Forschungszentrum Informatik
an der
Universität Karlsruhe
Forschungsbereich Softwaretechnik

Software-Architekturen für lokalitätsabhängige Dienstleistungserbringung auf mobilen Endgeräten

Cand. Inform. Marc Schanne

Diplomarbeit
am
Forschungszentrum Informatik
an der
Universität Karlsruhe (TH)
Forschungsbereich Softwaretechnik
Prof. Dr. Walter Tichy
Dr. Michael Philippsen; Dipl. Inform. Andy Walter

Patet omnibus veritas.[1]

[1] Die Wahrheit ist allen zugänglich. Seneca [Sch1994]

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die verwendeten Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Marc Schanne Karlsruhe, den 29. Oktober 2000

Kurzfassung

Das Heywow-System ist eine Infrastruktur für mobile Handcomputer. Über lokale Sendebaken soll dem Benutzer der Zugriff auf dezentrale Dienstleistungen ermöglicht werden. Neben dem Zugriff auf zentrale Dienste, private Daten, Termine und Informationen soll das mobile Endgerät zu einem umfassenden Reiseassistenten ausgebaut werden. Für zielgenaue Navigation können neben der Information globaler Positionierungssysteme auch einfache Dienste im lokalen Sendernetz eingesetzt werden.

In der vorliegenden Diplomarbeit wird ein Rahmensystem für die lokale Dienstleistungserbringung konzipiert und entwickelt. Dabei werden insbesondere folgende Probleme gelöst:

- Die einfache Diensterbringung und Dienstentwicklung

  Mit Einsatz einer Java-Klassenbibliothek auf Basis des abstrakten Netzwerkprotokolls Jini lassen sich Dienste für beliebige Handcomputer entwickeln und anbieten.
  Für die Entwicklung wird ein einfaches Programmskelett vorgegeben, das die Einbindung auch komplexer Dienstleistungen in das Heywow-System erleichtert und für den Benutzer Funktionsgarantien der lokalen Dienste bereithält.

- Ein Nebeneinander von zentralen und ortsabhängigen, lokalen Diensten

  Das Jini-Programmierparadigma erlaubt die Integration von zentralen und lokalen Dienstformen. Durch die Zwischenspeicherung der Jini-Stellvertreter-Objekte wird ein störungsarmer Dienst im Funknetzwerk möglich.

- Der ressourcensparende Betrieb auf den Handcomputern

  Für den energiesparenden Einsatz der tragbaren Handcomputer wird ein zusätzliches Kommunikationsprotokoll über Multicast integriert und dem Benutzer eine weitere Entscheidungsmöglichkeit bei der Kommunikation mit dem Jini-System eröffnet.
  Mit dem Aufbau auf ein Standard-IP-Netzwerk mit TCP und UDP bleibt die Klassenbibliothek offen für zukünftige Entwicklungen in der mobilen Netzwerktechnik bei Funk oder Infrarot.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Motivation

Der Einsatz leistungsfähiger mobiler Endgeräte - Mobiltelefone, persönlicher digitaler Assistenten (PDAs) oder Notebooks - im Alltag bietet unzählige neue Möglichkeiten. Neben ihrer originären Aufgabe will Heywow[1] ein Kommunikations- und Navigationssystem basierend auf mobilen Handcomputern anbieten [ARS2000a] und mit ihnen z. B. die Reiseplanung unterstützen. Eine detailliertere Beschreibung von Heywow findet sich in Abschnitt 2.4.

Für einen umfassenden, persönlichen Reiseassistenten ist neben dem Angebot zentraler Dienste auch der Zugriff auf lokale Dienstgeber in Gebäuden, U-Bahnstationen oder z. B. Flughäfen nötig. Die lokalen Dienste ermöglichen eine zielgenaue Navigation und den Zugriff auf zusätzliche lokale Dienstleistungen, von Kauf bis Fahrplanauskunft. Die lokalitätsabhängigen Dienste können in Kombination mit dem direkten Zugriff auf Termine und Daten der Benutzer eine personalisierte Dienstleistung erbringen.

Architekturen für den Einsatz zentraler Dienstgeber sind heute bereits durch die Entwicklung im Festnetz bekannt und erprobt. Auch der Zugriff von mobilen Endgeräten ist in der Kommunikation üblich. Aus Benutzersicht ist es irrelevant, ob der Dienst von einem zentralen Dienstgeber oder lokal erbracht wird. Für den Dienstnehmer muss ein transparenter Zugriff auf beliebige Dienstleistungen mit dem mobilen Handcomputer möglich werden. Diese Anforderung ist besonders relevant, weil das Endgerät in Heywow kein bisher üblicher autonomer Handcomputer mit eigenständigen Applikationen ist. Die Dienstleistung für den Benutzer kann durch die direkte Interaktion des Dienstnehmers (Handcomputer) mit dem Dienstgeber erbracht werden. Hier bieten lokale Dienste inhärent die zusätzliche Ortsinformation ohne Notwendigkeit zusätzlicher Kommunikation.

Die Mobilität der verwendeten Handcomputer und die geringen Reichweiten bei der Funkverbindung zwischen Handgeräten und lokalen Kommunikationsstationen stellt besondere Anforderungen an das verwendete Netzwerk und die angebotenen Dienstleistungen. Solange ein Dienstgeber in funkweiter Entfernung zu erreichen ist, muss der Dienst eine Garantie für den Zugriff auf seine Leistung geben können. Wenn solche

[1] http://www.heywow.com/

Garantien nicht vorhanden sind, darf der Handcomputer dem Benutzer die Dienstleistung nicht länger anbieten.

Der Einsatz von mobilen, batteriebetriebenen Geräten erfordert von den Diensten einen stromsparenden Betrieb auf dem Handcomputer. Dienste müssen dort besonders den stromintensiven Funksendebetrieb reduzieren. Durch die Verwendung offener Standards muss der Einsatz zukünftiger, vielleicht energiesparenderer Hardware-Entwicklungen möglich sein.

So ist auch die Anforderung an den Betrieb mit sehr unterschiedlichen Systemkomponenten zu beachten. Zwischen den verschiedenen Teilnehmern muss bei der Kommunikation die Sicherheit und Vertraulichkeit privater Daten garantiert werden können.

Da heute leider noch nicht der gesamte Funktionsumfang von zukünftigen lokal und zentral angebotenen Diensten fest bestimmt werden kann, muss ein möglichst offener Systemansatz gewählt werden, über den alle erdenklichen Anwendungen verfügbar gemacht werden können.

1.2 Zielsetzung

In dieser Arbeit wird das von Heywow benötigte Rahmensystem für lokale Dienstangebote modelliert.

Es wird eine Infrastruktur entworfen, die die Entwicklung von Diensten und Programmen für die Benutzung lokalitätsabhängiger Dienstleistungen ermöglicht.

Die Implementierung ist 100% pures Java, und mit der Verwendung von Jini als Basistechnologie kann das Rahmensystem auf einer beliebigen lokalen IP-Netzinfrastruktur aufbauen. Die konsequente Verwendung des Jini-Dienste-Entwurfsmusters ermöglicht eine transparente Integration auch globaler Dienstleistungen in das lokale System von Heywow.

Durch Erweiterung der Jini-Protokolle um eine zusätzliche Kommunikation über IP-Multicast können auf dem mobilen Handcomputer energiesparende Anwendungen, die Daten nur empfangen und nicht senden müssen, eingesetzt werden.

1.3 Verwandte Arbeiten

Die weitere Verbreitung von mobilen Handcomputern im Alltag und der Wunsch nach allgegenwärtigem Rechnen (*Ubiquitous Computing*), der Möglichkeit an jedem Ort auf IT-Dienstleistungen und eine personalisierte Systemumgebung zugreifen zu können, fördert die Entwicklung von System-Architekturen, die dies unterstützen.

Im Folgenden wird ein knapper Überblick und Vergleich mit bereits existierenden oder geplanten Systemen gegeben:

Im Rahmen eines Projekts an der Universität Saarbrücken, IRREAL[2], wurde ein Gebäude-Informations- und Navigationssystem entworfen. Durch die Installation leistungsstarker Infrarotsender können lokal Informationen gesendet werden [BBK2000]. Diese werden über die IrDA-Schnittstelle handelsüblicher PDAs empfangen und verarbeitet. Derzeit wird ein System, basierend auf IRREAL, von der Firma eyeled[3] unter dem Namen Redboard für den Einsatz in großen Gebäudekomplexen vertrieben.

Erst mit der Entwicklung eines speziellen Übertragunsprotokolls wird, trotz des geringen Datendurchsatzes über Infrarot, mit leistungsstarken Sendern eine eindirektionale Kommunikation zu mehr als einem mobilen Handcomputer möglich. Das Heywow-System baut auf ein vollwertiges IP-Netzwerk auf. Hier kann der Handcomputer auch selbst aktiv mit einem lokalen Dienstgeber kommunizieren. So wird es auch möglich, komplexe Dienstleistungen auf dem mobilen Handgerät anzubieten.

In einem weiteren europäischen Gemeinschaftsprojekt, Moby Dick[4], wird seit 1995 eine System-Architektur für mobile Multimedia-Computer entwickelt [HS2000]. Mit dem Bau von verschiedenen experimentellen Computersystemen wurden im Rahmen der Forschung verschiedene Aspekte eines Gesamtsystems näher untersucht [Mob1997].

Die Forschungsergebnisse mit Moby Dick können die gesamte System-Architekur einer neuen Generation von mobilen Handcomputern beinflussen. Der hier vorgestellte Ansatz von Heywow versucht durch Verwendung bestehender Standards, mit einer Implementierung in 100% purem Java und aufbauend auf das abstrakte Netzwerkprotokoll Jini schon jetzt auch für zukünftige Entwicklungen offen zu bleiben. Insbesondere die Ergebnisse der stromsparenden Verwendung von Multicast-Kommunikation (vgl. Abschnitt 2.1.6) sind in das Konzept für das Heywow-System aufgenommen worden.

Ebenfalls mit Interesse an allgegenwärtigem Rechnen wird in den Forschungslabors von Hewlett-Packard an der Idee für zukünftige Computer- und Informationssysteme gearbeitet. Mit einer eigenen Implementierung der Java Entwicklungsumgebung 1.1, Chai, wird ein auf offenen Web-Standards aufbauendes System entwickelt [KEa2000]. HP CoolTown[5] ist derzeit noch eine Vision von Hewlett-Packard. In Zukunft sollen alle Geräte und Dienste Teil des *World Wide Web* (WWW) werden und können mit einer eindeutigen Adresse (URL) identifiziert und verwendet werden. Hewlett-Packard strebt eine Komplettlösung für die Anbindung kleiner Geräte in einen Diensteverbund an. Im Vergleich zum bei Heywow verwendeten Jini sollen hier insbesondere auf Dienstgeberseite Ressourcen eingespart werden können. Für den Einsatz komplexerer Dienste muss für die Kommunikation zwischen Dienstgeber und Stellvertreter ein zusätzlicher

[2] http://w5.cs.uni-sb.de/irreal/
[3] http://www.eyeled.com
[4] http://wwwhome.cs.utwente.nl/~havinga/mobydick.html
[5] http://www.cooltown.hp.com/

Kontrolldienstgeber eingesetzt werden, der in Jini durch Einsatz von verteilten Ereignissen, Zeitzusicherungen (vgl. Abschnitt 2.2.4) und Transaktionen auf Basis von Java obsolet ist [Keh2000].

Ein Konzept außerhalb der Ideen für mobile Geräte und lokaler Dienstleistungserbringung stellt eine Arbeitsgruppe am Massachusetts Institute of Technology (MIT) vor. Mit Hive[6], einer Infrastruktur für die intelligente Küche [Gra1999], wird eine dezentrale, verteilte Plattform für mobile Agenten beschrieben. Ähnlich wie der Jini-Programmieransatz soll es mit Hive möglich sein, Anwendungen durch Verwendung von lokalen Diensten [MEa1999] zu implementieren. Der wesentliche Unterschied von Hive zu Jini liegt im erweiterten Ansatz der Mobilität bei Hive. Anders als bei Jini können nicht nur die Dienstnehmer, sondern auch die Dienstgeber (im Sprachgebrauch von Hive: Agenten) ihren Ort wechseln. Dies ermöglicht die Migration von Agenten, abhängig von den Leistungsanforderungen und Systemressourcen innerhalb des Netzwerks. Für das Heywow-System ist dieses allgemeine Konzept nicht notwendig. Die in den Dienst integrierte Ortsinformation der dezentralen Diensteanbieter widerspricht einer beliebigen Migration der Agenten.

1.4 Überblick

Im Kapitel 2 der Arbeit werden zunächst einige Grundlagen erläutert, die im Zusammenhang mit der Entwicklung eines serviceorientierten, plattformunabhängigen und drahtlosen Informationssystems für tragbare Handcomputer nötig sind. Oft stellen diese Informationen nur einen Einstieg in umfassendere Problembereiche dar und mit Literaturverweisen wird eine speziellere Vertiefung ermöglicht.

Das Kapitel 3 beschreibt alle Anforderungen an das Heywow-System. Ausgehend von den Problemen wird ein Konzeptvorschlag für das System von Heywow genauer dargestellt. Hier werden die nötigen Software- und Kommunikationskomponenten für die Plattform beschrieben und eine Implementierung nahe gelegt.

Als Java Klassenbibiothek wird dies in Kapitel 4 realisiert und besonders die Erweiterungsmöglichkeiten und die Übertragbarkeit näher beleuchtet. Neben einer ausführlichen Beschreibung des Rahmensystems für die Entwicklung von Diensten in Heywow wird anhand einiger einfacher Beispiele und einer einfachen Benutzer-Schnittstelle die Leistungsfähigkeit verdeutlicht.

[6] http://hive.sourceforge.net/

2 Grundlagen

Im vorliegenden Kapitel werden einige Grundlagen erläutert, die für die folgenden Überlegungen notwendig sind. Neben den Möglichkeiten der Programmiersprache Java und Erweiterungen durch das abstrakte Netzwerkprotokoll Jini wird in Abschnitt 2.4 auch das Gesamtsystem von Heywow näher beschrieben.

Der Abschnitt 2.3 erläutert besondere Einschränkungen durch die mobile Plattform und in Abschnitt 2.5 werden verschiedene Techniken für die drahtlose Kommunikation kurz vorgestellt.

2.1 Java

Java ist eine einfache, objektorientierte, verteilte, interpretierte, robuste, sichere, architekturneutral portable, hochleistungsfähige, Multithread-fähige und dynamische Sprache [Fla1998, Kap. 1], die Mitte der 90er Jahre von der Firma Sun Microsystems Inc. veröffentlicht wurde.

Mit Einführung der Java 2 Plattform, einer Sammlung von Spezifikationen und Programmierschnittstellen (im Folgenden auch kurz APIs) [Sun1999], hat Sun Microsystems das Java-System in drei unterschiedlich umfangreichen Ausgaben vorgestellt.

Grundlegend für diese Arbeit ist die Spezifikation der *Micro Edition* (J2ME)[1], eine eingeschränkte Version des Java-Systems für kleinere Geräte des wachsenden Markts für eingebettete Systeme in Geräten für den Endverbraucher. Die J2ME beinhaltet ein hochoptimiertes Java-Laufzeitsystem mit geringsten Ansprüchen an die Hardware-Ressourcen. Dies wird erreicht durch die Reduktion der Basisklassen. Diese können in bestimmten Anwendungsbereichen mit sogenannten *Profiles* um Klassen für ausgewählte Funktionalität ergänzt werden.

Um eine größere Bandbreite an Plattformen abzudecken, ist die *Java 2 Micro Edition* selbst in zwei Konfigurationen aufgeteilt [Gla2000]. Interessant für den Markt der Handcomputer und PDAs ist sowohl die *Connected Limited Device Configuration* (CLDC) für

[1] http://java.sun.com/j2me/

Geräte, die der Java Virtual Machine (JVM) nicht mehr als 128 KB Speicher bieten können [Sun2000], als auch die *Connected Device Configuration* (CDC), für etwas besser ausgestattete Endgeräte mit einer Java-Laufzeitumgebung mit mindestens 512 KB ROM und 256 KB Arbeitsspeicher [JSR2000a].

Die folgenden Abschnitte 2.1.1 bis 2.1.6 beleuchten diejenigen Eigenschaften von Java, die für die vorliegende Arbeit von Bedeutung sind.

2.1.1 Kontrollfäden

Bei der Entwicklung von Java war die Multithread-Fähigkeit ein wesentliches Entwurfskriterium. Das Konzept des Multitasking auf Threadebene ist in die Programmiersprache integriert und wird nicht erst durch zusätzliche Bibliotheken oder Erweiterungen ermöglicht [OW1997, Kap. 1]. In der Java Virtual Machine können verschiedene Kontrollfäden (*Threads*) einer Anwendung quasi parallel bearbeitet werden. Dies ermöglicht besonders im Einsatz bei netzweit agierenden Applikationen eine nichtblockierende Interaktion des Servers mit mehreren Klienten.

2.1.2 RMI

Solche verteilte Anwendungen sind in Java leicht mit Hilfe des entfernten Methodenaufrufs (*Remote Method Invocation*, RMI [Sun1998b]) zu entwickeln. Neben Klassen und Schnittstellen in `java.net` und `java.io` für die Benutzung von TCP/IP-Netzen ist das RMI-Paket seit der Version 1.1 des Java Development Kits Teil der Java Plattform. Mit RMI ist es möglich, Objekte in Java nicht nur lokal in einer Applikation, sondern auch aus einer anderen JVM anzusprechen und zu verwenden.

2.1.3 Aktivierung

Mit Einführung der Java 2 Plattform wurde der entfernte Methodenaufruf erweitert [Cou1999, Kap. 9]. Mit Aktivierung (*Activation*) kann ein Dienst, der sich länger im Leerlauf befindet, beendet werden. Erst wenn er aktiv auf Klientenanfragen reagieren muss, kann er in einer neuen JVM neu gestartet werden.

2.1.4 Objekt-Serialisierung

Mit Objekt-Serialisierung kann die Instanz einer Java-Klasse in einen Strom von einzelnen Bytes geschrieben werden. Durch den Versand der Daten kann anschließend das Objekt, evtl. auf einer völlig anderen Java Virtual Machine rekonstruiert (deserialisiert) werden [Cou1999, Kap. 7].

Eine ausführliche Beschreibung dieses Verfahrens bietet die Java Object Serialization Specification [Sun1998a].

2.1.5 Reflection

Mit der Reflection API in Java 2 ist es möglich, auch unbekannte Java-Objekte zu inspizieren und zu benutzen. Über einfache Klassen und Schnittstellen des `java.lang.reflect`-Pakets ist es möglich, auf Felder, Konstruktoren und Methoden ohne Kenntnis der Klassenschnittstelle zuzugreifen.

