *LocURI* Wert eine Datenbank in dem die Ursache des Alarms liegt. Im Gegensatz dazu beschreibt der Wert in einem *MapItem* einen einzelnen Datenbankeintrag.
Wenn eine URI angegeben wird muss entweder eine relative oder eine absolute URI benutzt werden. Eine relative URI muss benutzt werden, wenn der Empfängt aus dem Kontext (der SyncML Nachricht) die absolute URI ermitteln kann. Zum Beispiel kann die relative URI des *Target* Elementes in einem *Alert* Kommando leicht in eine absolute URI umgewandelt werden,
indem man den Wert des *Target* Elements aus dem *SyncHdr* der relativen URI voranstellt.
Die Ziel- und Quelladressen werden in den dazugehörigen Antworten der Empfänger in den Elementen *TargetRef* und *SourceRef* wiedergespiegelt. Diese Antworten sind zumeist *Status* Antworten, und haben den Zweck einer Bestätigung.

4. Device Managment Protokoll

Allgemein

Als grundsätzliches Problem bei der Synchronisation diverserer Gerätetypen stellt sic die Frage nach der Leistungsfähigkeit insbesondere mobiler Geräte und deren Abgleich zwischen den Partnern einer Sync-Session dar.

Hierbei sind neben der Frage nach dem verfügbaren Speicherplatz insbesondere die Frage der Art des Dateisystems und die Art der verarbeiteten Daten von Wichtigkeit.

Daher muß vor dem Beginn eines jeden Austausches von Daten zwischen dem Server und dem Klienten genau ausgehandelt werden welche Parameter der jeweilige Partner besitzt.
So kann es etwa Klienten geben, welche nur Kalendareinträge oder Telefonnummern speichern oder aber Server die nur die aktuellen eMails zur Verfügung stellen.

Zu Zweck des Austausches dieser gerätespezifischen Informationen dient das Device Managment Protokoll.

Betrachtet man das Problem näher so scheint es sinnvoll, daß sich verschiedene Typen von Servern und verschiedene Typen von Klienten möglichst gleich oder zumindest ähnlich verhalten.
Mit Hilfe des Protokolls versucht man daher durch eine baumartige Kategorisierung der Geräteeigenschaften das Verhalten nach Außen möglichst ähnlich zu gestalten.

Dazu dient das standardisierte Device Discription Framework.

Protokollaufbau – das Phasenmodell

Das Device Managment Protokoll selbst läßt sich in zwei grobe Phasen unterteilen, wobei die erste dem Austausch von Einstellungen und die zweite dem eigentlichen Verwalten von Informationen dient.

In der Regel geht man davon aus das die Anforderung einer Synchronisation von Seiten des Klienten zu erfolgen hat; eine Einleitung des Vorganges durch den Server, etwa durch Softwarebefehl ist aber ebenfalls im Protokoll vorgesehen.

Das erste wirkliche Datenpaket enthält sodann die gesamten Informationen über den Klienten und wird von diesem entsprechend zum Server gesendet.

Dieser beantwortet das Paket seinerseits indem er dem Klienten seine Leistungsfähigkeit mitteilt und zudem übermittelt, welche Art von Daten er zur Verfügung stellt bzw. in der Lage zu synchronisieren ist.
Außerdem enthält dieses Paket vom Server bereits den ersten Befehl zum Austausch von Daten oder aber Benutzereingaben, die auf dem Klienten ausgeführt werden sollen (etwa Befehle zur Einrichtung von Programmen auf dem Klienten).

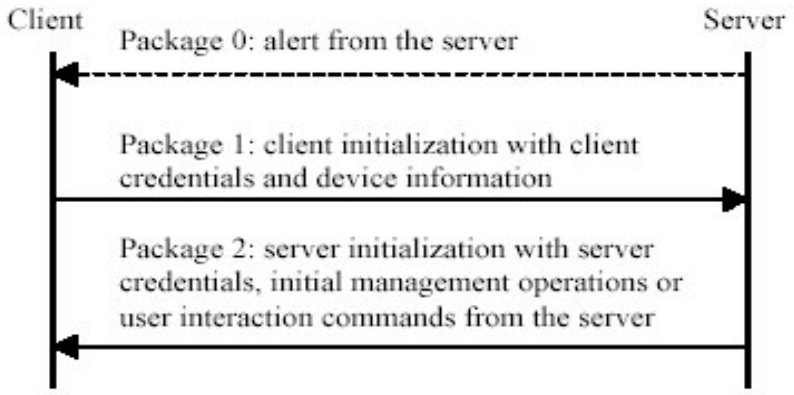
Nunmehr ist die sogenannte „Setup-Phase“ abgeschlossen und das Protokoll tritt in seine zweite Phase, die auch als „Managment-Phase“ bezeichnet wird, ein.

