ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semesterarbeit

JaCal – ein verteilter Kalender

Franziska Meyer

30. September 2005

Betreuer: Keno Albrecht
Prof. Roger Wattenhofer

Zusammenfassung

JaCal ist ein verteilter Kalender, der es ermöglicht, dass verschiedene Benutzer Termindaten ohne zentralen Server austauschen können. Um miteinander zu kommunizieren verwendet JaCal ein eigenes, auf Jabber aufbauendes Protokoll. Neben der Verteilung bietet JaCal ein mächtiges Datenmodell, das auf einer eingebetteten, relationalen Datenbank aufbaut.

Inhaltsverzeichnis

Einleitung

Die Motivation dieser Semesterarbeit war die Notwendigkeit, die Kalenderdaten verschiedener Leute ohne einheitliche Infrastruktur abzugleichen, um Termine zu planen. Insbesondere sollten nicht alle auf den gleichen zentralen Groupware-Server zugreifen müssen. Zugleich löst JaCal das Problem, dass viele Benutzer ihre verschiedenen Groupware-Daten für die verschiedenen Gruppierungen (Arbeitsplatz, Familie, ...) von Hand abgleichen müssten. Die Daten werden über ein P2P-Netz ausgetauscht und lokal gespeichert, wodurch auch ein Zugriff ohne Netzanbindung möglich wird.

Jabber, das als System für den Datenaustausch benutzt wird, ist zwar nicht wirklich ein P2P-Netz, doch es bietet mit sehr wenigen Einschränkungen sehr viele Möglichkeiten, die die verteilten Server mindestens rechtfertigen. Das erste Kapitel wird eine kleine Einführung in Jabber bieten.

Unter einem verteilten Kalender verstehen verschiedene Leute verschiedene Sachen. Die von JaCal unterstützte Verteilung wird in Kapitel 2 genauer erklärt. Kurz vorweg genommen unterstützt JaCal den Austausch von Kalenderdaten zwischen mehreren Personen ohne zentralen Server. Hingegen bietet JaCal aber keine Möglichkeit für den einzelnen Benutzer, von überall Zugriff auf seine Daten zu haben, da die Daten ausschliesslich lokal gespeichert werden.

Da Jabber an sich noch nur ein Framework ist, um Nachrichten zwischen Peers zu versenden, ist es nötig auf diesem Framework aufbauend ein Protokoll zu verwenden, welches die Funktionalität des Datenabgleiches und der Terminübermittlung erbringt. Dieses speziell für JaCal entwickelte Protokoll wird in Kapitel 3 behandelt.

Details zur Implementation gibt es im Kapitel 4, abschliessend folgt ein Fazit.

1 Jabber

Jabber ist ein Protokoll, das für Instant Messaging (IM) entwickelt wurde. Im Gegensatz zu anderen verbreiteten IM-Protokollen basiert es nicht auf einem einzigen zentralen Server, sondern benutzt viele verteilte Server, die miteinander kommunizieren. Das Protokoll ist offen und als RFC 3920 standardisiert (siehe [4]). Obwohl ursprünglich für IM entwickelt, wurde beim Design von Jabber darauf geachtet, dass das Protokoll erweiterbar ist. Unter Verwendung dieser ausgezeichneten Basis wurde das JaCal-Protokoll entwickelt (siehe Kapitel 3).

Aufgrund des IM-Hintergrundes ist Jabber asynchron, jedoch nicht nur aus der Sicht des Computers, sondern auch aus der Sicht des Benutzers: Es können auch Nachrichten an Peers gesendet werden, die offline sind. Diese Nachrichten werden auf einem Server gespeichert und zugestellt, sobald sich der Benutzer wieder anmeldet.

Gewählt wurde Jabber als Transportschicht wegen des dezentralen Aufbaus, der es jedem ermöglicht, einen eigenen Jabberserver einrichten, welcher sich nahtlos ins bestehende Netz eingliedert.

Für weiterführende Information zu Jabber und dessen Programmierung siehe [2].

1.1 Grundlagen des Jabber-Protokolls

Im Normalfall sind Sender und Emfpänger einer Nachricht nicht am selben Server angemeldet. Abbildung 1 zeigt, wie eine Nachricht von einem Sender über zwei Server an den Empfänger weitergeleitet wird: Der Sender sendet die Nachricht an den Server, an dem er angemeldet ist. Dieser Server leitet die Nachricht weiter an den Server des Empfängers (sofern er das nicht selber ist), der diese wiederum an den Empfänger schickt.