Zusammen mit der Deserialisierung eines über das Netzwerk erhaltenen Java-Objekts ist es somit möglich, mächtige Java-Anwendungen zu entwickeln.

2.1.6 Multicast-Übertragung

Um in IP-Netzen Daten von einer zentralen Stelle aus an viele Empfänger übermitteln zu können, kann *Multicasting* eingesetzt werden [Net2000].

Bei diesem Übertragungsverfahren speist der Sender die Daten einmal in das Netz ein. Router duplizieren dann die Pakete und leiten sie zu den Adressaten in den jeweiligen Netzsegmenten weiter. Anders als bei Verfahren der Punkt-zu-Mehrpunkt-Kommunikation im herkömmlichen Unicast-Modell, kann ein Empfänger ohne Kenntnis des Senders einer Multicast-Gruppe beitreten und diese wieder verlassen.

Seit der Version 1.1 wird diese nicht verbindungsorientierte Form der Kommunikation mit UDP-Datagrammpaketen auch durch Java-Klassen unterstützt [Cou1999].

2.2 Jini

Die *Jini Connection Technology* (JCT) wurde von Sun Microsystems 1999 als neuer Meilenstein für die Welt der verteilten und vernetzten Systeme vorgestellt [GS2000]. Jini ist eine in 100% purem Java entwickelte Klassenbibliothek, die auf RMI aufbaut und zur dynamischen Verwaltung vernetzter Dienste eingesetzt werden kann.

Hardwarekomponenten oder Softwaredienste können mit einem Netzwerk verbunden werden und dort ihre Dienstleistung anbieten [New2000, Kap. 1]. Mit Jini ist es möglich, leistungsstarke verteilte Anwendungen zu entwickeln, die durch einen Verbund von Software- und Hardwarediensten entstehen [Mar1999, Part 1]. Ohne zusätzlichen Konfigurationsaufwand können durch den Einsatz der einfachen Basiskonzepte in Jini komplexe Dienste implementiert werden.

Eine ausführlichere Einführung in die JCT bieten z. B. die beiden Bücher [Edw1999] und [OW2000] an.

In den folgenden Abschnitten 2.2.1 bis 2.2.5 werden nur die für das Heywow-System wesentlichen Basiskonzepte kurz vorgestellt.

2.2.1 Nachschlagedienste

Nachschlagedienste (*Lookup-Service*s, im Folgenden auch kurz LUS genannt) sind die zentralen Anlaufstellen für alle Jini-Dienste [Dö1999], Softwaredienste oder Hardwarekomponenten. Hier können Dienste (*Services*) Stellvertreter für die Kommunikation registrieren.

Ein Stellvertreter (*Proxy*) ist eine serialisierte Java-Klasse, die über das Netz von einem HTTP-Dienstgeber bezogen werden kann.

Nachschlagedienste bilden einen Verbund der bei ihnen registrierten Dienste.
Jini-Anwendungen sind nicht nur einfach objektorientierte Applikationen, sie entstehen durch Verteilung von Java Objekten über eigenständige Dienstgeber, die über ihre Stellvertreter-Objekte miteinander kommunizieren können [New2000, Kap. 1]. Klienten können durch Benutzung der registrierten Dienste eines Verbunds eigene Dienstleistungen bieten. Diese Zugriffsform auf Dienstebene gewährleistet das dynamische Verhalten von Jini-Anwendung.

Vor der Verwendung eines Nachschlagedienstes muss sich dieser bei den Dienstgebern und Klienten im Netzwerk erst bekannt machen. Hierfür benutzt er das *Multicast Announcement* (vgl. Abbildung 2.1). Interessierte Klienten oder Dienstgeber, die sich beim

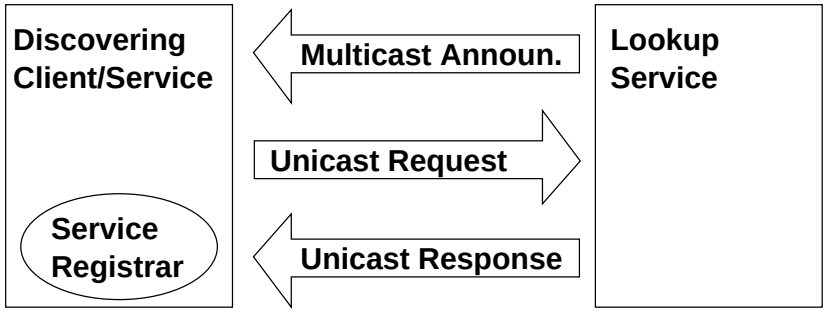

Abbildung 2.1: Multicast Announcement

Nachschlagedienst registrieren wollen, reagieren mit einer TCP-Unicast-Anfrage und erhalten in einer Unicast-Antwort ein `ServiceRegistrar`-Objekt zur Kommunikation mit dem LUS.

2.2.2 Discovery

Nachschlagedienste melden nur beim Start oder Neustart ihre Anwesenheit über UDP-Multicast. Der Prozess mit dem ein Dienstgeber oder Klient selbst aktiv wird, um im

Netzwerk existierende Nachschlagedienste zu suchen, heißt *Discovery*.

Unicast Request

Wenn die Adresse eines Nachschlagedienstes bekannt ist, kann über TCP mit einem *Unicast Request* (vgl. Abbildung 2.2) auf den Dienst zugegriffen werden. Dort kann di-

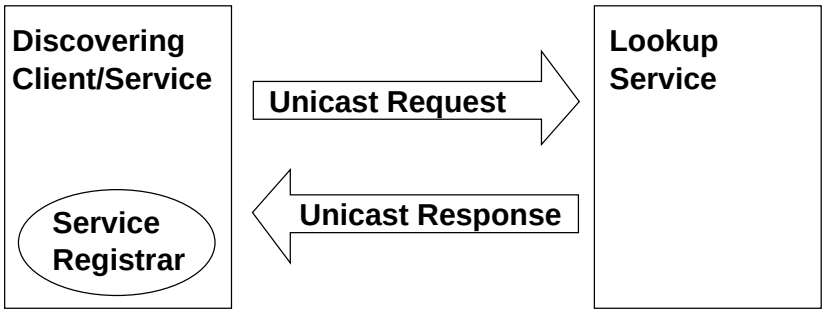

Abbildung 2.2: Unicast Request

rekt ein Stellvertreter-Objekt (`ServiceRegistrar`) des Nachschlagedienstes bezogen werden.

Multicast Request

Falls noch keine Adresse eines Nachschlagedienstes im Netzwerk bekannt ist, kann der Klient oder Dienstgeber mit einem *Multicast Request* (vgl. Abbildung 2.3) über UDP alle

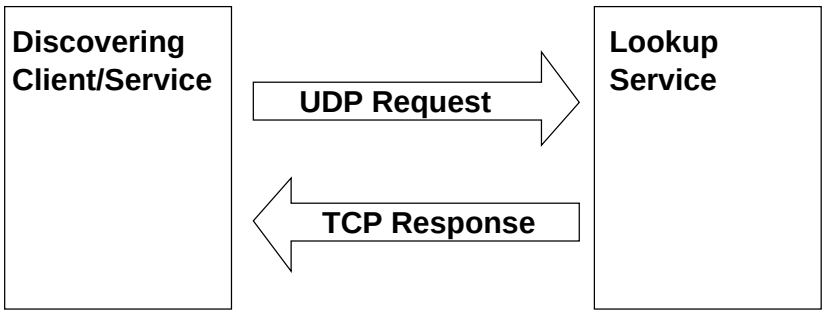

Abbildung 2.3: Multicast Request

erreichbaren LUS auf sich aufmerksam machen. Bei Erfolg melden die Nachschlagedienste mit TCP ihre Adresse an den Interessierten zurück. Jetzt kann der Klient mit einem *Unicast Request* ein `ServiceRegistrar` für die weitere Kommunikation anfordern.

2.2.3 Join und Lookup

In allen Nachschlagediensten, die sich durch ein Multicast-Announcement bekannt gemacht haben, oder die durch den Discovery-Prozess entdeckt worden sind, wird vom Diensterbringer ein Stellvertreter registriert. Diesen Vorgang nennt man *Join*. Der Nachschlagedienst speichert dieses Objekt für die spätere Kommunikation mit dem Dienstgeber und bietet es in seinem Verzeichnis für Klienten zur Benutzung an.

Ein Klient schickt an den Nachschlagedienst eine Anfrage nach einem Dienst mit einer bestimmten Schnittstelle. Erfüllt ein Stellvertreter diese Bedingungen, erhält der Klient als Antwort den gesamten Stellvertreter übermittelt. Diese Phase wird *Lookup* genannt. In Abbildung 2.4 wird ein Join und ein anschließendes Lookup schematisch dargestellt.

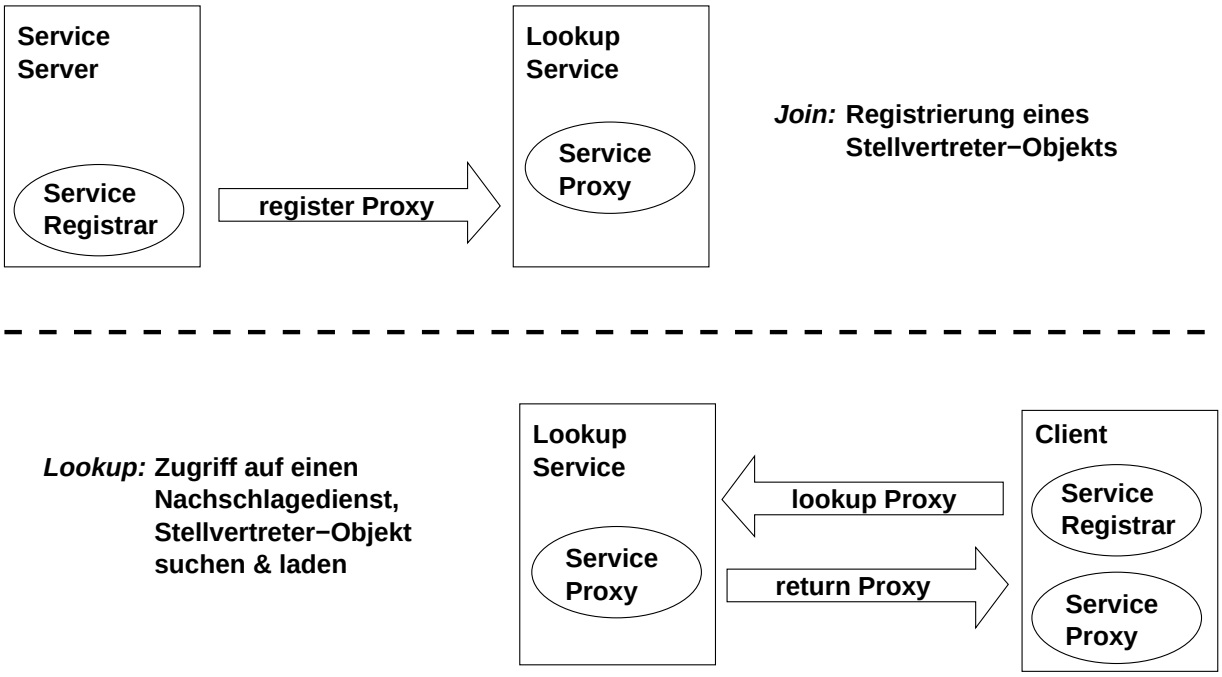

Abbildung 2.4: Join & Lookup

2.2.4 Leasing

Wichtige Voraussetzung für den fehlerfreien Ablauf einer Applikation mit Jini-Diensten sind Garantien über die in den Nachschlagediensten angebotenen Stellvertreter. In Jini ist eine Zeitzusicherung (*Leasing*) der Mechanismus, der von zwei Applikationen benutzt wird, um einen sicheren Zugriff auf Ressourcen für eine bestimmte Zeit zu gewährleisten [New2000, Kap. 7].

Mit der Registrierung eines Stellvertreters erhält der registrierende Dienstgeber eine Zusicherung (*Lease*), die Garantie des Nachschlagedienstes, dass das Stellvertreter-Objekt für eine vorgegebene Zeit Klienten angeboten wird. Bevor diese Garantiezeit abläuft und der Nachschlagedienst den Stellvertreter aus seiner Datenbank löscht, muss der Dienstgeber diese Zusicherung vom Nachschlagedienst verlängern lassen.

Das Verfahren der Lease-Vergabe zwischen Nachschlagedienst und einem sich registrierenden Dienst kann in Jini auch analog zwischen dem Dienstgeber und dem Dienstnehmer eingesetzt werden. Der Klient kann nur für die im Lease zugesicherte Zeit den Dienst sicher verwenden. Wenn er den Dienst länger in Anspruch nehmen möchte, muss er aktiv werden und eine Verlängerung beim Dienstgeber veranlassen.

Für die Verwaltung von Zusicherungen muss jeder Dienst eine Schnittstelle für das Anfordern, Erneuern und Löschen der Zeitverträge zur Verfügung stellen. Das Konzept, nach dem die Verwaltung von Zeitzusicherungen geregelt wird, ist in Jini nicht festgelegt. Hier kann eine beliebige Kombination von Verwalter und Zusicherung eingesetzt werden.

Eine Idee, das *Landlord-Leasing*, wird vom Prinzip der Lehenvergabe bei englischen Gutsherren (*Landlord*s) abgeleitet. Dieses einfache Verfahren kann auch im Heywow-System von beliebigen Diensten (Gutsherren) verwendet werden.

Landlord-Leasing

In England konnte ein Gutsherr an seine Lehennehmer Ländereien verpachten. Für die Verwaltung der Pachten und die Verhandlungen mit den Pächtern hat er einen speziellen Verwalter eingesetzt. Die Pächter konnten über die Lehen frei verfügen und sie selbst wieder an Bauern weiterverpachten. Der Bauer hatte kein direktes Vertragsverhältnis mit dem Gutsherrn oder dessen Verwalter und wenn der Lehennehmer, für den er das Land bewirtschaftete, starb, fiel das Lehen wieder an den Gutsherrn zurück. Der Bauer hatte keine Möglichkeit dies zu verhindern.

In Jini verhält sich ein Dienst ähnlich wie ein Gutsherr. Er setzt einen Verwalter ein, der Zeitzusicherungen nach den Vorgaben des Dienstes erzeugt. Mit der Angabe eines eindeutigen Schlüssels, über den die Zusicherungen von ihm verwaltet und evtl. veraltete gelöscht werden, wird von ihm eine Zeitzusicherung generiert. Im Sprachgebrauch der englischen Lehenvergabe ist jede Zeitzusicherung ein Lehennehmer. Die Zeitzusicherung enthält *intern* den Verweis auf den Verwalter.

Ein Klient, der einen Dienst benutzen möchte, fordert beim Dienstgeber eine Zeitzusicherung an.[2] Der Dienstgeber leitet diese Anforderung an seinen Verwalter weiter und dort wird dem Klienten ein Verweis auf eine Zeitzusicherung übergeben. Durch

[2]Hier bricht die Analogie mit der historischen Vorlage, früher konnte ein Bauer nicht direkt beim Gutsherrn einen zuständigen Lehennehmer erfragen.

die Existenz eines gültigen Zusicherungsobjekts wird dem Klienten ein funktionierender Jini-Dienst garantiert. Nur die Zeitzusicherung selbst kann mit dem Verwalter eine Verlängerung regeln.

In der aktuellen Implementierung der Jini Connection Technology definiert Sun Microsystems eine Schnittstelle nach diesem Konzept, die zusammen mit einer Fabrikmethode und den erzeugten Zeitzusicherungen verwendet werden kann [Edw1999, Kap. 11].

Durch den Einsatz von Landlord-Leasing ist es möglich im Heywow-System ein von den konkreten Diensten unabhängiges Zeitzusicherung-Verwalter-Modell einzubauen. Jeder Dienst kann einen Verwalter, passend für seine Notwendigkeit einsetzen und ihm die Verhandlungen mit den Zeitzusicherungen überlassen.

Das Landlord-Leasing ermöglicht es, ohne direkte Verträge zwischen Klienten und Verwalter eine Zeitzusicherung für die Dienstleistung zu garantieren.

2.2.5 Attribute

Zur Einschränkung beim Nachschlageprozess (vgl. Abschnitt 2.2.3) sind nicht nur die Vorgaben bestimmter Schnittstellen möglich, die der Stellvertreter unterstützen muss. In einem Nachschlagedienst kann mit jedem Dienst-Stellvertreter auch eine Menge von Attributen gespeichert und zur genaueren Auswahl angeboten werden. Beim Nachschlagen eines Stellvertreter-Objekts kann der Dienstnehmer neben einer Schnittstellenanforderung auch einen Katalog von (Soll-)Attributen, die der Dienst erfüllen soll, mitübergeben. Attribute sind einfache serialisierte Java-Objekte [Edw1999, Kap. 7].

2.3 Mobile Endgeräte

Mobile Endgeräte unterscheiden sich erheblich von Arbeitsplatzrechnern. Handcomputer sind batteriebetrieben und müssen deshalb mit ihren eingeschränkten Energieressourcen sparsam umgehen. Ein enthaltener Prozessor bietet weniger Rechenleistung und der verfügbare Speicherplatz ist beschränkt. Für die Kommunikation mit einer lokalen oder zentralen Gegenstellen ist ein Sende-Empfangs-Modul mit Funk oder Infrarot notwendig, dessen Energieverbrauch einen erheblichen Anteil des Gesamtsystems ausmacht.

Die erreichte Portabilität kann nur mit deutlichen Beschränkungen bei der Entwicklung von Anwendungen für mobile Endgeräte erreicht werden. Der Energieverbrauch kann durch geregelten Netzwerkzugriff und vorausschauendes Zwischenspeichern gering gehalten werden. Die beschränkten Speicherressourcen müssen deshalb verwaltet werden. Dieses Ressourcen-Management [DH1995] erfordert neben der Entwicklung spezieller Applikationen auf dem Handcomputer auch eine besondere Beachtung bei der Entwicklung von den lokal erreichbaren Diensten.

2.4 Heywow

Heywow baut auf die Möglichkeiten und besonderen Fähigkeiten von Java und Jini auf [ARS2000b]. Die Vorteile von Java, die Einfachheit tragbarer Handcomputer und die Leistungsfähigkeit größerer, lokaler und globaler Dienste werden verbunden.

Heywow stellt ein Architekturkonzept für die Kombination von Kommunikation und Navigationstechnologien dar [ARS2000a]. Der Ansatz, moderne Kommunikationstechniken und lokale Dienstbenutzung mit der Mobilität eines tragbaren Endgeräts zu kombinieren, erfordert neben der genauen Lokalisierung auch die Möglichkeit zur Kommunikation in drahtlosen Ad-Hoc-Netzwerken.

