Lehrstuhl für Messsystem- und Sensortechnik

Integration von heterogenen Bussystemen in die Heimautomatisierung unter Verwendung von Middleware

Kay Werthschulte

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing. K. Diepold
Prüfer der Dissertation:	1. Univ.-Prof. Dr.-Ing., Dr.-Ing. habil. F. Schneider, i.R.
	2. Univ.-Prof. Dr.-Ing. J.-U. Varchmin, Technische Universität Braunschweig

Die Dissertation wurde am 16.01.2003 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 08.05.2003 angenommen.

Technische Universität München
Lehrstuhl für Messsystem- und Sensortechnik

Integration von heterogenen Bussystemen in die Heimautomatisierung unter Verwendung von Middleware

Dipl.-Ing. Kay Werthschulte

Abstract

Stichwörter: Feldbus, Sensor/Aktor-Kommunikation, Systemverknüpfung, Middleware, Software-Agenten

Diese Arbeit stellt ein Gesamtkonzept für den Bereich der Heimautomatisierung vor, das die systemübergreifende Verbindung mehrerer Bussysteme ermöglicht. Das Konzept berücksichtigt dabei die Eigenschaften der einzelnen Bussysteme, erreicht aber dennoch eine Abstraktion der Buseigenschaften in Form von Diensten. Dazu wird eine hierarchische Systemstruktur entwickelt, deren Systemebenen durch Übergänge miteinander verbunden sind. Dabei können Geräteeigenschaften jeder Ebene auf Dienste abgebildet werden. Durch die Definition eines Basisdienstes können Funktionen des Systems exportiert werden. Zu den betrachteten Bussystemen (IEEE 1394, EIB, IEEE 802.3, IEEE 802.11, Bluetooth, DECT) wurden Dienstumsetzungen unter Verwendung von Middleware (HAVi, Jini, OSGi) realisiert.

Key words: fieldbus, HVAC, system interconnection, middleware, software agents

The concept given in this work is intended to be used in the home automation area to build an open platform for system interconnectivity. Although several different systems are being coupled by turning them into services the individual properties of the communication systems are maintained. Therefore a hierarchical communication structure was built up categorized by the bussystems' data transfer speed. The different levels in this model are tied together through gateways. Each service mapped to a device includes a set of basic operations which reveals the device's functionality. The service implementations built for several bussystems (IEEE 1394, EIB, IEEE 802.3, IEEE 802.11, Bluetooth, DECT) were realized by using middleware (HAVi, Jini, OSGi).

Inhaltsverzeichnis

1. EINLEITUNG 1

2. ZIELSETZUNG UND GLIEDERUNG 4

3. ÜBERBLICK ÜBER VERGLEICHBARE ANSÄTZE 6

3.1 WISSENSCHAFTLICHE ARBEITEN 6
3.2 AUTOMATISIERUNGSTECHNISCHE LÖSUNGEN 7
3.3 STANDARDISIERUNG VON HEIMAUTOMATISIERUNGSSYSTEMEN 9
3.4 BUSSYSTEME UND DEREN BEDEUTUNG IN DER HEIMAUTOMATISIERUNG 9
3.4.1 Vorhandene Bussysteme 9
3.4.1.1 Europäischer Installationsbus 10
3.4.1.2 LonWorks 11
3.4.1.3 IEEE1394 11
3.4.1.4 Ethernet 12
3.4.1.5 Wireless LAN 13
3.4.1.6 Digital Enhanced Cordless Telecommunication 13
3.4.1.7 Bluetooth 14
3.4.2 Analyse und Kategorisierung der Bussysteme 15
3.4.2.1 EIB 17
3.4.2.2 IEEE 1394 18
3.4.2.3 IEEE 802.3 und IEEE 802.11 18
3.4.2.4 Bluetooth 19
3.4.2.5 DECT 19

4. ERSTELLUNG DES GESAMTKONZEPTS 21

4.1 EINSTUFIGES SYSTEM 21
4.1.1 Auslastungsabschätzung der Einzelstufen 23
4.1.2 Ausfallbetrachtung 23
4.2 MEHRSTUFIGES SYSTEM 23
4.2.1 Konzeptionelles Vorgehen 27
4.2.1.1 Schnittstellenentkopplung durch Verwendung von Middleware 27
4.2.1.2 Konzeption und Systemmodell 30
4.2.2 Definition von Diensten 33
4.2.2.1 Dienste aus der Sicht des Anwenders 34
4.2.2.2 Dienstdefinitionen 36
4.2.3 Subsystemverknüpfung 42
4.2.4 Einsatz von Agenten 45
4.2.4.1 Definition eines Agenten 46
4.2.4.2 Aufbau des Agenten 47
4.3 VERGLEICH UND AUSWAHL 48

