Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung Projekt

Andreas Piening

RESCUE
Leitstand für Desaster-Szenarien

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andreas Piening

RESCUE
Leitstand für Desaster-Szenarien

Ausarbeitung Projekt eingereicht im Rahmen des Masterstudiums
im Studiengang Master of Computer Science
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck

Abgegeben am 1. März 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Diese Ausarbeitung behandelt die Inhalte der Veranstaltung “Projekt“. Im Rahmen dieser Veranstaltung wurde der Einsatz von Technologien und Frameworks für die Realisierung der Leitstand-Implementation des “Rescue: Leitstand für Desaster-Szenarien”-Projektes erprobt. Ein Überblick über die untersuchten Frameworks findet sich in Kapitel 2 auf Seite 6. Diese Arbeit ist aber weniger als Protokoll gedacht, sondern sie soll die gewonnenen Erkenntnisse aufbereiten, um damit Vorraussetzungen für die Master-Arbeit zu schaffen.

Das übergreifende Thema dieser Arbeit ist das von Prof. Dr. Kai von Luck inspirierte “Rescue Szenario“, welches in der Ausarbeitung der Ringvorlesung genauer definiert wurde.
Da der Bezug zu dem Szenario in dieser Ausarbeitung recht lose gekoppelt ist und dieser in der Ausarbeitung Ringvorlesung auch bereits ausführlich behandelt wurde, soll hier weder eine ausführliche Motivation noch eine Projekteinordnung erfolgen.

Das selbstdefinierte Ziel für die Veranstaltung “Projekt“ bestand neben der Untersuchung von interessanten Technologien, in der Realisierung von Teilaspekten der Kartendarstellung des Leitstandes. Wichtig war an dieser Stelle, bei der Beispiel-Implementation nicht den Bezug zu dem Gesamtprojekt zu verlieren. Die Kartendarstellung ist eine Möglichkeit den Bereich “Gebäudesensorik“ und “Wearable Computing“ zu integrieren.

2 Technologien und Frameworks

In diesem Kapitel sollen Frameworks vorgestellt werden, die in der Veranstaltung “Projekt“ Verwendung fanden. Die Darstellung ist dabei zunächst abstrakt gehalten; die Rolle der vorgestellten Technologien für die Realisierung wird in Kapitel 3 auf Seite 17 und Kapitel 4 auf Seite 21 vorgestellt.

2.1 Google Maps

Bei Google Maps[Google (2006a)] handelt es sich um einen webbasierten Dienst zum Anzeigen von Kartenmaterial. Google stellt weltweites Kartenmaterial bereit. Dabei gibt es neben dem Vektor-Kartenmaterial, welches auch die Straßenbezeichnungen beinhaltet, auch hochauflösende Satellitenkarten. In einem “Hybrid-Modus“ kann sogar eine Kartenansicht dargestellt werden, bei der Straßen inklusive der Bezeichner in die Satellitenbilder als Overlay eingeblendet werden.

Neben dem Zugriff auf das Kartenmaterial über ein Web-Portal stellt Google auch die Möglichkeit bereit, die Karten in die eigene Webseite zu integrieren. Hierzu wird eine ausführliche Beschreibung der API zur Verfügung gestellt [Google (2006b)].

Die Funktionsweise der Google-Maps-Karte ist dabei immer gleich: Der Browser lädt von dem Google Maps-Server einen Java-Script-Code herunter und bringt diesen lokal zur Ausführung. Hierbei kommen AJAX[1]-Technologien zum Einsatz. Die Benutzereingaben, wie zum Beispiel die Bewegung der Karte, werden lokal ausgeführt. Fehlt jedoch Kartenmaterial (z. B. weil der Ausschnitt verschoben wurde), so wird dieses dynamisch von dem Google-Maps-Server nachgeladen.

Ein Herunterladen des Kartenmaterials von dem Google-Maps-System ist zwar unter der Anwendung einiger technischer Tricks möglich, widerspricht jedoch den Benutzungsbedingungen, denen man bei der Verwendung von Google-Maps zustimmt.

[1] AJAX: Asynchronous Javascript and XML. Ein Java-Script-Code wird von einem Webserver geladen und in dem Browser zur Ausführung gebracht. Durch Nachrichten im XML-Format kann die Applikation im Browser Ereignisse und Informationen mit dem Webserver austauschen. [Garrett (2005)]

2.2 Graph Modelling Framework

Das “Graph Modelling Framework“ [Tarling und Robbins (2006)] ist eine Bibliothek bzw. ein Framework für die Programmiersprache JAVA, welches die Basis-Funktionalität, die zum Darstellen und grafischen Editieren von Diagrammen benötigt wird, bereitstellt. Das Framework entstammt der Open Source UML-Anwendung ArgoUML [Boger und Robbins (2006)], welches das GEF verwendet, um beispielsweise UML-Klassendiagramme darzustellen.

Das GEF besitzt folgende Features:

- Einfaches und strukturiertes Design, wodurch das Framework einfach zu benutzen und zu erweitern ist.
- Unterstützung für “verbundene Graphen“ durch das “Node-Port-Edge graph model“.
- Model-View-Controler[2] Design, welches auf die “Swing Java Benutzerinterface-Bibliothek“ aufsetzt und damit leicht in bestehende Datenstrukturen integrierbar ist.
- Benutzerinteraktionen wie Verschieben, Skalieren, Anordnen, Umgestalten etc.
- Standard-Zugriffsmethoden nach JavaBeans-Modell.
- XML-basierte Dateiformate basiertend auf den PGML[3]-Standard [Consortium (1998)] (SVG[4]-Support [Consortium (2003a)] in Planung).

2.3 Java Desktop Integration Components

Java ist eine plattformunabhängige Programmiersprache, somit kann ein Java-Programm prinzipiell auf mehreren verschiedenen Betriebssystemen ausgeführt werden. Ein oft zitiertes Problem dabei ist die Integration in den jeweiligen Desktop. Die Arbeitsflächen der unterschiedlichen Betriebssysteme haben einen unterschiedlichen Aufbau und Funktionsumfang, es mangelt an einer standardisierten Schnittstelle. Dies ist der Ansatzpunkt der “Java Desktop Integration Components“ [Laux (2004)]. JDIC stellt die wichtigsten Desktop-Funktionalitäten über einheitliche Schnittstellen dem Java-Entwickler zur Verfügung, der sich dann nicht mehr um die betriebssystemabhängige Implementierung dieser Funktionen zu kümmern braucht.