Jabber verwendet XML-Streams für die Kommunikation zwischen Client und Server sowie zwischen den Servern. Dabei bildet die gesamte Kommunikation von An- bis Abmelden ein langes XML-Dokument, in dem die einzelnen Nachrichten Fragmente bilden. Ein Beispiel einer solchen IM-Nachricht zeigt Listing 1.

```
<message type='chat' to='zis@swissjabber.ch/auryn'
    from='zis@swissjabber.ch/fuchur'>
  <thread>01</thread>
```

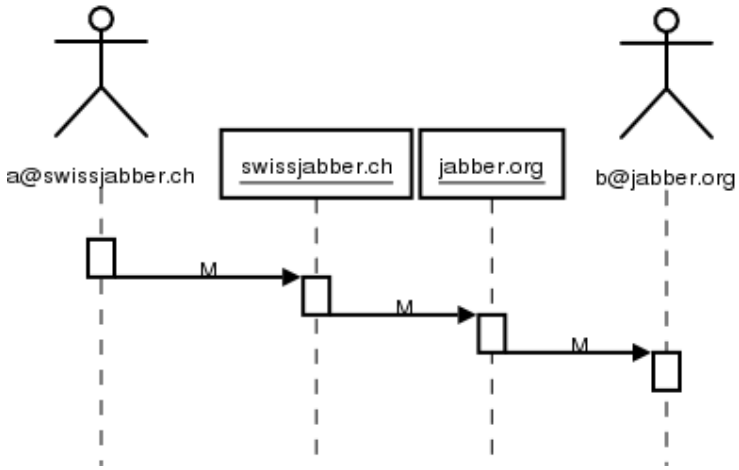

Abbildung 1: Ablauf Message versenden

```
  <body>Das ist die eigentliche Nachricht.</body>
</message>
```

Listing 1: Eine IM-Nachricht

Der Thread identifiziert Nachrichten, die zueinander gehören, denn es ist möglich, verschiedene Unterhaltungen parallel zu führen. Der Body enthält die Nachricht, die der Benutzer in seinem Diskussionsfenster angezeigt bekommt.

Neben `message`-Nachrichten werden `presence`-Nachrichten benutzt, die angeben, ob ein Benutzer gerade am Computer sitzt, keine Zeit zum Plaudern hat oder gar nicht angemeldet ist.

1.1.1 Info/Query

Als dritten wichtigen Nachrichtentyp neben `message` und `presence` gibt es `iq`, das beliebige Protokolle erlaubt. Der Name steht für Info/Query (I/Q), was an Request/Response angelehnt ist, und ausdrücken soll, dass zwischen Request und Response noch andere Nachrichten ausgetauscht werden können, die mit dem Request nichts zu tun haben. Ein Beispiel für die durch IQ möglichen Erweiterungen ist JEP 0009 (siehe [1]), das RPC über Jabber implementiert (ein Beispiel für mögliche Nachrichten gibt es am Anfang von [1]). Wie JaCal auf I/Q aufbaut, beschreibt Kapitel 3.

Erweiterbar ist Jabber über die I/Q-Nachrichten. Indem innerhalb des `iq`-Tags ein `query`-Tag mit einem Namespace untergebracht wird, können beliebige eigene Nachrichtenformate versendet werden. Die Listings 2 und 3 illustrieren die Frage nach der Client-Version über I/Q.

```
<iq type='get' from='user@host.ch/client'
    to='user2@host2.ch/client2' id='v1'>
  <query xmlns='jabber:iq:version' />
</iq>
```

Listing 2: Versions-Anfrage

```
<iq type='result' to='user@host.ch/client'
    from='user2@host2.ch/client2' id='v1'>
  <query xmlns='jabber:iq:version'>
    <name>Jabber Instant Messenger</name>
    <version>1.7.0.14</version>
    <os>95 4.10</os>
  </query>
</iq>
```

Listing 3: Antwort auf die Versions-Anfrage

Weitere Beispiele gibt es im Zusammenhang mit der Implementation von JaCal im Kapitel 3.2.