Diese beginnt damit, daß der Klient auf die soeben erhaltenden Befehle des Servers reagiert, wobei es sich hierbei um das Übertragen angeforderter Daten oder aber einfach um die Bestätigung einer ausgeführten Operation handeln kann.

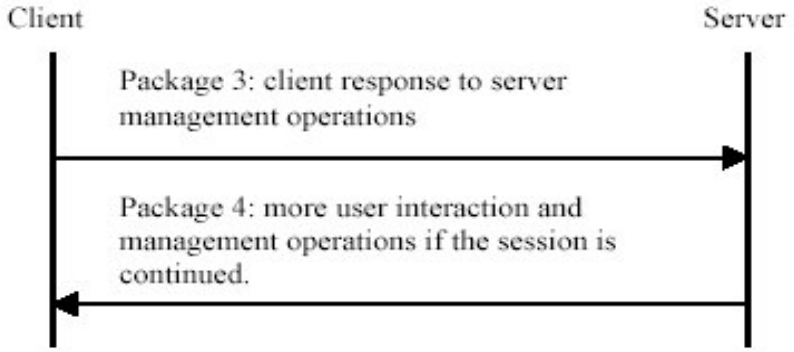
Sodann antwortet der Server den Erhalt des Klientenpaktes indem er entweder weitere

Operationen an diesen überträgt oder aber die laufende Sitzung durch Senden eines entsprechenden Endpaketes beendet und die Verbindung trennt.
Die letzten beiden Phasen des Ablaufes können sich entsprechend so oft wiederholen, bis die Sitzung von Seiten des Servers getrennt wird oder aber die Verbindung physikalisch verloren geht.

Der Veranschaulichung der genannten Abläufe dient folgendes Diagramm, welches der Onlinedokumentation zum SyncML-Standard entnommen wurde:



Setup phase



Management phase

Device Discription Framework

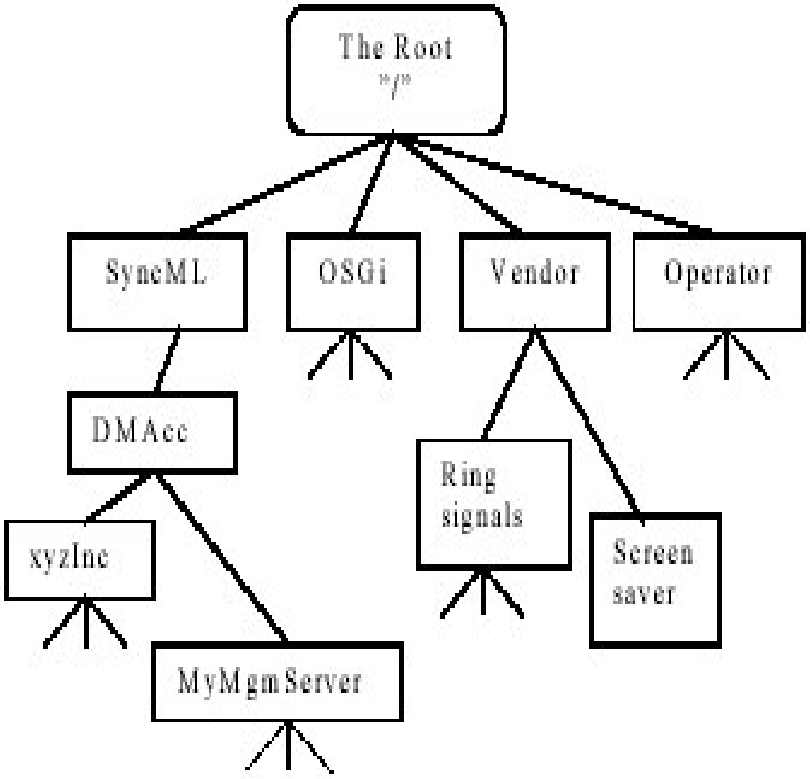
Um einen möglichst einheitlichen Standard zum Austausch der Fähigkeiten der Vielzahl von Geräten zu erhalten und den Standard offen für belieblige Erweiterungen zu bestalten, nutzt das SyncML Device Managment Protokoll als Grundlage ein auf dem XML-Standard definiertes Device Discription Framework.