[2] MVC: Zu deutsch “Modell-Präsentations-Steuerung“. Software-Architektur-Muster bei dem die Darstellung, das Datenmodell und die Logik in Module aufgeteilt wurden. [Gamma u. a. (2004)]

[3] PGML: Vorgänger von SVG, ähnlicher Aufbau

[4] Scalable Vector Graphics: Vektorbasiertes Format zum abspeichern von zweidimensionalen Grafiken in der XML-Syntax

JDIC stellt unter anderem Funktionen wie einen Dateibrowser, Erstellung eines Tray-Icons bzw. eines Floating Dock[5], Verändern des Desktop-Hintergrunds, Erstellung plattformunabhängiger Screensaver etc.
Ein besonders interessanter Teil dieses Projektes ist die Browser-Komponente. Hierbei handelt es sich um ein JPanel[6], welches je nach Betriebssystem die Engine des Standard-Browsers (z.B. Internet Explorer unter Windows, Gecko unter Linux, Webkit unter MacOSX) als Java-Komponente zur Verfügung stellt.

2.4 XML-RPC

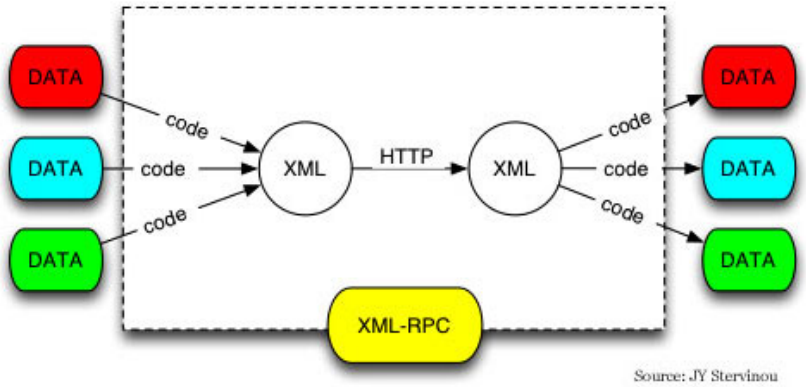

Abbildung 2.1: XML-RPC: Daten-Kodierung und -Übertragung [UserLand (1998)]

XML-RPC[7] [UserLand (1998)] ist eine Definition zum entfernten Funktionsaufruf. Um eine einfache Integration in möglichst viele Programmiersprachen zu ermöglichen, wurden zwei Standards miteinander verbunden: Die “Extensible Markup Language“ zum Kodieren und Übertragen der Informationen das durch das Internet bereits etablierte HTTP-Protokoll.

Eine der durch die Anforderung nach einem möglichst einfachen Protokoll diktierten Einschränkungen von XML-RPC ist, dass lediglich primitive Datentypen und einfache Verbund-Datentypen für einen entfernten Funktionsaufruf zur Verfügung stehen[8]. Hierbei werden

[5] Floating Dock: Status-Panel ohne Fenster-Kontrolleiste, welches an den Ecken des Desktops ausgerichtet werden kann.
[6] JPanel: Java GUI-Komponente zum optischen und logischen Kapseln von Controls
[7] XML-RPC: Extensible Markup Language Remote Procedure Call
[8] Die Datentypen von XML-RPC umfassen Integer-, Double, Boolean, String-, Date- und Binärtypen sowie die Verbund-Datentypen Struct und Array

die Daten auf Client-Seite in die XML-Darstellung transformiert, um dann über das HTTP-Protokoll an die Serverseite übertragen werden zu können (siehe Abbildung 2.1). Auf der Serverseite werden die Daten nach einem Unmarshalling[9]-Prozess der entsprechenden lokalen Funktion übergeben, dessen Ergebnis dann XML-kodiert über HTTP an den Aufrufer zurückgereicht wird.

Historisch betrachtet stellt XML-RPC den Vorgänger zu SOAP[10] dar. Da XML-RPC jedoch deutlich schlanker ist und eine hervorragende Unterstützung seitens der Programmiersprachen hat (unter anderem ASP, C, C++, Delphi, Flash, Haskell, Java, JavaScript, .NET, Perl, PHP, Python, Ruby und TCL), erfreut sich das Protokoll weiterhin großer Beliebtheit.

2.5 ICE

Die “Internet Communication Engine“ (ICE) [ZeroC (2006)] der Firma ZeroC ist eine Object-Middleware für verteilte Systeme ähnlich dem etablierten, aber betagten CORBA[11]. Das Projekt stellt sich selbst dem CORBA-Standard gegenüber und greift dessen Nachteile auf, um sich demgegenüber abzugrenzen. Hier ein Auszug aus der beeindruckenden Feature-Liste von ICE:

- Aufgrund eines hoch effizienten Protokolls ist ICE sehr performant. Durch Features wie “request batching“ und einer intelligenten Ereignis-Weiterleitung benötigt ICE darüber hinaus weniger Netzwerkressourcen als der Kontrahent CORBA.
- ICE bietet Unterstützung für die Sprachen C++, Java, Python, PHP, C# und Visual Basic, zudem werden die Plattformen Linux, MacOSX, Solaris, Windows und HP-UX unterstützt. Alle Implementierungen kommen hierbei aus “einem Haus“. Aufgrund der schlanken Architektur ist der Einsatz von ICE auch auf Handys, PDA’s und Embedded-Geräten möglich.
- Eine integrierte Persistenz-Unterstützung ermöglicht beispielsweise das Ablegen von ICE-Objekten in einer Datenbank. Das Persistenz-Layer “Freeze“ unterstützt sogar eine Versionisierung von ICE-Objekten.
- Sicherheitsmechanismen sind fest in ICE integriert. So wird standardmäßig SSL-verschlüsselung angeboten und ein integrierter Filter ermöglicht ein Eingrenzen der Aufrufer.
- ICE unterstützt asynchrone Kommunikation.

[9] Unmarshalling: Umwandeln der Daten von der Übertragungs-Darstellung in die lokale Datenrepräsentation
[10] SOAP: Simple Object Access Protocol, Standard zum entfernen Methodenaufruf [Consortium (2003b)]
[11] CORBA: Common Object Request Broker Architecture [CORBA (2000)]

- ICE steht unter der GPL und ist damit kostenlos verfügbar. Darüber hinaus ist die Software gut dokumentiert und ermöglicht durch Schnellstart-Anleitungen einen reibungslosen Einstieg.

Die Verwendung von ICE ist in der Tat sehr einfach gehalten. Um einen Objektzugriff über Netzwerkgrenzen hinweg zu ermöglichen, muss (ähnlich wie bei CORBA durch die IDL) die Schnittstelle beschrieben werden. Hierzu bietet ICE die “Specification Language for ICE“, kurz SLICE an. Dabei bietet SLICE eine an C++ angelehnte, aber stark vereinfachte Syntax an, in der Objektbezeichner, Typen und Strukturen definiert werden. Durch einen Slice-Compiler werden dann die Schnittstellen für die entsprechende Implementierungs-Sprache übersetzt. Nun wird eine Klasse erstellt, die von dem generierten Interface erbt und durch wenige Zeilen Quellcode kann auf die Klasse entfernt zugegriffen werden.