5. REALISIERUNG 49
5.1 GERÄTEBESCHREIBUNG 50
5.1.1 EIB-Geräte 50
5.1.2 Videokamera 50
5.1.3 Fernseher 52
5.1.4 Web Pad 52
5.1.5 PDA 52
5.2 BUSSYSTEMANKOPPLUNG 53
5.2.1 Ethernet 54
5.2.1.1 Basisdienst 57
5.2.1.2 Informationsdienst 59
5.2.1.3 Lokalisierungsdienst 62
5.2.1.4 Visualisierungsdienst 63
5.2.1.5 Auslastungsanalyse 64
5.2.2 IEEE 1394 72
5.2.2.1 Eingliederung in das Systemmodell 72
5.2.2.2 Aufbau des HAVi-Stacks 74
5.2.2.3 Übergang von EIB-HAVi 77
5.2.2.4 Erstellung der Kommunikationsschichten für den EIB 82
5.2.2.5 Übergang Jini-HAVi 86
5.2.2.6 Auslastungsanalyse 87
5.2.3 DECT 90
5.2.3.1 Auslastungsanalyse 93
5.2.4 Bluetooth 95
5.2.5 EIB 97
5.3 SERVICE-AGENTEN 101
5.3.1 Szenario einer Dienstverknüpfung 101
5.3.2 Agenten 102
5.4 EXTERNER ZUGANG 104
5.4.1 OSGi 104
5.4.2 RMI über TCP/IP 105
5.5 DARSTELLUNG 107
5.5.1 HAVi am Beispiel FAV 107
5.5.2 Applet mit Web Pad 108
5.6 SICHERHEITSASPEKTE 109
5.6.1 RMI und Jini 109
5.6.2 HAVi 111
5.6.3 DECT 112
5.6.4 Wireless LAN 112
5.6.5 Dienstzugriff 112
5.7 AUSFALLBETRACHTUNG 113
5.8 EFFIZIENZBETRACHTUNG DES GESAMTSYSTEMS 114

6. KONFIGURATION UND VERWALTUNG 115
6.1 ZUSTÄNDIGKEITEN 115
6.2 KONFIGURATION DER SYSTEMPARAMETER 117
6.2.1 EIB 117
6.2.2 Jini 118
6.2.3 Fernzugriff, HAVi und DECT 120
6.2.4 OSGi 122

7. ZUSAMMENFASSUNG 123

8. LITERATUR 128

ANHANG 135
ABKÜRZUNGSVERZEICHNIS 135
TABELLENVERZEICHNIS 138
ABBILDUNGSVERZEICHNIS 139

1. Einleitung

Die heutigen Haushalte werden noch zumeist mit konventionellen elektrischen Installationen ausgestattet. So wird die Verkabelung zwischen den Sensoren und Aktoren im Allgemeinen hartverdrahtet, also direkt durch elektrische Verbindungen miteinander gekoppelt. Dadurch sind die Beziehungen der sich im Haus befindlichen Geräte weitgehend festgelegt und sie können nicht ohne bauliche Maßnahmen verändert werden. Vergleichbar mit dem Einzug der Feldbusssysteme in den Bereich der industriellen Sensor/Aktor-Kommunikation in der Feldebene ist eine Vernetzung der Komponenten innerhalb von privaten Haushalten auf lange Sicht nicht mehr zu vernachlässigen. Bereits heute bestehen Bussysteme, die diese Aufgabe übernehmen. Die Sensoren und Aktoren werden über ein serielles Bussystem miteinander verbunden. Auf dem Markt sind mehrere solcher Systeme erhältlich. Für den Heizung/Klima/Lüftung-Bereich wird in Zweckgebäuden LonWorks[1] eingesetzt. Weiterhin gibt es X10, *Europäischer Installationsbus* (EIB), BatiBUS und *European Home System* (EHS). In Zukunft werden EIB, EHS und BatiBUS zu einem Standard verschmelzen (Konnex). Damit ist bereits ein Großteil der am Markt erhältlichen Bussysteme für die Gebäudeautomatisierung abgedeckt. Daneben gibt es noch herstellerspezifische Busse, die jedoch in der Regel keine große Marktdurchdringung finden und eher in sehr speziellen Bereichen (z.B. Lichtsteuerung) eingesetzt werden. Für die meisten Bussysteme gibt es Ansätze, nicht nur die Steuerung der Geräte zu koordinieren, sondern den Telegrammverkehr auch zur Fehlersuche und Anzeige zu verwenden. So werden beispielsweise in Zweckgebäuden Visualisierungen zum Facility Management eingesetzt.

Neben der Verbindung einfacher Aktoren und Sensoren zur Heizungssteuerung, Licht- und Raumtemperaturregelung sind in jedem Haushalt auch Multimediageräte zu finden. Als ein Medium für die Übertragung großer Datenmengen wird in der Netzwerkkommunikation Ethernet mit TCP/IP verwendet. Durch die hohe Verbreitung des TCP/IP-Protokolls wurden Protokolle spezifiziert, die den paketorientierten Transport von isochronen Daten beschreiben. Ein Beispiel dafür ist der Standard H.323[2], eine Anpassung von H.320 (Videokonferenz über ISDN).