Dieses wurde baumartig gestaltet, was es zudem bei der Benutzung stark vereinfacht etwa Eigenschaften zu definieren, die für alle Geräte eines Herstellers oder Types zutreffen.

Ein einfaches Beispiel wäre hier etwa die Definition von Klingeltönen, die für alle Handymodelle eines Herstellers auf einem Server zur Verfügung gestellt werden sollen.

Durch die beliebige Erweiterbarkeit eines solchen Baumes lassen sich somit für jede Anwendung zugeschnitten die nötigen Eigenschaften definieren und dem jeweiligen Kommunikationspartner auf einheitliche Weise mitteilen.
Das Verhalten des somit beschriebenen Gerätes ist dem Partner gegenüber also eindeutig beschrieben.

Ein mögliches Beispiel für einen solchen Baum für ein Mobiltelefon ist in der folgenden Grafik dargestellt:



5. Protokoll Kommando Elemente

Add:

Das *Add* Kommando wird im allgemeinen genutzt um den Empfänger über neue Daten in der Datenbank des Senders zu Informieren. Dieses Kommando kann nur innerhalb eines *Sync* Kommandos erfolgen.
Der Sender sollte vorher bestimmen was für Fähigkeiten und Eigenschaften der Empfänger in bezug auf die Datenelemente hat, und nur unterstützte Eigenschaften benutzen. Das *device information document* kann diese Informationen enthalten.
Das verbindliche *CmdID* Element gibt, eine innerhalb der Nachricht, eindeutige Nummer für das Kommando an.
Wenn angegeben, gibt das optionale *NoResp* Element an, dass kein Status Code für das Kommando zurückgegeben werden muss.
Das optionale *Cred* Element liefert eine Authentifizierungsbeglaubigung für dieses Kommando. Wenn es nicht angegeben ist, wird die standard Authentifizierung von dem Elternelement (das *Sync* Element) verwendet. Wenn es dort nicht angegeben ist, wird das des nächsten Elternelement genommen (das *SyncHdr* Element). Wenn es dort auch nicht angegeben ist, wird das Kommando ohne Beglaubigung ausgeführt.
Das optionaler *Meta* Element gibt die Metainformationen für dieses Kommando an. Zum Beispiel kann der allgemeine Datentyp der Daten angegeben werden. Die Reichweite der Metainformationen ist auf das Kommando begrenzt.
Mindestens ein *Item* Element muss angegeben werde. Das *Item* Element gibt die Datensätze die in die Datenbank hinzugefügt werden sollen an. Die *Target* und *Source* Elemente in dem *Item* sollten eine relative URI sein. Relativ zu dem umschliessenden *Atomic*, *Sequence* oder *Sync* Kommando.
Der Empfänger kann dem Datensatz eine neue lokale Kennung geben, muss in diesem Fall aber dem Sender mi einem *Map* Kommando über diese Zuweisung informieren.
Wenn das Kommando erfolgreich ausgeführt wurde, wird die (*201*) *Item added* Ausnahme erzeugt. In dem Fall das die der Datensatz bereits existiert, wird die Ausnahme (*418*) *Already exists* von Kommando erzeugt.
Wenn die angegebene Authentifizierung nicht für die Aufgabenausführung ausreicht (aufgrund von mangelnden Rechten) wird die (*401*) *Unauthorized* Ausnahme kreiert, sollte keine Authentifizierung gegeben worden sein, wo eine benötigt wird, hat dies eine (*407*) *Authentication required* Ausnahme zur Folge. Eine passende Herausforderung (*Challenge* – siehe Authentifizierung) kann (mit) zurückgegeben werden.
Wenn der Medientyp oder das Medienformat nicht vom Empfänger unterstützt wird, kommt eine (*415*) *Unsupported media type or format* Ausnahme zurück.
Sollte der Platz auf den Datenträger des Empfängers unzureichend sein, erhält der Sender eine (*420*) *Devive Full* Ausnahme.
Alle weiteren Fehler generieren eine (*500*) *Command failed* Ausnahme.