1.2 Die Jabber ID

Die Benutzer werden über die Jabber ID (JID) identifiziert, die sich aus einem Benutzernamen und einem Servernamen zusammensetzt: `zis@swissjabber.ch`. Für den Benutzer oft unsichtbar kommt die Ressource dazu: `zis@swissjabber.ch/auryn`.

Jabber erlaubt dem Benutzer, sich mehrmals anzumelden. Dabei muss pro Verbindung eine eindeutige Ressource gewählt werden, damit die vollständige JID wieder eindeutig ist. Falls der Benutzer sich mit einer Ressource doppelt anmeldet, wird die ältere Verbindung geschlossen. Diese Möglichkeit, sich mit verschiedenen Clients am Server anzumelden und über die Ressourcen gezielt einem Client eine Nachricht zu senden, nutzt JaCal aus, um bestehende Jabber-Accounts weiterverwenden zu können.

2 Use Cases

Die folgenden Use Cases definieren die Funktionalität, die eine zukünftige Version von JaCal bieten könnte. Da der Schwerpunkt dieser Semesterarbeit im Bereich des Datenaustausches liegt, ist bisher noch nicht alles vollständig implementiert.

Abbildung 2 zeigt die Bereiche, in die sich die Bedienung einer zukünftigen JaCal-Version einteilen lassen.

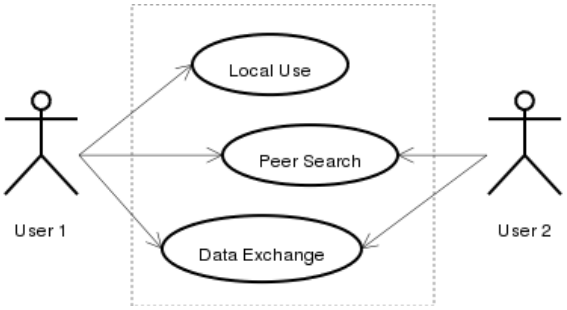

Abbildung 2: Allgemeine Use Cases

Ein User, der JaCal benutzt, hat grundsätzlich drei Möglichkeiten:

- Er arbeitet lokal (siehe Abschnitt 2.1).
- Er bearbeitet die Liste seiner Peers (siehe Abschnitt 2.2).
- Er tauscht mit seinen Peers Daten aus (siehe Abschnitt 2.3).

2.1 Lokale Use Cases

Die folgenden Tätigkeiten können ohne Netzanbindung gemacht werden:

Manage Data Der User trägt Daten ein, verändert bestehende Daten oder löscht solche. Dabei gibt es unterschiedliche Typen von Daten:

- eigene Termine
- Feiertage, Geburtstage und ähnliches
- fremde Termine

Fremde Termine können auch Feiertage, Geburtstage und Ähnliches sein. Jeder dieser Termin-Typen kann einfach oder wiederholt sein. Verschiedene Termintypen sind noch nicht implementiert.

Manage Data Groups Der User teilt seine Termine in Termingruppen ein, für die er jeweils festlegt, an welche Benutzergruppen sie verteilt werden. Die Termingruppen sind nicht zwingend disjunkt. Ausserdem kann festgelegt werden, dass eine ganze Gruppe von Terminen als neuer Termin verteilt wird (nach Möglichkeit abhängig von der Benutzer-Gruppe).

Manage User Groups Wie die Termine können auch die Peers in Personengruppen eingeteilt werden. Dadurch können Gruppen von Terminen einfach an ganze Gruppen von Personen verteilt werden.

2.2 Peer-Suche

Die Peer-Verwaltung setzt auf Jabber auf.

Diesen Bereich habe ich bisher kaum angeschaut, da er durch normale Jabber-IM-Clients erledigt werden kann. Es ist kein Problem, einen bestehenden Account mit JaCal mitzubenutzen.

2.3 Daten-Austausch

Beim Start von JaCal wird versucht, eine Verbindung zum Jabber-Server aufzunehmen.

Details zum Datenaustauschprotokoll gibt es im Kapitel 3.

Send Data Falls die Verbindung zum Server erfolgreich war, werden neuen Daten versendet. Dabei gibt es zwei Fälle:

1. Die Gegenseite ist online: Die beiden Peers gleichen ihre Daten in mehreren Schritten ab.
2. Die Gegenseite ist offline: Der Datenaustausch wird nicht durchgeführt.

