Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminararbeit

Thomas Steinberg

Erstellung eines universellen Konzepts
zur Kommunikation auf mobilen Geräten mit Hilfe
von Positionsbestimmung.

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

Einleitung

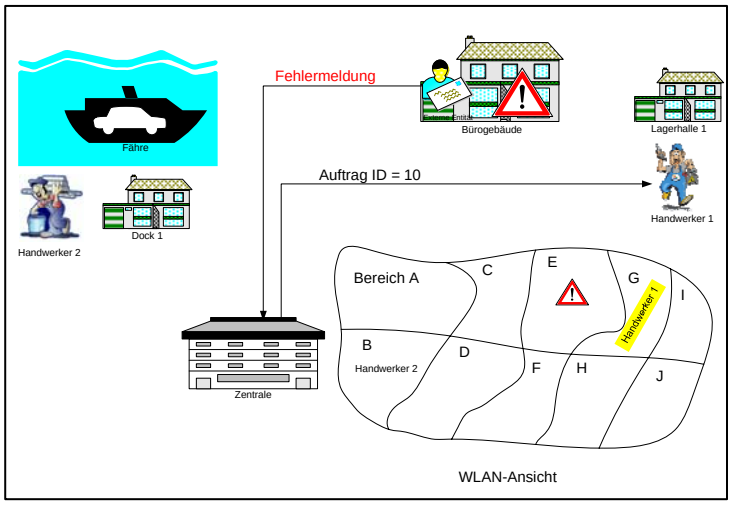

Abb. 1 Szenario Werft: Auftragsvergabe

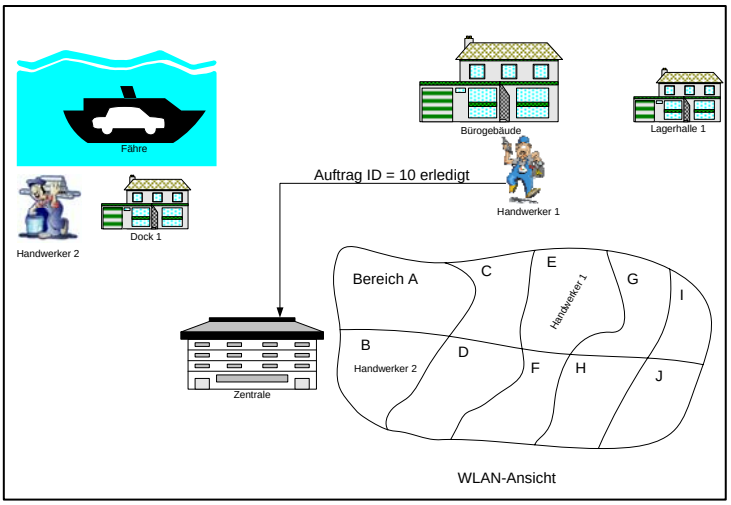

Abb. 2 Szenario Werft: Auftragserfüllung

Stellen sie sich folgendes Szenario mal vor. Sie befinden sich in einer Werft die übers ganze Gelände mit einem WLAN Netzwerk ausgestattet ist. Das Gelände beinhaltet mehrere Docks, Bürogebäuden, Lagerhallen und eine Zentrale. Im Bürogebäude1 bricht ein Rohr und ein Handwerker, vorzugsweise ein Klempner, wird dringend gebraucht. Einer der Sachbearbeiter im Bürogebäude meldet den Schaden in der Zentrale. In dieser wird ermittelt, wo der Schaden entstanden ist und welcher Handwerker mit der geeigneten Qualifikation sich in der Nähe befindet. Diesem Handwerker, nennen wir ihn „Handwerker1“ wird der Auftrag mit einer hohen Priorität (Server-Push) zugeteilt. Der Handwerker der mit einem PDA ausgestattet ist quittiert den Eingang des Auftrags (Client-Pusch) und meldet ihn in Bearbeitung (Abb. 1 Szenario Werft: Auftragsvergabe). Nach getaner Arbeit wird der Zentrale gemeldet, dass der Schaden behoben ist (Abb. 2 Szenario Werft: Auftragserfüllung).

Motivation

Kommt dieses Szenario so oder ähnlich nicht auch an anderen Stellen vor? Könnte man sich nicht vorstellen, dass man in einem Flughafen mit mehreren Terminals jeden Fluggast mit einem PDA ausstattet und das Netzwerk so aufbaut, dass man die Position nicht nur von Handwerkern, sondern auch von Passagieren feststellt. Somit könnte eine Software überwachen, wo sich die Passiere aufhalten und sie gegebenenfalls 15 min vor ihren Abflug darüber informieren, dass sie sich im falschen Terminal befinden.
Oder stellen sie sich mal den Ferienklub vor, wo sich Feriengäste für ein Ereignis wie das Wasseraerobic angemeldet haben. Der Swimmingpool durch einen Schadensfall aber in dieser Zeit nicht benutzbar wird. Dann musste nicht nur der nötige Handwerker, sondern auch die angemeldeten Gäste informiert werden.
Diese Liste lässt sich fast unendlich weiter spinnen, somit kann man diese Szenarien auf verschiedene Orte, wie z.B. Einkaufszentren, Fabriken oder Bahnhöfe reproduzieren.

All diese Szenarien zeigen ein gemeinsames Schema auf, wieso sollte man diese Kommunikationsstruktur nicht einheitlich gestallten und sie für mehrere Szenarien wieder verwendbar machen.

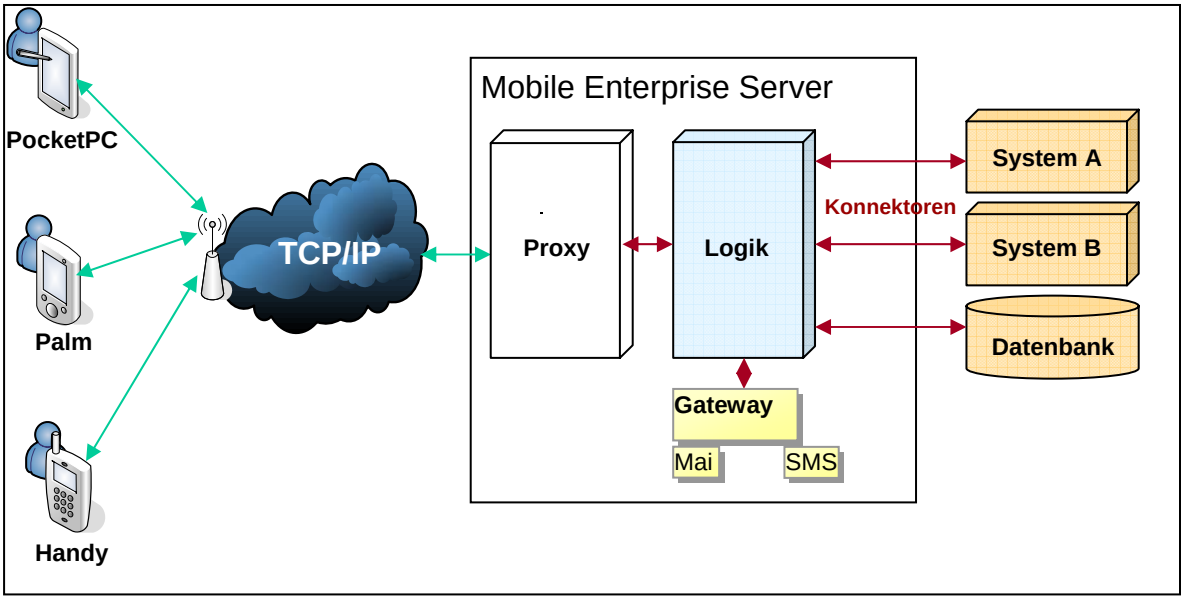

Abb. 3 Kommunikationsstruktur

Die Kommunikationsstruktur könnte folgendermaßen aussehen (siehe Abb. 3 Kommunikationsstruktur). Man hat mehrere mobile Client's (PocketPC's, Palm oder Handys) diese sind über ein TCP/IP Netzwerk mit einem Server verbunden, nennen wir diesen „Mobile Enterprise Server" oder kurz MES. Der MES-Server bietet eine Kommunikationsschnittstelle zum TCP/IP Netzwerk über einen Proxy. Desweitern besitzt dieser Konnektoren um die Logik mit anderen Drittsystemen, wie Systemen zur Auftragsverwaltung oder Systemen die Lokalisierungsaufgaben übernehmen, zu verbinden. Der MES-Server sollte auch Schnittstellen in Form eines Gateway zur Kommunikation über Mail oder SMS zur Verfügung stellen um Administrationsinformationen an vorgesehne Personen zu schicken.

Mehrwert

Die Struktur diese Konzeptes würde einen enormen Mehrwert für die Gesellschaft bringen, indem eine Kommunikationsstruktur an vielen verschieden Orten eingesetzt werden kann und nur an individuellen Stellen in der Logik des Clients und der MES-Logik den individuellen Ansprüchen des Kunden angepasst werden muss.

Anforderungen an das System

Die Anforderungen an solch eine Middleware, wären umfangreich.