Alert:

Das *Alert* Kommando wird speziell benutzt um Bekanntmachungen zu übertragen, wie eine Synchronizationsanfrage. Wenn ein mobiles Gerät beispielsweise einen „klientenseitig initierte, zwei-Wege Synchronisation“ starten möchte, würde es diese Kommando verwenden.

Die Elemente *CmdID*, *NoResp* sind bereits bei dem *Add* Kommando beschrieben, und haben hier die gleiche Funktion.

Das optionale *Data* Element spezifiziert die Art des Alarms.

Wenn das Kommando und der die damit assoziierte *Alert* Aktion erfolgreich ausgeführt wurde, wird ein (*200*) *OK* Ausnahme erzeugt. Wurde das Kommando erfolgreich akzeptiert, aber die *Alert* Aktion noch nicht ausgeführt, dann wird eine (*202*) *Accepted for processing* Ausnahme generiert. Eine folgende Ausnahme kann erzeugt werden um den Status der Ausführung des Kommandos anzuzeigen.

Sollten nicht genügend Rechte für die Aktion vorhanden sein, wird wie beim *Add* Kommando verfahren.

Wenn das Synchronisationsprotokoll nicht zulässt, dass das *Alert* Kommando in dem aktuellen SyncML Packet benutzt wird, wird eine (*405*) *Command not allowed* Ausnahme erzeugt.

Ist der angegebene *Alert* nicht von dem Empfänger unterstützt, schickt dieser die (*406*) *Optional feature not supported* Ausnahme.

Für fehlende Parameter in dem *Item* Element gibt es die (*412*) *Incomplete command* Ausnahme, und wenn das Format oder der Typ des Datenelements dem Empfänger unbekannt ist, schickt dieser eine (*415*) *Unsupported media type or format* Ausnahme.

Für alle anderen Fehler wird, wie auch beim *Add* Kommando die (*500*) *Command failed* Ausnahme erzeugt.

Atomic:

Es gibt wieder die *CmdID*, *Meta* und *NoResp* Elemente.

Das Innere eines *Atomic* Elementes besteht aus einem oder mehr *Add*, *Delete*, *Copy*, *Map*, *Replace*, *Sequence* oder *Sync* Kommandos. Diese werden entweder alle erfolgreich ausgeführt, oder andernfalls, sollte nur ein Kommando fehlschlagen, werden alle Kommandos nicht ausgeführt. Der Fehlschlag hätte eine (*507*) *Atomic failed* Ausnahme zur Folge. Es ist möglich, noch weitere Ausnahmen zu erzeugen, die den Fehler präziser beschreiben. Verschachtelte *Atomic* Kommandos sind nicht erlaubt. Diese hätte eine (*500*) *Command failed* Ausnahme zur Folge.

Copy:

Das *Copy* Kommando dient dazu Dateien auf dem Empfänger des Kommandos zu kopieren. Ob es sich bei der Kopie um eine physikalische Kopie oder nur um einen Pointer auf das Quellelement handelt bleibt der jeweiligen Implementation überlassen.

Das C*opy* Kommando soll nicht den Medientypen ändern, oder irgendwie sonst das Datenelement verändern.

Auch in diesem Kommando gibt es das obligatorische *CmdID,* sowie die optionalen *NoResp*, *Cred* und *Meta* Elemente (siehe *Add*). Wie auch bei dem *Add* Kommando muss das *Copy* Kommando ein oder mehr *Item* Elemente beinhalten. Es müssen in den gleichen Kontexten wie bei dem *Add* Kommando relative bzw. absolute URI angegeben werden. Wenn der Empfänger der Kopie einen anderen Bezeichner (ID) gibt, als in dem Kommando angegeben, muss er dem Initiator darüber mit einem *Map* Kommando informieren.

Wenn die Aktion erfolgreich ausgeführt wurde, wird eine (*201*) *Item added* Ausnahme erzeugt. Sollte das Element auf der Datenbank des Empfängers bereits existieren, wird eine (*418*) *Already exist* Ausnahme zurück gesendet. Desweiteren gibt es, wie beim *Add* Kommando die (*420*) *Device full* Ausnahme. Wenn ein Fehler beim kopieren der Daten in der Datenbank auftritt, wird eine (*510*) *Data store failure* Ausnahme erzeugt. Ansonsten gibt es für alle unspezifischen Fehler die (*500*) C*ommand failed* Ausnahme.