Speziell für Multimediaanwendungen wurde 1995 der Standard IEEE 1394-1995 verabschiedet. Mit diesem Bussystem können Daten mit einer Übertragungsrate von bis zu 400 MBit/s transportiert werden. Das Protokoll sieht sowohl eine asynchrone als auch eine isochrone Datenübertragung vor. Mittlerweile wird der IEEE1394 auch für andere Bereiche, beispielsweise in der Fertigungstechnik, eingesetzt. Der Standard beschreibt jedoch nur den Physical und Data Link Layer. Für die Applikationschicht hat sich noch kein Standard zur Vernetzung von Geräten unterschiedlicher Hersteller durchgesetzt,

[1] Dies ist ein Feldbussystem der Firma Echelon. S. Kap. 3.4.1.2.

[2] S. [ITUH.323]

obwohl bereits einige Lösungen auf dem Markt erhältlich sind. So gibt es mit der Norm IEC 61883 einen Standard zur Übertragung von Daten über IEEE 1394[3].
Die Anforderungen in diesem Bereich sind nicht vergleichbar mit der Sensor/Aktor-Kommunikation aus der Feldbustechnik. Während es in der Feldbustechnik um die sichere Übertragung kleiner Informationsmengen geht, steht bei Multimediageräten die Übertragung großer Datenmengen im Vordergrund. Der Verlust eines kleinen Teils dieser Daten ist nicht weiter von Interesse, da es sich zumeist um Audio- und Videodaten handelt, die isochron übertragen werden und eine Verfälschung der Daten durch den Datenkanal nur in einer fehlerhaften Wiedergabe eines Audio- oder Videofragments resultiert.
Die drahtgebundene Anbindung von Geräten ist aus der Vernetzung nicht wegzudenken, dennoch sind mit dem Fortschreiten der Miniaturisierung immer kompaktere mobile Eingabe- und Telekommunikationsgeräte verfügbar. Die *Personal Digital Assistants* (PDA) verfügen mittlerweile über Farbdisplays und die Rechenleistung der verwendeten Prozessoren erlaubt sogar den beschränkten Einsatz von Mutimediaanwendungen. Damit rücken auch PDAs, die über Funksysteme an das Gesamtsystem angebunden werden können, in das Umfeld der Heimautomatisierung zur Steuerung und Anzeige. Von den vorhandenen Funksystemen eignen sich unter anderem Bluetooth, *Digital Enhanced Cordless Telecommunictaion* (DECT) und Wireless LAN für die Verwendung mit digitalen Assistenten. Die Übergänge zwischen diesen Systemen und Ethernet sind durch Bridges möglich.
Die meisten der oben genannten Systeme werden für die hausinterne Kommunikation verwendet. Für die externe Anbindung eines Gebäudes zu Wartungs- und Überwachungszwecken werden andere Medien benötigt. Denkbar sind Einwahlverbindungen (Analog, ISDN), feste Anbindungen an das Internet (xDSL) oder aber drahtlose über Funk-Gateways. Auch dort gibt es eine Vielzahl von Variationen, die sich in der Bereitstellung von Diensten für die Sprach- und Datenübermittlung und in den verfügbaren Bandbreiten unterscheiden.

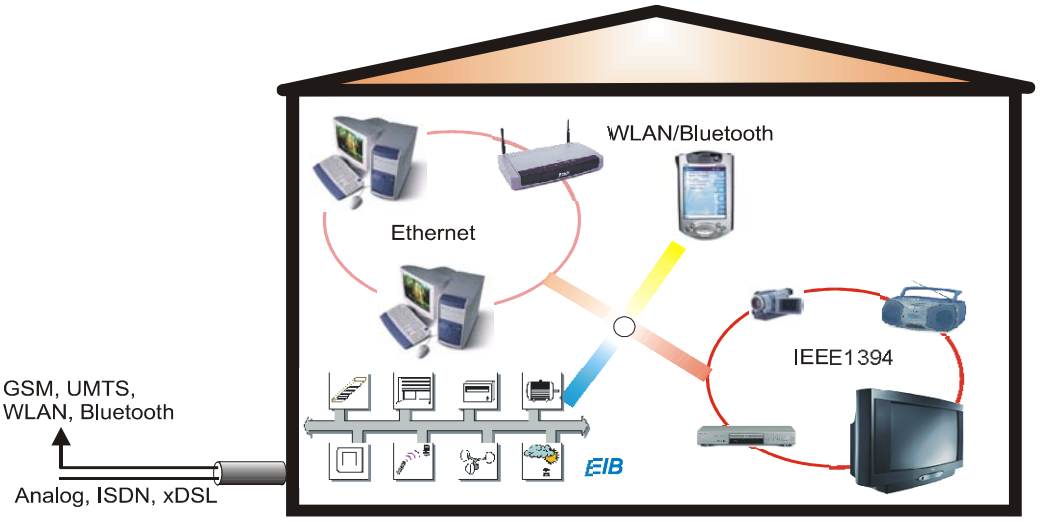

Abbildung 1.1: Bussysteme innerhalb eines Gebäudes

[3] Siehe dazu Kap. 2.2.4

In der Abbildung 1.1 ist die Struktur eines Gebäudes mit einigen der erwähnten Bussystemen dargestellt. Damit lassen sich jeweils die Einzelgewerke miteinander vernetzen, ohne jedoch Verbindungen untereinander geschaffen zu haben. Obwohl für einzelne Bussysteme Übergänge zwischen unterschiedlichen Medien vorhanden sind, fehlt ein gemeinsamer Ansatz zur Vereinheitlichung der Kommunikation. Zumeist werden die Daten bei der Übertragung nur transparent verpackt und am anderen Ende ausgewertet, ohne dass eine Abstrahierung stattgefunden hat. Die Bildung von Übergängen, die nicht nur die Tunnelung der Daten übernehmen, wie das bei Bridges üblich ist, soll im weiteren Verlauf näher beschrieben werden.
Obwohl es bereits einige Ansätze aus anderen Bereich gibt, ist bis heute noch kein geschlossener Ansatz für den Einsatz in der Gebäudeautomatisierung verfügbar.