Grundsätzlich cachen Jabber-Server Nachrichten, die aus irgendeinem Grund nicht zugestellt werden können. Da die meisten User ihren Jabber-Account nicht nur für JaCal benutzen werden, macht es keinen Sinn, dieses Caching der Jabber-Server zu verwenden, denn diese stellen die Nachrichten unabhängig von der Ressource dem ersten Client zu, der sich mit dem passenden Usernamen anmeldet. Das heisst, dass die meisten der für JaCal gespeicherten Nachrichten bei einem IM-Client landen würden, der diese einfach löscht.

Receive Data Falls die Verbindung zum Server erfolgreich war, werden die empfangenen Messages ausgewertet. Dabei hat der User die folgenden Möglichkeiten:

- Er akzeptiert den empfangenen Termin als einen eigenen.
- Er akzeptiert den empfangenen Termin als einen fremden.

- Er akzeptiert den Termin nicht.

3 Datenabgleich

3.1 Das Protokoll in Worten

Das Protokoll für den Datenabgleich ist einfach gehalten. Eine JaCal-Instanz, die sich am Server anmeldet, bittet den Server um seinen Roster[1] und überprüft für jeden Eintrag, ob für den angegebenen Benutzer eine JaCal-Instanz mit der Ressource „JaCal“ angemeldet ist. Jede der angemeldeten JaCal-Instanzen erhält einen Timestamp, der angibt, wann aus der Sicht der sendenden Instanz der letzte Datenabgleich stattgefunden hat.

Eine JaCal-Instanz, die einen Timestamp bekommt, überprüft, ob sie dem sendenden Peer schon einen Timestamp geschickt hat. Falls dies nicht der Fall ist, sendet sie ihren eigenen Timestamp und benutzt den früheren der beiden für das weitere Protokoll.

Sobald eine JaCal-Instanz einem Peer einen Timestamp gesendet und von ihm einen solchen empfangen hat, beginnt sie, alle seit dem Timestamp geänderten Termine in einzelnen Nachrichten an den Peer zu senden. Empfangene Termine werden einzeln bestätigt. Beim Beenden speichert JaCal die momentane Zeit für die Verwendung als nächsten Timestamp ab, sofern alle gesendeten Events bestätigt wurden. Sonst wird der alte Timestamp beibehalten[2].

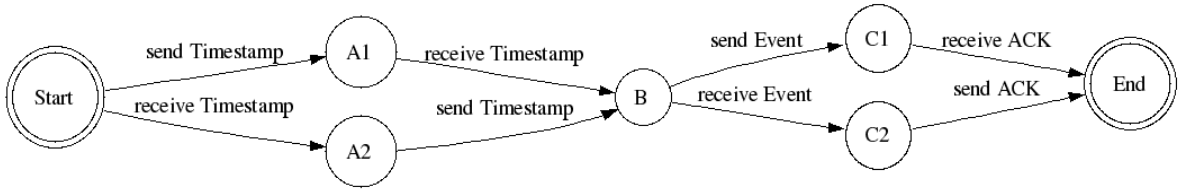

Abbildung 3: Ablauf des Verteilungsprotokolls

Bild 3 stellt das Protokoll graphisch dar. Das Protokoll besteht aus zwei Phasen:

1. Timestamp-Austausch: Der Timestamp definiert, wann aus der Sicht des sendenden Peers der letzte erfolgreiche Datenaustausch stattgefunden hat.
2. Datenaustausch: Die beiden Peers senden dem anderen die seit dem letzten Abgleich veränderten Daten zu und bestätigen sie einzeln.

Auf das Bild übertragen findet die erste Phase zwischen Start und B statt, die zweite zwischen B und End. Da JaCal das Protokoll mehrfach oder auch gar nicht durchlaufen kann, sind Start und End identisch und wurden nur für die übersichtlichere Darstellung getrennt aufgezeichnet.

3.2 Nachrichtenformate

3.2.1 Timestamp

Die Nachricht, die das ganze Protokoll auslöst, ist die Timestamp-Nachricht (siehe Listing 4). Sie gibt an, wann (nach Ansicht des sendenden Peers) der letzte Datenabgleich mit dem Emp-

[1] Die Liste der bekannten Kontakte heisst im Jabber-Jargon Roster.