- Die Middleware müsste ein möglichst kleines Datenübertragungsvolumen haben um die Kosten bei Systemen wie GPRS[1] oder UMTS[2], die nach Transfervolumen abgerechnet werden, möglichst gering zu halten.
- Um die Übertragung vom Client, sowie vom Server aus zu starten muss die Middleware auch asynchrone Kommunikation unterstützen. Hier wären auch andere möglichen Konzepte vertretbar, wenn sie sie nicht gegen Anforderung eins „möglichst kleines Transfervolumen" verstoßen würden.
- Um auf die Funktionen bereits bestehender Produkte zurückzugreifen müssen entsprechende Schnittstellen geschaffen werden. Die Schnittstellen müssen allgemein gehalten werden, um sie für viele verschiedene Produkte wiederverwendbar zu machen.

[1] GPRS: "General Packet Radio Service", engl. „Allgemeiner paketorientierter Funkdienst"
[2] UMTS: Universal Mobile Telecommunications System

Steinberg-Thomas@web.de

- Um ein gewisses Maß an Sicherheit für die Middleware zu schaffen, muss es für die Benutzer eine Art der Authentisierung und eine Verschlüsselung der Daten geben, um sie vor Dritten zu schützen.
- Damit mehrere Benutzer mit dem System arbeiten können, muss eine Art der Benutzerverwaltung eingerichtet werden. Die die Hardware den Benutzer und die Benutzer der jeweiligen Hardware zuordnen kann.
- Wenn man eine Art der Presence Funktion braucht müssen so genante buddy listen [3] geführt werden, um jeder Zeit darüber informiert zu sein, wer in diesem Moment verfügbar ist oder nicht.

Technologien

Das Kapitel Technologie soll die Struktur der Middleware, aber auch die verschiedenen Arten der Kommunikationsprotokolle, die für diese Middleware denkbar wären veranschaulichen.

Protokolle für Kommunikationsschnittstellen

Es gibt eine Vielzahl von Kommunikationsprotokollen. Natürlich können nicht alle hier aufgeführt werden, deswegen werden exemplarisch drei Standardprotokolle mit denen eine Implementierung des Konzepts mögliche wäre vorgestellt.

HTTP[4]

HTTP ist ein etablierter Protokollstandard, der auf allen hier (Kapitel Entwicklungsplattformen) aufgeführten Entwicklungsplattformen genutzt werden kann, ohne externe Bibliotheken einbinden zu müssen. Der HTTP Standard basiert auf einem Request/Response Modell und definiert unterschiedliche HTTP-Methoden. Zu den gebräuchlichsten Methoden zählen GET und POST. Mit einem GET fordert der Browser direkt Ressourcen vom Webserver an. Das kann eine HTML-Seite oder eine Datei sein. Um Daten vom Browser an den Webserver zu schicken wird POST verwendet, dies kann in Form z.B. eines HTML Formular sein.

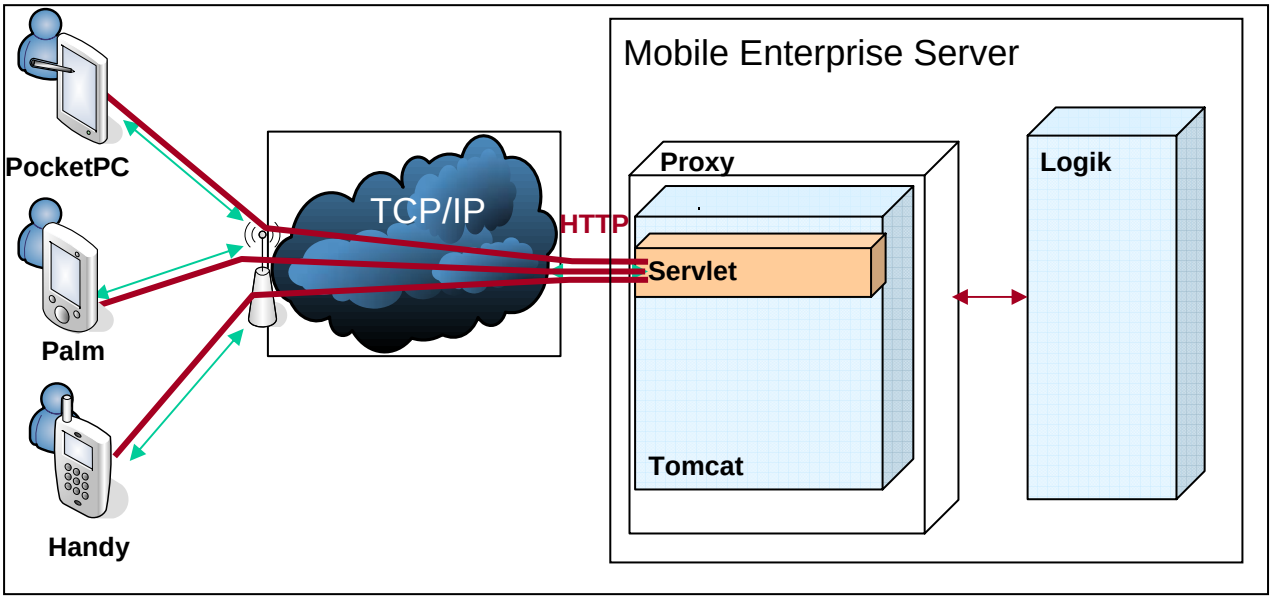

Abb. 4 Mögliche Implementierung über HTTP

[3] Buddy list: Liste der online/offline verfügbaren Freunden/Bekannten/Mitarbeiter
[4] HTTP: Hypertext Transfer Protokoll

Steinberg-Thomas@web.de

In diesem Lösunkszenario (Abb. 4 Mögliche Implementierung über HTTP) ist die Umsetzung des Konzepts durch einen Apache Tomcat4 Servlet Containers, der ein Java Servlet beinhaltet, realisiert. Gemeinsam stellen diese den Proxy dar. Alle Verbindungen der mobilen Clients zum MES, sowie die Sitzungsverwaltung werden vom HTTP-Konnektor des Tomcat-Servers gesteuert. Eine Sitzung wird automatisch ungültig gesetzt, wenn der Benutzer über einen bestimmten Zeitraum keine Anfragen ausgeführt hat.
Die Kommunikation würde folgendermaßen ablaufen: Das Java-Servlet nimmt alle HTTP Anfragen auf und leitet sie transparent an die Logik weiter. Diese verarbeitet die Daten und generiert eine Antwort, welche anschließend über den MES-Proxy an den mobilen Client übermittelt wird. Dies geschieht, weil der MES-Proxy den Response, solange wie das Logikmodul die Anfrage bearbeitet, zurück hält. Erst wenn die Antwort von der Logik kommt wird der Response freigegeben und die Antwort darin übertragen.
Auch die Verschlüsselung der Daten ist mit HTTP in der Form des HTTPS[5] möglich und erfüllt einer der Anforderungen an das System.
Leider hat das Protokoll mehrere Nachteile für die mobile Welt und somit auch für unser universelles Konzept. HTTP hat einen recht hohen Overhead bei der Übertragung durch den HTTP-Header, somit ist es nicht wirklich effizient bei kleinen Datenmengen. Durch das Request/Response Modell das HTTP zugrunde liegt, ist es den Servlet nicht möglich eine Verbindung zum mobilen Client zu initiieren (Server push). Der Client ist gezwungen in regelmäßigen Abständen den Server anzupollen, um den aktuellen Stand der für ihn gerichteten Nachrichten zu erfahren. Ein solche Polling-Verfahren verursacht unnötigen Transfervolumen, und somit Kosten bei Verbindungen über GPRS oder UMTS. Es wäre natürlich möglich eine Verbindung vom Server in Richtung Client mit HTTP zu realisieren, hierfür müsste auf dem mobilen Client ein HTTP–Server aufgesetzt werden, der die Anfragen vom MES-Server beantwortet. Diese Lösung belastet die heutigen Geräte in Punkto Speicherplatz und Prozessorleistung zu sehr, dass sie nicht mehr in der Lage wären etwas anderes zu tun.

SOAP[6]

SOAP ist ein standardisiertes Protokoll für den Informationsaustausch in einer heterogenen, verteilten Welt. Die SOAP Spezifikation umfasst einen einfachen XML Container, den SOAP Envelope, in dem die Informationen als Nachricht verpackt werden. Der Envelope besteht aus einem optionalen Header und einem Body indem die eigentliche Nachricht verpackt ist. Der Envelope gibt Information darüber, um was für eine Nachricht es sich handelt, wie sie zu verarbeiten ist und in wessen Zuständigkeit die Verarbeitung fällt. Es gibt zwei Arten von SOAP Nachrichten: Nachrichten zum Transport von XML Dokumenten (document-style SOAP) oder Nachrichten, mit denen ein in XML formulierte Remote Procedure Call(RPC) ausgeführt werden kann.
Mit der RPC-Variante kann via SOAP eine Funktion (bzw. Prozedur oder Methode) auf einem entfernten Rechner ausgeführt werden. Die XML Daten im SOAP Body beschreiben den Namen der Funktion sowie deren Übergabeparameter.

SOAP ist von der Transportschicht unabhängig, muss aber auf eine Art transportiert werden. Die Spezifikation des W3C5 bezieht sich auf den Transport per HTTP, wobei sich der Overhead drastisch erhöht. Das RPC Prinzip passt gut zu dem Request/Response Modell, dass SOAP mit HTTP als Transportprotokoll adaptiert. Das Ergebnis des Funktionsaufrufes wird einfach als SOAP Nachricht im HTTP-Response an den Aufrufer zurückgeliefert.