2. Zielsetzung und Gliederung

Obwohl die vorhandenen Automatisierungsbussysteme innerhalb des Gebäudes die zentrale Rolle für die Sensor/Aktor-Kommunikation übernehmen und damit die Funktionsweise der einzelnen Gewerke innerhalb des Hauses sicherstellen, ist deren Funktion dem Anwender weitgehend verborgen. Die zusätzlichen Bussysteme stellen in diesen Zusammenhang nur einen Gewinn an Komfort dar, der durch die fehlende herstellerübergreifende Kompatibilität der Einzelsysteme innerhalb des Gebäudes jedoch in seinem Nutzen beschränkt ist.

In dieser Arbeit wird ein Ansatz entwickelt, der netzübergreifend nicht nur Übergänge zwischen unterschiedlichen Medien schafft, sondern darüber hinaus auch eine vereinheitlichte Schnittstelle anbietet. Die Anbindung und Umsetzung der Einzelsysteme auf eine gemeinsame Schnittstelle wird anhand einiger Beispiele exemplarisch erläutert. Für die medienunabhängige Bereitstellung von Diensten werden *Java Intelligent Network Infrastructure* (Jini), *Home Audio/Video Interoperability* (HAVi), *Universal Plug and Play* (UPnP) und *Multimedia Home Platform* (MHP) in Betracht gezogen. Die buseigenen Fähigkeiten werden dabei in Diensten abgebildet. Diese Dienste sind so zu gestalten, dass die besonderen Fähigkeiten der Bussysteme berücksichtigt werden, gleichzeitig aber eine gemeinsame Übereinkunft über die Nutzung der Dienste gefunden werden kann. Konzepte der Informatik, die sich in der recht jungen (Ingenieur-)Wissenschaft abzeichnen, werden hier als Basis dienen und so die dort vorgeschlagene Methodik in das Gebiet der Heimautomatisierung einbringen.

Neben der Erstellung einer einheitlichen Verwaltung der Ressourcen aus den einzelnen Netzen, die in Form von Diensten dargestellt werden, ist auch das Exportieren der Gesamtheit oder Teile der Dienste Ziel dieser Arbeit. Anhand von vorhandenen Konzepten wird eine Implementierung mittels *Open Services Gateway Initiative* (OSGi) am System demonstriert. OSGi ermöglicht nicht nur die Vereinheitlichung von Schnittstellen für Dienstanbieter, sondern auch die Festlegung des Lebenszyklus von Diensten, also deren Installation, Inbetriebnahme und Entfernung. Durch Verwendung von OSGi beschränkt sich die durchgängige Nutzung der Hausfunktionen nicht auf das Hausinnere, sondern erlaubt eine abstufbare Freigabe der Dienste für externe Nutzer. Gleichzeitig können jederzeit Dienste für die externe Nutzung installiert, gestartet, gestoppt und entfernt werden.

Sowohl die Interaktion des Nutzers mit dem System als auch die Umsetzung des Gesamtsystems wird anhand von UML-Diagrammen beschrieben und es werden Möglichkeiten erarbeitet, unter Verwendung von Quasistandards zum Austausch von Daten (Nutzung von XML) das Gesamtsystem zu konfigurieren und die Automatismen aufzuzeigen, die durch die Verwendung von Middleware erreicht werden können. Die Aufgabe von UML wird in [UML2001] so beschrieben: „The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.”. Da UML eine Definition eines Metamodells ist, kann dieses Modell wiederum

für jede Art von Systemen verwendet werden. So gibt es Arbeiten, die sich mit der Modellierung und Umsetzung von Fehlerbaumanalysen in UML [Ganesh2002] beschäftigen. Dies ist nur ein Beispiel für die Einsatzfähigkeit von UML in Bereichen außerhalb der Software-Entwicklung.
Das System soll weitgehend autonom arbeiten, also ohne Eingriff des Benutzers. Dennoch ist eine Parametrierung des Gesamtsystems zur erstmaligen Inbetriebnahme unumgänglich. Im Rahmen dieser Arbeit werden auch Werkzeuge geschaffen, mit denen diese Parametrierung erfolgen kann. Dabei wird besonderer Wert auf einen einheitlichen Datenaustausch gelegt.
Auslastungsuntersuchungen der Einzelsysteme an einem zentralen sowie auch an einem dezentralen Konzept werden Auskünfte über die Ausfallsicherheit des realisierten Systems geben.
Die erarbeiteten Konzepte werden an einer vorhandenen Forschungsplattform demonstriert und damit die Fähigkeiten eines solchen Konzepts unter Ausschluss von Laborbedingungen unter Beweis gestellt. Die dazu genutzte Doppelhaushälfte bestehend aus der Haushälfte VisionWohnen, die am Wochenende öffentlich besichtigt werden kann, und *tele*-Haus, das zu Forschungszwecken mit neuen Komponenten der Gebäudeautomatisierung ausgestattet wird, wurde 1999 in der Nähe von München von der Bauland GmbH und ihren Entwicklungspartnern errichtet und der Technischen Universität München als Forschungsplattform zur Verfügung gestellt.