Delete:

Das *Delete* Kommando wird im allgemeinen genutzt, um Daten permanent aus der Datenbank des Empfängers zu löschen. Es kann aber auch dazu benutzt werden um Daten nur temporär zu löschen, um zum Beispiel Platz für ein folgendes *Add* Kommando zu schaffen. Im letzterem Fall handelt es sich um einen *Soft Delete*. Implementationen die beide Eigenschaften unterstützen, das *Mark for Deletion* und das pysikalische *Delete*, müssen letzteres bei dem *Delete* Kommando ausführen, und für ersteres die *Mark* Metainformationen in einem *Replace* Kommando benutzen (das Datenelement als gelöscht markieren).
Hierbei gibt es das optionale *Archive* Element, das angibt, dass der Empfänger sicherstellt das er eine Kopie der Daten besitzt, bevor er die Daten löscht. Sollte der Empfänger das Archive *Kommado* nicht unterstüzen, muss er eine (*210*) *Delete without archive* Ausnahme zurückgeben.
Auch in diesem Kommando gibt es die *CmdID*, *NoResp*, *Cred* und *Meta* Elemente (siehe *Add*).

In Anwendungen (z.B.: E-mail Clients), in denen das Konzept des Löschens bedeutet, dass man die „gelöschten“ Daten in einen speziellen Ordner namens gelöscht verschiebt, werden zwei Kommandos benötigt (sinnvollerweise von einem *Atomic* umschlossen). Erst ein *Add* Kommando, das die Daten in den gelöscht Ordner hinzufügt, und dann ein *Delete* Kommando, das die Daten aus dem alten Ordner entfernt.
Der Empfänger des *Delete* Kommandos kann jede Untermenge der Angegebenen Daten löschen. Sollte er dies nicht tun, muss er eine (*206*) *Partial content* Ausnahme ausgeben. Wenn ein *Status* Kommando mit der Ausnahme zurückgegeben wird, sollten die ID' s der nicht gelöschten Teildaten in dem *Status* enthalten sein. Wenn der Empfänger feststellt, das die Daten in seiner Datenbank nicht existieren, wird ein (*211*) *Item not deleted* Ausnahme erzeugt.
Wenn für die Bezeichner (ID' s) der zu löschenden Daten ein mapping existiert (siehe *Map*), muss der Server immer die Bezeichner des Klienten benutzen. Andernfalls wird eine (*412*) *Incomplete command* Ausnahme zurückgegeben werden. Unspezifische Fehler erzeugen wie immer eine (*500*) *Command failed* Ausnahme.

Exec:

Mit dem *Exec* Kommando führt man entfernte Prozedurenaufrufe auf einem entfernten Netzwerkgerät durch (remote Procedure Calls). Dieses Kommando erlaubt es entweder dem mobilen Gerät Funktionen auf dem Server auszuführen, mit dem es sich Synchronisieren will, oder aber dem Server Funktionen auf dem mobilen Gerät auszuführen (z.B. um Konflikte von dem Benutzer lösen zu lassen). Hierbei sollten der der Programmierer und Benutzer beide Maßnahmen ergreifen, um Sicherheitsbedrohungen die durch remote Procedure Calls entstehen zu vermeiden.

Dieses Kommando beinhalted wieder das *CmdID*, *NoResp* und *Cred* Element.

Wenn das Kommando erfolgreich ausgeführt wurde wird die (*200*) *OK* Ausnahme gesendet, solange das Kommando akzeptiert wurde, und noch im ausführen begriffen ist, wird die (*202*) *Accepted for processing* Ausnahme erzeugt.

Sollte eine andere Einheit als die aufgerufene Prozedur anfragende Statuscodes an den Auslöser zurückgeben wird die (*203*) *Non-authoritative response* Ausnahme erzeugt. Das kann püassieren, wenn die aufgerufende Prozedur einen daemon des Unterliegenden Systems aufruft.