[2] Als Optimierung könnten die Events chronologisch sortiert werden. Dann könnte der neue Timestamp so gewählt werden, dass die erfolgreich ausgetauschten Termine beim nächsten Mal nicht mehr ausgetauscht werden.

fänger stattgefunden hat.

```
<iq id='9wX75-4' to='zis@jabber.atreju.ch/JaCal' type='set'
                 from='franziska@jabber.atreju.ch/JaCal'>
  <query xmlns='ch:atreju:jacal'>
    <timestamp>2005-06-14</timestamp>
  </query>
</iq>
```

Listing 4: Timestamp-Nachricht

3.2.2 Event

Sobald ein Peer weiss, wann der letzte erfolgreiche Datenabgleich mit dem Peer stattgefunden hat, kann er beginnen, Events zu senden. Listing 5 zeigt eine Event-Nachricht für einen Event ohne Wiederholung.

```
<iq id='9wX75-4' to='franziska@jabber.atreju.ch/JaCal' type='result'
                 from='zis@jabber.atreju.ch/JaCal'>
  <query xmlns='ch:atreju:jacal'>
    <event>
      <id>zis@jabber.atreju.ch:13</id>
      <title>event title</title>
      <dateStart>2005-06-14</dateStart>
      <dateEnd>2005-06-14</dateEnd>
      <timeStart>12:00</timeStart>
      <timeEnd>13:00</timeEnd>
      <lastModBy>zis@jabber.atreju.ch</lastModBy>
      <lastModTime>2005-06-16 00:00</lastModTime>
    </event>
  </query>
</iq>
```

Listing 5: Event-Nachricht

Für die Wiederholung wird zusätzlich ein `repetition`-Element verwendet, welches die für die Wiederholung relevanten Daten enthält.

3.2.3 ACK

Jeder Event wird einzeln mit einer ACK-Nachricht (siehe Listing 6) bestätigt, wobei der Inhalt des `ack`-Elementes die Event-ID des bestätigten Events ist.

```
<iq id='9wX75-4' to='zis@jabber.atreju.ch/JaCal' type='set'
                 from='franziska@jabber.atreju.ch/JaCal'>
  <query xmlns='ch:atreju:jacal'>
    <ack>zis@jabber.atreju.ch:13</ack>
  </query>
</iq>
```

Listing 6: ACK-Nachricht

3.2.4 Fehler

Als Fehler können nur die folgenden Fälle auftreten:

- Der Empfänger unterstützt das JaCal-Protokoll nicht und signalisiert einen Fehler. Das kann nur vorkommen, wenn die Gegenseite Probleme mit der Initialisierung der Jabber-Library Smack hat.
- Eine Nachricht kommt nicht an. Unabhängig davon, ob Events oder ACKs verloren gehen, die Daten werden dann einfach beim nächsten Abgleich ausgetauscht.

Abgelehnte Events werden wie angenommene Termine bestätigt, da es für den Sender keine Rolle spielt, ob der Empfänger die Daten speichert oder nicht.

4 Implementation

Für die Implementation wurde Java 1.4.2 verwendet [6]. Obwohl Java 1.5.0 während der Arbeit erschienen ist, wurde auf den Versionenwechsel verzichtet.

4.1 Jabber-Subsystem

Das Jabber-Subsystem bildet eine Abstraktionsschicht über der Jabber-Verbindung und implementiert auch das JaCal-Austauschprotokoll aus Kapitel 3. Abbildung 4 zeigt ein vereinfachtes Klassendiagramm der Implementation, wobei `PacketFilter`, `IQProvider`, `PacketListener` und `IQ` von der Library Smack[3] (siehe Abschnitt 4.1.1 und [5]) zur Verfügung gestellt werden. Die nach aussen verwendbaren Schnittstellen des Subsystems sind die dunkler eingefärbten Klassen und Interfaces.