Für die Bestimmung der genauen Ortsinformation existieren mehrere technische Möglichkeiten. Neben den global erreichbaren Diensten und Informationen, wie dem Global Positioning System (GPS) oder der Lokalisierung über Zellen im Mobilfunk, sieht Heywow vor, dass auch Informationen von lokalen Sendeeinrichtungen zur Ortsbestimmung und Navigationsunterstützung verwendet werden. In Heywow werden diese Lokalisierungsdienste über die Benutzung von Jini vereinheitlicht angeboten.

Die lokale Diensterbringung des Typs 'Navigation/Positionierung' kann auf diese Weise auch auf allgemeine Dienstleistungen für die Reiseplanung, wie z. B. Einkaufs- und Gastroinformationen, erweitert werden. Über lokale Sendebaken soll dem Benutzer der Zugriff auf dezentrale Dienstleistungen ermöglicht werden. Mit dem Zugriff auf zentrale Dienste, private Daten, Termine und Informationen soll das mobile Endgerät zu einem umfassenden Reiseassistenten ausgebaut werden.

Diese Arbeit stellt in Kapitel 3 den Entwurf eines Rahmensystems für die Entwicklung von Diensten zur Leistungserbringung im lokalen Bereich vor und beinhaltet in Kapitel 4 die genauere Beschreibung einer Realisierung.

2.5 Drahtlose Kommunikationssysteme

Heywow verlangt für die lokale Kommunikation die Möglichkeit, zwischen den tragbaren Endgeräten, Sendebaken und Dienstgebern einfach Ad-hoc-TCP/IP-Netzwerke mit Multicast-Funktionalität zu bilden.

Drahtlose Kommunikation kann elektromagnetische Wellen als Übertragungsmedium, üblicherweise Infrarot- oder Radiowellen benutzen [Wet2000]. Für die mobile Kommunikation und den Zugriff auf lokale Informations- und Diensteinheiten wird oft das international genormt freie Industrial, Scientific, Medical (ISM) Band ab 2,4 GHz des Frequenzspektrums eingesetzt.

Allgemein beschränkt sich die Reichweite von Funk-LAN-Systemen (Radio-LANs) bei nicht gerichteter Ausstrahlung, je nach Energieverbrauch, auf eine Reichweite zwischen

10 und mehreren 100 Metern [DH1995, Kap. 3]. Als technische Systeme für Heywow werden im Folgenden kurz Bluetooth und IEEE 802.11 vorgestellt.

2.5.1 Bluetooth

Schon 1994 hat Ericsson begonnen, die Möglichkeiten für eine energiesparende und günstige Radio-Funk-Schnittstelle bei Mobiltelefonen und tragbaren (Kleinst-)Computern zu erforschen [WM1999, Kap. 2.1]. *Bluetooth-Technology* ist ein Codename für die Spezifikation eines kleinen Nahbereichs-Radio-Funknetzes zwischen mobilen Computern, Telefonen und sonstigen portablen Endgeräten. Sie dient der schnellen Einrichtung von leistungsfähigen *Ad-hoc-Netzen*, die sowohl Sprache als auch Daten übertragen können sollen [Hof2000].

Der Bluetooth Chip bietet eine Sendereichweite von 10 Metern und mit einem zusätzlichen Funkmodul lassen sich, bei einer Leistung von 100mW, bis zu 100 Meter erreichen.

Für die Verwendung mit Java und Jini kann Bluetooth über einen Protokoll-Stapel ein TCP/IP-Netzwerk aufbauen. Ausgehend von einem *Profile* für das Punkt-zu-Punkt-Protokoll (PPP) kann ein vollwertiges IP-Netzwerk angeboten werden (vgl. [BP1999, Kap. 2.1].
Nähere Beschreibungen finden sich in der Spezifikation von Bluetooth ([BC1999] & [BP1999]).

Durch die Integration kleiner, günstiger und energiesparender Sender in existierende mobile Endgeräte wird es jetzt möglich, allgegenwärtiges Rechnen zu schaffen [HE2000].

2.5.2 IEEE 802.11

Die Produkte für die Bluetooth-Technologie befinden sich im Moment noch in der Entwicklung hin zur angestrebten Massenproduktion.

Der bestehende Funk-LAN-Standard IEEE 802.11 sendet ebenfalls über das 2,4 GHz Band. Über ihn ist es eingeschränkt auch möglich, einen drahtlosen asynchronen Transfermode (WATM) zu realisieren [ALM1998].

Für Heywow interessant ist aber die Möglichkeit, in einem Funknetz über IEEE 802.11 stationäre, portable oder sich bewegende Geräte, bei einer Reichweite von 30 bis 360 Metern, über einen TCP/IP-Netzwerk Stapel verbinden zu können [Wet2000].

2.5.3 Vergleich

Beide Produkte sind für die lokale Kommunikation in Heywow einsetzbar. Mit beiden Funknetzwerken kann ein Standard-TCP/IP-Netz angeboten werden, auf dem das folgende Konzept mit der Implementierung in Java (vgl. Kapitel 4) aufbaut.

3 Konzept

Dieses Kapitel motiviert ausgehend von den Anforderungen an das Heywow-System ein mögliches Konzept für das Gesamtsystem. In Abschnitt 3.1 werden die nötigen Hardwarevoraussetzungen für das in Abschnitt 3.2 folgende Software-Modell vorgestellt. Hier werden die wesentlichen Entwurfsentscheidungen für die Software-Architektur genauer begründet.

Abschließend geht Abschnitt 3.3 nur kurz auf die Schwierigkeiten bei Sicherheit und Stabilität des Heywow-Systems ein.

3.1 Hardwarekomponenten

Im Heywow-System wird die Struktur der Hardwarekomponenten durch den möglichen Zugriff auf zentrale Dienste und die spontane Integration in ein lokales Funknetzwerk für die Kommunikation mit den lokalen, dezentralen Diensteanbietern, bestimmt. Diese Bedingungen ergeben einige konkrete Anforderungen an das System:

- Angebot lokaler Dienste von dezentralen Diensteanbietern
- Verfügbarkeit zentraler Dienste
- Ad-Hoc-Funknetzwerk für unterschiedliche tragbare Handcomputer

Dies bedeutet, dass neben einem Zugriff auf zentrale Dienstgeber über Mobilfunk für die lokale Dienstleistungserbringung, in Gebäuden oder U-Bahn-Stationen, im wesentlichen drei verschieden lokale Hardwarekomponenten notwendig sind.
Die Abbildung 3.1 zeigt schematisch die erforderlichen Netzwerke und alle Teilnehmer.

Für den Einsatz von zentral verfügbaren Dienstleistungen muss zwischen dem *Handcomputer* (WID) und einem *zentralen Server* (Heywow.com) eine Verbindung über Mobilfunk aufgebaut werden können.

Außerdem kann sich der Handcomputer per Funk in ein bestehendes lokales Festnetz integrieren. Dort stellen die *Dienstgeber* der lokalen Diensteanbieter ihre Jini-Dienste

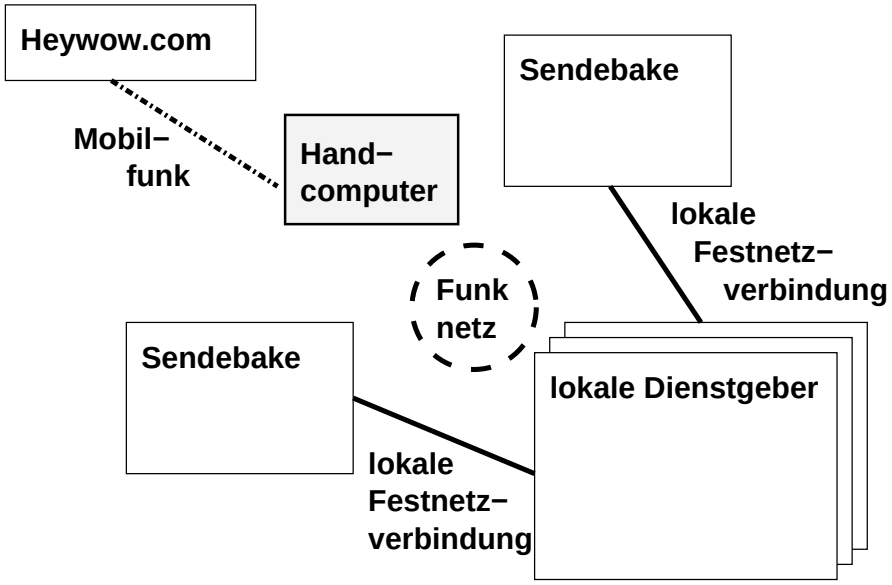

Abbildung 3.1: Hardwarekomponenten des Heywow-Systems

über strategisch verteilte *Sendebaken* (Beacons) bereit. Der Dienstgeber registriert bei einer allgemeinen Sendebake von Heywow einen Stellverteter für den Zugriff auf seinen Dienst.

Die Stellvertreter erbringen die Dienstleistung in Kommunikation mit dem eigenen Dienstgeber.

Dieses Konzept stellt an die Hardware in Heywow bestimmte Anforderungen. Der tragbare Handcomputer ist durch seine beschränkten Ressourcen bestimmt, er muss die eingeschränkten Speicher- und Stromressourcen verwalten. Für die Kommunikation mit dem Restsystem muss der WID sein Funkmodul energiesparend betreiben.

Für die im Festnetz integrierten Komponenten gelten diese Einschränkungen nicht. Die lokalen Dienste sind auf Dienstgebern mit Zugriff auf das Fest- und Funknetz implementiert. Sie besitzen eine direkte Verbindung zu den Sendebaken, die für alle Dienste einen Nachschlagedienst realisieren.

3.2 Softwarekomponenten

Die Teile der Software von Heywow richten sich nach der Struktur der Hardwarekomponenten. In den folgenden Unterkapiteln wird eine Klassenbibliothek für die Implementierung von lokalen Diensten mit unterstützenden Klassen für deren Verwendung entworfen.

Eine Anwendung auf dem mobilen Handcomputer kann nur durch die Kommunikation mit lokalen und zentralen Diensten für den Benutzer eine Leistung erbringen. Der

Zugriff für den Benutzer muss transparent und möglichst einfach gestaltet werden. Außerdem muss für die breite Einsetzbarkeit die große Vielfalt von mobilen Handcomputern unterstützt werden:

- Integration lokaler und zentraler Dienste
- Heterogenität der Klienten

Beide Anforderungen an das Softwaresystem von Heywow werden durch die Wahl der Programmiersprache und Programmstruktur gelöst. Der Einsatz der Java 2 Micro Edition (vgl. Abschnitt 2.1) und die Verwendung des Programmierkonzepts von Jini (vgl. Abschnitt 2.2) ermöglichen den Einsatz auf einer breiten Plattformpalette und den Zugriff auf lokale oder zentrale Dienstleistungen ohne Unterschied mit dem Gebrauch eines Stellvertreter-Objekts.

3.2.1 Dienste

Teil der weiteren Ausführungen ist jetzt besonders die Implementierung und der Einsatz von Diensten im lokalen Netzwerk. Beim Zugriff auf lokal verfügbare Ressourcen und Dienste muss WID-intern ein Jini-Stellvertreter-Objekt des verfügbaren Dienstes die Kommunikation mit dem Dienstgeber regeln.

Allgemeine Anforderungen ergeben sich wieder aus einem Benutzerszenario. Ein mobiler Handcomputer kann sich über das Funk-LAN in das bestehende Heywow-Netz einbinden. Er muss Zugriff auf alle Dienste in seinem Aktionsradius bekommen und diese kontrolliert einsetzen können:

- Lokale Dienstleistungen von dezentralen Diensteanbietern
- Funktionsgarantie der Dienste

Der Dienst wird auf einem dezentralen Dienstgeber gestartet und nach dem Start muss er sich mit einem Stellvertreter-Objekt im Jini-Netzwerk, d.h. bei allen erreichbaren Nachschlagediensten, registrieren.

Der Dienst kann auf einem im Netzwerk integrierten Dienstgeber des Anbieters oder auf einem zentralen Prozessor arbeiten.

Die Leistung des Dienstes erbringt der über einen Nachschlagedienst auf den WID bezogene Stellvertreter, evtl. in Zusammenarbeit mit der Dienstkomponente auf dem Dienstgeber. Durch die Kommunikation zwischen dem Stellvertreter auf dem WID und dem Dienstgeber kann der Benutzer mit dem Diensteanbieter interagieren. So sind komplexe Dienste mit Vertragsverhandlungen und Abstimmung der Beteiligten möglich.

Über die Funktionalität des Dienstes selbst hinaus, ist eine Klasse auf dem Dienstgeber auch nötig, für die ordentliche Integration in das Jini-Netzwerk von Heywow. Jeder Dienst muss seinen Benutzern Zeitzusicherungen (vgl. Abschnitt 2.2.4) über die Verfügbarkeit seiner Dienstleistung geben können. Eine solche Funktionsgarantie wird durch Verwendung des Lease-Prinzips aus Jini ermöglicht.

Der Dienstgeber-Teil eines Heywow-Dienstes ist für die Zeitzusicherungen zuständig. Diese Aufgabe wird durch eine zusätzliche Klasse erfüllt, die von allen Diensten instanziiert werden kann und für die Verwaltung eingesetzt wird.

Durch das Auslagern der Verwalter-Funktionalität in eine zusätzliche Klasse kann, wie in Jini üblich, eine beliebige Kombination von Zusicherung und Verwaltung (vgl. Vorschlag Landlord-Leasing in Abschnitt 2.2.4, S. 19) eingesetzt werden.

Mit dem Auslagern kann für die Verwalter-Aufgaben auch ein spezieller Dienst im Heywow-System eingeführt werden. Jeder registrierte Dienst kann die Anforderungen nach Zeitzusicherungen an einen Verwalter weitergeben und diesem die Kommunikation mit der speziellen Zeitzusicherung übertragen.

Falls der Stellverteter die gesamte Funktionalität des Dienstes beinhaltet, und keine Verbindung zu einem Dienstgeber nötig ist, kann die Vergabe von Zeitzusicherungen und deren Verwaltung auch umgangen werden. Nach einmaligem Bezug des Stellvertreters ist dieser bis auf weiteres gültig und es muss auch nicht für die Zeitzusicherung des Dienstes eine Verbindung zum Dienstgeber aufgebaut werden.

- Mächtige Stellvertreter-Objekte, die auch kommunikationsfrei Dienste erbringen können

Aus dem Einsatz von solchen mächtigen Dienst-Stellvertreter-Objekten ergibt sich die Notwendigkeit für einen WID-internen Zwischenspeicher für Stellvertreter-Objekte. Die eigenständigen Objekte, aber auch die normalen Stellvertreter-Objekte für die Kommunikation mit Dienstgebern können nach dem Bezug jederzeit verwendet werden. Für den Zugriff auf schon bekannte Stellvertreter-Objekte kann im mobilen Endgerät ein WID-interner Nachschlagedienst angeboten werden, über den eine Applikationen auf dem Handcomputer auf die Dienstleistungen zugreifen kann (vgl. Abschnitt 3.2.2).

Schwierig bei einem 'immer' gültigen Dienst ist in diesem Fall aber der Austausch einer veralteten Programm-Version. Wenn ein Stellvertreter bereits über einen Nachschlagedienst im Heywow-System bezogen worden ist, wird er WID-intern zwischengespeichert. Der Dienst erfüllt auf dem tragbaren Handgerät seinen Dienst und es besteht keine Notwendigkeit für den Bezug einer aktuelleren Version. Dieses Caching-Problem kann nur durch die Einführung einer Protokollerweiterung von Jini für Heywow behoben werden. Beim Einsatz eines Stellvertreter-Objekts auf dem Handcomputer kann durch einen Kontrollzugriff auf das Versionsattribut des Dienstes über die Aktualität der gespeicherten Version entschieden und evtl. ein neues Stellvertreter-Objekt bezogen werden.

Um die aufwendigen Kontrollen zu umgehen, ist es nützlich, das Prinzip der Zeitzusicherungen aus Jini für die implementierten Dienste zu unterstützen. Ein Dienst sichert für eine festgelegte Zeit die Gültigkeit seines Stellvertreters zu. Erst bei einer späteren Benutzung muss der Klient diese Zusicherung verlängern. Falls dies nicht möglich ist, muss evtl. ein Abgleich mit der aktuellen Version auf dem Dienstgeber vorgenommen werden.

Ein großer Nachteil des bisher vorgestellten Dienstgebers ist, dass er fortwährend einen ganzen Betriebssystemprozess, einschließlich Speicher und sonstigen Ressourcen benötigt. Nach dem Start wird der Prozess für die Verwaltung der Leases und die eigene dauerhafte Registrierung im Jini-Netzwerk benötigt.

Durch Registrierung des Dienstes im Aktivierungssystem (vgl. Abschnitt 2.1.3) können auf diese Weise auch auf dem Dienstgeber Systemressourcen eingespart werden.

Bei der Implementierung konkreter, lokaler Dienste ergibt sich noch eine weitere wesentliche Anforderung:

- Einfache Implementierung konkreter Dienste

Da die grundlegende Struktur und Aufgaben für die Verwaltung bei allen Diensten im Heywow-System gleich sind, empfiehlt sich für die Entwicklung der Dienste der Einsatz einer Klassenbibliothek unter Verwendung des Entwurfsmusters Schablonenmethode. Hier kann die Defintion eines Skeletts für die notwendigen Aufgaben eines Dienstes im Heywow-System abstrahiert werden.

Mit Verwendung unterstützender Basismethoden kann durch Klassenvererbung die bestehende Struktur für konkrete Dienste einfach angepasst und ergänzt werden.

3.2.2 Handcomputer

Um den Einsatz der über Funk in den lokalen Nachschlagediensten erhältlichen Stellvertreter-Objekten durch die Anwendungen auf dem Handcomputer zu erleichtern, kann auf dem mobilen Handgerät ein WID-interner, erkundender Nachschlagedienst als Zwischenspeicher eingerichtet werden. Jeder WID-interne Nachschlagedienst enthält eine Sammlung der für den Benutzer interessanten, lokal erreichbaren Dienste. Durch diese Form des Zwischenspeichers ist es möglich die aus Jini bekannten Programmierkonzepte für Applikationen auf dem Handcomputer einzusetzen.

Auch bei einer nicht dauerhaft mit voller Bandbreite verfügbaren Funkverbindung zu den lokalen Sendebaken können die Anwendungen auf dem WID ohne Beinträchtigung ihre Dienstleistung anbieten. Bei der Kommunikation mit dem Benutzer fallen kleinere Sendestörungen so nicht mehr ins Gewicht.

Die Implementierung von WID-internen Nachschlagediensten auf den mobilen Handcomputer ermöglicht die Verwendung des Jini-Programmierparadigmas (vgl. Abschnitt

2.2). Ein solch mobiles Endgerät stellt einen Kompromiss dar, zwischen einem völlig autonom arbeitenden Handcomputer, mit allen Diensten für den Benutzer, und einem Gerät mit ständiger Kommunikation zu einem zentralen Dienstgeber, mit den dort gespeicherten Informationen über den Benutzer. Ein Nachschlagedienst auf dem WID speichert die Objekte aus den über Funk lokal erreichbaren Nachschlagediensten des Heywow-Systems und bietet den Benutzer-Anwendungen auf dem WID diese Stellvertreter an.