Wenn die Zielprozedur permanent verschoben wurde, kommt es zu einer (*301*) *Moved permanently* Ausnahme, und wenn sie nur temporär verschoben wurde, kommt es zu einer (*302*) *Found* Ausnahme. In dem Fall, das die Prozedur nicht mehr auf dem Gerät exitiert wird eine (*410*) *Gone* Ausnahme erzeugt. Sollte die Funktion durch einen Proxy hindurch ausgeführt werden, wird dies mit der (*305*) *Use Proxy* Ausnahme angezeigt.

Wenn der Syntax der *Exec* Kommandos nicht richtig angegeben wurde erzeugt dies eine (*400*) *Bad Request* Ausnahme. Wenn der Aufrufende nicht die benötigten Rechte hat, um das Kommando auszuführen wird eine (*403*) *Forbidden* Ausnahme geworfen, oder, alternativ, eine (*404*) *Not Found* Ausnahme sofern der Empfänger nicht mitteilen will warum die Anfrage abgelehnt wurde. Für weitere Ausnahmen die durch Authorization erzuegt werden siehe *Add.*

Wenn die auszuführende Prozedur gerade läuft, und deshalb erst später ausgeführt werde kann wird eine (*417*) *Retry later* Ausnahme zurückgegeben. Wenn es zu einem Fehler beim Ausführen der Prozedur kommt, wird dies mit der (*506*) *Processing error* Ausnahme angezeigt.

Get:

Es gibt keine Synchronisationszustandsinformationen über Daten, die durch ein *Get* Komanndo gewonnen wurden. Das *Get* Kommando darf nicht in einem *Sync* Element enthalten sein. Die Daten, die durch ein *Get* Kommando angefordert wurde, werden in einem *Results* Element in der folgenden SyncML Nachricht übergeben.
Für dieses Kommando gibt es wieder die üblichen *CmdID*, *NoResp*, *Lang*, *Cred* und *Meta* Elemente (siehe *Add*).
Wenn das Kommando erfolgreich ausgeführt wurde, wird die (*200*) *OK* Ausnahme erzeugt. Sollte das Kommando erfolgreich durchgeführt worden sein, aber es gibt keinen Inhalt zurückzugeben (bei einer Anfrage wie beispielsweise: „Adressen aller Meiers“), hat das ein (*204*) *No content* zur Folge. Wenn das Kommando erfolgreich ausgeführt wurde, aber nicht alle Daten auf einmal zurückgegeben wurden, gibt es die Ausnahme (*206*) *Partial content* als Anzeige dafür, das noch Daten folgen werden.

Wenn das Ziel nicht eindeutig angegeben wurde, sondern es mehrere Treffer für die Anfrage gäbe, wird die Ausnahme (*300*) *Multiple Choices* erzeugt. Desweiteren gibt es für dieses Kommando die (*305*) *Use Proxy*, die Authorization Ausnahmen und die (*404*) *Not found* Ausnahmen.
Wenn das angefragte Datenelement zu groß ist, um zu diesem Zeitpunkt übertragen zu werden, wird statt dessen eine (*413*) *Request entity too large* Ausnahme geschmissen. Wenn der Medientyp nicht vom Empfänger unterstützt wird, wird dies mit der Ausnahme (*415*) *Unsupported media type or format* angezeigt, und alle unspezifischen Fehler haben eine (*500*) *Command failed* Ausnahme zur Folge.

Map:

Das *Map* Kommando informiert über neu hinzugekommene oder gelöschte Elemente mit der *Map table* des Empfängers. In dieser *Map table* wird die Zuweisung von den längeren Bezeichnern des Servers in die kürzeren der Klienten vorgenommen. Beispielsweise hat das mobile Gerät nur 2 Byte lange Bezeichner (um Platz zu sparen) für seine Datensätze, während der Server mit 16 Byte langen Bezeichnern arbeitet. In diesem Fall werden die langen Bezeichner des Servers zu den entsprechenden kürzeren Bezeichnern des mobilen Gerätes gespeichert. Der Sinn hiervon ist, das der Server seine Anfragen mit den Bezeichnern des Klienten machen muß. Deshalb werden diese *Map tables* im allgemeinen auch auf den Servern gepflegt. Diese *Map tables* sind nicht nötig, wenn der Server und der Klient mit demselben physikalischen Schema arbeiten (exakte kopien voneinander sind), und deshalb die gleichen Bezeichner haben.

Die neuen Bezeichner werden von dem Empfänger (hier: dem Klienten) zugewiesen. Dieser muss dann den Server mit dem *Map* Kommando über den neuen Bezeichner informieren.