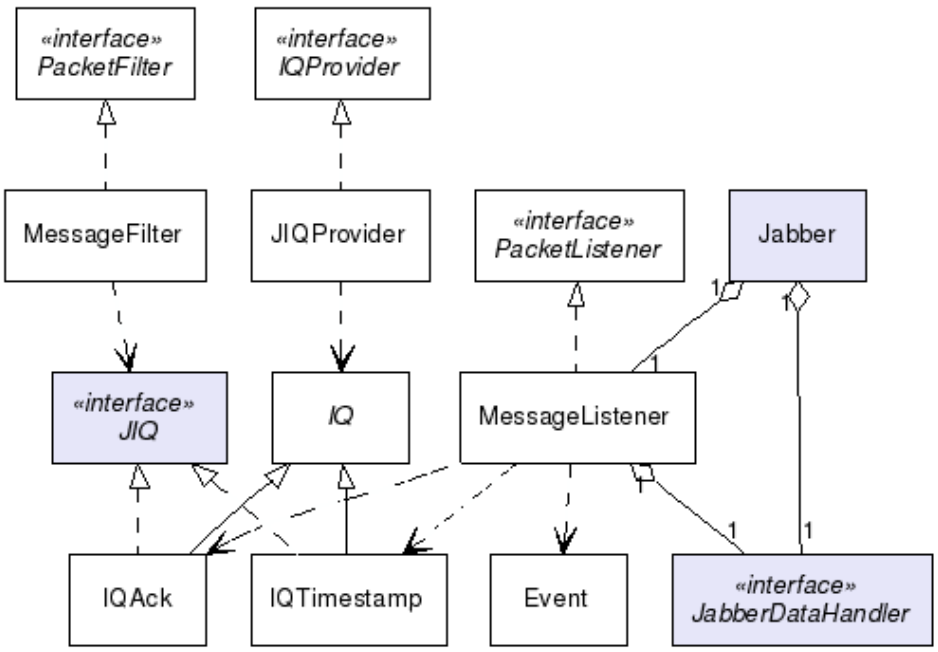

Abbildung 4: Jabber-Subsystem

[3] auf den Versionswechsel von 1.4.1 auf 1.5.0 von Smack wurde verzichtet

4.1.1 Smack

Für das Jabber-Protokoll verwendet JaCal die Library Smack, welche ein sehr einfaches Interface für IM und auch vollen Zugriff auf die Möglichkeiten von I/Q bietet. Da die Dokumentation für den Umgang mit den Erweiterungsmöglichkeiten mit I/Q eher knapp ist, folgt nun ein kurzer Überblick über die in JaCal verwendete Lösung:

- Die Datei `META-INF/smack.providers` hat folgenden Inhalt:

```
<?xml version="1.0"?>
<smackProviders>
  <iqProvider>
    <elementName>query</elementName>
    <namespace>ch:atreju:jacal</namespace>
    <className>ch.atreju.jacal.jabber.JIQProvider</className>
  </iqProvider>
</smackProviders>
```

Listing 7: Die Datei smack.providers

- Wie aus Abbildung 4 ersichtlich ist, implementiert die in `smack.providers` eingetragene Klasse `JIQProvider` das Interface `IQProvider`. Die geerbte Methode `public IQ parseIQ(XmlPullParser parser)` parst den XML-Stream und generiert dabei die verschiedenen Nachrichtentypen (siehe Abschnitt 3.2). Dabei ist es wichtig, dass der Parser am Schluss auf dem `query`-Endtag steht (`XmlPullParser` bietet ein Iterator über die XML-Elemente).
- `MessageFilter` implementiert `PacketFilter` und überprüft, ob die empfangenen (und bereits von `JIQProvider` geparsten) Pakete vom Typ I/Q sind und das Markerinterface `JIQ` implementieren.
- `MessageListener` implementiert `PacketListener` und verarbeitet in der geerbten Methode `public void processPacket(Packet packet)` die von `MessageFilter` akzeptierten Pakete. Im Fall von JaCal implementiert `MessageListener` das Datenaustauschprotokoll, indem die Klasse aufgrund des empfangenen Paketes entscheidet, was der Absender als nächstes erhalten muss.
- `Jabber.java` setzt die Komponenten zusammen, indem es die Verbindung zum Server öffnet und der Verbindung den `MessageFilter` und den `MessageListener` zuweist. `JIQProvider` wird über die erwähnte Datei `smack.providers` registriert.