2.6 JINI

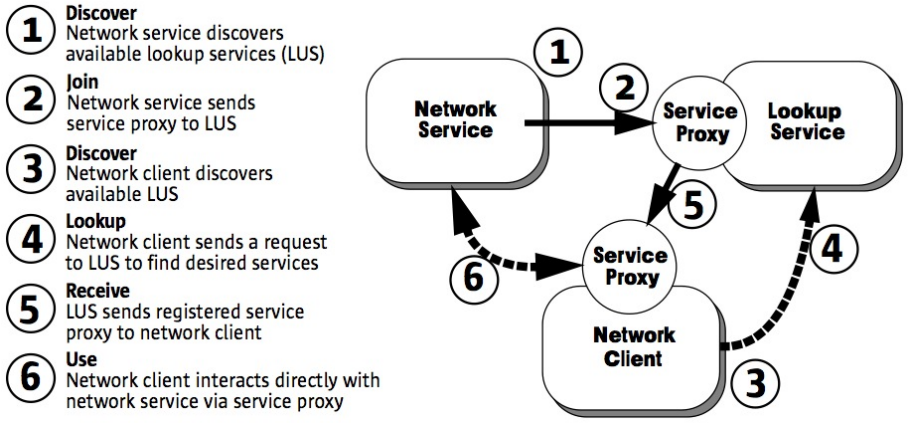

Abbildung 2.2: JINI: Bekanntmachung eines Dienstes [SUN (2006)]

JINI ist eine Netzwerk-Technologie, die das Erstellen von netzwerkspezifischen Anwendungen ermöglicht, welche spontan kommunizieren und sich ändernden Netzwerk-Bedingungen anpassen können [SUN (2006)].

Hierbei stellt JINI Lösungen für folgende Anforderungen bereit:

- Auffinden von Netzwerk-Diensten und Herstellung einer Verbindung zu diesen.
- Erstellung von zuverlässigen Dienst-Gruppen aus unzuverlässigen Teilen; dies gilt auch für das Netzwerk selbst.

- Arbeiten mit sehr großen Netzwerken bzw. Netzwerken, die über einen sehr langen Zeitraum bestehen.
- Komponenten eines Dienstes ermöglichen, sich jederzeit zu ändern, ohne dabei die Verfügbarkeit des Dienstes zu unterbrechen.

JINI legt besonderen Wert auf Einfachheit. Zum einen soll JINI für den Entwickler einfach einzusetzen sein, zum anderen soll es speziell für den Anwender Vorteile bringen: So benötigt es z. B. keine Konfiguration, um Dienste einbinden zu können.

Als Schnittstellendefinition dient bei JINI ein Java-Interface, welches von “Remote“ erbt und die Signaturen der aufzurufenden Methoden enthält. Der Server muss dann dieses Interface implementieren. Auf Clientseite muss das Interface ebenfalls vorliegen, da er das von dem Registrar zurückgegebene Service-Objekt auf das Interface casten muss, bevor er den Service zur Ausführung bringen kann.
An dieser Stelle ein Hinweis zur Begrifflichkeit: Bei Jini wird eigentlich nicht von einer Client/Server-Aufrufsemantik gesprochen, sondern eher von einer Client/Service-Aufrufsemantik. Jeder Netzwerkteilnehmer hat dabei die Option, einen Service anzubieten und welche anzufordern.

Der Ablauf eines Service-Aufrufes, wie er auch in Abbildung 2.2 dargestellt wird, soll hier noch einmal schrittweise verdeutlicht werden:

- Ein aktives JINI-Netzwerk verwendet einen Server, bekannt als “JINI Lookup Service (LuS)“.
- Wenn ein neues Gerät ein JINI-Netzwerk betritt, verwendet es die JINI-Klassenbibliothek, um den Lookup-Service ausfindig zu machen. Sofern das Gerät einen Service anbieten will, so registriert es diesen nun an dem LuS. Als Teil des Registrierungsprozesses kopiert das Gerät die Java-Klasse, die für den Aufruf notwendig ist, auf den LuS.
- Will ein Gerät einen Service in Anspruch nehmen, so fragt es den LuS nach der Liste der registrierten Dienste. Durch Filter-Vorgaben, kann das Gerät die zurückgelieferte Liste des LuS dabei einschränken. Zusammen mit der Antwort liefert der LuS die Interface-Klasse für jeden Service in der Liste zurück.
- Der Benutzer des neuen Gerätes kann nun jeden Service der Liste in Anspruch nehmen und beispielsweise mit Hilfe einer von dem Service angebotenen GUI mit diesem interagieren. Wenn das GUI-Interface entscheidet, dass der Benutzer eine Anfrage gestellt hat, so leitet es diese an den Service weiter, empfängt die Antwort und stellt diese dem Benutzer dar.

Bei JINI handelt es sich um eine reine Java-Technologie damit ist diese nicht sprachunabhängig. Die Tatsache, dass bei verteilten Applikationen meist plattformunabhängige Technologien zum Einsatz kommen und Java in diesem Feld äußerst stark vertreten ist, relativieren diesen Nachteil wieder ein wenig. Darüber hinaus ermöglicht die Java-Technologie ein besonderes Feature von JINI: “Moving Code“. Hierbei wird ausführbarer Java-Bytecode an den Aufrufer übermittelt, welcher dann in der Lage ist, diesen auszuführen. Dabei benötigt der Client keine weiteren Kenntnisse über den Dienst, da dieser die Klasse zur Benutzung selbst liefert.

2.7 Berkeley: iROS

Bei iROS handelt es sich um ein eigenständiges aus Basis-Technologien zusammengesetztes Framework. Es handelt sich um einen Teil des “iWORK“-Projektes der Berekeley University Of California. “iROS“ steht für “Interactive Room Operating System“, entgegen der Überschrift handelt es sich jedoch nicht um ein Betriebssystem im eigentlichen Sinne, sondern mehr um eine Plattform bzw. Middleware, auf deren Basis Anwendungen speziell für kollaborative Umgebungen implementiert werden können [Winograd u. a. (2002)].

Die Basis der iROS Implementation ist das “Tuplespace-Model“ welches den Systemübergreifenden Zugriff auf Objekte und Ereignisse zulässt. Bei einem TupleSpace handelt es sich um einen assoziativen Speicher, der Informations-Tuple ablegen und mehreren Prozessen zum parallelen Zugriff zur Verfügung stellen kann [Cunningham (2006)]. Eine Informationsverteilung wird darüber erreicht, daß ein Prozess in den TuppleSpace schreiben kann, während mehrere aus diesem lesen.

“iROS“ definiert folgende Umgebungscharakterisierung mit den damit verbundenen Anforderungen:

Heterogenität: Unterschiedliche Geräte mit verschiedenen Fähigkeiten, Displays, Eingabemöglichkeiten und Softwarekomponenten nehmen an einer Kollaboration teil.