Neben der Kopie des Stellvertreter-Objekts für den lokalen Gebrauch speichert der WID-interne Nachschlagedienst auch Zeitzusicherungen, mit denen die Verfügbarkeit des Dienstes garantiert wird. Bei Auslauf dieser Garantie wird, wenn die Zusicherung nicht erneuert werden kann, der Stellvertreter aus der lokalen Datenbank gelöscht.

Der mobile Handcomputer arbeitet batteriebetrieben, deshalb muss ein Dienst mit seinem Stellvertreter-Objekt auf dem Handcomputer den Energieverbrauch dort gering halten:

- Stromsparender Betrieb

Die Anforderung stromsparenden Betriebs gilt nicht nur für die Dienste, sondern auch für die Anwendung auf dem Handcomputer selbst. Durch den Einsatz von Jini-Diensten entsteht ein großer Bedarf für Kommunikation mit Nachschlagediensten und Dienstgebern.

Eine Möglichkeit, die Funkleistungsanforderungen auf dem mobilen Endgerät zu verringern, ist die Verwendung von Multicast oder Broadcast Techniken [HS1997]. Wie in Abschnitt 2.3 beschrieben, verbraucht besonders das Funk-Modul des Handcomputers viel Energie. Grundsätzlich ist aber das Empfangen von Daten für das Handgerät stromsparender als der Sendebetrieb.

Ein Simulationsexperiment in [SEa1996] belegt, dass das Empfangen von Datenpaketen nur unwesentlich mehr Energie verbraucht als der Verbrauch des Empfangsgeräts im Leerlauf. Wenn das mobile Gerät Daten in das Netz senden muss, kostet dies mehr Energie. Besonders signifikant wird dieser Unterschied wenn eine größere Datenmenge gesendet wird, was bei jedem Heywow-Dienst möglichst vermieden werden muss.

Zur Erstellung der WID-internen Datenbasis von erreichbaren Diensten kann deshalb auch das energiesparendere Belauschen einer Multicast-Gruppe eingesetzt werden. Die Topologie der Netzzellen in Heywow ermöglicht es, einfache Dienste mit stark lokalem Bezug, durch das in UDP implementierte Abhören einer Multicast-Gruppe an die Klienten zu verteilen.

Bei Multicast-Kommunikation kann keine Garantie für den Empfang aller Datenpakete gegeben werden. Der Verlust wesentlicher Dateninformationen muss anderweitig abgefangen werden. Anders als bei verbindungsorientierten TCP-Unicastverbindungen gibt es bei Multicast über UDP keine Rückmeldung an den Sender. Die zusätzliche

Einführung eines Rückkanals würde zwar eine Fehlerkorrektur bei verlorenen Datenpaketen erlauben, dies würde aber einen unvertretbaren Energie-Mehraufwand bedeuten. Günstiger bleibt auch hier der stark redundante Versand der Datenpakete. So kann eine praktisch zuverlässige Kommunikation implementiert werden [Riz1997].

Nach Erhalt aller Datenpakete eines Stellvertreter-Objekts kann auf dem Handcomputer das entsprechende Objekt den WID-internen Anwendungen zur Verfügung gestellt werden. Für die Integration der neue Bezugsstrategie in das Jini-System ist es hier möglich, eine Verbindung mit dem entsprechenden Dienstgeber aufzubauen und eine evtl. notwendige Zeitzusicherung für den Dienst anzufordern.

3.2.3 Sendebake

Für den problemlosen Einsatz eines Jini-Netzwerks beim Zugriff auf lokale Dienste erfordert der tragbare Handcomputer eine andauernde Integration in das lokale Festnetzwerk über Funk:

- Allgegenwärtiges Rechnen

Für die Einrichtung eines flächendeckenden Jini-Funknetzwerks ist auf den strategisch verteilten Sendebaken (Beacon) des Heywow-Systems das Angebot eines Nachschlagedienstes mit Unterstützung eines HTTP-Dienstgebers notwendig (vgl. Abbildung 3.1, S. 24).

Um den mobilen Endgeräten im Einzugsbereich den Bezug von Dienst-Stellvertretern über Multicast zu ermöglichen, müssen auf den einzelnen Sendebaken die lokalen Nachschlagedienste durchforstet werden und der Inhalt unter einer Multicast-Gruppe ins Netz gesendet werden. Dieses zusätzliche Multicast-Protokoll erlaubt nicht nur den Versand kompletter Stellvertreter-Objekte[1], es ist auch möglich auf der Sendebake nur die Schnittstelle oder eine Beschreibung der vorhandenen Objekte zu versenden. So lässt sich die zweite Bezugsstrategie in den Einsatz des Jini-Systems integrieren. Die Anwendung auf dem Handcomputer erhält die Möglichkeit, nur bei Bedarf das vollständige Stellvertreter-Objekt des Dienstes unter der mitgesendeten Adresse anzufordern.

3.3 Überlegungen zu Sicherheit und Stabilität

Beim Einsatz persönlicher, digitaler Assistenten und Handcomputer mit privaten Daten müssen an das System von Heywow besondere Anforderungen von Sicherheit und Vertraulichkeit gestellt werden.

[1] verpackt in evtl. mehrere UDP-Datenpakete

Heywow, als Mittler zwischen lokalen Dienstleistern und Benutzern der Infrastruktur von Heywow, muss seinen Kunden Garantien über die Sicherheit der im System angebotenen Dienste und die Stabilität der Software auf dem privaten Handcomputer garantieren:

- Authentifizierung der Teilnehmer
- Autorisierung der Daten
- Garantie für Systemstabilität auf dem tragbaren Handcomputer

Die Stabilität und Sicherheit des Systems von Heywow beruht auf der sicheren Kommunikation zwischen den Hardwarekomponenten. Probleme, die durch ein unsicheres Netzwerkprotokoll schon von Hardwareseite verursacht werden, sind nicht Teil dieser Überlegungen. Im Folgenden wird ein funktionierendes TCP/IP-Netzwerk vorausgesetzt.

Heywow setzt ein Vertrauensverhältniss zwischen dem Dienstevermittler Heywow auf der einen Seite und den Benutzern und den Diensteanbietern auf der anderen Seite voraus. Jeder Klient muss in die Sicherheit und Funktionstüchtigkeit der registrierten Stellvertreter bei Heywow vertrauen und zwischen Heywow und den lokalen oder globalen Diensteanbietern muss ein Vertrauensverhältnis über die angebotenen Dienste bestehen.

Angriffe in das System von Heywow beruhen deshalb meist auf einer Störung oder einem Unterlaufen der bestehenden Vertrauensverhältnisse. Die Abbildung 3.2 zeigt

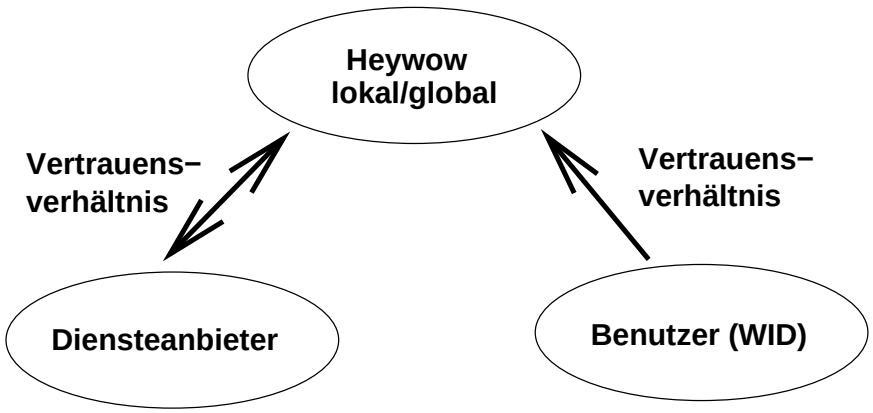

Abbildung 3.2: Vertrauensverhältnisse

die direkten Beziehungen der Teilnehmer.

Einige Angriffsmöglichkeiten:

1. Funktionalität des Benutzer-WIDs wird beeinträchtigt, z. B. durch zu große oder nicht authentifizierte Stellvertreter-Objekte
2. Nicht autorisierte Dienste werden über Heywow angeboten
3. Nicht-Heywow-Nachschlagedienste bieten sich zur Registrierung von Diensten an

Zur Unterstützung der einzelnen Vertrauensverhältnisse und bei der Weitergabe vertrauenswürdig befundener Daten über die Nachschlagedienste von Heywow ist leicht ein Authentifizierungs und Verschlüsselungsverfahren unter Einsatz von public-key Algorithmen möglich.

4 Realisierung

Aufbauend auf die in Kapitel 3 vorgestellten Anforderungen und die daraus abgeleiteten Konzept-Ideen für die Software-Architektur des Heywow-Systems wird in diesem Kapitel eine Klassenbibliothek für die Entwicklung und Benutzung von Heywow-Diensten vorgestellt.

Durch den Einsatz von 100% purem Java wird der Einsatz im heterogenen System aus verschiedensten Handcomputern als Klienten und leistungsstarken Dienstgebern auf Diensteanbieterseite ermöglicht. Mit dem Aufbau auf die Klassen und Schnittstellen des Jini-Pakets[1] können die mit der Bibliothek entworfenen Dienste und Anwendungen in einem beliebigen IP-Netzwerk mit Funk- und Festvernetzung eingesetzt werden.

Da die verfügbaren Versionen der Micro Edition (vgl. Abschnitt 2.1) den Einsatz entfernter Methodenaufrufe (vgl. Abschnitt 2.1.2 RMI) noch nicht unterstützen, ist die Java 2 Standard Edition als Grundlage verwendet worden. Außer dem Einsatz von RMI-Klassen und Schnittstellen ist aber der eingeschränkte Umfang der Micro Edition beachtet worden. In Abschnitt 4.10 werden die wesentlichen Parameter für die Übertragung auf die mobile Plattform ausführlicher diskutiert.

Die Klassenbibliothek bietet Klassen und Schnittstellen, die die Implementierung von Heywow-Diensten vereinfachen. Durch den Einsatz des Entwurfsmusters Schablonenmethode wird ein Skelett für den Aufbau von Diensten im Heywow-System vorgegeben. Mit Erweiterung und Überschreiben der abstrakten Bibliotheksklassen ist es möglich, beliebige, komplexe Dienstleistungen lokal verfügbar zu machen.

Der Abschnitt 3.2.1 stellt zwei verschiedene Entwicklungsstufen für einen Heywow-Dienst vor. Sowohl die einfache Variante, mit Verwendung der Verwalterklassen aus den `net.jini`-Paketen, als auch die Möglichkeit des Einsatzes spezieller Jini-Dienste wird näher beschrieben. Mit der Unterstützung des Aktivierungssystems auf den Dienstgebern der Diensteanbieter lassen sich auch Dienste entwickeln, die dort Systemressourcen einsparen können.

Für die Reduktion der Leistungsanforderungen auf dem mobilen Handcomputer unterstützt die Bibliothek neben den Standardprotokollen des Jini-Systems auch ein weiteres Protokoll für den Versand (vgl. Abschnitt 4.5) und Empfang von Dienst-Stellvertreter-Objekten über Multicast.

[1] Verwendet wurde die Implementierung von Sun Microsystems in der Version 1.1 beta2.

Für den Einsatz der abstrakten Dienst-Basisklassen werden in Abschnitt 4.3 drei Musterdienste implementiert, die die Funktionalität der gesamten Bibliothek an einfachen Beispielen vorstellen. Im Folgenden werden in Abschnitt 4.7 die Möglichkeiten für die Kommunikation mit dem Heywow-System und den Diensten vom Handcomputer aus in einer einfachen Anwendung implementiert.

Bei der Realisierung der Klassenbibliothek für Heywow wurde Wert gelegt auf eine klare Trennung der Klassen nach ihren Funktionen. Die Aufteilung in Klassenpakete richtet sich nach Verfügbarkeit auf den verschiedenen Hardwarekomponenten.

4.1 Dienste

Das Paket `heywow.service` bietet Basisklassen für die einfache Konstruktion von Diensten im Heywow-System. Neben den Dienstgebern und Sendebaken von Heywow sind diese Klassen auf den Dienstgebern der Diensteanbieter und den Endgeräten der Benutzer verfügbar.

Ein Dienst besteht aus 3 Komponenten:

1. Dienstgeberkomponente (`ServerObject`)
2. Stellvertreter-Objekt (`ServiceInterface`)
3. Starter

Abbildung 4.1: `ServiceInterface` und `ServerObject`

Jedes der Stellvertreter-Objekte muss die Schnittstelle `heywow.service.ServiceInterface` unterstützen. Mit dieser Schnittstelle ist es möglich, beim Benutzen eines Jini-Nachschlagedienstes Heywow-Dienste auszuwählen (vgl. Abschnitt 2.2.3). Außerdem wird über die `ServiceInterface`-Schnittstelle sichergestellt, dass jedes Stellvertreter-Objekt eine Referenz auf die Instanz seines `heywow.service.ServerObject`s zurückgeben kann. Diese Dienstgeberkomponente muss das unter Jini notwendige Zeitzusicherungsverfahren (vgl. Abschnitt 2.2.4) implementieren und mit `getLease()` eine gültige Zeitzusicherung für den Dienst zurückgeben können. Die Abbildung 4.1 zeigt in einem UML-Klassenstrukturdiagramm diese beiden grundlegenden Schnittstellen.

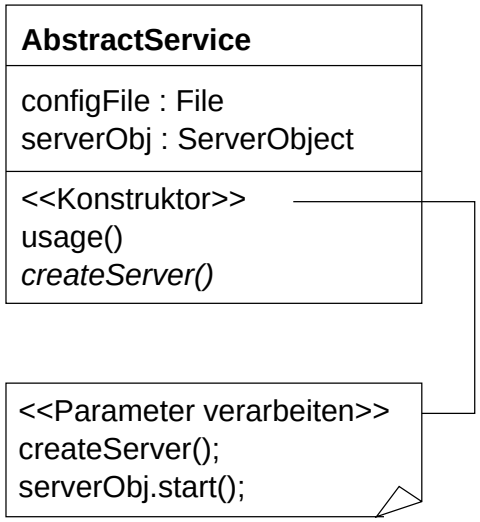

Abbildung 4.2: `AbstractService`

Für den einheitlichen Start eines Heywow-Dienstes wird neben diesen zwei Schnittstellen auch die abstrakte Starterklasse `heywow.service.AbstractService` (vgl. Abbildung 4.2) angeboten. In ihr wird die Integration in das Betriebssystem auf Dienstgeberseite kontrolliert und der Dienst gestartet.

4.1.1 Abstrakte Dienstgeberkomponente

Zwei abstrakte Klassen, die die beiden Schnittstellen (vgl. Abbildung 4.1) implementieren und allgemeine Funktionalität ergänzen, sind:

`heywow.service.AbstractProxy` und `heywow.service.AbstractServer` (vgl. Abbildung `4.3`). Der `AbstractServer` stellt Methoden zur Sicherung und Wiederherstellung des Dienst-Zustands nach dem ersten Aufruf zur Verfügung. In Jini muss jeder Dienst eindeutig durch einen 128bit Schlüssel identifiziert werden. Diese `ServiceID` muss auch über Neustarts des Dienstes hinweg, in allen Nachschlagediensten bei der Registrierung gleich bleiben. Die `ServiceID`, Attribute des Dienstes und erlaubte Gruppen von Nachschlagediensten werden so für jeden Dienst verwaltet.

Beim ersten Start des Dienstes wird die `ServiceID` nach den Jini-Konventionen (vgl. Jini 1.1 API Dokumentation der Klasse `net.jini.core.lookup.ServiceID`) mit einem Objekt der Klasse `heywow.util.ServiceIDFactory` zufällig neu erzeugt. Hierauf hat der Diensteanbieter keinen Einfluss. Anders die Attribute, sie müssen von einem konkreten Dienstgeber durch die Implementierung von `createAttributes()` festgelegt werden. Die Gruppennamen interessanter Jini-Nachschlagedienste werden ebenfalls in der Implementierung festgelegt.

Bei späteren Neustarts stellt `AbstractServer` diesen Zustand wieder vollständig her.

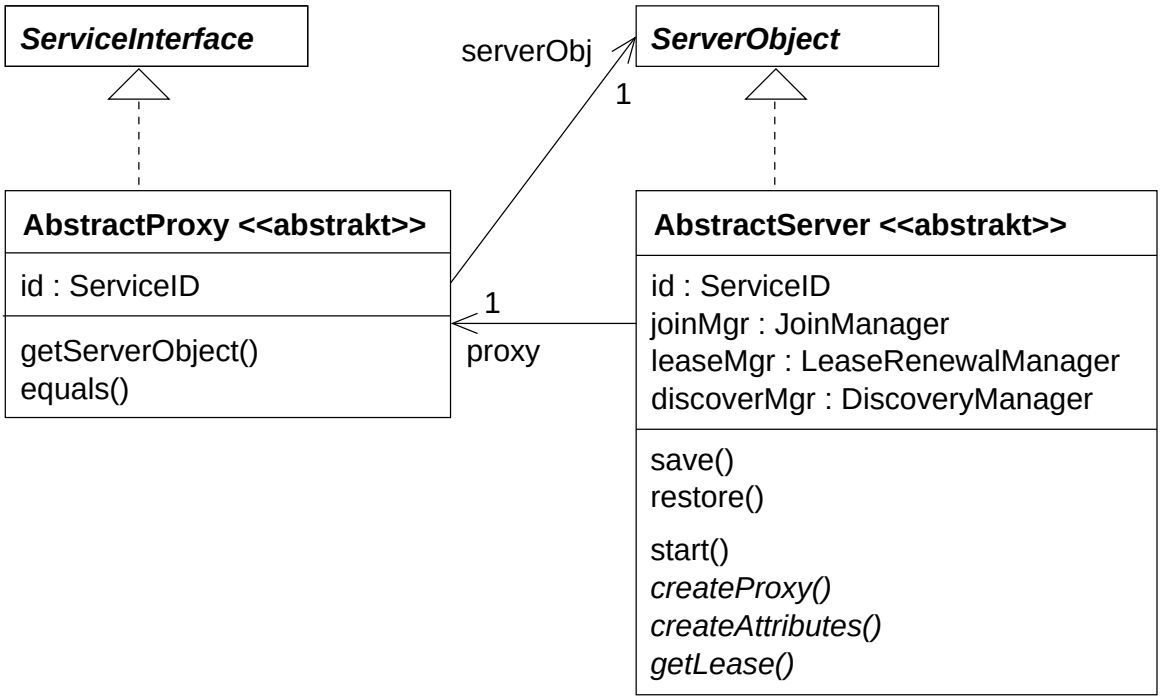

Abbildung 4.3: `AbstractProxy` und `AbstractServer`

Die abstrakte Dienstgeberkomponente implementiert die Methode `start()` aus der `ServerObject`-Schnittstelle und führt abhängig von Art und Zeitpunkt des Aufrufs die (Wieder-)Registrierung des konkreten Stellvertreter-Objekts in allen interessanten Nachschlagediensten durch. Hierzu muss die konkrete Dienstgeberkomponente die beiden Methoden `createAttributes()` und `createProxy()`, mit der Instanziierung des Stellvertreters, implementieren.