5 HTTPS: Hypertext transfer protocol secure
6 SOAP: Simple Object Access Protocol

Steinberg-Thomas@web.de

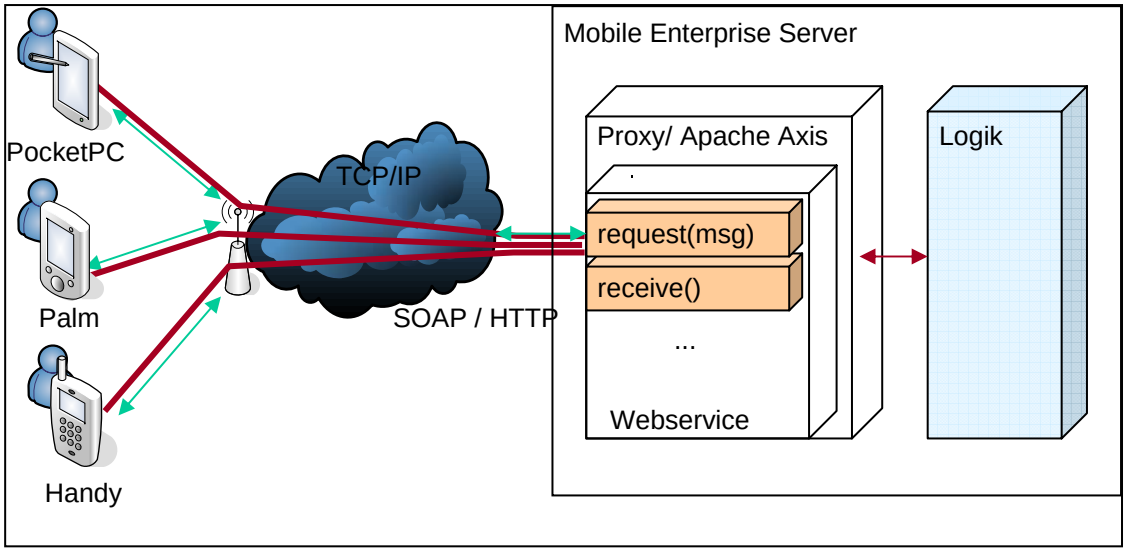

Abb. 5 Mögliche Implementierung über SOAP

In diesem Lösunkszenario (Abb. 5 Mögliche Implementierung über SOAP) ist die Umsetzung des Konzepts durch einen Web Service implementiert. Ein Web Service ist eine Applikation, deren Schnittstellen mit Hilfe von Internetprotokollen offen gelegt und verwendbar werden. Schnittstellen eines Web Services können nicht nur über HTTP, sondern auch beispielsweise mit dem Versand und dem Empfang von E-Mails über das Simple Mail Transfer Protocol (SMTP) genutzt werden. Der Web Service stellt den Proxy dar. Der mobile Client kann jetzt den Web Service nutzen um mit dem MES zu kommunizieren. Hierzu stellt der Web Service dem Client verschiedene Funktionen zu Verfügung. Auch eine Authentisierung des mobilen Benutzers kann mit Hilfe einer login- Funktion behandelt werden. Zum Empfangen von Nachrichten kann der mobile Client eine spezielle receive-Funktion verwenden.

Die Vorteile eines Web Service liegen in der einfachen Implementierung auf der Server Seite in Form des Web Services und auf der (mobilen) Client Seite in Form von Stubs des Web Services. Sie bieten den Client-Applikation Schnittstellen, die den Zugriff auf den Web Service transparent machen. Für die Erzeugung von Stubs gibt es diverse Toolkits wie z.B. Wireless Toolkit von Sun, es bringt einen Stub-Generator mit sich der Stub-Klassen für CLDC/MIDP Plattformen erzeugt. Als Grundlage für die Stub-Erzeugung wird eine WSDL-Datei (Web Service Description Language) benötigt. Die WSDL ermöglicht eine genaue Beschreibung von Web Services im XML-Format.
SOAP ist vor allem unabhängig vom Betriebssystem und der Entwicklungsplattform und bietet somit viele Möglichkeiten. In Verbindung mit HTTP kann SOAP auf allen CLDC/MIDP Profilen eingesetzt werden, ist aber auch auf dem .NET Compact Framework implementierbar. Entsprechende Bibliotheken für SOAP werden direkt von Sun kostenlos zum Download angeboten.
Lizenzkosten fallen nicht an, wenn Open-Source Projekte wie Axis für die Kommunikationsschnittstelle eingesetzt werden. Die Sitzungsverwaltung kann bei SOAP im HTTP-Server mit Hilfe von HTTP-Sessions erfolgen.

Leider gibt es kein perfektes Protokoll, auch wenn es sich am Anfang so anhört. Wenn man SOAP via HTTP benutzt, ist der Overhead durch den zusätzlichen HTTP-Header noch größer als bei der reinen HTTP Lösung. Der Aufruf einer einfachen Funktion erfordert bereits eine Menge Protokolldaten. Auch bei der Benutzung von SOAP via HTTP ist ohne weiteres keine vom Server initiierte Verbindung zum mobilen Client möglich. Der mobile Client muss ein Polling-Verfahren verwenden, um bereitliegende Nachrichten zu empfangen. SOAP kann zwar mit anderen Protokollen als HTTP kombiniert werden, um einen asynchronen Empfang von Nachrichten zu ermöglichen, doch wurden im Rahmen der Recherche keine brauchbaren Bibliotheken für die mobilen Plattformen gefunden.

XMPP[7]

XMPP ist der Kern des Jabber Protokolls, dieser wurde von Jabber Software Foundation (JSF) angepasst und als Standard verabschiedet.

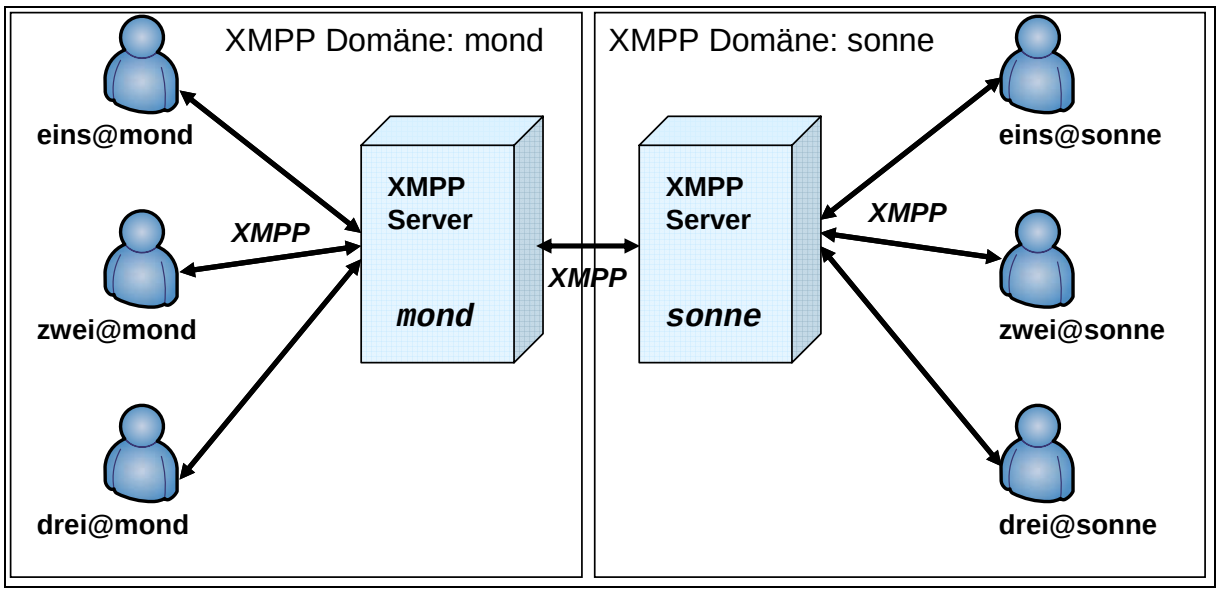

Abb. 6 Domäneverhalten von XMPP

Im Allgemeinen ist der Aufbau des Protokolls schnell erklärt. Jeder Server besitzt seine eigene Domäne mit den zughörigen Benutzern. Somit können bei größeren Infrastrukturen mehrere Server jeweils mit ihren Domänen miteinander verbunden werden.
Jeder XMPP Benutzer (Client) hat eine eigene eindeutige Adresse, die aus historischen Gründen als Jabber ID (JID) bezeichnet wird. Die JID ähnelt einer E-Mail Adresse, da sie aus den Benutzernamen und den Domänennamen, getrennt durch ein @-Zeichen, zusammengesetzt wird. Die Abbildung (Abb. 6 Domäneverhalten von XMPP) zeigt einen allgemeinen Aufbau mit zwei Domänen und jeweils drei Benutzern. Da ein Benutzer gleichzeitig an mehreren Geräten sich an einem XMPP-Server anmelden kann, wurde die JID mit einer Angabe der Ressource erweitert. Eine vollständige JID lautet also **benutzername@domänenname/ressourcenname**.

Die Kommunikation zwischen den Clients geschieht über sogenannte Kurznachrichten, die sich in Subnachrichten unterteilen, dazu aber später mehr. Die Nachrichten werden zwischen den Benutzern unter Angabe der JID als Empfängeradresse ausgetauscht. Jeder Benutzer besitzt eine Kontaktliste, die über den aktuelle Online Status von anderen Benutzern informiert. Dabei liegt es in der Entscheidung des jeweiligen Benutzers, wen er seiner Kontaktliste hinzufügt und ob er in anderen Kontaktlisten erscheinen möchte. In der Regel kann der XMPP Server so eingerichtet werden, dass er Nachrichten zwischenspeichert, wenn der Empfänger gerade nicht erreichbar ist. Die Nachrichten werden dem Empfänger dann nachträglich zugestellt, wenn er sich das nächste Mal mit dem Server verbindet.