Multiplizität: Mehrere Benutzer nehmen gleichzeitig Eingaben innerhalb eines gemeinsamen Kontextes vor, die Anwendungen sind dabei parallel aktiv.

Dynamik: Geräte werden der Kollaboration hinzugefügt und wieder von dieser entfernt. Langfristik kommen neue Geräte mit neuen Fähigkeiten hinzu. Eine schnelle Fehlerbehandlung ist die Vorraussetzung für eine erfolgreiche Kollaboration.

Sofern man die Kommunikation in den Mittelpunkt der Betrachtung legt, lassen sich hieraus folgende Anforderungen ableiten:

Moving Data: Es müssen Daten (z. B. Dokumente, Bilder und Texte) zwischen unterschiedlichen Geräten ausgetauscht werden können.

Moving Control: Jeder Benutzer soll mit jedem Dienst von überall aus interagieren können.

Dynamic Application Coordination: Informationen sollen von Ihren "Standard-Applikationen" entkoppelt werden und von möglichst vielen Diensten dargestellt werden können.

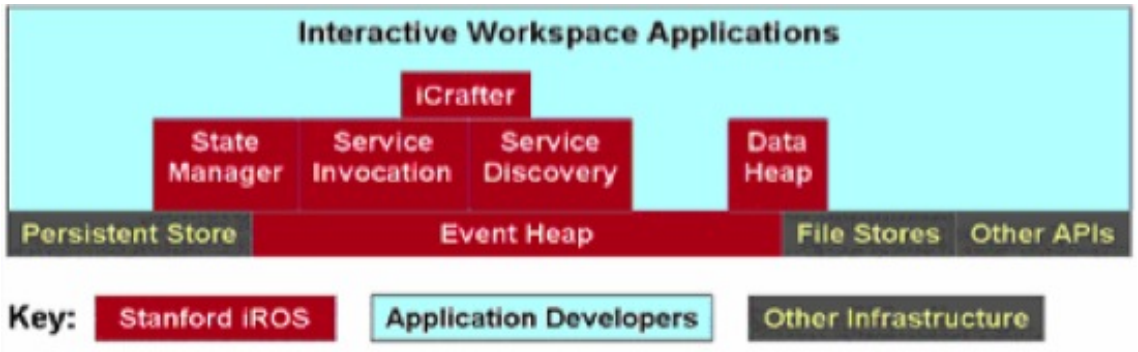

Abbildung 2.3: iROS: Architektonischer Aufbau

Das iROS ist in Java realisiert um plattformübergreifend zum Einsatz kommen zu können. Über C++-Schnittstellen soll auch von anderen Anwendungen aus auf iROS zugegriffen werden können. In Abbildung 2.3 ist der Aufbau einer iROS-Anwendung auf Komponenten-Ebene zu sehen, iROS selbst ist in drei Realisierungsbereiche aufgeteilt:

Event Heap: Ein "TupleSpace" auf dem jeder Prozess Events ablegen kann, dabei kann eine "Verfallszeit" für das Event angegeben werden. Durch "Pattern-Matches" kann ein Filter auf den TupleSpace angewendet und somit auf bestimmte Ereignisse reagieren. Dabei können Events sowohl zerstörend als auch nicht zerstörend gelesen werden.

Data Heap: Der Data Heap ermöglicht das "verschieben" von Daten zwischen den an der Kollaboration teilnehmenden Clients. Jeder kann durch Attribute spezifizierte Daten ablegen, welche dann über diese selektiert werden können. Der Empfänger kann beim Allozieren der Daten ein gewünschtes Format vorgeben, wodurch die Informationen dann gegebenenfalls transformiert werden.

iCrafter: Der iCrafter stellt einen Service zum Bekanntmachen von Diensten bereit, neue Dienste werden durch ein "Advertisement Event" bekannt gegeben. Er kann bei einer Interaktions-Anfrage durch einen Client automatisch ein angepasstes Benutzerinterface generieren und ausliefern. Die Kommunikation mit dem iCrafter erfolgt ausschließlich über den Event Heap.

2.8 Zeroconf (Bonjour)

Zeroconf ist eine Spezifikation, die das konfigurationsfreie Einbinden von Netzwerkressourcen ermöglicht [Steinberg und Cheshire (2005)] [Cheshire (1999)]. Es gibt etliche Ansätze[12], die Herausforderung der automatischen Konfiguration von Netzwerkgeräten zu bewältigen. Das Ziel der Zeroconf Working Group ist es, einen Mechanismus zu definieren, der auf offene und gut dokumentierte und damit leicht zu implementierende Standards setzt. Folgende Probleme sollen durch Zeroconf gelöst werden:

- Automatische Zuweisung von IP-Adressen ohne DHCP-Server
- Namensauflösung ohne DNS Server
- Automatisches Finden und Identifizieren von Diensten im lokalen Netzwerk ohne einen zentralen Verzeichnisdienst

Ein wichtiger Punkt hierbei ist, dass ZeroConf eine eventuell bereits bestehende Netzwerk-Infrastruktur erkennen und verwenden kann, ohne diese dabei zu beeinträchtigen. In besonderem Maße interessant wird dieser Standard neben der hervorragenden Dokumentation auch durch die bereits vorhandene Verbreitung. ZeroConf (ehemals unter der Bezeichnung Bonjour bekannt) ist fester Bestandteil von MacOSX (ab Version 10.2) und wurde in vielen Unix-Derivaten inklusive Linux implementiert. Sogar für Windows gibt es einsatzfähige Implementierungen.

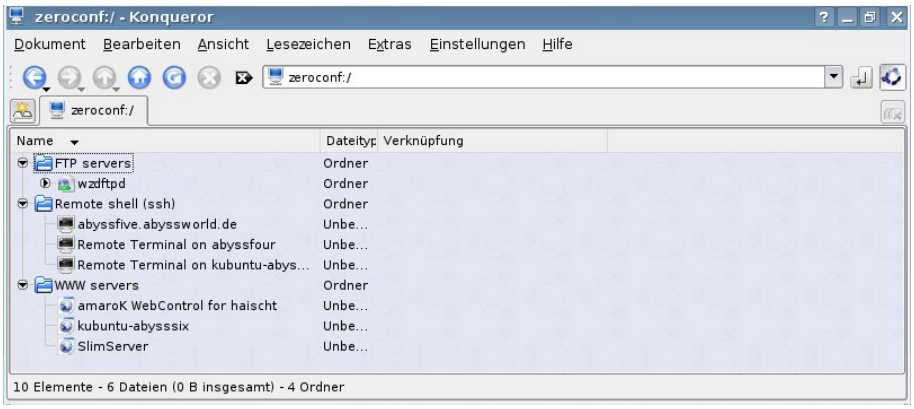

Abbildung 2.4: ZeroConf: Netzwerkdienste durch Konqueror/Avahi unter KDE Anzeigen