3. Überblick über vergleichbare Ansätze

3.1 Wissenschaftliche Arbeiten

Die Vielzahl der Kommunikationssysteme führte in der Vergangenheit dazu, definierte Schnittstellen durch Standardisierung zu schaffen. Auf Grund der sehr unterschiedlichen Zielsetzungen der Systeme zeigte sich jedoch bald, dass dies nicht in allen Bereichen möglich ist. In den informationstechnischen Abhandlungen zeichnet sich eine Konvergenz der Übergänge auf Anwendungsebene ab. Dabei werden die Systemcharakteristiken weitgehend verdeckt, um eine einheitliche Kommunikationsbasis zu finden.
Bei der Erstellung einer einheitlichen Infrastruktur sind mehrere Teilaspekte zu beachten:

- Erstellung von Kommunikationsverbindungen
  Auf der untersten Ebene werden Verbindungen zu dem Übertragungsmedium aufgenommen, Daten entgegengenommen, weiterverarbeitet und verteilt. Dazu müssen die herstellerspezifischen Protokolle berücksichtigt und umgesetzt werden.
- Aufbereitung der Daten
  Die netzbezogenen Daten werden aufbereitet und gebündelt, um die Art des Informationsgehalts an eine netzübergreifende Struktur anzupassen und in Dienste umzuwandeln.
- Regelung des Informationsflusses
  Vor der Verwendung der Dienste müssen die geeigneten Dienstnehmer und -anbieter ermittelt werden. Dazu gehört auch die Verteilung der Ressourcen im System. Dies ist besonders bei Multimediaanwendungen zu bedenken. Werden paketorientierte Übertragungsschichten verwendet, so muss die geforderte Dienstgüte sichergestellt werden. Diese Aufgabe ist parallel zum schichtweisen Aufbau der Softwarestruktur zu sehen und erstreckt sich bis zur Kommunikationsverbindung.
- Nutzung der Dienste
  Die zur Verfügung gestellten Dienste werden von Anwendungen genutzt. Dabei gibt es in einem verteilten, dienstorientiertem System keine scharfe Trennung zwischen Dienstnehmer und Dienstanbieter, da hier Dienste wechselseitig beansprucht werden.

Ein direkter Bezug zur Gebäudeautomatisierung im Hinblick auf die Abbildung eines Gebäudebussystems in eine dienstorientierte Umgebung ist in [Dietrich2000] gegeben[4]. Dort werden Arbeiten der Technischen Universität Wien vorgestellt, die sich mit der Umsetzung einer Abbildung des EIB auf ein verteiltes Netzwerk zur Auffindung und Verwaltung von Geräteressourcen durch die Verwendung von Jini beschäftigen. Dazu werden einzelne EIB- Geräte in Funktionsgruppen gegliedert und dann als Dienste in ein Netzwerk überführt. Es werden Szenarien vorgestellt, in denen die Dienste, die aus der EIB-Abbildung hervorgehen, mit anderen Teilnehmern interagieren.
In [Kellerer2001] wird ein Vorgehen zur Nutzung von Diensten aus dem Bereich *Telekommunikation, Information, Medien und Entertainment* (TIME) beschrieben. Die dort

[4] Siehe dazu S. 297-302 in [Dietrich 2000].

vorgestellte Dienststeuerplattform verwaltet zuvor definierte Dienstarten und soll „[...] höhere Dienste erbringen, die über die Funktionalität der einzelnen Netze hinausgehen. “[Kellerer2001]. Der Schwerpunkt dieser Arbeit wird eindeutig auf die Netzunabhängigkeit gelegt, ohne im Einzelnen auf die umgesetzten Dienste einzugehen. Dazu werden die Anwendungen durch Verwendung von Komponenten für den Teilnehmerzugang, Kommunikationsbeziehungen und Netzressourcen von der Dienststeuerung getrennt. Ein zentrales Element der Arbeit sind die Adaptoren, Komponenten, die die netzabhängige Signalisierung auf eine übergeordnete Architektur abbilden. Diese Adaptoren sind stark an den Telekommunikationsbereich angelehnt und verwenden zur Kommunikation eine Abwandlung des *Session Initiation Protocol* (SIP) der *Internet Engineering Task Force* (IETF)[5].