Sieht man sich eine Ebene darunter an, besteht eine XMPP Verbindung aus einem XML-Stream in dem sog. XML-Stanzas[8] (Kurznachrichten) verschickt werden. Der Standard sieht als Transportebene TCP/IP-Verbindungen vor, die für die Dauer einer Sitzung aufrecht gehalten werden. Diese Verbindung, ermöglicht den Client sowohl auch den Server jederzeit asynchron Daten zu übertragen. Zu Beginn einer XMPP Verbindung wird der XML-Stream initiiert, indem sich beide Endpunkte einen <stream>-Tag übermitteln. Der XML-Stream bleibt solange aktiv, bis ein Endpunkt die Verbindung mit einem </stream>-Tag beendet.

[7] XMPP: Extensible Messaging and Presence Protocol
[8] XML-Stanzas: XML Kurznachricht(strukturierte Informationsblöcke)

Steinberg-Thomas@web.de

Dazwischen können die Endpunkte eine unbegrenzte Anzahl an XML-Stanzas über den XML-Stream schicken. XML-Stanzas sind strukturierte Informationsblöcke, die in einem XML-Stream übertragen werden. Der XMPP Standard legt hauptsächlich drei Typen von XML-Stanzas fest: <*message*/>, <*presence*/>und <*iq*/>. Daneben existieren XML-Stanzas für die Aushandlung von Sicherheitsfunktionen. *Message-Stanzas* dienen für den Transport von Kurznachrichten zwischen den Benutzern. *Presence-Stanzas* informieren den Server oder einen Client über den Online-Status eines Benutzers. *IQ-Stanzas* bilden einen synchronen Request-Response Mechanismus in dem sonst asynchron gearteten XMPP-Protokoll.

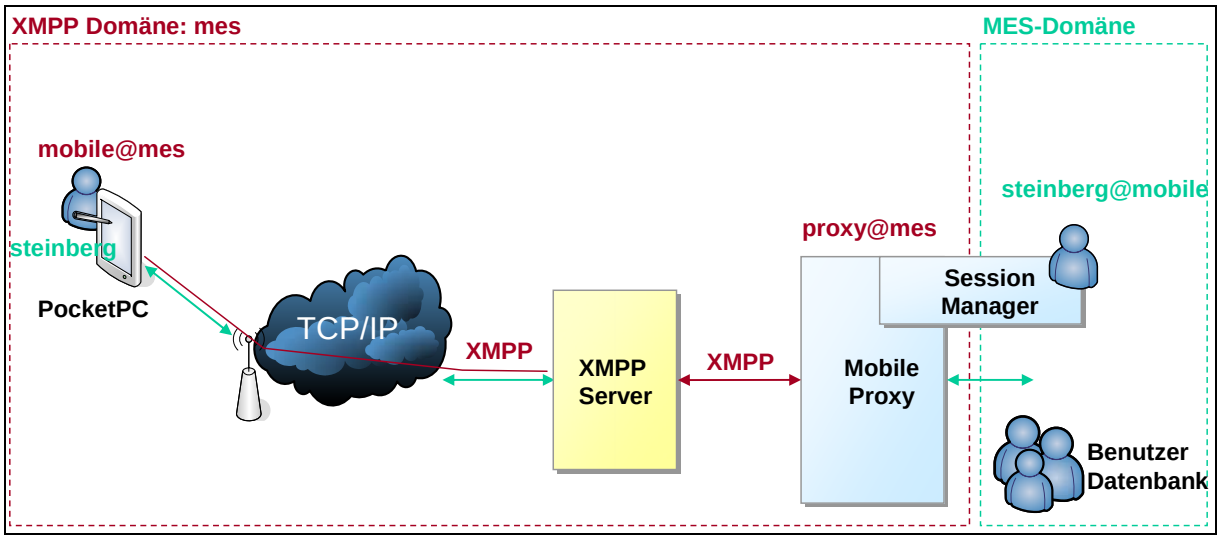

Abb. 7 Benutzer und Ressource bei XMPP

Die Abbildung (Abb. 7 Benutzer und Ressource bei XMPP) zeigt, wieso XMPP als „Exentsible“ bezeichnet wird. Den Anforderungen entsprechend können eignene Kindelemente in einem Namespace definiert werden, somit werden sogenannte XMPP-Extension erzeugt. Wird dieses XML-Stanzas an einen Empfänger verschickt der diese Kindelemente nicht versteht, wird die Nachricht vom Empfänger einfach verworfen und der Absender wird darüber informiert.

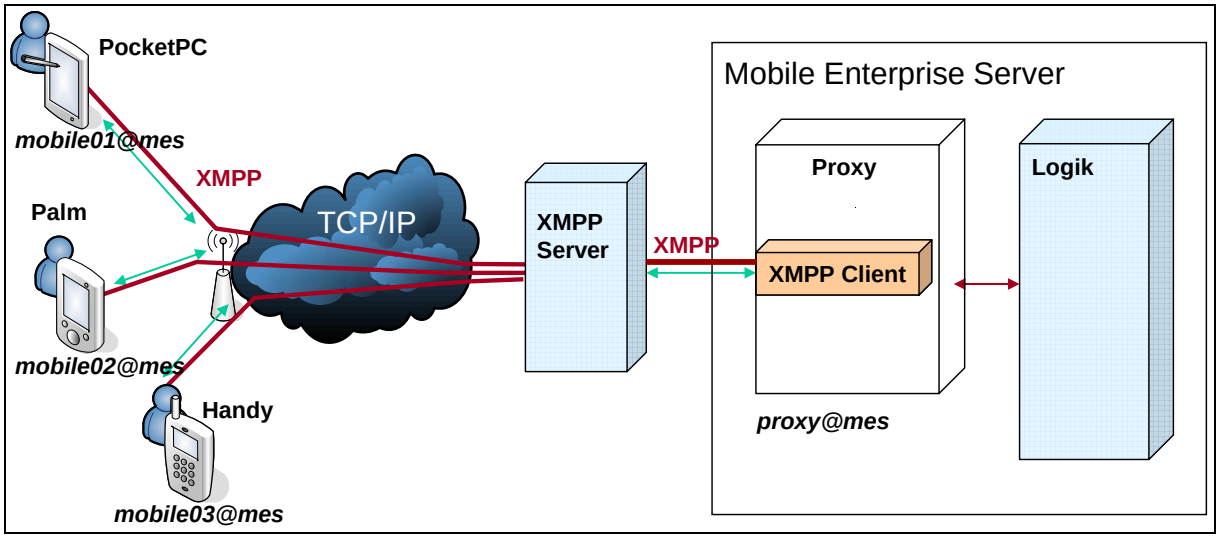

Abb. 8 Mögliche Implementierung über XMPP

Die Abbildung(Abb. 8 Mögliche Implementierung über XMPP) zeigt ein mögliches Lösungszenario mit XMPP. Es wird ein XMPP-Server verwendet der mehrere Clients und den MES-Proxy in seiner Domäne verwaltet. Gegenüber dem XMPP Server tritt der MES-

Steinberg-Thomas@web.de

Proxy wie ein normaler XMPP-Client auf. Sowohl die mobilen Clients, als auch der MES-Proxy erhalten eine JID, mit der sie eindeutig angesprochen werden können.
Bereits in diesem einfachen Aufbau können die Clients miteinander Standard-Nachrichten austauschen. Das gibt den Server (MES-Proxy) bereits die Möglichkeit aktiv Kurznachrichten an die mobilen Clients (server push) zu schicken.Zudem hat der Server über den MES-Proxy die Übersicht über den Online Status jedes einzelenen Clients, der sich am XMPP Server angemeldet hat und in der Kontaktliste vom MES-Proxy geführt wird.