Für die dauerhafte Registrierung in den Nachschlagediensten des Heywow-Systems werden die unterstützenden Klassen des Jini Starter Kits in `net.jini.discovery`, `net.jini.lease` und `net.jini.lookup` verwendet. Mit den verschiedenen Verwalterklassen werden die Standardaufgaben im Jini-System geregelt.

4.1.2 Abstraktes Stellvertreter-Objekt und Zeitzusicherungen

Die Elternklasse aller Stellvertreter-Objekte im Heywow-System, `heywow.service.AbstractProxy`, unterstützt die Schnittstelle `ServiceInterface`. Für den Vergleich von Stellvertreter-Objekten aus verschiedenen Nachschlagediensten bietet `AbstractProxy` die Methode `equals()`, die die Gleichheit anhand der gespeicherten `ServiceID` feststellt. Das Stellvertreter-Objekt eines Dienstes wird in allen interessanten, lokalen Nachschlagediensten registriert. Beim Lookup-Prozess (vgl. Abschnitt 2.2.3) auf einem Klienten werden somit über Jini auch mehrere Objekte aus verschiedenen Nach-

schlagediensten bezogen. Dubletten müssen deshalb beim Aufbau des WID-internen Zwischenspeichers eliminiert werden. Außerdem wird in `AbstractProxy` eine Referenz auf die zugehörige `ServerObject`-Instanz gespeichert.

Ein Klient kann über das bezogene Stellvertreter-Objekt von der zugehörigen Dienstgeberkomponente eine Zeitzusicherung anfordern. Dieser `AbstractServer` tritt wie-

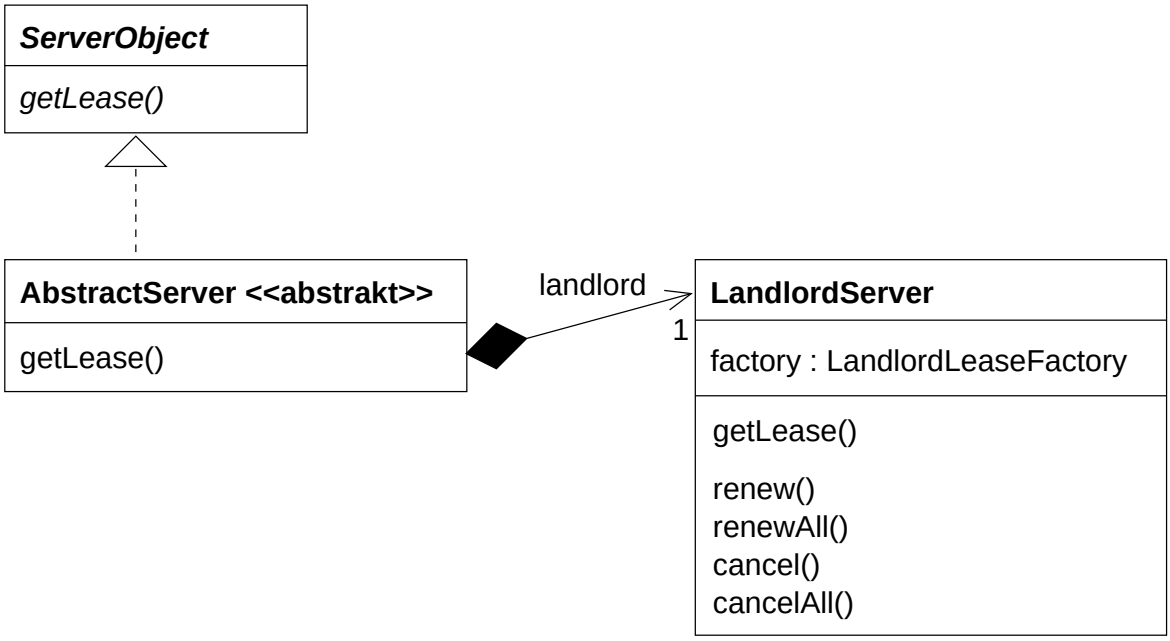

Abbildung 4.4: `AbstractServer` und `LandlordServer`

derum als Vermittler auf und leitet die Anfrage an eine Instanz von `heywow.service.support.LandlordServer` weiter (vgl. Abbildung 4.4).

Die Klasse `LandlordServer` implementiert das Protokoll eines Landlords (vgl. Abschnitt 2.2.4) und verwaltet für den Dienst vergebene Zusicherungen. Die Dienstgeberkomponente des Dienstes kann auch ohne einen `LandlordServer` erzeugt werden, wenn auf die Vergabe von Zeitzusicherungen verzichtet wird.

4.1.3 Abstrakter Starter

Für den Start eines Dienstes bietet das Rahmensystem eine abstrakte Starterklasse (vgl. UML-Klassenstrukturdiagramm in Abbildung 4.2, S. 34) an. Durch Vererbung von der Basisklasse `heywow.service.AbstractService` kann ein Dienst eine Starterklasse mit einer `main()`-Methode implementieren. Die abstrakte Elternklasse enthält in ihrem Konstruktor das Skelett für den Start eines Dienstes. Eine konkrete Klasse muss die Methode `createServer()` implementieren und dort die Dienstgeberkomponente des Dienstes instanziieren. `AbstractService` startet diese nach der Erzeugung.

4.1.4 Abstrakte, aktivierbare Dienstgeberkomponente

Im Gegensatz zur einfachen Instanziierung

```
serverObj = (ServerObject) new ConcreteServer();
```

kann bei der aktivierbaren Variante eines Dienstes die Dienstgeberkomponente im Java-Aktivierungssystem registriert und dort eine aktiviertes Objekt erzeugt werden.

Mit dieser Form des Dienststarts können auf den Dienstgebern der Diensteanbieter Systemprozesse, die nur im Leerlauf sind, eingespart werden und auch dort die nötigen Systemressourcen reduziert werden.

Die Starter-Variante des aktivierbaren Dienstes bietet eine zusätzliche Methode `activate()`, die den vollständigen Name der Dienstgeber-Klasse und die nötigen Parameter, verpackt als `MarshalledObject`, als Parameter entgegennimmt.

`AbstractActivatableService` erbt selbst von `AbstractService` und ergänzt den Konstruktor der Elternklasse nur um das Beenden des Starterprogramms nach der erfolgreichen Registrierung der Dienstgeberkomponente im Aktivierungssystem.

Das Sequenzdiagramm in Abbildung 4.5 veranschaulicht das dynamische Zusammenspiel der einzelnen Klassen und Komponenten im Heywow-System:

Nach dem Aufruf des Starters eines konkreten, aktivierbaren Dienstes muss durch die Methode `createServer()` die Dienstgeberkomponente instanziiert werden. Beim aktivierbaren Dienst wird dies mit der Registrierung im Aktivierungssystem verbunden. Hierzu bietet `AbstractActivatableService`[2] die Methode `activate()` an.
Nach der Registrierung wird ausgehend von der zurückgegebenen eindeutigen `ActivationID` die Dienstgeberkomponente aktiviert und eine Referenz zurückgegeben. Mit diesem Objekt wird der Dienst, wie ein nichtaktivierbarer, gestartet.

Da die Dienstgeberkomponente nicht dauerhaft rechnen muss, und sich bei Leerlauf selber beendet, kann sie nicht für die dauerhafte Registrierung im Jini-System verwendet werden. `AbstractActivatableServer` besitzt keine Verwalterklassen aus dem Jini Starter Kit.

Für aktivierbare Dienste bietet Sun Microsystems Jini-Dienste, die in Nachschlagediensten registriert werden, an. Teil des Jini Starter Kits ist ein Nachschlagedienst-Entdeckerdienst (*LookupDiscoveryService*) namens `fiddler`, der eingesetzt werden kann, um interessante Nachschlagedienste zu finden. Wenn dieser Unterstützerdienst einen neuen Nachschlagedienst entdeckt, aktiviert er die Dienstgeberkomponente des Dienstes und diese registriert das Dienst-Stellvertreter-Objekt dort.

Auch die Verwaltung der dort erhaltenen Zeitzusicherung wird über einen Jini-Dienst (`norm`) geregelt. Mit diesem Nutzungszeitverlängerungsdienst (*LeaseRenewalService*)

[2] Eine spezialisierte abstrakte Klasse, die von `heywow.service.AbstractService` erbt.

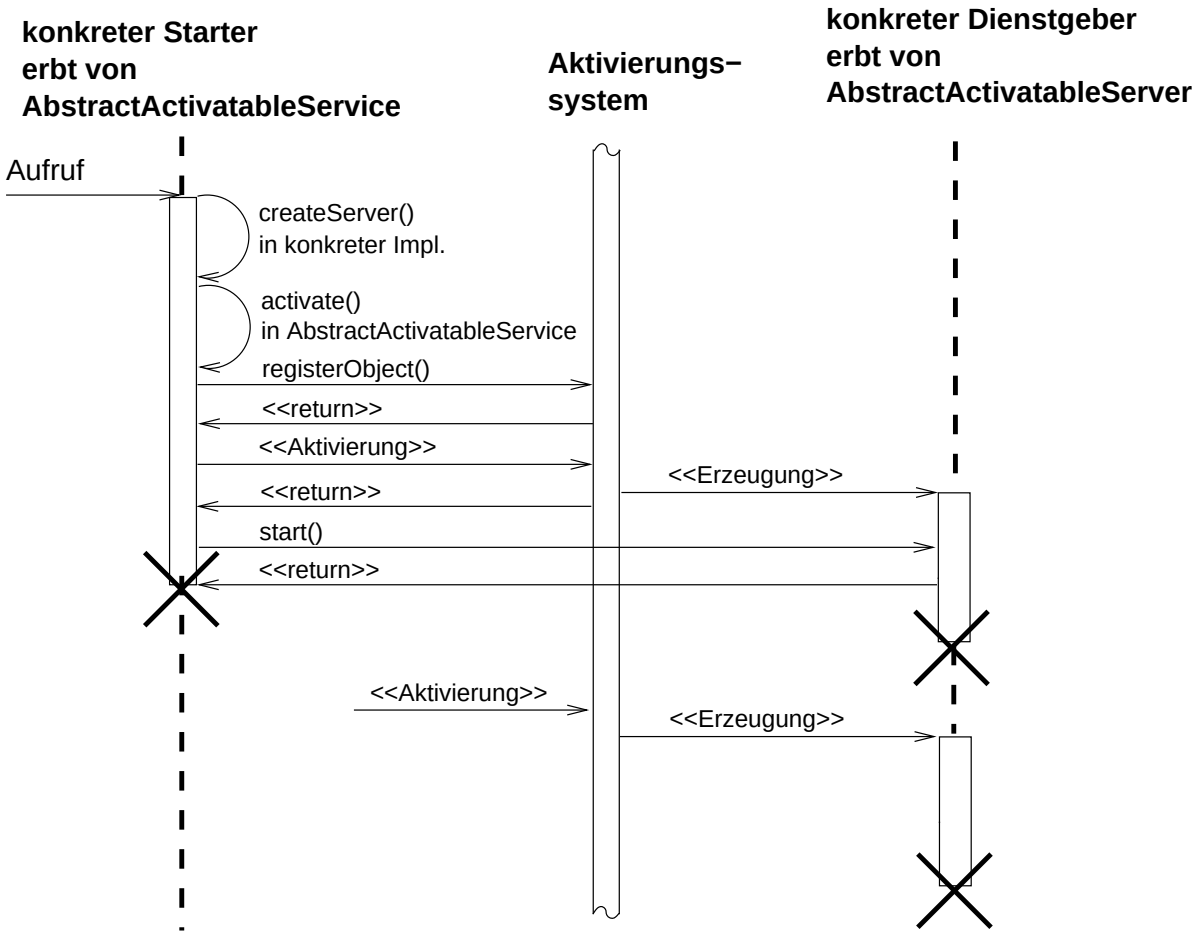

Abbildung 4.5: Zusammenspiel der konkreten Klassen eines aktivierbaren Dienstes mit dem Aktivierungssystem von Java (vgl. Abschnitt 2.1.3)

aus dem Jini Starter Kit kann die Dienstgeberkomponente die regelmäßige Erneuerung von erhaltenen Zeitzusicherung an diesen Dienst delegieren.

Der `AbstractActivatableService`, bzw. der konkrete Starter kann nach erfolgreicher Registrierung der Dienstgeberkomponente im Aktiverungssystem beendet werden. Für diese Einsparung von Systemressourcen sind aber laufende Prozesse für den RMI-Activation-Daemon (`rmid`), den Nachschlagedienst und die Entdecker- und Verlängerungsdienste notwendig.

Wird der `heywow.service.AbstractActivatableServer` über das Aktivierungssystem instanziiert, wird sofort ein Stellvertreter-Objekt mit der Information von `ServiceID` und `ActivationID` erzeugt. Jede konkrete Dienstgeberkomponente muss hierfür wieder eine Methode `createProxy()` implementieren, die dieses Muster erzeugt. Außerdem wird im Konstruktor auch die Methode `createAttributes()` aufgerufen.

Neu ist eine Methode `notify()`, aus der `RemoteEventListener`-Schnittstelle, die mit einem `net.jini.core.event.RemoteEvent` durch die unterstützenden Dienste aufgerufen wird. Das Aktivierungssystem ist dafür zuständig, dass bei Bedarf die aktivierbare Dienstgeberkomponente in einer neuen JVM instanziiert wird. Nach einer Benachrichtigung kann über das `RemoteEvent` entschieden werden, welche Aktivität vom Dienstgeber-Objekt erforderlich ist.

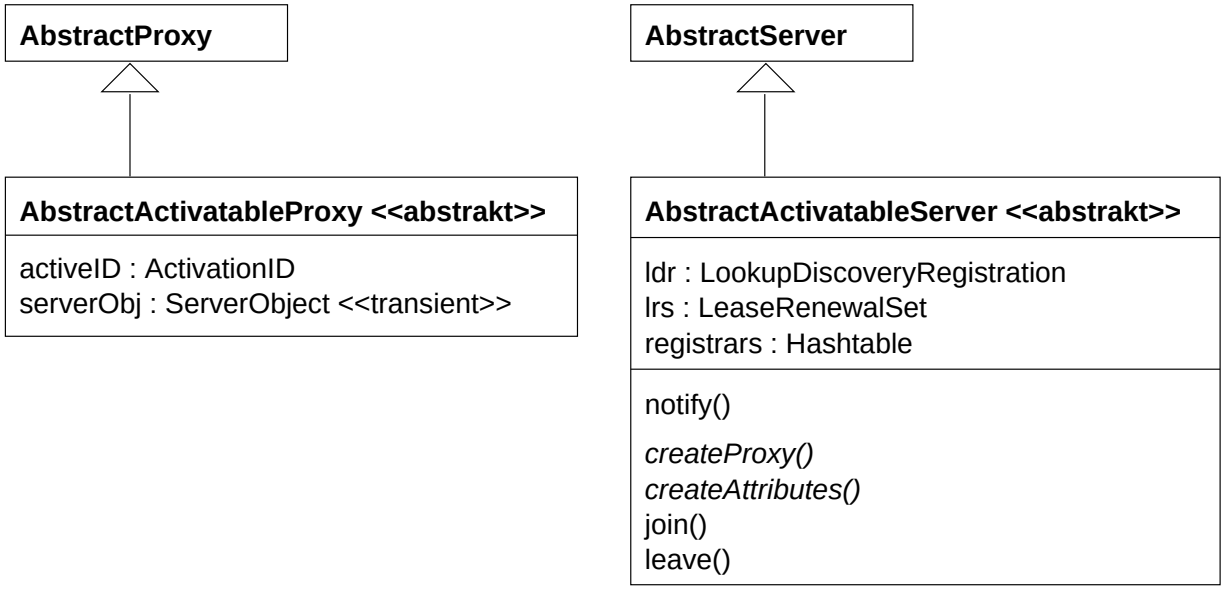

Abbildung 4.6: `AbstractActivatableProxy`, `AbstractActivatableServer`

Die Abbildung 4.6 zeigt das UML-Klassendiagramm von `AbstractActivatableProxy` und `AbstractActivatableServer`, mit den wesentlichen Methoden und Variablen.

Auch für die Verwaltung der Zeitzusicherungen, die der Dienst an seine Klienten vergibt, ist die Dienstgeberkomponente zuständig. Falls über die `getServerObject()`-Methode im Stellvertreter eine Instanz der Dienstgeberkomponente referenziert wird, aktiviert die Klasse `AbstractActivatableProxy` evtl. die nicht aktive Komponente über die gespeicherte `ActivationID`.

Und wenn nach dieser Aktivierung eine Zeitzusicherung für den Dienst angefordert wird, muss der Prozess auf dem Dienstgeber aktiv bleiben um mit seinem `Landlord-Server`-Objekt die vergebenen Zeitzusicherungen zu verwalten.

Der `AbstractActivatableServer` implementiert eine neue Methode `start()`, die an das neue Verfahren mit den Unterstützer-Jini-Diensten angepasst ist. Mit dem Einsatz der Jini-Dienste werden die abstrakten Methoden `(re-)register()` aus `AbstractServer` nicht weiter unterstützt.

4.1.5 Versionsattribut

Eine konkrete Klasse im Paket `heywow.service` ist die Implementierung für ein Jini-Dienste-Attribut. Mit `heywow.service.Version` kann die Programmversion eines angebotenen Dienstes näher beschrieben werden. Dieses Dienste-Attribut ist für den Vergleich, z. B. mit dem eines zwischengespeicherten Stellvertreters, nötig.

4.2 Unterstützerklassen

Die Klasse `LandlordServer` (vgl. Klassendiagramm in Abbildung 4.4, S. 36) aus dem Paket `heywow.service.support` implementiert die Schnittstelle für einen Landlord (vgl. Abschnitt 2.2.4). Über entsprechende Objekte können Zeitzusicherungen nach diesem Verfahren verwaltet werden.

Die Klasse, die von einer Dienstgeberkomponente instanziiert werden kann, speichert erzeugte Zusicherungen in einer Datenbasis. Regelmäßig wird diese nach veralteten Einträgen durchsucht und gereinigt. Bei der Konstruktion eines `LandlordServer`-Objekts kann die gewünschte Zusicherungsdauer übergeben werden. Zeitzusicherungen, die mit der Methode `getLease()`, aus `ServerObject`, angeforderten werden, trägt der Verwalter dann mit einer Referenz auf den Klienten, als Schlüssel, in seine Datenbasis ein.