[12] Biespielsweise AppleTalk unter Mac OS [Sudhu u. a. (1990)] und Netbios unter Windows [Winston (2005)]

2.8.1 Automatische Zuweisung von IP-Adressen ohne DHCP-Server

Die Realisierung der automatischen Adressvergabe wurde durch “Link-Local IPv4-Adressen“ realisiert. Hierbei handelt es sich um eine IP-Adresse, welche aus dem von der IANA[13] für diesen Zweck vorgegebenen Addressbereich 169.254.0.0/16 ausgewählt wird. Bei dieser zufälligen Auswahl wird die MAC-Adresse des Netzwerk-Interfaces berücksichtigt, um nach Möglichkeit immer dieselbe Adresse für das Interface auszuwählen [IANA (2002)]. Durch eine spezielle ARP[14]-Anfrage (ARP-Probe) stellt das System vor dem Zuweisen der Adresse sicher, dass diese nicht bereits vergeben ist. Wird ein Konflikt entdeckt, so wird so lange eine neue Adresse generiert und über ARP getestet, bis eine freie gefunden wurde.

2.8.2 Namensauflösung ohne DNS-Server

Der von ZeroConf verwendete Ansatz basiert auf Multicast-DNS (mDNS) [Cheshire (2002)]. Dies ist ein Standard, der definiert, wie DNS-Anfragen an Multicast-Adressen gesendet werden sollen und wie die Empfängergruppe damit umzugehen hat, ohne dass redundanter Netzwerkverkehr entsteht. Für mDNS wird definiert, dass als Adressen nur die in Kapitel 2.8.1 auf Seite 15 beschriebenen Link-Local-Adressen verwendet werden. Der mDNS-Name für das System kann dabei frei ausgwählt werden. Ein Verfahren zur Konfliktlösung wurde bewusst nicht dokumentiert, da die Mehrfachbelegung eines Namens, z. B. in High-Availibility-Lösungen eine konkrete Anwendung findet.

2.9 JXTA

Bei JXTA[15] [CollabNet (2006)] handelt es sich um eine Ansammlung offener, generalisierter “peer-to-peer“ - Protokolle, die es ermöglichen, jedes an ein Netzwerk angeschlossenes Gerät, egal ob Handy, PDA oder PC, zu gleichberechtigten Kommunikationspartnern zu machen. JXTA unterstützt Java, C, C++ und C#, und durch die Unterstützung von JavaME [SUN (2005)] wird der Einsatz auf Miniaturgeräten plattformunabhängig angeboten.

Die JXTA-Protokolle definieren die Art in der P2P-Teilnehmer:

- Sich über Firewall-Grenzen hinweg gegenseitig finden

[13] IANA: Internet Assigned Numbers Authority, regelt die Vorgabe von IP-Adressen, Top Level Domains und Dienst-Ports (0-1024)
[14] ARP: Address Resolution Protocol, Übersetzung von IP- in MAC-Adressen et vice versa
[15] JXTA ist keine Abkürzung, sondern die Kurzform für “Juxtapose“, was so viel bedeutet wie “Seite an Seite“

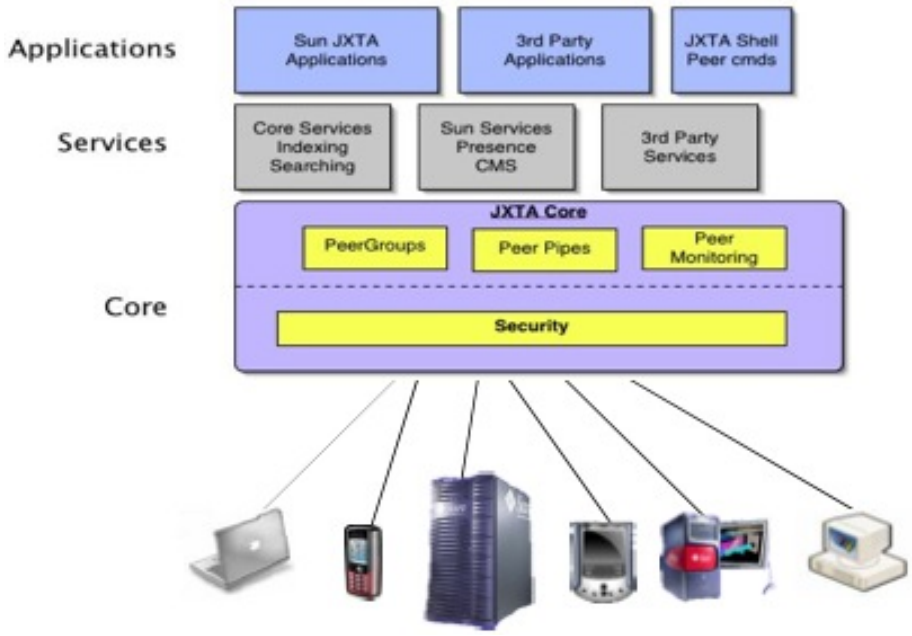

Abbildung 2.5: JXTA: Aufbau der Protokoll-Architektur [CollabNet (2006)]

- Sich in P2P-Gruppen selbst organisieren, z. B. um eine Gruppe von P2P-Teilnehmern gemeinsam einen Dienst anbieten zu lassen
- Miteinander kommunizieren und Daten sicher austauschen
- Sich gegenseitig überwachen

Die bei JXTA verwendeten Protokolle sind transportprotokollunabhängig ausgelegt, sie können auf Basis von TCP/IP, HTTP, Bluetooth, HomePNA oder anderen Protokollen implementiert werden. Das JXTA-Projekt wurde ursprünglich von einem SUN-Mitarbeiter ins Leben gerufen, wurde dann aber als Community-basiertes Open Source-Projekt bekannt. JXTA ist sehr gut dokumentiert und anhand von Tutorials kann das Framework sehr schnell in Betrieb genommen werden.

3 Beispielimplementierungen und Realisierungsansätze

Dieses Kapitel könnte im Prinzip auch “Auswertung“ heißen, der Schwerpunkt liegt hier jedoch auf der Untersuchung der in Kapitel 2 auf Seite 6 genannten Frameworks hinsichtlich der Verwendbarkeit für eigene Implementierungen im Rahmen des Szenarios “Rescue: Leitstand für Desaster-Szenarien”.
Im Rahmen des Projektes wurden neben der Untersuchung und Bewertung von aktuellen Technologien auch einige konkrete Ansätze verfolgt, um einen Prototypen für eine Referenzimplementierung zu erstellen. Die Realisierungsziele geben hierbei die Struktur dieses Kapitels vor.