Das *Map* Kommando muß *Atomic* sein. Das heisst, wenn mehrere Zuweisungen durch ein Kommando nötig werden, müssen entweder alle übertragen werden, oder gar keine. Wenn die Operation fehlschlägt, muss der Empfänger dies in der nötigen Anwort durch einen Fehlerkode angeben. Das *Map* Kommando ist idempotent, das heisst, dass wenn das Kommando zwei mal ausgeführt wird, das Resultat das selbe ist, wie wenn sie nur einmal ausgeführt wurde (im Gegensatz zu „multipliziere mit 2“).

Das *Target* und *Source* Element muss in diesem Kommando spezifiziert werden. Hierbei gibt das *Target* Element die Daten des mobilen Gerätes und das *Source* Element die Daten auf dem Server an.

Es muss ein oder mehr *MapItem* Elemente angegeben werden. Ein *MapItem* gibt eine einzlene Zuweisung an.

Wenn das Kommando erfolgreich ausgeführt wurde, wird die *200 OK* Ausnahme erzeugt.

Das *Map* Kommando besitzt das verpflichtende *CmdID* Element, sowie die optionalen *Meta* und *Cred* Elemente. Desweiteren können bei diesem Kommando die Authentifizierungsausnahmen, die (*420*) *Device full*, (*510*) *Data store* failure oder das unspezifische (*500*) *Command failed* erzeugt werden.

Put:

Das Kommadno *Put* schiebt Daten auf den Empfänger. Es gibt keine Synchronisationszustands-informationen über Daten, die mit Hilfe eines *Put* Komanndos übertragen wurden.
Es gibt hier die üblichen *CmdID*, *NoResp*, *Lang*, *Cred* und *Meta* Elemente (siehe *Add*). Das *Put* Kommando gibt nicht die Grösse der zu übertragenden Datei an. Dies passiert im *Meta* Element mit dem Attribut *Size*. Wenn dort die Grösse nicht angegeben wird, lösst dies eine (*411*) *Size required* Ausnahme aus. Sollte der zu übertragenden Datei zu gross sein, hat dies eine (*413*) *Request entity too large* Ausnahme zu Folge. Im Unterschied dazu steht die Ausnahme (*416*) *Requested size too big*, die erzeugt wird, wenn die Grösse des *Size* Attributes im *Meta* Element zu gross für den Puffer oder den Speicher des Empfangsgerätes ist. Ansonstonsten gibt es die (*200*) *OK* Ausnahme, die (*305*) *Use proxy* Ausnahme, die Authentifizierungsausnahmen, die (*415*) *Unsupported media type or format* Ausnahme, die (*420*) *Device full* Ausnahme und die unspezifische (*500*) *Command failed* Ausnahme für dieses Kommando.

Replace:

Mit dem *Replace* Kommando ersetzt man Daten auf dem Empfänger. Dieses Kommando muss innerhalb eines *Sync* Kommandos stehen. Die Ausnahmen die enstehen können sind die gleichen die auch beim *Add* Kommando entstehen können.

Search:

Das Ergebnis der Suche wird in einem *Result* Kommando geliefert, sofern nicht in dem *Search* Kommando das *NoResults* Kommando angegeben wurde.

Als zusätliche Kommandos für diesen Befehl gibt es *CmdID*, *NoResp*, *Cred*, *Target*, *Source*, *Lang*, *Meta* und *Data*. Davon müssen *CmdID*, *Source*, *Data* und *Meta* angegeben werden. Hierbei gibt *Source* die Datenbank auf der gesucht werden soll an, *Meta* den Typus der Suchgrammatik und *Data* die Suchgrammatik (oder auch Suchanfrage).

Wenn das optional *Target* Element angegeben wird, muss das *NoResult* Kommando gegeben werden (umgekehrt genauso). Dieses Kommandokombination bedeutet, das kein Ergebniss zurückgegeben werden soll, sondern temporär auf den Empfänger gespeichert werden soll, wobei der Ort für die temporäre Speicherung mit *Target* angegeben wird. Diese temporären Daten werden dann als Quelle für ein folgendes *Sync* Kommando benutzt.