Mögliche Implementierung

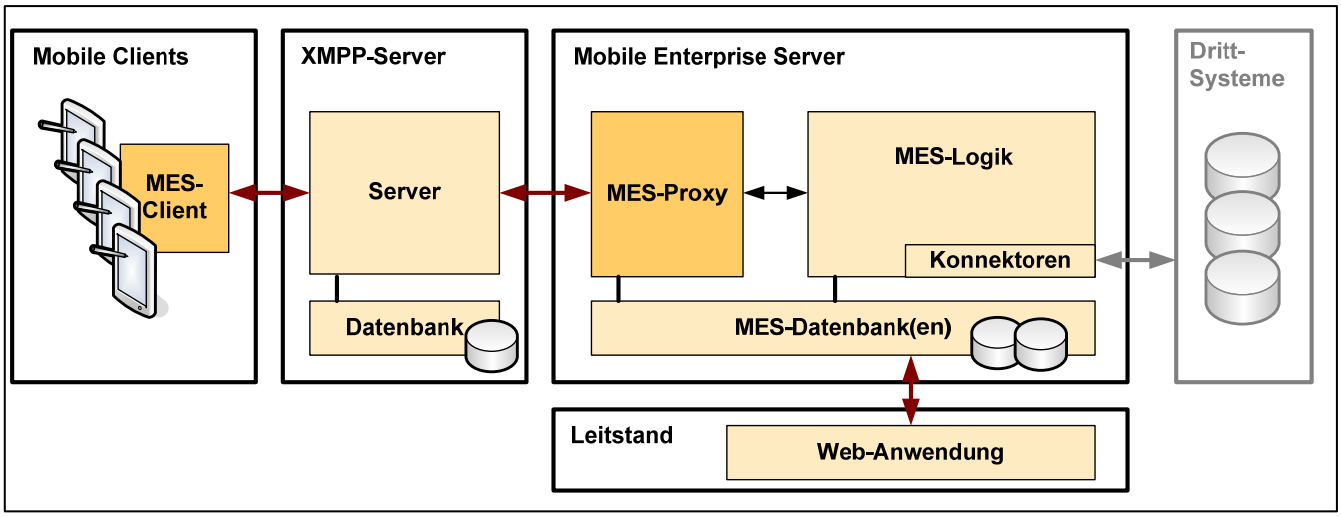

Abb. 9 Mögliche Implementierung mit XMPP

In der Abbildung (Abb. 9 Mögliche Implementierung mit XMPP) wird detallierter auf die mögliche Implementierung mit XMPP eingegeangen. Der XMPP-Server ist das Bindeglied zwischen den mobilen Clients und dem MES-Server. Das XMPP-Protokoll bietet eine kosten günstige Verbindung zu den mobilen Clients und gewährleistet eine asynchrone Verbindung vom Sender zum Empfänger.

Der MES-Server ist losegekoppelt durch eine TCP/IP-Verbindung über den MES-Proxy mit dem XMPP-Server verbunden. Aus Sicht des XMPP-Servers agiert der MES-Proxy dabei wie ein „normaler“ Client. Der MES-Server kann über verschiedene Konnektoren mit Drittsystemen verbunden werden, die in unseren Fall, die Auftragsverwaltung und die Lokalisierung von Personen übernehmen können. Alle Daten, die für den Betrieb der MES-Lösung notwendig sind, werden in einer eigenen Datenbank gehalten. Um diese Bedingung zu gewährleisten muss die Möglichkeit der Synchronisation von Daten gewährleistet werden, damit MES keine Daten im Drittsystem mit direktem Zugriff manipulieren muss.

Mit der Leitstand Anwendung soll auf den Datenbestand im MES zugegriffen werden, z.B. damit der Disponent neue Störaufträge anlegen oder vorhandene betrachten kann.

Mobile Clients

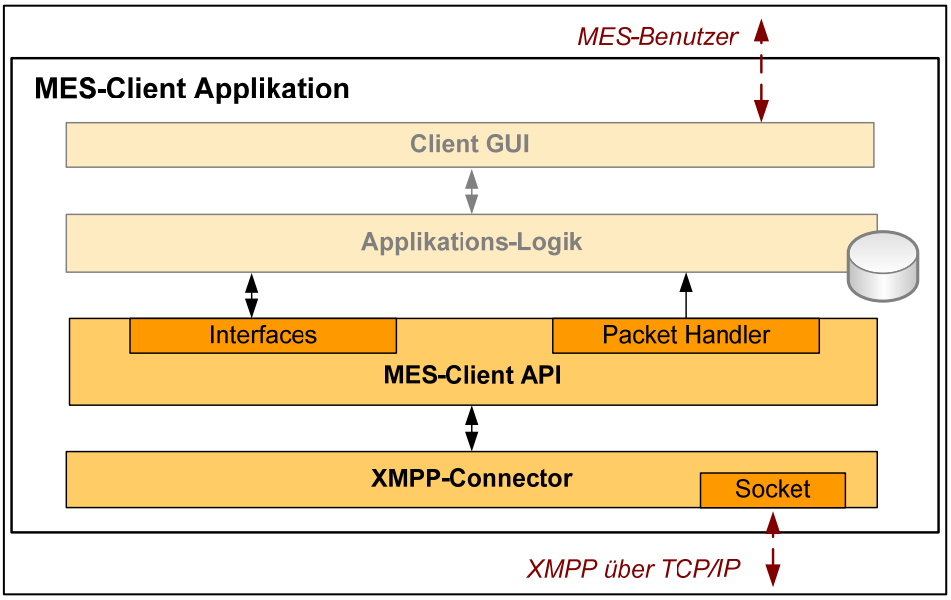

Abb. 10 Implementierung des mobilen Clients

In der Abbildung (Abb. 10 Implementierung des mobilen Clients) wird die mögliche Struktur des mobilen Clients aufgezeichnet. Diese setzt sich aus der Client Gui, der Applikations-Logik, der MES-Client API und dem XMPP Connector zusammen.
Die **Client Gui** ist die Kommunikationschnittstelle zwischen den Benutzern und dem System. Die Daten die der Benutzer in die GUI eingibt werden von der **Applikation Logik** entgegengenommen und auf der Paket Ebene verarbeitet. Unter einem Paket wird in diesem Zusammenhang eine Datenstruktur verstanden, die ein XML-Stanza abbildet, z.B. ein Objekt vom Typ IQ-Packet. Wenn die Pakete verschickt werden sollen, werden Funktionen und Methoden von der **MES Client API** benutzt, in der Abbildung allgemein als **Interfaces** bezeichnet. Die MES-Client API soll nach einem asynchronen Prinzip arbeiten, d.h. Aufrufe zur API sind nicht-blockierend. Der Aufrufer einer API-Funktion erhält sofort die Kontrolle zurück, im Hintergrund sendet die MES-Client API XML-Stanzas an den MES. Währenddessen ist die Anwendung nicht durch das Warten auf die Antwort blockiert. Auch in die entgegengesetzte Richtung soll es ähnlich ablaufen.
Ein **Packet Handler** empfängt innerhalb der MES-Client API ankommende XML-Stanzas vom XMPP-Connector und wandelt sie mit Hilfe eines XML-Parsers in Pakete um. Anschließend informiert der Packet Handler die Applikations-Logik über die Ankunft eines neuen Paketes (per Event oder Listener). Dabei werden nur die „rohen“ Pakete an die Logik übergeben. Was mit dem Paket passiert, bzw. welche Reaktion auf ein bestimmtes Paket erfolgt, muss in der Applikations-Logik festgelegt werden. Somit bleibt die MES-Client API wiederverwendbar, während die Applikations-Logik ohnehin neu auf ein kundenindividuelles Projekt angepasst werden muss.
Für den Aufbau der TCP/IP Verbindung und den Aufbau für den XML-Stream ist der **XMPP Connector** zuständig. Je nach Entwicklungsplattform kann der XMPP-Connector mit einer fertigen Bibliothek (Open-Source oder kommerziell) abgebildet werden. Um eine XMPP-Verbindung aufzubauen, öffnet der Connector einen TCP/IP Socket und initiiert einen XML-Stream mit dem XMPP-Server. In dem XML-Stream werden XML-Stanzas asynchron versendet und empfangen. Der XMPP-Connector stellt für den Versand von IQ-, Message- und Presence- Stanzas Schnittstellen bereit, die von der MES-Client API genutzt werden.

MES-Proxy

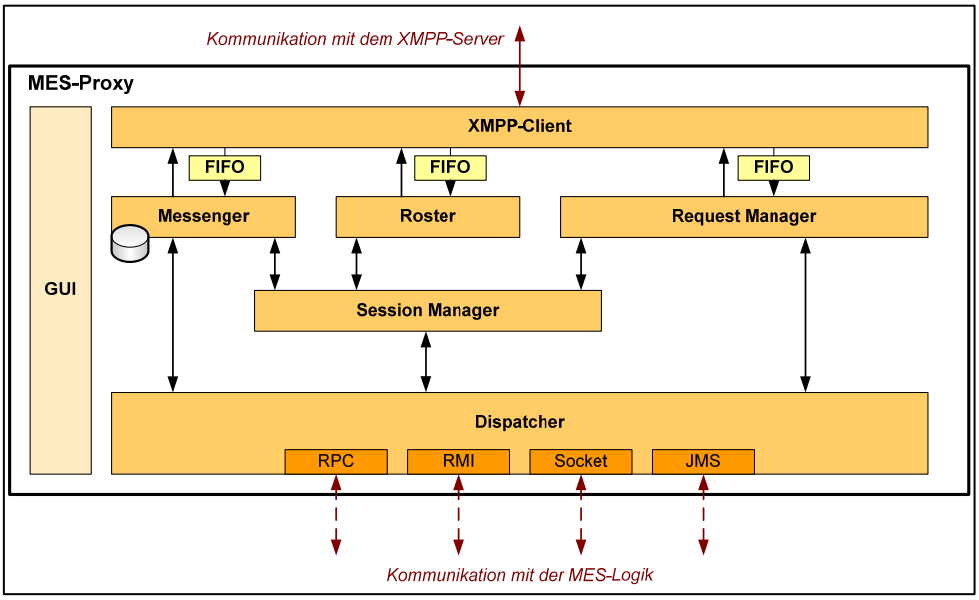

Abb. 11 Implementierung des MES-Proxy

In der Abbildung (Abb. 11 Implementierung des MES-Proxy) wird die mögliche Struktur des MES-Proxy's aufgezeichnet. Diese setzt sich aus den XMPP-Client, den Messenger, den Roster, den RequestManager, den Session Manager und dem Dispatcher zusammen.
Der **XMPP-Client** bildet die Kommunikationsschicht zum XMPP-Server. Innerhalb des XMPP-Clients kann eine freie oder kommerzielle Bibliothek eingebunden werden, die eine Verbindung zum XMPP-Server ermöglicht. Die Komponente meldet sich beim Start mit Benutzernamen und Kennwort am XMPP-Server an. Die JID des XMPP-Client-Proxy gilt als Empfänger, für alle an den MES gerichteten XML-Stanzas. Eingehende XML-Stanzas mit MES-Extension werden an die Unterkomponenten zur weiteren Verarbeitung vermittelt. Beim Eingang eines XML-Stanzas entscheidet der XMPP-Client, welcher Unterkomponente das Stanza zugeteilt wird. Die Zuteilung erfolgt mittels eines zwischengeschalteten FIFO's (siehe Abb. 11 Implementierung des MES-Proxy), um Anfragen bei großer Last der MES-Logik puffern zu können. Message-Stanzas mit MES-Extension werden vom XMPP-Client an die *Messenger-Komponente* weitergereicht. IQ-Stanzas mit MES-Extension vermittelt der XMPP-Client an den *Request Manager*. Ankommende IQ-Stanzas ohne MES-Extension oder mit einer anderen Extension werden vom XMPP-Client verworfen. Der anfragende Client erhält die XMPP- Fehlermeldung <featurenot-implemented/>. Presence-Stanzas leitet der XMPP-Client an die *Roster-Komponente* weiter.