Zu der Übertragung von Daten, besonders bei großen, zeitkritischen Datenaufkommen, gehört eine Regelung der Dienstgüte. Dies wird in den Protokollen zur Datenübertragung bereits festgelegt, so beispielsweise bei H.323[6]. Da jedoch in der Regel die Anwendung direkt auf dem Protokoll aufbaut, muss bei Zwischenschaltung von Verwaltungsebenen eine Möglichkeit zur Lastregelung über diese Schichten hinaus gegeben sein. In einem anderen Kontext, der aber im Allgemeinen durchaus auf diesen Fall zu beziehen ist, wurden in [Groh1998] Methoden aufgezeigt, um die Ressourcen eines verteilten Prozesses möglichst gleich zu verteilen.

Während sich die oben besprochene Arbeit mit der Abstraktion der Netzwerk- und Kommunikationsschnittstellen beschäftigt, wird in der Arbeit von [Nöhmeier1998] der Einsatz von Agenten[7] für die automatisierte Informationssuche von Daten im Internet beschrieben. Die Ergebnisse sind im Bereich der *Computer Supported Cooperative Work* (CSCW) anzusiedeln, der in [Nöhmeier1998] so eingeführt wird: „Als der Begriff CSCW in den frühen 80er Jahren [...] geprägt wurde, verstand man darunter zunächst die Grundlagen und Methodologien zum Verständnis von Gruppenarbeit und deren Unterstützung durch Computer“. [Nöhmeier1998] führt Methoden zur Nutzung von Agenten im Umfeld der Internetdatenbanken und Informationsdiensten ein, die Anwender im Zusammentragen von Informationen und damit ein kooperatives Arbeiten unterstützen sollen. Genau wie das Internet eine verteilte Informationsquelle darstellt, so kann auch die Verknüpfung mehrere Subsysteme und deren bereitgestellten Dienste als solche angesehen werden. Insofern kann der dort gemachte Ansatz in dieser Arbeit auf die Gebäudeautomatisierung erweitert werden.

3.2 Automatisierungstechnische Lösungen

Die zur Zeit existierenden Lösungen weisen noch keinen in sich geschlossenen Ansatz auf. Dies ist auch in absehbarer Zeit kaum möglich, da es keine Standardmethode gibt, verschiedene Systeme miteinander zu koppeln.

[5] Siehe dazu [IETFRFC3261].

[6] Siehe dazu [ITUH.323].

[7] Siehe dazu Kap. 4.2.4

Es kann eine grobe Unterteilung in verschiedene Anwendungsgebiete unternommen werden. Dabei kommt eine Strömung aus der Automatisierungstechnik, etwas spezieller aus der Netzwerktechnik und im weiteren Umfeld aus dem Telekommunikationssektor, bei der es ähnliche Bestrebungen gibt, eine netzübergreifende Nachrichtenübermittlung zu errichten.

Im Bereich der Netzwerkkommunikation gibt es Ansätze, die unter dem Begriff *Residential Gateway* geführt werden. Die meisten Hersteller verstehen darunter Bridges oder Router. So ist es durchaus üblich, einen WLAN Access Point oder DSL-Router als Residential Gateway zu bezeichnen. Außerhalb der Netzwerktechnik wird dieser Begriff nicht verwendet. Am besten lässt er sich in das Umfeld von verteilten Systemen oder verteilten lokalen Netzen einordnen. Eine Definition dieses Begriffs fällt schwer, da er zwar in unterschiedlichen Zusammenhängen genannt wird, jedoch im Allgemeinen ein Gerät zur Verbindung oder Protokollumsetzung unterschiedlicher Netztopologien und -systeme ist, wobei sich die Umsetzung zumeist auf eine Schnittstelle zwischen zwei Systemsegmenten beschränkt.

In der Automatisierungstechnik wurde mit unterschiedlichen Ansätzen der Versuch einer Vereinheitlichung, besonders im Feldbusbereich, unternommen, der sich jedoch bis auf wenige Ausnahmen auf die untersten Übertragungsschichten beschränkt hat. So wurden Teile von *Process Field Bus* (PROFIBUS) in unterschiedlichen Normen spezifiziert (DIN 19245 Teil 1-4, EN 50170-2, IEC 61158).

Unter dem Namen *Interface for Distributed Automation* (IDA) wird eine Verbindung unterschiedlicher Bussysteme mit Hilfe von Ethernet/TCP/UPD-IP angestrebt. IDA verwendet zur Kommunikation neben den oben genannten Standardprotokollen auch HTTP, FTP und SMNP zur Nachrichtenübermittlung. Auf Basis dieser Protokolle werden durch [IDA2001] Schnittstellen für den (Echtzeit-)Datenzugriff sowie eine Beschreibungssprache (*Extensible Device Description Markup L*anguage – XDDML) aufbauend auf XML zur systemunabhängigen Darstellung von Geräteinformationen definiert, ohne dabei die Anwendungen zu spezifizieren. Die interne Daten- und Dienstrepräsentation wird durch die IDA objects näher beschrieben. Dazu gehört die Vorgabe von Datentypen, Daten- und Dienstobjekten, auf die über *Application Programming Interface* (API) zugegriffen werden kann.