Dies ist bei der Verwendung eines speziellen Verwalters für jede Dienstgeberkomponente ein ausreichender, eindeutiger Schlüssel. Erst wenn ein `LandlordServer`-Objekt für mehrere Dienste die Zeitzusicherungen verwaltet, muss der Schlüssel anders berechnet werden.

Für die Rückgabe gültiger Zeitzusicherungen wird die Implementierung einer `LandlordLeaseFactory` verwendet [Edw1999, Kap. 11]. Der `LandlordServer` wird für die erzeugten Zeitzusicherungen als Landlord registriert und implementiert die nötigen Methoden für die Verwaltung.

Eine weitere von den Dienst-Basisklassen benötigte Klasse in diesem Heywow-Paket ist `FindAService`. Mit Objekten dieser Klasse kann in den Nachschlagediensten des Heywow-Systems ein Stellvertreter-Objekt mit einer vorgegebenen Schnittstelle gesucht werden. Aufbauend auf die Kern-Klassen von Jini wird ein einfacher `DiscoveryListener` implementiert. Mit der Methode `getService()` kann synchronisiert das erste gefundene Objekt dieser Schnittstelle, z. B. für die Suche nach Nachschlagedienst-Entdeckerdiensten und Nutzungszeitverlängerungsdiensten bei aktivierbaren Heywow-Diensten, bezogen werden.

4.3 Beispieldienste

Im `heywow.example`-Paket sind drei einfache Heywow-Dienste implementiert, die die Verwendung der abstrakten Basisklassen und Schnittstellen aus Abschnitt 4.1 im Paket `heywow.service` erklären.

4.3.1 'Hej Marc'-Dienst

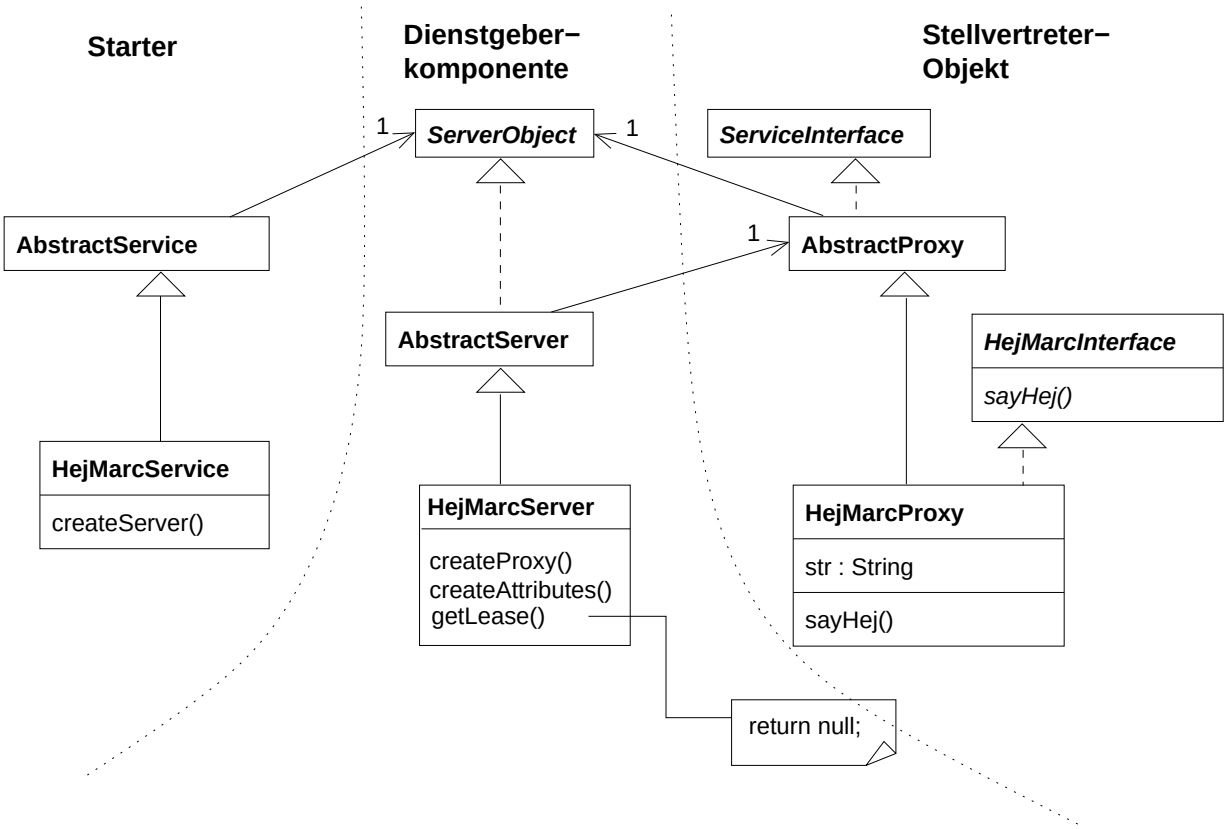

Abbildung 4.7: Klassen des 'Hej Marc'-Dienstes

Der 'Hej Marc'-Dienst (vgl. Abbildung 4.7) implementiert mit seinem Stellvertreter-Objekt die Schnittstelle `HejMarcServiceInterface`, die eine Methode `sayHej()` mit einer Begrüßungszeichenkette als Rückgabewert definiert.

Der Starter `HejMarcService` erbt von `AbstractService` und ergänzt die abstrakte Klasse um eine einfache `main()`-Methode. Hier wird der korrekte Aufruf des Starters kontrolliert und der Starter instanziiert.

Der Dienstgeber besitzt keinen `LandlordServer` zur Erzeugung und Verwaltung von Zeitzusicherungen. Die Methode `getLease()` wird überschrieben, gibt nur die `null`-Referenz zurück und zeigt damit an, dass das Stellvertreter-Objekt immer gültig bleibt.

Ein solches Stellvertreter-Objekt wird, nachdem es in den WID-internen Zwischenspeicher kopiert worden ist, dort bis auf weiteres für Anwendungen auf dem Handcomputer bereitgehalten. Eine Funk-Verbindung vom Stellvertreter-Objekt zur Dienstgeberkomponente ist nicht nötig.

Da die Zeitzusicherungen für den dauerhaften Eintrag in einen Nachschlagedienst aber regelmäßig erneuert werden müssen, kann das Dienstgeber-Programm nicht beendet werden. Der Dienstgeber erzeugt einen zusätzlichen Kontrollfaden (vgl. Abschn. 2.1.1), der ohne Unterbrechung weiterläuft und kann so mit den Unterstützerklassen aus dem Jini Starter Kit seine Registrierungen erneuern und sich in neuen Nachschlagediensten eintragen.

4.3.2 'Totally Different'-Dienst

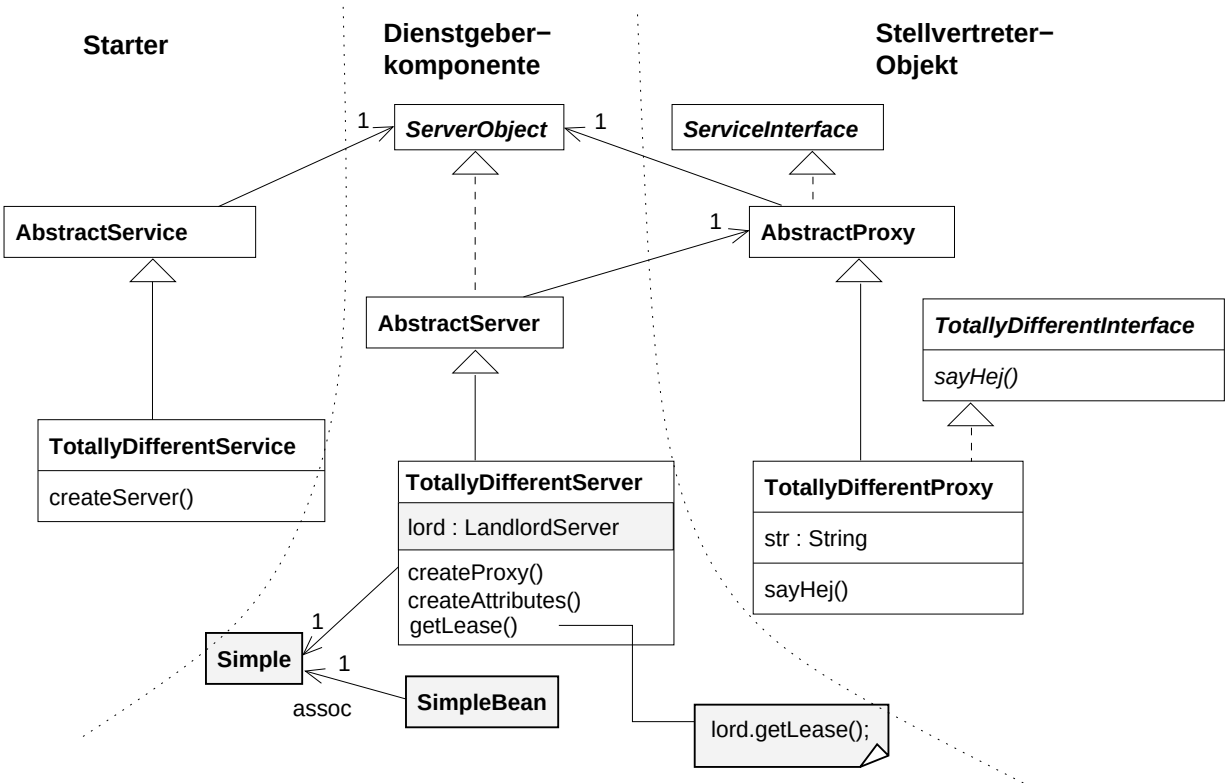

Abbildung 4.8: Klassen des 'Totally Different'-Dienstes

Der zweite Muster-Dienst, mit dem Starter `heywow.example.TotallyDifferentService` (vgl. Abbildung 4.8), bietet den gleichen Funktionsumfang, implementiert aber eine unabhängige Schnittstelle.

`heywow.example.TotallyDifferentInterface` definiert zwar ebenfalls eine Methode `sayHej()`, steht aber in keinerlei Beziehung zum ersten Dienst.

Zum Starter des 'Hej Marc'-Dienstes besteht ebenfalls kein großer Unterschied. Erst die Dienstgeberkomponente des Dienstes zeigt neue Möglichkeiten der Klassenbibliothek. In `TotallyDifferentServer` wird durch einen anderen Konstruktor der Elternklasse (`AbstractServer`) ein `LandlordServer`-Objekt zur Verwaltung von Zeitzusicherungen instanziiert. Die Methode `getLease()` wird jetzt an die Methode des Verwalters weitergereicht.

Außerdem wird bei der Registrierung des Stellvertreter-Objekts zusätzlich auch eine Menge von Attributen mit im Nachschlagedienst gespeichert (vgl. Abschnitt 2.2.5). Die Beispielklasse `heywow.example.Simple` erweitert die Klasse `AbstractEntry` aus dem Jini-Paket und kapselt einen Integer-Wert. Mit die Klasse `SimpleBean` kann dieser auch vom Dienst oder einem Klienten geändert werden.

4.3.3 Aktivierbarer Dienst

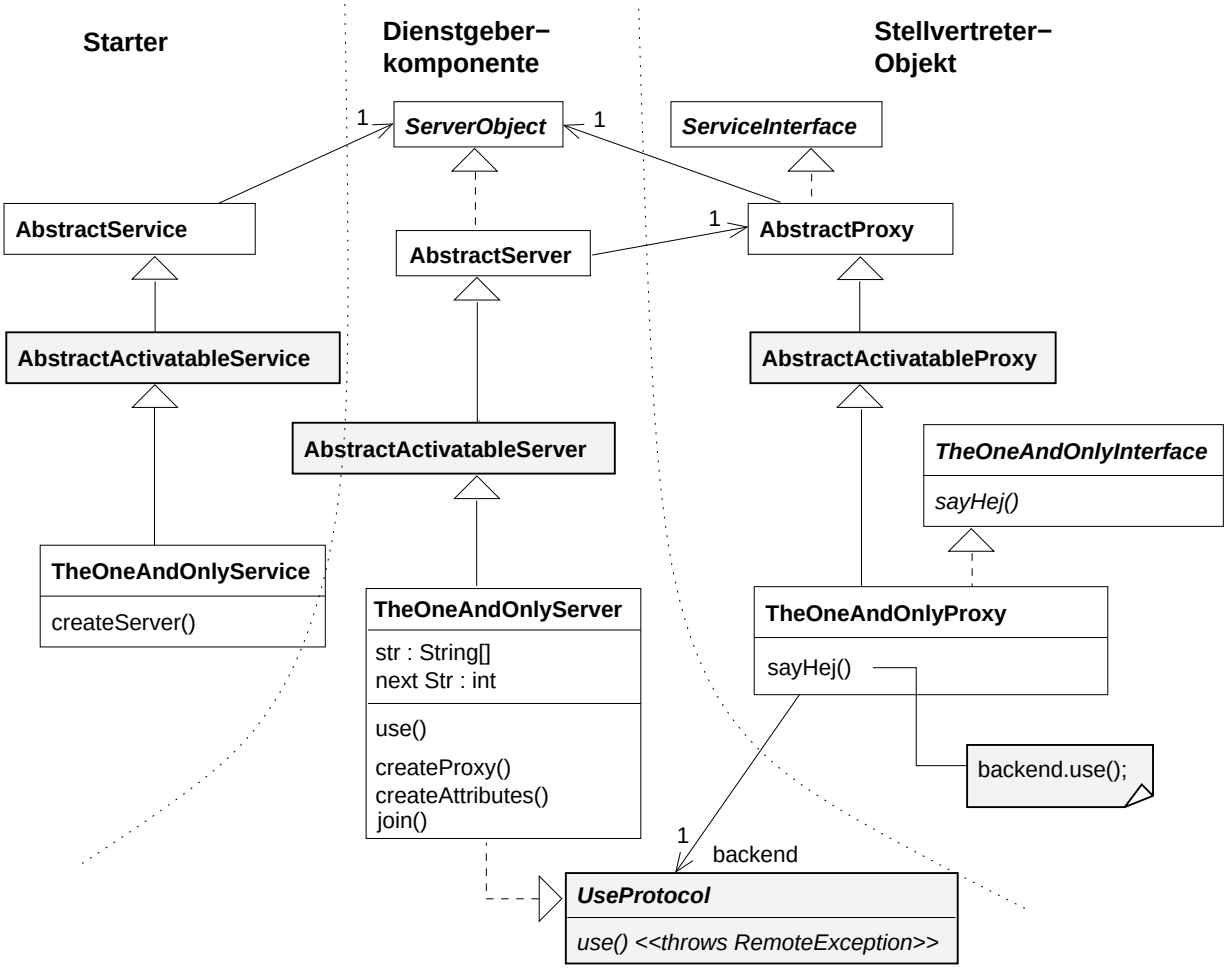

Abbildung 4.9: Klassen eines aktivierbaren Dienstes (`TheOneAndOnly*`)

Die Verwendung der Basisklassen für einen aktivierbaren Dienstgeber wird mit der Starterklasse `heywow.example.TheOneAndOnlyService` (vgl. Abbildung 4.9) verdeutlicht.
`TheOneAndOnlyServer` ist ein `ServerObject` und erbringt neben der Funktionalität für die Verwaltung der Zeitzusicherungen an Klienten auch den eigentlichen Dienst.

Das Stellvertreter-Objekt (`heywow.example.TheOneAndOnlyProxy`) leistet jetzt nur die Kommunikation zum Dienstgeber. Die Funktionalität gleicht wieder der bisherigen Aufgabe, es wird eine einfachen Begrüßungszeichenkette zurückzugeben. Der `TheOneAndOnlyProxy` verwendet dazu aber das `TheOneAndOnlyServer`-Objekt auf einem Dienstgeber. Der Zugriff erfolgt über einen entfernten Methodenaufruf mit Verwendung der Schnittstelle `heywow.example.UseProtocol`.

Die Methode `use()` reagiert auf den Aufruf mit einer Zeichenkette, die im Umlauf aus einer kleinen Datenbasis ausgewählt wird.

Neben der Verwendung der Dienstgeberkomponente für die Erbringung des Dienstes, zeigt die Dienstgeberkomponente `TheOneAndOnly` auch die Java-Möglichkeiten des Aktivierungssystems.

Der Starter, `heywow.example.TheOneAndOnlyService`, erbt wie die anderen Komponenten des Dienstes, von den `*Activatable*`-Klassen. Nach der Aktivierung durch den Starter kann sich die Dienstgeberkomponente, wenn sie sich im Leerlauf befindet, beenden.

Das einfache Beispiel unterstützt keinerlei Sicherheitsprotokoll. Beliebig viele Klienten können sich gegenseitig beeinflussen und die Rückgabe für die anderen verändern.

4.4 WID-interner Zwischenspeicher

Auf Seiten des Handcomputers unterstützt die Klassenbibliothek den Entwurf von Anwendungen für den Benutzer.

Der Benutzer will auf die lokal verfügbaren Dienstleistungen, die seinen Interessen entsprechen, zugreifen. Dafür müssen aus den Nachschlagediensten des Heywow-Systems die entsprechenden Stellvertreter-Objekte bezogen werden. Für die Verwendung der Dienste auf einem mobilen Endgerät wird ein WID-interner, erkundender Nachschlagedienst als Zwischenspeicher eingesetzt (vgl. Abschnitt 3.2.2). Durch Einsatz eines `DiscoveringLUS`-Objekts kann die Anwendung auf dem Handcomputer alle im lokalen Netzwerk erreichbaren Nachschlagedienste erforschen und dort interessante und vom Benutzer gefragte Dienst-Stellvertreter-Objekte nachschlagen lassen.

Mit dieser Bezugsform baut der `DiscoveringLUS` im Paket `heywow.util` auf die Kommunikationsprotokolle von Jini auf und erzeugt einen WID-internen Zwischen-

speicher. Dieser funktioniert eingeschränkt unabhängig von der Funkverbindung[3] des Handcomputers zu den Sendebaken des Heywow-Systems.

Durch die Methode `getService(ServiceTemplate)` wird der Zwischenspeicher für das übergebene Stellvertreter-Muster erzeugt und die Suche gestartet. Ein `ServiceTemplate` ermöglicht die Einschränkung der Suche nach interessanten Stellvertreter-Objekten auf die Implementierung bestimmter Schnittstellen, eine Menge von Attributen oder eine bestimmte `ServiceID`. Bei erfolgreicher Suche im lokalen Netz benachrichtigt der WID-interne, erkundende Nachschlagedienst die Anwendung, und die übergebene Referenz auf die Datenbasis kann verwendet werden, um die entsprechenden Stellvertreter-Objekte einzusetzen.