3.1 Indoor-Kartendarstellung

In der “Ausarbeitung Ringvorlesung“ wurden bereits Anforderungen definiert, die an eine Kartenanwendung gestellt werden. Da die Anforderungen an Collaborations-Unterstützung und Usability-Aspekte nicht Ad-Hoc zu befriedigen sind, musste zunächst eine Basis geschaffen werden, auf deren Grundlage die entscheidenden Aspekte herausgearbeitet werden konnten.

Als zentraler Punkt der Kartenanwendung ist wohl die Darstellung zu bezeichnen. Hier wurde die Hauptanforderung auf eine möglichst große Implementierungsfreiheit gelegt, es kommt also nur ein offenes Framework in Frage, welches bei der Umsetzung die Darstellungs-Metaphern nicht einschränkt.
Das in Kapitel 2.2 auf Seite 7 diskutierte Framework GEF scheint diese Anforderung zu erfüllen. Zwar gibt es hier keine direkte Unterstützung für Kartendarstellung, aber der generische Ansatz scheint gut geeignet, um ein “Proof of Conzept“ zu erreichen. Bei der Arbeit mit dem GEF-Framework stand im Mittelpunkt, die übergreifende Bedeutung des Projektes deutlich zu machen, also die Kopplung der Themen “Gebäudesensorik“, “Wearable Computing“ und “Leitstand“ mit folgenden Anforderungen zu integrieren:

- Darstellung einer Kartenansicht einer fiktiven Etage, welche in einem Raum der Hochschule für Angewandte Wissenschaften Hamburg nachgebildet wird. Hierbei soll die architektonische Struktur der Räume und Durchgänge erkennbar sein.
- Darstellung von Sensoren und deren Werten. Hierzu ist eine Anbindung an den von Arno Davids [Davids (2007)] implementierten Location-Server notwendig, um an die Anzahl der Sensoren, deren Positionen und Werte zu gelangen.
- Darstellung einer Rettungskraft, in diesem Falle eines Feuerwehrmannes. Hierzu wird eine Kopplung zu dem Bereich “Wearable Computing“ hergestellt, welches von Stephen Hinck bearbeitet wurde Hinck (2007). Steffen Hinck stellt die über IMAPS[1] bestimmte Position der Rettungskraft und einige Statuswerte als Webservice zur Verfügung.

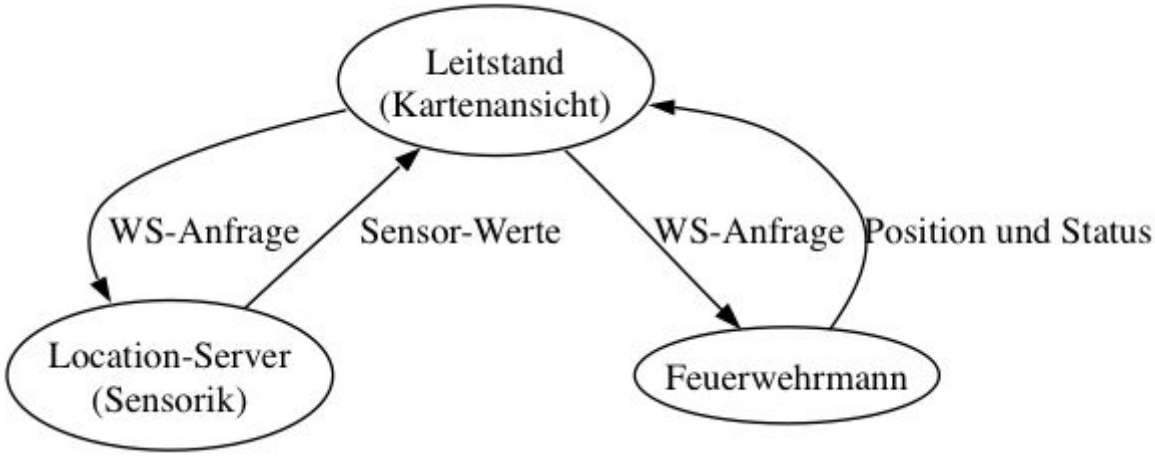

Abbildung 3.1: Übersicht über die Projekt-Kopplung hinsichtlich der Datenübergabe (WS = WebService)

Die Karte läßt sich hinein- und hinauszoomen und durch Scroll-Balken ist der Ausschnitt verschiebbar. Sensoren und Rettungskräften wurden Symbole zugeordnet, die auf der Karte dynamisch erscheinen und verschwinden bzw. ihre Position ändern (siehe Abbildung 3.2). Ein Thread sorgt dabei im Hintergrund für die Aktualisierung der dargestellten Werte. Durch eine eindeutige Zuordnung zwischen den abgefragten Sensorwerten und dem Sensor auf der Karte mittels einer ID wird das Hinzufügen und Entfernen von Sensoren erkannt und kann somit zeitnah auf der Karte dargestellt werden. Die Struktur der Kommunikationsanbindung kann Abbildung 3.1 entnommen werden.

[1] IMAPS: Indoor Distance Measurement and Positioning System, Positionsbestimmung über Bluetooth und Untraschall [Kutak (2006)]

Da für einen produktiven Einsatz eines Kartensystems das Kartenmaterial nicht wie in dem hier vorliegenden Prototypen manuell erzeugt werden kann, ist eine Import-Schnittstelle für Kartenmaterial von großer Bedeutung. Glücklicherweise liefert das GEF eine XML-basierte Import-Schnittstelle, die auch erweiterbar ist.

Abbildung 3.2: Bildschirmfoto der Beispiel-Implementierung für eine Indoor-Karte

3.2 Outdoor-Kartendarstellung

Die Anforderungen, die an die Outdoor-Kartenansicht gestellt werden, unterscheiden sich von den Anforderungen der in Kapitel 3.1 auf Seite 17 beschriebenen Indoor-Kartendarstellung. Zum einen spielt die Interaktion mit dem Kartensystem dort keine so große Rolle wie in dem Gebäude, zum anderen ist hier die Erfassung von Kartenmaterial eine andere. Auf Gebäude-Ebene gibt es Grundriss-Pläne, häufig bereits in digitaler Form. Für eine Outdoor-Karte wäre ein detaillierter und hochauflösender Kartenfundus wünschenswert.

Google Maps (siehe Kapitel 2.1 auf Seite 6) stellt dieses Kartenmaterial in der erforderten Qualität zur Verfügung. Die Tatsache, dass es sich bei Google-Maps um einen webbasierten Dienst handelt, macht die Integration in eine Desktop-Anwendung nicht gerade leicht, da die Kartendarstellung nur in einem Webbrowser erfolgen kann. Die Lösung, oder zumindest ein Workaround, ist der Einsatz von JDIC (siehe Kapitel 2.3 auf Seite 7), welches eine Browser-Komponente zur Verfügung stellt, die in die GUI einer Java-Anwendung integriert werden kann.