Ausnahmen dier von diesem Kommando erzeugt werden sind (*200*) *OK*, (*204*) *No content*, (*206*) *Partial content*, (*400*) *Bad request*, (*412*) *Unknown search grammer*, (*401*) *Unauthorized*, (*407*) *Authentication required*, (*403*) *Forbidden*, (*405*) *Command not allowed* (wenn eine Suche prinzipell nicht auf dem Empfänger erlaubt ist), (*404*) *Not found*, (*413*) *Request entity too large*, (*420*) *Device full*, (*415*) *Unsupported media type or format* oder (*500*) *Command failed*. Eine Beschreibung der Ausnahmen findet sich in den oberen Abschnitten.

Eine alternative für das *Search* Kommando bietet das *Exec* Kommando in Verbindung mit einem suchenden CGI Skript.

Sequence:

Das *Sequence* Kommando wird benutzt um mehrere Befehle in einer Sequenz zu bündeln

und die Reihenfolge ihrer Ausführung festzulegen (die ohne *Sequence* beliebig ist).
In dem *Sequence* Kommando müssen deshalb auch mindestens ein *Add*, *Replace*, *Delete*, *Copy, Atomic*, *Map* oder *Sync* Kommando enthalten sein.
Weitere Elemente für dieses Kommando sind *CmdID*, *NoResp* und *Meta*. Hievon ist nur *CmdID* verbindlich.
Ausnahmen sie von diesem Kommando geworfen werden können sind (*200*) *OK*, (*406*) *Optional feature not supportet* (wenn der Empfänger kein *Sequence* unterstützt) und (*500*) *Command failed*.
Verschachtelte *Sequence* Kommandos sind nicht erlaubt. Sollten sie vorkommen würde eine (*500*) *Command failed* Ausnahme erzeugt werden.

Sync:

Sync ist Wurzelelement der SyncML Nachricht. Elemente die in diesem Kommando angegeben werden können (und teilweise müssen) sind *CmdID*, *NoResp*, *Cred*, *Target*, *Source* und *Meta*. Desweiteren müssen eine oder mehre *Add*, *Replace*, *Delete*, *Copy*, *Atomic* oder *Sequence* Kommandos angegeben werden. Die Reihenfolge in der die Kommandos ausgeführt werden ist beliebig, sofern sie nicht in einem *Sequence* Kommando gegeben werden.

Mögliche Ausnahmen die von diesem Kommando erzeugt werden sind (*200*) *OK*, (*401*) *Unauthorized*, (*407*) *Authentication required*, (*403*) *Forbidden*, (*404*) *Not found*, (*405*) *Command not allowed*, (*508*) *Refresh required* und (*500*) *Command failed*. Neu hierbei ist die (*508*) *Refresh required* Ausnahme. Sie wird erzeugt, wenn der Empfänger feststellt, dass mit hoher Wahrscheinlichkeit die Daten der Synchronisationspartner nicht mehr synchronisiert sind. Wenn der Auslöser des *Sync* Kommandos diese Ausnahme empfängt sollte er eine *slow synchronisation* starten.

6. Schluß

Endbetrachtung

Bei der Betrachtung des vorliegenden Protokolles fallen deutliche Faktoren ins Auge, welche zu begründen meinen, man habe einen idealen Standard zur Synchronisation von Daten zwischen diversen Gerätetypen vorliegen.

Zum einen zeigt der Aufbau nach dem XML-Standard eine deutliche Ausrichtung hin zu Kompatibilität insbesondere mit Übertragungsstandards aus dem Internetbereich und zum anderen deutet das Engagement vieler renommierter Hard- und Softwarehersteller auf ein großes Interesse hin dieses Protokoll als Standard einzuführen.
Zudem kommt hinzu, daß es bisher kein ernsthaftes Konkurrenzprodukt zu dem vorliegenden Modell auf dem Markt gibt.

Jedoch bleibt zu beachten, daß bereits die Erfahrungen mit vielen Protkollentwürfen insbesondere im Mobilfunkbereich bezeigt haben, daß diese Gründe allein noch nicht als Beweis für den Erfolg eines solchen Vorschlages herhalten können, zumal bis heute keinerlei Geräte auf dem Markt zu finden sind die SnycML bereits in praktischer Anwendung nutzen.

Literatur:

Zur Recherche für die vorliegende Arbeit wurden die Seiten der SyncML-Interessengemeinschaft im Internet unter www.syncml.org verwendet.