Beim Konzept des entfernten Methodenaufrufs (vgl. Abschnitt 2.1.2 RMI) wird die Objekt-Serialisierung für die Übertragung beliebiger Java-Objekte verwendet. Die serialisierbaren Objekte, insbesondere die Stellvertreter-Objekte, unterstützen die `java.io.Serializable`-Schnittstelle, eine Schnittstelle ohne Methodenbeschreibung oder Konstanten. Sie ist im Java-System nur als Marke notwendig. Die Versendung der in Nachschlagediensten registrierten Stellvertreter-Objekte erfordert dies und benötigt zusätzlich den Bezug der Bytecode-Datei für die Objekt-Schnittstelle. Für diesen Zweck müssen die übersetzten Programmdateien von einem Web-Dienstgeber mit HTTP bezogen werden können.

Da Java aber auch die Möglichkeit bietet beliebige Java-Objekte ohne Kenntnis ihrer Schnittstelle zu benutzen (vgl. Abschnitt 2.1.5 Reflection), können die Anwendungen auf dem Endgerät die Objekte auch ohne Bezug der Bytecode-Dateien inspizieren und benutzen.

Für den stromsparenden Betrieb des Handcomputers kann der energieverbrauchende, sendende Zugriff über Jini noch weiter eingeschränkt werden. Mit einem speziellen Protokoll über Multicast können alle Proxy-Objekte, die an eine Multicast-Gruppe gesendet werden, durch energiesparenderes Belauschen der Adresse mit `heywow.wid.ProxySniffer` bezogen werden.

Der `ProxySniffer` erweitert den allgemeinen `heywow.wid.MulticastSniffer`, der ein Empfangsprotokoll für beliebige Objekte unterstützt.

Mit `MulticastSniffer` wird eine IP-Adresse belauscht und für jedes erhaltene Datenpaket eine Datenstruktur aufgebaut. Die zusammengehörenden Datagramm-Pakete werden wieder in der richtigen Reihenfolge zusammengesetzt. Erst wenn ein Objekt vollständig erlauscht ist, wird von der konkreten Kindklasse `ProxySniffer` die Methode `handleData()` aufgerufen, um das erhaltene Objekt zu deserialisieren und in eine WID-interne Datenbasis aufzunehmen.

Analog zur Einschränkung der Suche nach interessanten Stellvertreter-Objekten bei `DiscoveringLUS` kann auch mit dem `ProxySniffer` der Suchraum durch Übergabe

[3] Nur beim Bezug der Stellvertreter-Objekte und bei der evtl. Verhandlung über Zeitzusicherungen ist eine unterbrechungsfreie Funkverbindung notwendig.

eines `ServiceTemplate`-Objekts eingeschränkt werden. Objekte, die mit den Methoden aus `MulticastSniffer` empfangen werden, werden dynamisch geprüft und nur bei Unterstützung der gewünschten Schnittstellen oder Attribute in den Zwischenspeicher aufgenommen.

4.5 Sendebake

Neben dem Empfänger-Handgerät muss auch ein Sender auf den Sendebaken das Multicast-Protokoll unterstützen. Die Klasse `heywow.beacon.MulticastAdvertiser` implementiert das Protokoll abstrakt. Die über UDP versendeten Datagramm-Pakete enthalten neben dem Java-Objekt auch einen Protokoll-Kopf mit zusätzlichen Informationen. UDP-Datagramm-Pakete sind selbst wieder Java-Objekte, die über entsprechende Kommunikationsstellen (`MulticastSockets`) von Java-Anwendungen gesendet und empfangen werden können. Die maximale Größe der gesendeten Daten in den Datagramm-Paketen ist auf 1024 Byte festgelegt. Von diesem Kilobyte Daten werden 20 Bytes für das Protokoll verwendet.

0-15: Schlüssel (`ServiceID`)

16: Paketegesamtanzahl (`Byte`)

17: Paketnummer (`Byte`)

18,19: Version (`Integer`)

20-: Daten

Jedes Paket identifiziert seinen Dateninhalt über einen eindeutigen 128bit-Schlüssel. Über ein Byte für die Gesamtanzahl der Pakete zu jedem Schlüssel und einem Index für das aktuelle Datagramm-Paket kann die Reihenfolge auch auf der Seite des Empfängers wieder richtig zusammengesetzt werden.

Wenn für den Stellvertreter ein Versionsattribut in die Nachschlagedienste eingetragen ist, wird dies unter Index 18 und 19 ebenfalls in die Multicast-Datagramme verpackt und somit eine einfache Auswahl aktueller Versionen auf dem Handgerät ermöglicht.

Von Index 20 bis zum Ende des Datensatzes ist es dann möglich, ganze Objekte oder indizierte Objektstücke über Multicast zu versenden.

`MulticastAdvertiser` bietet allen erbenden Klassen dazu zwei Methoden an:

```
protected void advertiseBytes(byte[] data,
                ServiceID serviceID,
                byte[] versionBytes, int port)
```

```
protected void advertiseObj(Object obj,
                ServiceID serviceID,
                Version version, int port)
```

Die Methode `advertiseObj()` nimmt ein beliebiges Java-Objekt, wandelt es in einen Byte-Strom und versendet diesen mit `advertiseBytes()`.

Mit einem Objekt der konkreten Klasse `heywow.beacon.ProxyAdvertiser` kann der lokale Nachschlagedienst auf der Sendebake mit einem `DiscoveringLUS` beobachtet und der erhaltetene Inhalt über Multicast mit `advertiseObj()` aus `MulticastAdvertiser` versendet werden.

Neben dem Versand vollständiger Stellvertreter-Objekte, die in bis zu 256 Datagramm-Paketen verpackt werden können, kann für ein Stellvertreter-Objekt auch lediglich die `ServiceID` des Dienstes und die Adresse des Nachschlagedienstes versendet werden. Die Entscheidung, ob der Dienst für den Benutzer notwendig ist, kann dann die Anwendung auf dem Handcomputer anhand der Dienst-Attribute und der mitgesendeten Methoden-Schnittstelle entscheiden. Für diese Verwendung wird kein Stellvertreter-Objekt direkt verschickt, sondern ein Ersatzobjekt (vgl. Abbildung 4.10), das Informationen über das tatsächliche Objekt enthält.

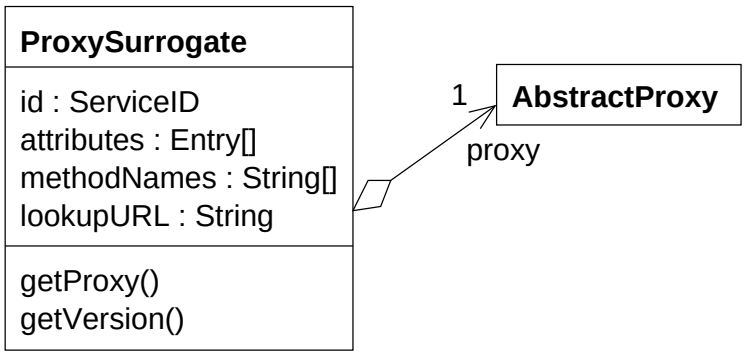

Abbildung 4.10: `ProxySurrogate`

In `heywow.service.ProxySurrogate` wird entweder ein Stellvertreter-Objekt gekapselt oder die URL des Nachschlagedienstes, in dem das Objekt über Jini bezogen werden kann, gespeichert. Um der Anwendung auf dem Handcomputer die Entscheidung, ob das Stellvertreter-Objekt nachgeschlagen wird, zu ermöglichen werden die mit dem Dienst gespeicherten Attribute, sowie eine Liste der verfügbaren Methoden mitgesendet.

In Java ist es möglich, mit einem `MulticastSocket` die Multicast-Kommunikation zu implementieren. Der `MulicastSocket` erlaubt das Betreten einer Multicast-Gruppe und das Belauschen des dortigen Datenverkehrs.

Multicast-Adressen oder Multicast-Gruppen sind normale IP-Adressen. Sie beginnen mit der Bytefolge 1110 und werden als Class D Adressen bezeichnet. Der Multicast-Adressbereich ersteckt sich von 224.0.0.0 bis 239.255.255.255 [Har1997, Kap. 13]. Eine Multicast-Adresse beschreibt Rechnern im Netz, die an dieser Gruppe teilnehmen. Das IANA (Internet asssigned Number Authority) ist verantwortlich für die Vergabe von permanenten Multicast-Adressen[4]. Dort ist im Moment auch die Registrierung der IP-Adressen für das Multicast-Verhalten des Discovery-Protokolls in Jini (vgl. Abschnitt 2.2.2) beantragt. Mit der Adresse ALL-SYSTEMS.MCAST.NET (224.0.0.1) können alle Systeme auf dem lokalen Subnetz erreicht werden. Für weitergehende Netzstrukturen kann den einzelnen Datenpaketen (UDP-Datagramm-Paketen) im Netzwerk auch ein Time-To-Live (TTL) Zähler mitgegeben werden. Dieser Zähler wird bei jedem passierten Router um mindestens 1 verringert und wenn der Wert 0 erreicht hat wird das Datenpaket nicht weitergesendet.

Ein Nachteil des Multicast-Versands ist zwar, dass die Netzlast in denjenigen Segmenten unnötig hoch werden kann, für welche die Daten nicht bestimmt sind, aber insbesondere in Heywow kann dies umgangen werden, weil die Datenpakete eines lokalen Senders nur in seinem lokalen Subnetz verschickt und empfangen werden müssen.

Neben einem Sender (`ProxyAdvertiser`) für den Versand der auf der Bake verfügbaren Stellvertreter-Objekten bietet jede Sendebake auch die nötigen Dienste für ein Jini-Netzwerk.

Für die erste Entwicklungsstufe ist der Standard-Nachschlagedienst aus dem Jini-Paket, `reggie`, ausreichend. Auch an den Web-Dienstgeber werden keine außergewöhnlichen Voraussetzung gestellt. Den Zugriff auf Java-Klassen über HTTP bietet auch schon das einfache Tool aus der Standard-Implementierung für Jini von Sun. Wenn evtl. das Angebot anderer Quellen auf diesem Weg nötig ist, kann auch ein vollwertiger Dienstgeber für alle Dateien im lokalen Dateisystem verwendet werden.

4.6 Nutzklassen und Ereignisse

Im Paket `heywow.util` sind einige nützliche Klassen und Schnittstellen des Heywow-Systems gesammelt. Mit `Constants` werden die notwendigen Konstanten definiert und damit die einfache Anpassung an andere Systemparameter ermöglicht.

Für die Verwendung von `ServiceID`-Objekten bietet die Klasse `ServiceIDFactory` Methoden zur Erzeugung und deren Umwandlung aus und in einfache Bytefolgen an.

Durch den Einsatz der zwei verschiedenen Bezugsformen für die Dienst-Stellvertreter-Objekte ist eine einheitliche Schnittstelle zusammen mit Ereignissen (*Event*s) und Beobachtern (*Listenern*) für die synchronisierte Verarbeitung der Ereignisse notwendig.

[4] aktuelle Liste: ftp://ftp.isi.edu/in-notes/iana/assignments/multicast-addresses

Sowohl `heywow.util.DiscoveringLUS` als auch `heywow.wid.ProxySniffer`[5] erben deshalb von `heywow.util.AbstractExplorer`. In dieser abstrakten Klasse werden die Gemeinsamkeiten der beiden Protokollformen, beim Aufbau einer WID-internen Datenbasis von Stellvertreter-Objekten, zusammengefasst. Außerdem werden hier für beide Klassen Beobachter (`heywow.event.ExploredListener`) registriert und verwaltet. Objekte, die die Schnittstelle `ExploredListener` implementieren, werden bei erfolgreicher Suche nach Stellvertreter-Objekten mit einem Ereignis (`heywow.event.ExploredEvent`) benachrichtigt und können dann auf den erstellten Zwischenspeicher zugreifen.

Durch diese zusätzliche Vererbungshierarchie wird der Gebrauch der verschiedenen Protokolle vereinheitlicht. Auf diese Weise können beide Protokolle für den Benutzer transparent eingesetzt und ausgetauscht werden.

4.7 WID-Anwendungen

Neben den drei Beispieldiensten (vgl. Abschnitt 4.3) enthält die Java-Bibliothek in `heywow.client` auch noch verschiedene Muster für Anwendungen auf dem Handcomputer.

Mit der Java-Anwendung `heywow.client.LUSContent` kann der Inhalt aller im lokalen Netz erreichbaren Nachschlagedienste angezeigt werden. Die Applikation baut auf den Kernklassen des Jini-Programmiersystems auf. Es werden allgemeine Informationen über die gefundenen Nachschlagedienste, eine Liste aller mit einem Dienst registrierten Attribute, sowie die Klassen der gespeicherten Stellvertreter aufgelistet.

Diese Anwendung eignet sich für den Test der Funktionstüchtigkeit der entwickelten Dienste. Es werden keine weiteren Klassen aus der Heywow-Bibliothek und nur Basisklassen des Jini-Systems verwendet.

Ein Muster für eine Anwendung, wie sie auch auf einem Endgerät denkbar wäre, ist `heywow.client.UI`. Die Anwendung (vgl. Abbildung 4.11) setzt beide Bezugsmethoden für Stellvertreter-Objekte, die das Heywow-System anbietet, ein.

Die Klasse implementiert einen `heywow.event.ExploredListener` und kann sowohl mit `DiscoveringLUS` als auch mit `ProxySniffer` eine WID-interne Datenbasis der lokal verfügbaren Stellvertreter aufbauen.

Die Anwendung selbst ist unter Verwendung der Grafikklassen von kAWT[6] entwickelt. Diese Klassenbibliothek bietet für die Java 2 Micro Edition eine fast vollständige Implementierung der Basisklassen der `java.awt`-Pakete.

[5] `ProxySniffer` erbt über `MulticastSniffer` und implementiert die notwendigen Methoden.
[6] Informationen und Quellen unter: http://www.kawt.de

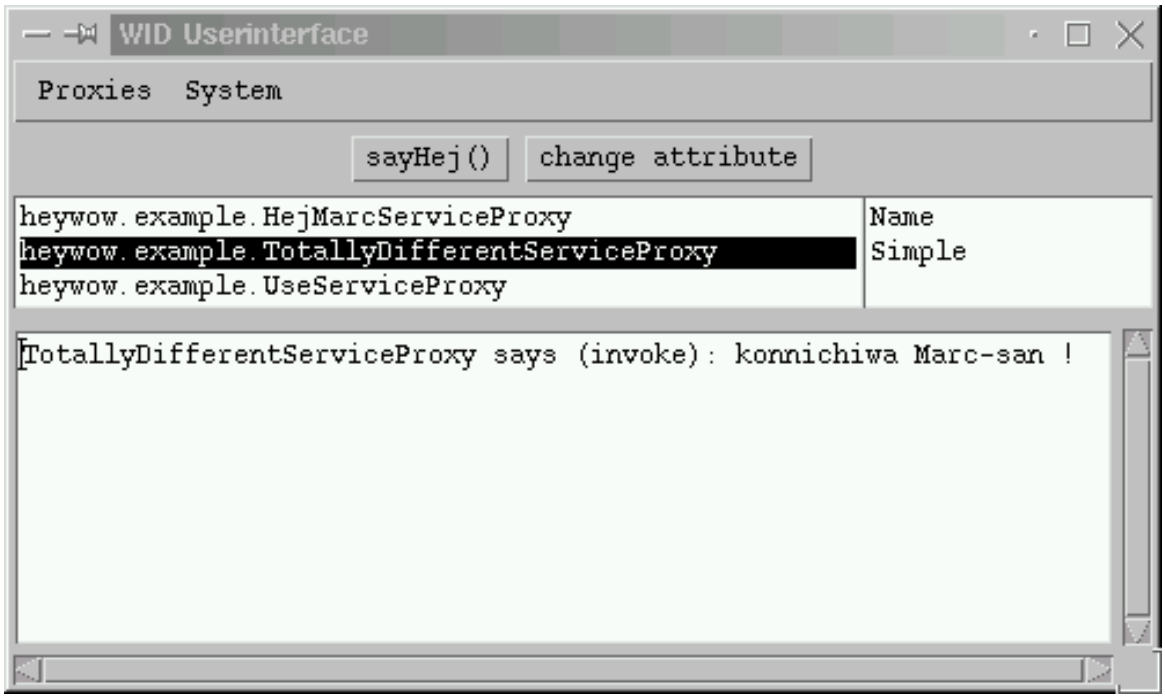

Abbildung 4.11: WID-Musteranwendung

4.8 Konfiguration

Bei der Entwicklung der Klassenbibliothek für das Heywow-System wurde Wert auf eine funktionale Trennung der einzelnen Klassen gelegt.

Somit ist es möglich, durch Erzeugung von Java-Dateiarchiven die nötigen Klassen für die einzelnen Softwarekomponten zu trennen.

Bei der Verwendung beliebiger Heywow-Dienste auf einem mobilen Endgerät, ist es möglich, die Kommunikation, z. B. für den Bezug von Klassen-Dateien über HTTP-Dienstgeber, auf den Handcomputer zu verringern. Die Basisklassen auf denen alle Dienste aufbauen, können schon fest installiert sein und nur konkrete Dienste-Klassen müssen über Jini oder Multicast dynamisch bezogen werden.

4.9 Test

Zum Test der Klassenbibliothek mit den implementierenden Beispieldiensten wurde die Java 2 Standard Edition (vgl. Abschnitt 2.1), die eine Unterstützung von RMI bietet, eingesetzt. Als Hardwarebasis wurde ein Festnetz aus Arbeitsplatzrechnern verwendet, und dort der Einsatz der Dienste getestet.

Der Ansatz von 100% purem Java, mit der Verwendung von Jini als Basistechnologie, ermöglicht aber auch eine einfache, kostensparende Entwicklung der Heywow-Dienste in bestehenden lokalen Netzwerken. Mit der Portierung der aktuellen Version der Connected Limited Device Configuration CLDC 1.0 nach Linux besteht eine professionelle Entwicklungsumgebung [Hei2000].

Außerdem ermöglicht eine Beschränkung im Funktionsumfang der Standard Edition schon jetzt die Implementierung und den Test von Diensten.

4.10 Übertragung

Eine große Hürde bei der Übertragung der Dienste und Anwendungen im Heywow-System auf die mobile Plattform ist im Moment noch das Fehlen einer vollständigen RMI-Unterstützung für eine Implementierung der Mirco-Edition.

Mit dem J2ME RMI-Profil [JSR2000b] gibt es zur Zeit nur eine Spezifikation im Entwicklungsstadium, die auf die Java 2 Micro Edition CDC (vgl. Abschnitt 2.1) aufbaut. Mit einer Implementierung nach dieser Spezifikation soll der entfernte Methodenaufruf analog zur Standard Edition auch auf kleineren Geräten ermöglicht werden.