4.2 Datenspeicherung

Die vielen Mehrwegbeziehungen, die sich aus Peers, Peergruppen, Events und Eventgruppen ergeben, lassen sich mit einer relationalen Datenbank relativ einfach abbilden. Doch im Normalfall haben relationale Datenkbanken den Nachteil, dass sie die Daten binär speichern und dass sie einen laufenden Datenbankserver brauchen. HSQLDB [3] umgeht diese beiden Nachteile, indem es eine eingebettete SQL-Datenbank bietet, die ihren Inhalt als Dump in Klartext auf die Festplatte schreibt. Der Zugriff läuft wie unter Java gewohnt über JDBC.

Abbildung 5 zeigt das für JaCal verwendete Datenbankschema.

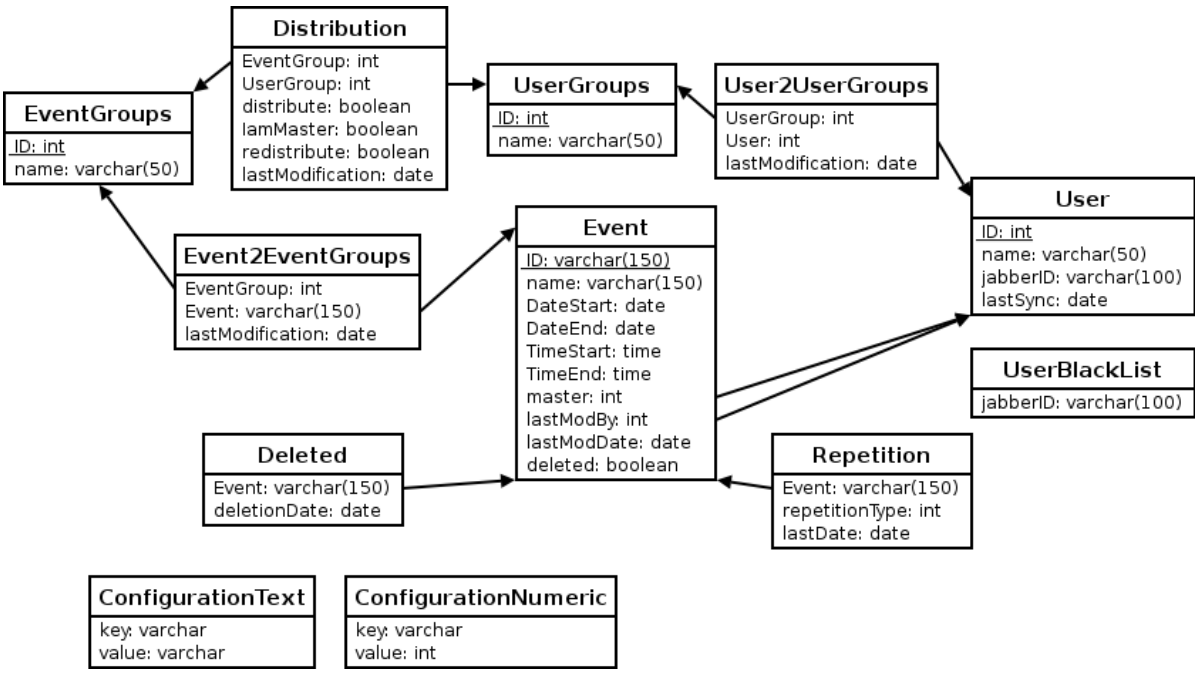

Abbildung 5: Datenbankschema

4.3 Userinterface

Das Userinterface ist in Swing implementiert, als Hilfsmittel habe ich SwiXML [7] verwendet, das eine XML-Beschreibung der Oberfläche einliest und daraus die Oberfläche baut. Da dies beim Starten gemacht wird, gibt es (im Gegensatz zu den mir bekannten graphischen GUI-Buildern) keinen Code, der von SwiXML verwaltet und nicht von Hand editiert werden darf.

5 Fazit

Die Arbeit hat gezeigt, dass ein Terminaustausch in einer anderen Form als einer Groupware ohne Probleme möglich ist. Ausserdem wurde bestätigt, dass das Jabber-Protokoll flexibel genug ist, um auch als Framework für eher artfremde Anwendungen zu dienen. Die verwendeten Libraries haben ihren Zweck erfüllt und können für weitere Arbeiten weiterempfohlen werden.

JaCal kann im momentanen Zustand benutzt werden, doch weitere Funktionalität wäre insbesondere im Bereich der einfacheren Bedienung und verbesserten Darstellung wünschenswert.