Die Auswahl des Kartenausschnittes erfolgt bei dem Webportal von Google-Maps durch ein HTML-Formular, welches an dieser Stelle aber nur bedingt geeignet scheint. Besser wäre die Auswahl direkt über die Java-Anwendung. Somit könnte eine Historie der besuchten Ausschnitte geführt werden und die Karte könnte kontextabhängig strategisch wichtige Punkte auf der Karte darstellen. Google bietet die Möglichkeit der Integration eines Kartenausschnittes in die eigene Webseite durch ein HTML-Code, der bereits die Angaben über den darzustellenden Kartenausschnitt enthält. Zum Berechnen der geografischen Position bietet Google einen “Geo-Coder“ an, der auch über XML-RPC angesprochen werden kann (siehe Kapitel 2.6 auf Seite 10. Mit diesen Informationen kann der HTML-Code für den Kartenausschnitt erzeugt werden. Um diesen nicht in einer ständig zu aktualisierenden Datei auf dem Festspeicher ablegen zu müssen, wurde ein HTTP-Server in die Applikation integriert, welcher den HTML-Code für den Kartenausschnitt bei einer Anfrage automatisch generiert und zurückliefert.
Das Ergebnis ist eine Kartendarstellung, die unter Verwendung des Google-Maps Kartenmaterials einen durch Freitext angegebenen geografischen Punkt direkt anspringen kann (siehe Abbildung 3.3).

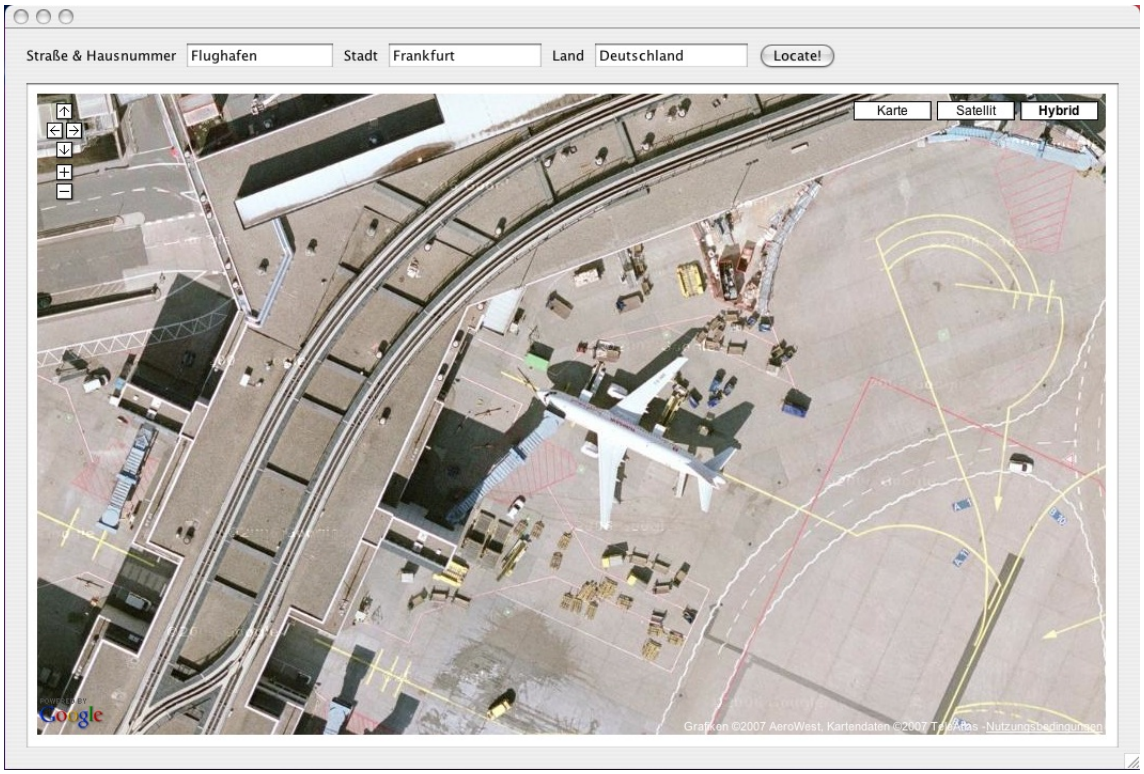

Abbildung 3.3: Bildschirmfoto der Beispiel-Implementierung für eine Outdoor-Karte

4 Bewertung und Ausblick

Die Anforderungen an eine Kartenanwendung, wie sie in der “Ausarbeitung Ringvorlesung“ definiert wurden, liegen außerhalb des Realisierungs-Horizontes der Veranstaltung “Projekt“. Die Ziele, die an den Prototypen für die Kartenansicht gestellt wurden, konnten aber erreicht werden. Es ist gelungen, die benötigten Komponenten und Technologien ausfindig zu machen und auf deren Basis ein “Proof of concept“ für das Vorhaben zu erreichen.

In Kapitel 3 auf Seite 17 wurde lediglich auf eine Untermenge der in Kapitel 2 auf Seite 6 beschriebenen Technologien eingegangen. Das ist darauf zurückzuführen, dass mit der Erstellung eines Prototyps für eine Kartendarstellung nur ein Bereich der Aufgabenstellungen, die durch das Rescue-Szenario beschrieben werden, angegangen wurde.
Auf den Bereich der AdHoc-Dienstanbindung soll daher im Rahmen dieser Bewertung eingegangen werden.

4.1 AdHoc Dienstanbindung & Kommunikation

Die Bedeutung von Kommunikation (im Sinne von Informationsübertragung) und dem dynamischen Einbinden von Diensten in eine kollaborative Umgebung ist mit dem Fortschreiten des Projektes immer deutlicher geworden. Während die funktionalen Anforderungen an eine Kartendarstellung relativ hart definiert und damit konkret angegangen werden können, sind die Anforderungen, die durch “Usability“ und “kollaborative Arbeitsumgebunden“vorgegeben werden, insgesamt schwerer greifbar.