Ohne diese Erweiterung ist derzeit noch kein Kommunikationstandard auf RMI-Basis unter Java für tragbare Kleinstcomputer verfügbar. Alle Implementierungen basieren bisher auf anderen Schnittstellen. Deren Verwendung wurde beim Entwurf der Java-Klassenbibliothek, zugunsten eines reinen Java-Standards verworfen.

Durch den Aufbau der gesamten Klassenbibliothek auf die Basis eines IP-Netzwerks ist es neben der einfachen Integration in bestehende lokale Festnetze auch möglich, zukünftige Entwicklungen bei den lokalen Funknetzen zu nutzen.

4.11 Zusammenfassung und Bewertung

Ziel der Arbeit war es, eine Software-Architektur für die lokale Dienstleistungserbringung im Heywow-System zu schaffen, die es ermöglicht, von einem mobilen Handcomputer aus, lokal in einem IP-Funknetzwerk auf beliebige Dienste zuzugreifen.

Nach einer Einführung verschiedener Grundlagen in Kapitel 2 wurden in Kapitel 3 schrittweise die wesentlichen Anforderungen an das System erarbeitet und daraus Entscheidungen für die Architektur abgeleitet.

Schließlich wurde in 100% purem Java eine Klassenbibliothek implementiert (vgl. Kapitel 4), die das Skelett für lokale Dienste bietet und leicht mit Dienst-Funktionalität zu konkreten Diensten erweiteret werden kann.

Durch die Einführung eines Java-Rahmensystems für lokale Dienste, aufbauend auf Jini, basierend auf einer TCP/IP-Netzwerk-Architektur, ist es auch möglich das Angebot zentraler Dienstleistungen tranparent in das System zu integrieren. Dem Benutzer eines mobilen Handcomputers kann durch den Zugriff auf private Daten, Termine und Informationen eine personalisierte Leistung angeboten werden, die dezentral von lokalen Diensteanbietern entwickelt werden kann.

Das Skelett für Dienste im Heywow-System ermöglicht dem Benutzer den Zugriff auf völlig neue und unbekannte Dienstleistungen. Die Dienste können sowohl einfache Daten, wie z. B. Ortinformationen für zielgenaue lokale Navigation ermöglichen, als auch den Bezug von Stellvertreter-Objekten für die Kommunikation mit Dienstgebern lokaler Diensteerbringer für z. B. Kauf und Verhandlungsinteraktion bieten.

Für batteriebetriebene, mobile Handcomputer ist der sparsame Umgang mit den Energieressourcen besonders wichtig. Mit der Erweiterung des Jini-Systems um eine weitere Protokollform über Multicast kann der Handcomputer energiesparend die Stellvertreter-Objekte eines Dienstes beziehen.

Durch die flexible Gestaltung des Gesamtsystem, aufbauend auf Java/Jini und den Einsatz von standardisierten IP-Netzwerken ist die Software-Architektur offen für weitere Entwicklungen.

5 Erweiterungsmöglichkeiten und Ausblick

5.1 Dienste

Aufbauend auf der bestehenden Klassenbibliothek können beliebige lokale Dienste im Heywow-System entwickelt werden. Das Rahmensystem vereinfacht die Entwicklung von konkreten Dienstleistungen und ist offen für das Angebot auch komplexer Anwendungen.

Um den störungsrobusten Betrieb des mobilen Handcomputers gewährleisten zu können, lässt sich der Einsatz von Zeitzusicherungen noch benutzerfreundlicher ausgebauen. Für einen Dienst, bei dem die aktuelle Zeitzusicherung ausläuft und die Verlängerung auf Grund von Problemen in der Funkverbindung zwischen Handcomputer und Verwalter der Zeitzusicherungen scheitert, kann das Stellvertreter-Objekt, zumindest in einem Sicherungszwischenspeicher, bestehen bleiben. Auf diesem Weg wird bei einem späteren Gebrauch mit einer dann erhaltenen Zeitzusicherung, die Funkkommunikation des Handcomputers reduziert und Energie eingespart. Hier kann für speicherseitig schwächer ausgerüstete Handgeräte bei Bedarf auch der Benutzerwunsch miteinbezogen werden.

Der Einsatz aktivierbarer Dienste ist auch noch nicht optimal geregelt. Nachdem die Dienstgeberkomponente von einem Klienten aktiviert wird, um eine Zeitzusicherung zu vergeben und zu verwalten, kann sich der Prozess erst nach Ende der spätesten vergebenen Zusicherungszeit beenden. Die Verwendung des Landlord-Leasing-Konzepts erlaubt es dem Dienst hier, statt einer Instanz von `LandlordServer` auch einen zusätzlichen Dienst des Heywow-Systems für die Verwaltung einzusetzen, der für mehrere Dienste als Verwalter der Zeitzusicherungen auftritt.

Erweiterbar sind auch die Entscheidungen über den Versand von Stellvertreter-Objekten über Multicast. Hier kann z.B. zusätzlich zur Spezifizierung der Heywow-Dienste durch die Implementierung der Schnittstelle `ServiceInterface` eine umfangreichere Hierarchie von Schnittstellen, die von `ServiceInterface` erben, eingeführt werden, über die die Art der Dienste genauer beschrieben wird. Durch den Versand der unterschiedlichen Stellvertreter-Objekte über verschiedenen Multicast-Adressen verringert

sich der überflüssige Empfang von uninteressanten Daten auf dem Handcomputer und der Benutzer erhält einen schnelleren Zugriff auf relevante Dienste.

Außerdem ist die Integration und der Zugriff auf zentrale Dienste im Heywow-System, z. B. über Mobilfunk, bisher nur durch das Konzept der Verwendung von Jini als abstraktem Netzwerkprotokoll sichergestellt. Mit der Definition einer allgemeinen Schnittstelle für den transparenten Zugriff auf beide Dienstformen ist dies zu verbessern.

5.2 Sicherheit

Die Sicherheit in Jini basiert auf dem Sicherheitssystem von Java 2. Insbesondere für netzwerkweit interagierende Anwendungen ist ein Sicherheitsverwalter (*Security Manager*) notwendig. Durch die Verwendung feingranular eingerichteter Policy-Dateien ist es möglich, das Java zugrundeliegende Sicherheitssystem auch gegen feindliche Angriffe auf das Jini-System einzusetzen [New2000, Kap. 12].

Mit dem Einsatz eines Sicherheitsverwalters können einer Anwendung einzelne Aktionen und Ressourcen freigegeben werden. Dies ist ein additives Verfahren, und jede Funktion wird einzeln erlaubt.

Für die verschiedenen Teilnehmer am Heywow-System gelten unterschiedliche Sicherheitsanforderungen. Spezielle Policy-Dateien für die einzelnen Teilnehmer tragen den verschiedenen Sicherheitsbedürfnissen Rechnung.

Besonders der Handcomputer des Benutzers im Heywow-System muss vor Angriffen geschützt werden. Dort sind evtl. persönliche und sicherheitsrelevante Daten unverschlüsselt abgelegt. Problematisch ist die Notwendigkeit, dass für den Einsatz eines Dienstes hier der Benutzer ein über das Jini-Netzwerk bezogenes Stellvertreter-Objekt ausführt. In den Nachschlagediensten werden diese Java-Objekte nur gespeichert und unterliegen deshalb nicht direkt so hohen Sicherheitsanforderungen. Auf Dienstgeberseite werden wieder andere Anforderungen gestellt. Ein Benutzer und seine Aktionen müssen auch dort authentifiziert werden.

Um den Benutzern ein hohes Vertrauen in ein 'fremdes' Java-Objekt zu ermöglichen, kann sich in Java-Systemen der Dienstgeber authentifizieren. Durch den Einsatz von signierten Java-Archivdateien wird die Echtheit der Java-Klassen bezeugt. Dieses Sicherheitskonzept ist schon in den Java-Klassenlader integriert.

In [AK2000] wird für den sicheren Einsatz von Stellvertreter-Objekten in einem Ad-Hoc-Netzwerk ein einfaches Protokoll für den sicheren Einsatz von Diensten über Jini in acht Schritten vorgestellt. Nach dem Bezug eines Stellvertreter-Objekts, wird dessen Authentifikation über signierte Java-Dateiarchive verifiziert und im Anschluss ein sicherer Kommunikationskanal zwischen Klient und Dienst aufgebaut.

Für den sicheren Einsatz von Diensten dezentraler Diensteanbieter über das Heywow-System muss das in [AK2000] vorgeschlagene Sicherheitskonzept erweitert werden.

Der Benutzer eines mobilen Handcomputers kann nicht mit allen Diensteanbietern ein einzelnes, vertragliches Vertrauensverhältnis aufbauen. Ein Benutzer vertraut auf das Heywow-System und schon der Bezug eines Stellvertreter-Objekts aus einem Nachschlagedienst muss mit bestimmten Zusicherungen verbunden sein.

Der Einsatz eines speziellen Nachschlagedienstes kann die Sicherheit im Heywow-System verbessern. Der Abschnitt 3.3 stellte den Nachschlagedienst als wichtige Vermittlerkomponente zwischen Klienten und Diensteanbietern im Heywow-System vor. Durch die Einführung von sicheren Protokollen für die Kommunikation mit dem Nachschlagedienst kann Heywow die Authentizität und zusätzliche Garantien für die Dienst-Stellvertreter-Objekte gegenüber den Benutzern übernehmen.

Glossar

GHz: Gigahertz. Milliarde Hertz.

Hertz (Hz): Frequenzeinheit.

HTTP: Das Hypertext Transfer Protocol ist ein Protokoll, aufbauend auf TCP, für den Informationsaustausch über das WWW.

IP-Netzwerk: Ein auf dem Internet-Protokoll basierendes Computernetzwerk.

J2ME: Java 2 Micro Edition.

J2SE: Java 2 Standard Edition.

LAN: local area network. Ein lokales Computernetzwerk.

TCP: Transmission Control Protocol. Ein verbindungsorientierter Dienst auf Basis des Internet Protokolls (IP).

UDP: User Datagram Protocol. Ein einfacher aber unzuverlässiger Datagramm-Paket-Dienst auf Basis des Internet Protokolls (IP).

URL: Unifom Resource Locator. Eine protokollbehaftete eindeutige Adresse für ein Objekt im Internet.

WID: wireless information device. Handgerät mit Funkverbindung zu Diensten.

WLAN: Wireless local area network. Ein lokales Computernetzwerk ohne physikalische Verkabelung der Netzwerkknoten.

Literaturverzeichnis

[AK2000] Fredrik Andersson, Magnus Arlsson: Secure Jini Services in Ad Hoc Networks. Master of Science Thesis, Royal Institute of Technology (KTH), Department of Teleinformatics, Stockholm, 2000.
http://www.e.kth.se/~e95_fan/exjobb/Thesis.pdf

[ALM1998] Guiseppe Anastasi, Luciano Lenzini, Enzo Mingozzi: MAC Protocols for Wideband Wireless Local Access: Evalution Toward Wireless ATM. aus: IEEE Personal Communications Magazine, 1998, Band 5, Heft 5, S. 53ff.

[ARS2000a] Michael Angermann, Patrick Robertson, Alexander Steingass: Integration of Navigation and Communication Services for Personal Travel Assistence using a Jini and Java Based Architecture. German Aerospace Center (DLR), Oberpfaffenhofen, 2000.
http://www.heywow.com/download/gnss99ab.zip

[ARS2000b] Michael Angermann, Patrick Robertson, Alexander Steingass: Heywow. Key-note, Präsentation. German Aerospace Center (DLR), Oberpfaffenhofen, 2000.
http://www.heywow.com/download/gnss99.zip

[BBK2000] Andreas Butz, Jörg Baus, Antonio Krüger: Augmenting Buildings with Infrared Information. University Saarbrücken, 2000.
http://www.coli.uni-sb.de/sfb/publications/butzetal2000e-en.html

[BC1999] Specification of the Bluetooth System. Voilume 1, Core. Version 1.0B, 29.11.1999.

[BF1999] Daniel J. Berg, J. Steven Fritzinger: Advanced techniques for Java developers. Revised Edition, Wiley Computer Publishing, 1999.

[BP1999] Specification of the Bluetooth System. Volume 2, Profiles. Version 1.0B, 29.11.1999.

[Cou1999] Justin Couch: Java 2 networking. McGraw-Hill, 1999.

[DH1995] Norbert Diehl, Albert Held: Mobile Computing. Thomson, 1995.

[Dö1999] Andreas Dörr: Einstieg in die Jini-Programmierung. aus: Java Magazin 3/1999, S. 23ff.

[Edw1999] W. Keith Edwards: Core Jini. Prentice Hall, 1999.

[Fla1998] David Flanagan: Java in a Nutshell, 2. Auflage. O'Reilly, 1998.

[Gla2000] Kay Glahn: Java allgegenwärtig. aus: Java Magazin 1/2000, S. 40ff.

[Gra1999] Matthew Konefal Gray: Infrastructure for an Intelligent Kitchen. Master of science in Media Arts and Sciences Thesis, MIT Media Labs, 5/1999. http://hive.www.media.mit.edu/projects/hive/mkgray-thesis/ps/thesis-with photos.ps.gz

[GS2000] Martin Gitsels, Jochen Sauter: Spontane Vernetzung mit Jini. aus: Java Spektrum 4/2000, S. 16ff.

[Har1997] Elliotte Rusty Harold: Java Network Programming. O'Reilly, 1997.

[Hei2000] Andreas Heilwagen: Frisch geröstet. J2ME unter Linux. aus: iX 10/2000, S. 172ff.

[HE2000] Jaap C. Haartsen, Ericsson Radio Systems B. V.: The Bluetooth Radio System. aus: IEEE Personal Communications Magazine, 2000, Band 7, Heft 1, S. 28ff.

[Hof2000] Lars Hoffmann: Bluetooth. Mobile Kommunikation in kleinen Netzwerken. Proseminar: Mobile Computing, Forschungszentrum Informatik an der Universität Karlsruhe (FZI), 2000. http://palmpower.fzi.de/Proseminar/Ausarbeitung/Papers/LarsHoffmann_ Bluetooth.pdf

[HS1997] Paul J.M. Havinga, Gerard J.M. Smit: Minimizing energy consumption for handheld computers in Moby Dick. Proceedings Euromicro 97, S. 196ff, 7/1997. http://wwwhome.cs.utwente.nl/~havinga/papers/energy.MD.euro97.pdf

[HS2000] Paul J.M. Havinga, Gerard J.M. Smit: Moby Dick, on the design of a Swiss army knife of computing. International Conference on Advances in infrastructure for Electronic Business, Science, and Education on the internet (SSGRR 2000), 2000. http://wwwhome.cs.utwente.nl/~havinga/papers/havinga_ssgrr2000.pdf

[JSR2000a] JSR #000036: J2ME Connected Device Configuration. Proposed Final Draft, 8/2000. http://java.sun.com/aboutJava/communityprocess/jsr/jsr_036_j2mecd.html

[JSR2000b] JSR #000066: J2ME RMI Profile. Community Review draft Approved 9/2000.
http://java.sun.com/aboutJava/comminityprocess/jsr/jsr_066_rmime.html

[KEa2000] Tim Kindberg, et al.: People, Places, Things. Web Presence for the Real World. White paper, 2000.
http://www.cooltown.hp.com/papers/webpres/WebPresence.htm

[Keh2000] Roger Kehr: Spontane Vernetzung. Infrastrukturkonzepte für die Post-PC-Ära. aus: Informatik Spektrum, Band 23, Heft 3, 6/2000, S. 161ff.

[Mar1999] Benoit Marchal: Java Resource Center: Jini Part 1-6. Digital Cat, Februar - August 1999.
http://www.digitalcats.com/US/articles/Ben/Jini[1..6].html

[MEa1999] Nelson Minar, et al.: Hive: Distributed Agents for Networking Things. MIT Media Labs, 8/1999.
http://nelson.www.media.mit.edu/people/nelson/research/hive-asama99/hive-asama99.ps

[Mob1997] Moby Dick - The Mobile Digital Companion. Result of the first phase. Final Report. 7/1997.
http://wwwhome.cs.utwente.nl/~havinga/frl.html

[Net2000] IP-Multicasting. 3tlge Artikelserie aus: NetworkWorld Germany, 3-5/2000
http://www.networkworld.de/

[New2000] Jan Newmarch: Guide to JINI Technologie. Version 2.04, 2000.
http://pandonia.canberra.edu.au/java/jini/tutorial/Jini.xml

[OW1997] Scout Oaks, Henry Wong: Java Threads. O'Reilly, 1997.

[OW2000] Scout Oaks, Henry Wong: Jini in a Nutshell. A Desktop Quick Reference. O'Reilly, 2000.

[Riz1997] Luigi Rizzo: Effective Erasure Codes for reliable Computer Communication Protocols. aus: ACM Computer Communication Review, Vol.27, n.2, S. 24ff, 4/1997.
http://www.iet.unipi.it/~luigi/fec_ccr.ps

[Sch1994] Georg Schoeck: Seneca für Manager. Insel Verlag, 1994

[SEa1996] Mark Stemm, et al.: Reducing Power Consumption of Network Interfaces in Hand-Held Devices. Vortrag: 3rd International Workshop on Mobile Multimedia Communications (MoMuc-3), Princeton, NJ, USA, 9/1996.
http://HTTP.CS.Berkeley.EDU/~stemm/publications/MoMuc3.ps.gz

[Sun1998a] Sun Microsystems, Inc.: Java Object Serialization Specification. 11/1998.
ftp://ftp.javasoft.com/docs/jdk1.2/serial-spec-JDK1.2.pdf

[Sun1998b] Sun Microsystems, Inc.: Java Remote Method Invocation Specification. Rev. 1.50, JDK 1.2, 1998.
ftp://ftp.javasoft.com/docs/jdk1.2/serial-spec-JDK1.2.pdf

[Sun1999] Sun Microsystems, Inc.: Java 2 Plattform. 12.12.1999.
http://java.sun.com/java2/

[Sun2000] Sun Microsystems, Inc.: Java BibliographySun Microsystems, Inc.: CLDC. Specification (First Release). J2ME. Palo Alto, 4/2000.
http://java.sun.com/aboutJava/communityprocess/final/jsr030/CLDCSpecification1.0.zip

[Wet2000] Nicolas Wettstein: Grundlagen der Vernetzung: drahtlose und mobile Kommunikation. Fachseminar: Ubiquitous Computing. ETH Zürich, 2000.
http://www.inf.ethz.ch/vs/education/SS2000/UC/papers/Wettstein.pdf

[WM1999] Markus Wallmyr, Ola Markström: Bluetooth and Jini. The future of Networking. Seminararbeit, Department of Computer Systems, Institution of Information Technology, Uppsala University, 1999.
http://www.docs.uu.se/~perg/course/datakom2/it99/Bluetooth_Jini.ps

[WZ1999] Ralph Wittmann, Martina Zitterbart: Multicast. Protokolle und Anwendungen. DPunkt, 1999.