Der **Messenger** verarbeitet Kurznachrichten mit der MES-Extension, die er vom MES-Client bekommt, und stellt sie der Logik zu. Anders als bei einer IQ-Anfrage, erwartet der Absender der MES-Kurznachricht keine Antwort vom Server.
Der **Roster** verwaltet die Kontaktliste des MES-Proxy, wo die aktuellen Online-Stati der mobilen Clients aufgeführt sind. Damit der Roster den Status eines mobilen Clients vom XMPP-Server mitgeteilt bekommt, muss sich der Client zunächst der Kontaktliste hinzufügen und dem MES-Proxy erlauben, seinen Status zu abonnieren. Sobald der Roster die Meldung erhält, dass ein mobiler Client seinen Status auf „Offline" verändert hat, wird der Session Manager über dieses Ereignis unterrichtet. Zudem wird der mobile Client aus der Kontaktliste entfernt, da sich JID's durch die anonyme Anmeldung am XMPP-Server ständig ändern. Die Meldung an den Session Manager stößt dort die Zerstörung einer Sitzung an, welche aber erst nach einem vordefinierten Zeitraum (timeout) wirklich ausgeführt wird.

Dem **Request Manager** werden vom XMPP-Client die IQ-Stanzas mit MES-Extension zur Verarbeitung zugeteilt. Der Request Manager prüft zunächst, ob der anfragende mobile Client eine gültige Sitzung besitzt. Dazu wird dem Session Manager die JID des Absenders übergeben. Ist dem Session Manager eine Sitzung für die JID bekannt, erhält der Request Manager eine positive Bestätigung unter Angabe der MES User-ID.Aufgrund der vorhandenen Sitzung geht der Request Manager davon aus, dass der mobile Client sich ordnungsgemäß am Session Manager authentisiert hat. Die individuellen XML-Daten, die als <custom-data/> gekennzeichnet sind, werden aus dem IQ-Stanza und der MES-Extension extrahiert und über den Dispatcher an die MES-Logik übergeben. Nachdem die Anfrage in der MES-Logik verarbeitet wurde, übergibt sie die Antwort (<customresult/>) unter Angabe der User-ID und der IQID zurück an den Request Manager. Dort wird die Antwort in einem IQ-Stanza mit type=’result’ verpackt. Als ID wird der durchgereichte IQID-Wert übernommen, um standardkonform auf das ursprüngliche IQ-Stanza zu antworten.

Die Entscheidung, nur eine Benutzerdatenbank im MES zu pflegen und die mobilen Clients eine anonyme Anmeldung am XMPP-Server durchführen zu lassen, bringt mit sich, dass eine Komponente im MES-Proxy die Verknüpfung von JID zu MES User-ID vornehmen muss. Zudem sollte diese Komponente die Aufgabe übernehmen, MES-Benutzer zu authentisieren und eine Sitzung zwischen An- und Abmeldung zu verwalten. Für die soeben benannten Aufgaben ist der **Session Manager** zuständig.
Der Session Manager besitzt die Information, welcher MES-Benutzer zurzeit mit welchem mobilen Client am MES angemeldet ist. Er nimmt An- und Abmelde Anfragen vom Request Manager entgegen und verarbeitet sie in Zusammenarbeit mit dem Dispatcher und dem Roster. Eine Sitzung am MES wird im Session Manager durch ein MES-Session Objekt abgebildet, welches im weiteren Verlauf einfach als Session bezeichnet wird. Bei der erfolgreichen Anmeldung eines MES-Benutzers erstellt der Session Manager eine neue Session. Sie wird entweder zerstört, wenn sich ein Benutzer abmeldet, oder zerstört sich nach einer bestimmten Zeit von selbst, wenn der Session Manager die Nachricht vom Roster erhält, dass der mobile Client des Benutzers in den „Offline“-Status gewechselt hat.

Der **Dispatcher** schlägt die Brücke zwischen der MES-Logik und den internen Komponenten des MES-Proxy. Alle Anfragen, die von der Logik verarbeitet werden sollen, werden über den Dispatcher geleitet. Innerhalb des MES-Proxy stellt der Dispatcher Schnittstellen zur Verfügung, um den anderen Komponenten die Interaktion mit der Logik zu ermöglichen. Ob diese Schnittstellen synchroner oder asynchroner Natur sind, soll wiederum offen gehalten werden und muss im Feinkonzept, d.h. in dem detaillierten Systemdesign, festgelegt werden. Nicht zuletzt, da die Entscheidung für eine synchrone oder asynchrone API von der Art und der Stärke der Kopplung zur MES-Logik abhängt. Im Grunde können Proxy und Logik auf jede beliebige Weise miteinander in Kontakt treten.

MES-Logik

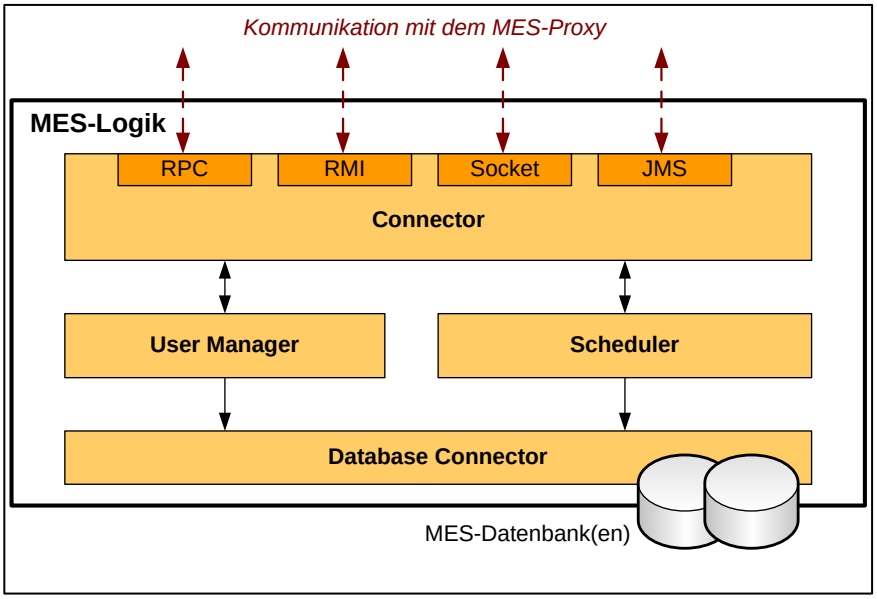

Abb. 12 Implementierung der MES-Logik

In der Abbildung (Abb. 12 Implementierung der MES-Logik) wird die mögliche Struktur der MES-Logik aufgezeichnet. Diese setzt sich aus dem Connector, den User Manager, den Schedular und dem Database Connector zusammen.

Über den **Connector** werden MES-Logik und MES-Proxy miteinander gekoppelt. Die Art der Schnittstelle kann je nach Anforderung gewählt werden. Der Connector unterstützt einen bidirektionalen Zugriff, d.h. Daten können von Seiten des MES-Proxy an die Logik übergeben werden, als auch von Seiten der Logik an den MES-Proxy. Letzteres ist für den asynchronen Versand von MES-Kurznachrichten erforderlich.

Der **User Manager** erhält alle Login- und Logout-Nachrichten der mobilen Clients vom Connector. Bei der Anmeldung eines Benutzers überprüft der User Manager die übergebenen Anmeldedaten in der Datenbank. Stimmen sie überein, übergibt der User Manager eine positive Antwort. Andernfalls schlägt der Login-Vorgang fehl und es wird eine negative Rückmeldung gegeben. Sobald sich ein Benutzer abmeldet, kann der User Manager bei Bedarf diese Benachrichtigung nutzen und einen Status in der Datenbank umschreiben.

Dem **Scheduler** werden allgemeine Anfragen von den mobilen Clients übergeben. Der Scheduler ist in der Lage eine individuelle Anfrage im XML-Format zu interpretieren und eine entsprechende Verarbeitung einzuleiten. Das Ergebnis der Anfrage wird anschließend über den Connector zurück an den anfragenden mobilen Client gesendet. Um z.B. Auftragsdaten zu beziehen, nutzt der Scheduler die Datenbank. Bei der Anfrage nach einem neuen Auftrag wählt der Scheduler einen Datensatz aus der Auftragstabelle und wandelt ihn in das XML-Format. Die XML-Daten werden im Anschluss an den mobilen Client übertragen.