5.1 Future Work

Mögliche Erweiterungen wären natürlich in erster Linie die Vervollständigung der Implementation der im Kapitel 2 beschriebenen Funktionalität. Weiter wäre der Datenabgleich zwischen zwei Computern des gleichen Benutzers sinnvoll.

Sicher auch bald gewünscht wäre die Möglichkeit, dass ein Benutzer von überall auf seine Daten Zugriff hat, ohne seinen eigenen Computer dabeihaben zu müssen. Eine Möglichkeit

für die Realisierung wäre, die Datenbank sowie die Jabber-Infrastruktur von der Oberfläche zu trennen (diese Trennung ist bereits weitgehend vollzogen) und mit einer Web-Oberfläche zu versehen. Dadurch hätte der Benutzer von überall her Zugriff auf seine Daten, dafür müsste er auf den Zugriff ohne Netzanbindung verzichten.

Sinnvoll wäre möglicherweise die Integration einer Todo-Liste sowie einer Möglichkeit, Notizen zu erfassen. Oft hängen nämlich Termine und Aufgaben zusammen, was eine getrennte Führung von Agenda und Todo-Liste bei Terminverschiebungen mühsam macht.

5.2 Erfahrungen

Dank Jabber war die Verteilung einfacher als erwartet (einmal abgesehen von den in Abschnitt 4.1.1 angesprochenen Problemen mit Smack), während ein Kalender, der mehr kann als eine Papieragenda, doch ziemlich schnell erstaunlich komplex wird. Der Hauptgrund dafür ist das sehr vielseitige Datenmodell, das wesentlich mehr Möglichkeiten bietet als alle Kalenderapplikationen, die ich bisher gesehen habe. So sind zum Beispiel Termine, die alle zwei Wochen von Freitag Mittag bis Montag Abend dauern, kein Problem, sie können als einen einzigen Termin eingetragen werden.

Mit zur Komplexität beigetragen hat die Entscheidung, die Woche in einer Tabelle darzustellen. Aufgrund des Designs der Swing-Tabellen muss jede Zelle einzeln nur mit Hilfe ihrer Koordinaten mit Daten gefüllt werden. Das ergibt einerseits sehr viele Datenbankabfragen, wenn man im Tabellenmodell nicht alle Daten cachen will, andererseits aber auch unnötig komplizierte Queries, weil nach Datum und Uhrzeit und nicht nach Terminen gesucht wird.

Während die Oberfläche und das Datenmodell öfters für Kopfzerbrechen sorgten, machte die Arbeit mit Jabber Spass. Smack bietet einen einfachen Debugger, der alle gesendeten, empfangenen und verarbeiteten Pakete anzeigt (die in Abschnitt 3.2 gezeigten Pakete wurden aus diesem Debugger kopiert), der sehr hilfreich ist. Ich würde jederzeit wieder ein Projekt mit Jabber und Smack machen.

Zu meinem Arbeitsstil habe ich gelernt, dass die völlige Freiheit, die mir mein Betreuer liess, durchaus auch ihre Tücken hatte. Einerseits konzentrierte ich mich manchmal zu lange auf Sachen, die eigentlich gar nicht so wichtig und schon gar nicht dringend waren, andererseits war auch der Zeitdruck so klein, dass ich im ersten Semester zuviele Vorlesungen besuchte und die Arbeit schlussendlich um ein Semester verlängern musste.

Literatur

[1] ADAMS, DJ, *Jabber-RPC*, 2002
URL http://www.jabber.org/jeps/jep-0009.html

[2] ADAMS, DJ, *Programming Jabber*, O'Reilly, 1. Aufl., January 2002, ISBN 0–596–00202–5

[3] *HSQLDB 1.7.1*
URL http://hsqldb.sourceforge.net/

[4] JABBER SOFTWARE FOUNDATION, *RFC 3920: Extensible Messaging and Presence Protocol: Core*, 2004
URL http://www.ietf.org/rfc/rfc3920.txt

[5] JIVE SOFTWARE, *Smack API 1.4*
URL http://www.jivesoftware.org/smack/

[6] SUN MICROSYSTEMS, INC., *Java 1.4.2*
URL http://java.sun.com/

[7] *SWIXML*, This product includes software developed by the SWIXML Project
URL http://www.swixml.org/