Die von der IDA Group erarbeiteten Vorschläge werden zusammen mit der *Open DeviceNet Vendors Association*[8] (ODVA), der *Industrial Automation Open Networking Association* (IAONA) und seit April 2002 auch mit der PROFIBUS Nutzerorganisation abgestimmt[9].

Die in [IDA2001] enthaltene Middleware zur Echtzeitkommunikation bietet Dienste zur Verteilung von Daten, Benachrichtigungsdienste (*Event Notifications*) und Fernaufruf-

[8] DeviceNet ist ein offenes System von Rockwell Automation und stellt eine Schicht 7 Implementierung von CAN dar. Mit DeviceNet ist ein Multi-Master/Slave Betrieb für den Zusammenschluss von Sensoren und Aktoren möglich.

[9] Pressemitteilung vom 16.04.2002.

methoden (*Remote Method Invocation*) an. Sie dient jedoch nicht dazu, Geräte selbstständig aufzufinden oder aber neue Dienste lokalisieren zu können.

3.3 Standardisierung von Heimautomatisierungssystemen

Neben dem wissenschaftlichen Interesse an der Entwicklung neuer Methoden zur Vernetzung unterschiedlicher Bussysteme, beschäftigt sich ein internationales Komitee, zusammengesetzt aus Mitgliedern der *International Organization for Standardization* (ISO) und der *International Electrotechnical Commission* (IEC) mit der Standardisierung von busübergreifender Kommunikation unter dem Arbeitstitel „Home Electronic System". Abgesehen von der Schaffung einheitlicher Schnittstellen, die eine Interoperabilität von Produkten verschiedener Hersteller sicherstellen soll, geht ein Teil dieser Bemühungen auch in Richtung Bildung eines Residential Gateways. Die Arbeitsgruppe, die sich mit diesem Thema beschäftigt, wird unter dem Titel „CD1 15045-01: Information technology - Interconnection of information technology equipment - Architecture for HomeGate, the residential gateway (AHRG)" in einer Arbeitsgruppe zusammengefasst. Bisher wurden mehrere Entwürfe ausgearbeitet, die jedoch in sehr großen zeitlichen Abständen vorgestellt wurden und daher der Gefahr ausgesetzt sind, die aktuellen Entwicklungen nicht mehr berücksichtigen zu können. Daher wurde innerhalb der Arbeitsgruppe nach Methoden gesucht, auch die neuesten Techniken einbeziehen zu können. Dies wird durch geschichtete Anforderungsbeschreibungen gelöst, wobei die Bedürfnisse, die Funktionalität, das Gesamtkonzept und die Semantik in den oberen Schichten statisch bleiben sollen, während die Protokolle, die Datenformate und die APIs in den untersten Schichten angeordnet sind und einem dynamischen Wandlungsprozess unterliegen dürfen [Farance2001]. Bisher wurden Entwürfe erarbeitet, die aber in diesem Stadium noch nicht vollständig sind.
Insgesamt ist das Vorhaben in seinem Umfang sehr weitreichend, sowohl hinsichtlich der Anzahl der eingeschlossenen Bussysteme, als auch hinsichtlich der Erstellung der Spezifikationen für Software-Schnittstellen. Es bleibt abzuwarten, ob der Standard nach Fertigstellung in die Produkte der Hersteller einfließen wird.

3.4 Bussysteme und deren Bedeutung in der Heimautomatisierung

3.4.1 Vorhandene Bussysteme

Die in der Heimautomatisierung eingesetzten Bussysteme sollen hier nur kurz erläutert und in ihrer Bedeutung für den Anwender sowie in ihrer Verwendbarkeit kategorisiert werden. Bussysteme werden meistens im Hinblick auf ihre Verwendbarkeit für den jeweiligen Anwendungsfall entwickelt. Damit vermeidet man die Entwicklung von unwirtschaftlichen Einzelkomponenten, die durch ihre Universalität nur schwer an die jeweilige Anwendung anpassbar sind und zudem für den Markt auf Grund ihres hohen Preises nicht geeignet wären.

Daraus ergeben sich spezialisierte Bussysteme für die jeweiligen Marktsegmente. Ein großer Bereich, der an dieser Stelle in der Mess- und Automatisierungstechnik zu nennen ist, sind die Feldbussysteme. In der Vielzahl der angebotenen Systeme (ASI, BITBUS, PROFIBUS, INTERBUS, CAN) zeigt sich die starke Spezialisierung der Bussysteme auf die jeweilige Anwendung, wobei sich auch hier die Marktsegmente mehrerer Systeme überschneiden. Im Folgenden wird nur eine Auswahl der Bussysteme vorgestellt, die in der Heimautomatisierung Anwendung finden. Alle vorgestellten Bussysteme besitzen die Fähigkeit sowohl Punkt-zu-Punkt- als auch Punkt-zu-Multipunkt-Verbindungen zu erstellen, wodurch sie in den dezentralen Ansatz des mehrstufigen Systems integrierbar sind, das in dieser Arbeit entwickelt wird.
Für die Bildung von Untersystemübergängen, die in der vorliegenden Arbeit angestrebt werden, ist die Abstrahierung der Informationen unabdingbar. Eine Modellierung der einzelnen Bussysteme ist zu umfangreich – es ist nur bedingt notwendig, einen solchen Abstraktionsgrad anzusetzen. Dennoch müssen die Datenmodelle bei ihrer Integration berücksichtigt werden, damit ein kommunikationsfähiges Gesamtsystem entstehen kann.