Der Zugriff auf den Datenbestand in den MES-Datenbanken erfolgt über den **Database Connector**. Dazu stellt der Database Connector den anderen Komponenten eine Reihe von Zugriffsfunktionen zur Verfügung. Durch diese Abstraktion kann das Datenbanksystem unabhängig von der Implementierung des User Managers oder des Schedulers gewählt werden.

Risiken

Die Risiken des Projektes sind vielfälltig und können hier nur kurz angerissen werden. Kurz angeschnitten werden die verschiedenen gängigsten Mobilen Betriebssysteme, die gängigsten Entwicklungsplattformen, die Sicherheit in Kommunikationssystemen und die Risiken mit Drittsystemen.

Mobile Betriebssysteme

Es gibt eine Vielzahl von Betriebssystemen und nicht anders sieht es bei den mobilen Betriebssystemen aus. Hier sollen nur die gängigsten kurz vorgestellt werden.

Das Betriebssystem **Windows Mobile** bringt gleich die erste Verwirrung mit sich, da es eine Flut von unterschiedlichen Versionen wie Windows CE, Pocket PC, Windows Mobile for Pocket PC oder einfach nur Windows Mobile hat. Hier ein kleiner Überblick:

- Bis WinCe3.0 war die Bezeichnung nur Windows CE
- Änderung aus Marketinggründen Windows CE in Pocket PC.
- Bei Vermarktung von Microsoft.NET Framework wurde Windows CE zu Windows CE.NET(WinCE4*) und Pocket PC zu Windows Mobile.
- Nebenbei bürgerte sich der Begriff Pocket PC ein, somit wurde Windows Mobile in der ersten Version zu Windows Mobile for Pocket PC's.
- Alle Versionen basieren auf Betriebskern Windows CE(WinCE)

Im Betriebssystem **Palm OS** laufen alle Programme in einer emulierten Umgebung um die Kompatibilät zu älteren Palm OS zu wahren, dem Palm Application Compatibilty Environment (PACE).
Das Betriebssystem **Symbian OS** ist zurzeit das führende Betriebssystem für Smartphones. Symbian ist ein Konsortium aus führenden Mobilfunkherstellern, wie Siemens, Sony Ericsson, Nokia und Samsung. Symbian bildet nur den Betriebssystemkern. Die Oberflächen müssen selbst erstellt werden.
Das Betriebssystem **Blackberry OS** ist eher eine Kommunikationslösung der Firma "Research in Motion". Es ist hauptschlich ein E-Mail und Organizer Lösung, die aus den Blackberry Enterprise Server, einer Destop Software und einem Blackberry PDA besteht.

Um das Risiko einzugrenzen, wird im Projekt erstmal auf das BetriebssystemWindows Mobile zurückgegriffen, da der zu Verfügung gestellte iPAQ ein Windows Mobile 2003 Betriebssystem vorinstalliert hat.

Entwicklungsplattformen

Ähnlich wie bei dem Betriebssystem gibt es eine Flut von Entwicklungsplattformen. Und auch hier gibt es ein reichhaltiges Angebot für mobile Anwendungen. Zu denen die bekanntesten, wie Java 2 Micro Edition, Microsoft .NET Compact Framework, C / C++ zählen. Auch hier sollen die gängigsten kurz vorgestellt werden.

Die Entwicklungsplattform **Java 2 Micro Edition (J2ME**) bietet eine Umgebung für Applikationen, die auf Mobiltelefonen, PDA's oder auf Embedded-Systemen ausgeführt werden. In J2ME sind virtuelle Maschinen (Java Virtual Machine - JVM) und eine Reihe von Standard Java-API's definiert. Die J2ME Applikationen werden in Java geschrieben und sind auf allen Geräten ausführbar, die eine Unterstützung von J2ME von Haus aus integriert haben, wie bei den meisten Betriebssystemen von Mobiltelefonen der Fall ist. An dieser Stelle ist interessant, dass von Sun selber keine (kostenlosen) JVM's für PDA's angeboten werden. Hierzu bietet aber IBM in Rahmen seiner WebSphere Suite seine „J9" JVM für Windows Mobile und Palm OS 5.0 an.

Die Entwicklungsplattform **Microsoft .NET Compact** Framework (.NET CF) von Microsoft ist für die Entwicklung von Applikationen für mobile Geräte in der Windows Welt gedacht. Als Entwicklungsumgebung wird das Microsoft Visual Studio 2003 benötigt, die zum testen und debuggen Simulatoren für Pocket PC und Windows CE .NET Geräte mitbringt.
Das Prinzip von .NET ähnelt dem von Java insofern, dass bei der Kompilierung kein direkt vom Prozessor ausführbarer Code sondern ein Zwischencode erzeugt wird. Der Zwischencode, auch managed code genannt, ist auf der .NET Runtime Komponente auszuführen. Im weitesten Sinne kann die .NET Runtime mit der JVM von Java verglichen werden. Aus dem managed code erzeugt die .NET Runtime mit einem just-in-time (JIT) Compiler den nativen Code der Hardwareplattform. Eine in .NET implementierte Applikation kann also auf verschiedenen Plattformen ausgeführt werden, ohne neu kompiliert zu werden.

Die Entwicklungsplattform **C / C++** ist die bevorzugte Hochsprache für Applikationen auf der Palm OS Plattform. PalmSource selbst stellt die PalmOS Developer Suite IDE zum kostenlosen Download zur Verfügung. Die Developer Suite basiert auf Eclipse und beinhaltet einen C/C++ Compiler.

Wie man an der Länge der Textabschnitte sehen kann, werden im Projekt um die Eingrenzung des Risikos die beiden Hochsprachen Java und .Net eine Rolle spielen, dies ergibt sich durch die vorhandene Hardware, sowie die Vorkenntnisse und den Wissensdrang des Entwicklers für die jeweiligen Hochsprachen.

Sicherheit

Natürlich darf in einen solchen Kommunikationssystem die Sicherheit nicht ausser Acht gelassen werden. Zu diesem Thema könnte man allein schon eine ganze Masterarbeit schreiben, da aber dieses Thema eine hohe Priorität hat, sollen hier kurz nochmal die wichtigsten Eckpunkte aufgeführt werden.
Der erste Punkt wäre die **Authentifizierung**, diese gewährleistet die Identifikation der Kommunikationspartner im System. Die **Autorisierung**, die die Zugriffskontrolle bzw. die Rechteverwaltung der Benutzer für den Zugriff auf Anwendungen verwaltet. In einem Kommunikationssystem ist die **Vertraulichkeit** ein wichtiges Argument, damit die Übertragung nur zu erwünschten Kommunikationspartnern erfolgt.
Ein wichtiges Thema in Sicherheitrelevanten Teilbereichen ist die **Verschlüsselung** der Daten bei der Übertragen und der Speicherung auf mobilen Geräten. Hierzu gehört auch die **Integrität**, diese verhindert die Manipulation der Daten während der Übertragung (Man in the Middle Attack). Im Bankengeschäft ist die **Non-Repudiation** (Unbestreitbarkeit) von Transaktionen, besonders wichtig. Hier das Stichwort: „Gerichtstauglicher Nachweis“.
Was eher von der Hardware und dem vorhanden Netzwerk verlangt wird ist ein **Geräteschutz**, der z.B. durch einen Fingerabdruck-Scanner gewährleistet werden kann. Oder ein **Medienzugriff**, z..B.Zugriff auf’s WLAN nur nach einer Registrierung vorab.
Und zur guter letzt die **Privacy** des einelnen, die Berücksichtigung des Datenschutzes, insbesondere nur die Lokalisierung mit Einverständnis des Nutzers durchzuführen.

Dritte Systeme

Mit Dritte Systeme sind hier Systeme zur Lokalisierung von Personen und Systeme zur Verwaltung von Aufträgen gemeint.Da die Auswahl auch hier sehr groß ist, stellen sie ein zeitliches Risiko dar. Leider war die Zeit zu kurz sich ausführlicher mit dem Thema auseinander zusetzen und wird in der Masterarbeit vertieft.

Ausblick

Im Anhang wird in der Abbildung (Abb. 13 Masterarbeit Gesamtvorhaben) das Vorhaben was bis jetzt im Einzelnen vorgestellt wurde in der Gesamtansicht gezeigt und untergliedert in die Middlewar und die Implementierung.