Eine genauere Betrachtung lässt deutlich werden, dass an die Kommunikation und Dienstanbindung besondere Anforderungen gestellt werden, die sich mit herkömmlichen Client-/Server-Architekturen nicht optimal abbilden lassen. Am ehesten gelingt es dem Projekt um iROS (siehe Kapitel 2.7 auf Seite 12), diese Anforderungen zu definieren. Die Ziele des Projektes sind dabei höher gesteckt als man nach einer oberflächlichen Betrachtung denken mag. Aus diesem Grunde wurden z. B. Sicherheitsaspekte und andere äußere Anforderungen ausgeklammert. Dies ermöglicht zum einen eine Konzentration auf die speziellen Bedingungen in einem kollaborativen Arbeitsumfeld, lässt aber auch Ansätze zur Kritik und Verbesserungsansätze zu.
Auf das Erkennen und Einbinden von neuen Diensten geht iROS insgesamt wenig ein, wie

JINI (siehe Kapitel 2.6 auf Seite 10), ZeroConf (siehe Kapitel 2.8 auf Seite 14) aber auch JXTA (siehe Kapitel 2.9 auf Seite 15) jedoch zeigen, gibt es hierzu bereits vielversprechende Ansätze, zudem handelt es sich um ein hochbrisantes und auch aktuelles Thema. Es ist kein Geheimnis, dass diese Technologien lediglich eine Basis schaffen um eine Kommunikation zwischen den Systemen aufbauen zu können. Um tatsächlich eine hohe Dynamik bei flüchtigen Applikationen zu erreichen, ist eine spezielle Auslegung der Softwarearchitektur notwendig, wie durch die Verwendung des “TupleSpace“-Speichers in iROS zu sehen ist.

4.2 Realisierungen im Rescue-Szenario

Die Erstellung einer Kartenansicht hat im Verhältnis zu dem Bereich “Kommunikation und Dienstanbindung“ ohne Zweifel die stärkere Bindung an das Rescue-Szenario, außerdem ist eine konkrete Anwendung mit definierbaren Aufgabenstellung einer der Voraussetzungen die für die Durchführung von Usability-Tests erfüllt werden müssen. Insgesamt scheint die Auswahl der bei dem Realisierungsbeispiel verwendeten Softwarekompontenten brauchbar. Durch die solide Softwarearchitektur des GEF’s und die Verfügbarkeit des Quellcodes ist der Realisierungsansatz auch für die Umsetzung der in der “Ausarbeitung Ringvorlesung“ erklärten Eingabe-Metaphern verwendbar.

Der Bereich der Kommunikation und Dienstanbindung ist zwar weniger plastisch, lässt sich dafür jedoch in seiner Anwendung weitaus besser verallgemeinern. Ein eng verzahntes, kollaboratives Arbeiten findet sich nicht nur in einem Katastrophen-Szenario wieder, sondern gewinnt für die Art, wie wir mit Computersystemen arbeiten, zunehmend an Bedeutung.

Literaturverzeichnis

[Boger und Robbins 2006] BOGER, Marko ; ROBBINS, Jasin: *ArgoUML: A UML design tool with cognitive support*. 2006. – URL http://argouml.tigris.org/

[Cheshire 1999] CHESHIRE, Stuart: *Zero Configuration Networking (Zeroconf)*. 1999. – URL http://www.zeroconf.org/

[Cheshire 2002] CHESHIRE, Stuart: *Multicast DNS*. 2002. – URL http://www.multicastdns.org/

[CollabNet 2006] COLLABNET: *JXTA: Get connected. Perr to peer Framework*. 2006. – URL http://www.jxta.org/

[Consortium 1998] CONSORTIUM, World Wide W.: *Precision Graphics Markup Language (PGML)*. April 1998. – URL http://www.w3.org/TR/1998/NOTE-PGML-19980410

[Consortium 2003a] CONSORTIUM, World Wide W.: *Scalable Vector Graphics (SVG) 1.1 Specification*. January 2003. – URL http://www.w3.org/TR/SVG/

[Consortium 2003b] CONSORTIUM, World Wide W.: *SOAP Version 1.2 Part 1: Messaging Framework*. June 2003. – URL http://www.w3.org/TR/soap12-part1/

[CORBA 2000] CORBA, Object Management G.: *Catalog of OMG CORBA®/IIOP® Specifications*. 2000. – URL http://www.omg.org/technology/documents/spec_catalog.htm

[Cunningham 2006] CUNNINGHAM: *Tuple Space: An implementation of AssociativeMemory that provides a repository (mathematically a bag) for tuples, generatively.* 2006. – URL http://c2.com/cgi/wiki?TupleSpace

[Davids 2007] DAVIDS, Arno: *Ad-hoc Sensornetzwerk zur Gebäudeüberwachung und Navigationsunterstützung (Ausarbeitung Seminar/Ringvorlesung)*. 2007

[Gamma u. a. 2004] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design patterns*. Addison-Wesley, 2004. – ISBN 0-201-63361-2

[Garrett 2005] GARRETT, Jesse J.: *Ajax: A New Approach to Web Applications*. Februar 2005. – URL http://www.adaptivepath.com/publications/essays/archives/000385.php

[Google 2006a] GOOGLE: *Google Maps*. 2006. – URL http://maps.google.de/maps

[Google 2006b] GOOGLE: *Google Maps API-Description*. 2006. – URL http://www.google.com/apis/maps/

[Hinck 2007] HINCK, Steffen: *Einsatz von Wearable Computing in Desaster Szenarien (Ausarbeitung Ringvorlesung)*. 2007

[IANA 2002] IANA: *Special-Use IPv4 Addresses*. 2002. – URL http://tools.ietf.org/html/rfc3330

[Kutak 2006] KUTAK, Edyta: *IMAPS: Hardwareplattform für Positionsbestimmung innerhalb von Gebäuden*. 2006. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master06-07-aw/kutak/folien.pdf

[Laux 2004] LAUX, Thorsten: *Java Desktop Integration Components*. Juni 2004. – URL https://jdic.dev.java.net/

[Steinberg und Cheshire 2005] STEINBERG, Daniel ; CHESHIRE, Stuart: *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., 2005. – ISBN 0596101007

[Sudhu u. a. 1990] SUDHU, Gursharan S. ; ANDREWS, Richard E. ; OPPENHEIMER, Alan B.: *Inside AppleTalk: Second Edition*. 1990. – URL http://developer.apple.com/MacOs/opentransport/docs/dev/Inside_AppleTalk.pdf

[SUN 2005] SUN: *The Java ME Platform — The Most Ubiquitous Application Platform for Mobile Devices*. 2005. – URL http://java.sun.com/javame/index.jsp

[SUN 2006] SUN, Microsystems: *Jini Network Technology*. 2006. – URL http://www.sun.com/software/jini/

[Tarling und Robbins 2006] TARLING, Bob ; ROBBINS, Jasin: *Graph Editing Framework*. 2006. – URL http://gef.tigris.org/

[UserLand 1998] USERLAND, Software: *XML-RPC: Extensible Markup Language Remote Procedure Call*. April 1998. – URL http://www.xmlrpc.com/

[Winograd u. a. 2002] WINOGRAD, Terry ; JOHANSON, Brad ; FOX, Armanda: *Stanford Interactive Workspaces Project Overview*. Februar 2002. – URL http://iwork.stanford.edu/

[Winston 2005] WINSTON, Gavin: *Netbios Specification*. 2005. – URL http://www.netbiosguide.com/

[ZeroC 2006] ZEROC: *Internet Communication Engine*. 2006. – URL http://www.zeroc.com/