Literaturverzeichnis

- Tanenbaum, Andrew S.: Moderne Betriebssysteme. Pearson Studium, 2004. – ISBN 3–8273–7019–1
- Young, Michael J.: XML Schritt für Schritt. 2. Ausgabe. Microsoft Press, 2001. – ISBN
- 3–86063–789–4
- Saint-Andre, Peter. Extensible Messaging and Presence Protocol (XMPP): Core. IETF Request For Comments (RFC) 3920. http://www.ietf.org/rfc/rfc3920.txt. November 2005
- Project, Jabberd: Homepage des JabberD Projektes. http://jabberd.jabberstudio.org
- Homepage der Jabber Software Foundation. http://www.jabber.org/
- Kort, Todd ; Anavitarte, Luis: Gartner Group - Windows CE Surpasses Palm OS in 3Q04. http://www.gartner.com/DisplayDocument?ref=g_search&id=459718
- Berners-Lee, T. Hypertext Transfer Protocol (HTTP-1.0). IETF Request For Comments(RFC) 1945. http://www.ietf.org/rfc/rfc1945.txt.
- Mauerer, Jürgen: PDA und Smartphones im Unternehmen. http://www.zdnet.de/mobile/print_this.htm?pid=39129995-20000102c
- Heinrich, Hayat: Auswahl einer Middleware für mobile Clients / Hochschule für Angewandte Wissenschafen Hamburg. 2003. – Diplomarbeit
- Roth, Jürgen: Mobile Computing. 1. Auflage. dpunkt Verlag, 2002. – ISBN 3–89864165–1
- Kruse, Mirko: Entwicklung einer synchronen Kommunikationsanwendung auf Basis von Web- und Java-Technologien / Hochschule Bremen. 2005. – Studienarbeit
- Kruse, Mirko Diplomarbeit „Konzept und prototypische Realisierung einer Kommunikationsschnittstelle für kundenindividuelle Mobile Enterprise Lösungen am Beispiel ”Auftragsverwaltung“
- Snell, James: Programming Web Services with SOAP. 1. Ausgabe. O’Reilly, Dezember 2001. – ISBN 0–596–00095–2
- Saint-Andre, Peter. Extensible Messaging and Presence Protocol (XMPP): Instant
- Messaging and Presence. IETF Request For Comments (RFC) 3921. http://www.ietf.org/rfc/rfc3921.txt.
- Dvorak, Petr: Build a SOAP-based Chat Application with Java Web Services, Part 2. In: DevX.com (2004), Januar. – http://www.devx.com/webdev/Article/18203
- Free Software Foundation - GNU General Public Licence - Version 2.
- http://www.gnu.org/licenses/gpl.txt
- http://www.interbay.de/BILD: Handwerker 1
- http://www.hkservice.de BILD: Handwerker 2
- http://www.wermelskirchen.de BILD: Schwimmgäste
- http://www.niedoba.de BILD: Auto im Swimmingpool
- http://derstandard.at BILD: End Windows Session
- http://www.blaukreuz-zentrum-hagen.de BILD: Fragen

Abbildungsverzeichnis

Anhang

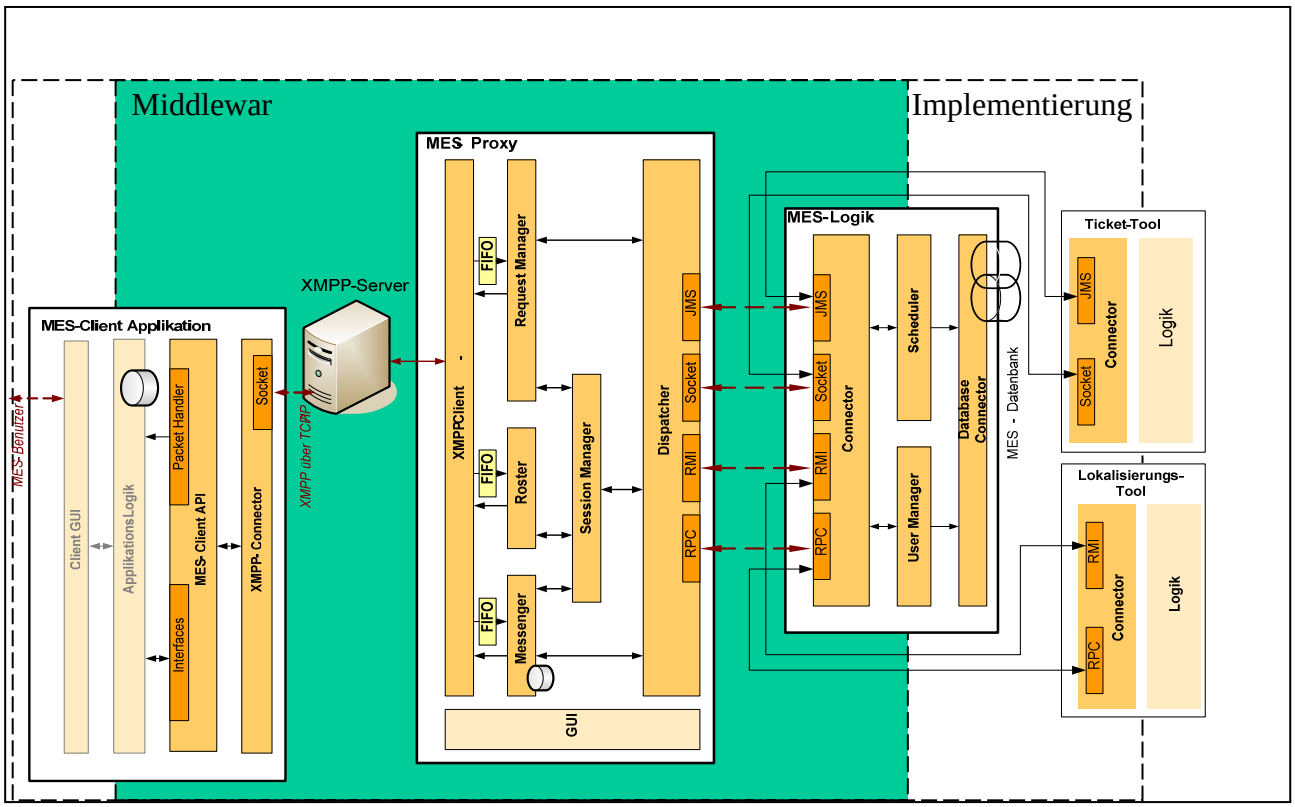

Abb. 13 Masterarbeit Gesamtvorhaben

Vor-bzw. Nachteile von XMPP im Zusammenhang

Abschliessend soll nocheinmal auf die Vor- und Nachteile von XMPP zum Vergleich zu den anderen Kommunikationsprotokollen eingegangen werden.

Ergänzend zum alllgemeinen XMPP Aufbau sollte noch erwähnt werden, dass XMPP als Sicherheitsfunktion TLS Verschlüsselung der TCP/IP Verbindung unterstützt. Somit kann Authentisierung auf der TLS-Ebene mit Zertifikaten durchgeführt werden. Als weitere Schutzfunktion sollte erwähnt werden, dass sich der Client erst am XMPP-Server anmelden muss bevor er Nachrichten verschicken kann.

Ein besonderer Vorteil von XMPP gegenüber der HTTP oder SOAP Lösung liegt in dem asynchronen Austausch von XML-Stanzas. Das ermöglicht u.a. einen vom MES ausgehenden Nachrichtenversand an die mobilen Clients. Ganz nebenbei lässt sich mit XMPP eine Instant Messaging Infrastruktur für die mobilen Clients aufbauen. Die Presence-Funktion kann vom MES genutzt werden, um z.B. einen Auftrag zurückzuziehen, wenn der Client für eine längere Zeit „Offline“ ist. XMPP ist durch die Extensions erweiterbar und kann voraussichtlich an die Anforderungen des MES angepasst werden, ohne den IETF Standard zu verlassen.

Für XMPP stehen nahezu auf jeder Entwicklungsplattform Bibliotheken zur Verfügung. Der wichtigste Vorteil ist, dass die einzelnen XML-Stanzas erhalten im Vergleich mit HTTP-Requests und SOAP-Nachrichten nur einen kleinen Protokolloverhead. Eine XMPP Verbindung zum MES wird aus diesem Grund ein kleineres Transfervolumen erzeugen, wodurch es interessant für die Mobilfunknetze wird.

Leider sind die meisten XMPP Server unter der GPL veröffentlich, was voraussetzt dass eine lose Kopplung zum restlichen System bestehen muss um die Software kommerziell zu verwenden. Desweiteren verlangt XMPP, dass die TCP-Verbindung während der gesamten Sitzung bestehen bleibt. Es muss dafür gesorgt werden, dass ein getrennter TCP-Socket von der Anwendung erkannt wird. Als Reaktion auf eine abgebrochene Verbindung muss die Anwendung automatisch die Verbindung wiederherstellen. Sollte der XMPP Server nicht zuverlässig getrennte TCP-Sockets erkennen und den mobilen Benutzer „Offline“ melden, bietet die Presence-Funktion von XMPP keinen Mehrwert.

Um die löse Kopplung des XMPP-Servers zum MES-Server zu gewährleisten, darf der MES-Server weder auf die XMPP-Server Benutzerverwaltung zugreifen noch sie integrieren. Um diese Trennung zu ermöglichen, ist für den MES-Server eine eigene Benutzerdatenbank vorzusehen. Somit kann eine kundenindividuelle Rechtestruktur im MES umgesetzt werden.
Um den Administrationsaufwand gering zu halten, werden für mobile Clients keine Benutzerkonten im XMPP-Server angelegt. Die Anmeldung erfolgt als anonymer Benutzer, wobei der XMPP-Server automatisch eine JID vergibt. XMPP verwendet SASL als Authentisierungsmechanismus, eine anonyme Anmeldung wird mit dem SASL-Mechanismus „ANONYMOUS“ unterstützt.
Der MES führt eine eigene Authentisierung durch, um einen unerlaubten Zugriff auf seine Dienste zu vermeiden. Dabei ordnet er dem Benutzer für die Dauer der Sitzung automatisch eine JID zu. Mit dieser Maßnahme muss nur eine einzige Benutzerdatenbank im MES verwaltet werden.