3.4.1.1 Europäischer Installationsbus

Der *Europäische Installationsbus* (EIB) ist ein offenes System, das durch die 1990 erfolgte Gründung einer Dachorganisation – der EIB Association –, Akzeptanz durch eine große Anzahl von Herstellern gewann [EIBA1999]. Der EIB dient der Verknüpfung von Sensoren und Aktoren innerhalb eines Gebäudes über ein gemeinsames Bussystem. Dabei stehen zur Verbindung der Komponenten unterschiedliche Medien zur Verfügung: Powerline für die nachträgliche Nachrüstung, ein Funksystem und die konventionelle Twisted Pair Verkabelung.
Bei Twisted Pair wird zur seriellen Übertragung der Informationen ein geschirmtes, verdrilltes Zweidrahtkabel mit Kodierung durch Differenzspannungen verwendet. Die Übertragungsrate beträgt dabei 9,6 kBit/s bei Verwendung von CSMA/CA als Zugriffsverfahren. Die Informationen werden einer Gleichspannung von 20-30 V überlagert, die der Versorgung der Teilnehmer dient. Die angeschlossenen Teilnehmer bestehen – unabhängig vom Übertragungsmedium – in der Regel aus zwei Komponenten:

- Busankoppler (Bus Coupling Unit)
  Über den Busankoppler wird die Hilfsenergie zur Versorgung des Teilnehmers entnommen. Gleichzeitig werden die Telegramme verarbeitet und an das Anwendungsmodul weitergegeben. Die Steuerung des Busankopplers übernimmt ein maskenprogrammierter Motorola Mikrocontroller der Serie 68HC05. Einige I/O Leitungen des Mikrocontrollers sind zu einer Schnittstelle geführt – dem Periphal External Interface. An dieses wird das Anwendungsmodul angeschlossen.
- Anwendungsmodul
  Für einfache Anwendungen kann beispielsweise ein Zweifachtaster oder eine LC-Anzeige dienen. Das Busankopplungsmodul erkennt das angeschlossene Anwendungsmodul über den Eingang des AD-Wandlers des Mikrocontrollers. Nach

[EIBA1999] sind 20 Typen definiert, die vom einfachen Taster bis zu einer asynchronen seriellen Schnittstelle reichen.

Der Busankoppler entspricht im Kommunikationsprotokoll dem Aufbau nach dem ISO/OSI-Referenzmodell, wobei die Schichten 1-4 und 7 implementiert sind.
Durch die lange Existenz des EIB auf dem Markt gibt es bereits eine Vielzahl von Komponenten, die über die Lichtsteuerung hinausgehen. So gibt es Anwendungen im Bereich der Heizungs- und Klimatechnik wie auch sicherheitsrelevante Lösungen.
Im Rahmen des Convergence Prozess wurden die drei Systeme BatiBUS, EHS und der EIB zu einem gemeinsamen Standard vereint (KNX). Auch dafür wird es in Zukunft eine Dachorganisation geben, nämlich die Konnex Association [Weinzierl2001].

3.4.1.2 LonWorks

Dieses Feldbussystem wurde von der Firma Echelon entwickelt und ist ebenfalls ein serielles Bussystem, das mehrere Übertragungsmedien unterstützt, darunter Zweidrahtleitung, Powerline, Funk, Infrarot und Lichtwellenleiter. Ähnlich wie beim EIB werden die Applikationen durch einen medienabhängigen Empfänger/Sender (Transceiver) mit dem Übertragungsmedium verbunden, der die Aufgabe übernimmt, die Daten der Anwendung unabhängig vom Medium zu vermitteln. LonWorks verwendet alle sieben Schichten des ISO/OSI Modells. Für die Datensicherung werden 16 Bit Fehlerpolynome berechnet.
In [Echelon2001] wird bei Powerline Transceivern eine Übertragungsrate von 5,4 kBit/s, bei Twisted Pair Transceivern von bis zu 1,5 MBit/s angegeben. Für die anderen Medien werden keine Angaben gemacht. Je nach Topologie werden Netzabschlüsse benötigt. Es gibt eine Vielzahl von Medienübergängen und Routern, daher ähnelt LonWorks eher einem LAN. Es werden fertige Lösungen für einfache Steuerungs- und Kontrollaufgaben wie digitale Ein- und Ausgänge angeboten. Daneben können mit einem von Echelon erhältlichen Entwicklungssystem auch eigene Anwendungen erstellt werden, die auf den Transceivern aufsetzen.

3.4.1.3 IEEE1394

Zur drahtgebundenen Datenübertragung in Multimedianetzen wurde bereits 1995 ein Standard vom *Institute of Electrical and Electronics Engineers* (IEEE) geschaffen [IEEE1394-1995] , der eine Gesamtkapazität von bis zu 400 MBit/s bietet. Die mit einem weltweit einmaligem Identifizierungsmerkmal – die *Global Unique ID* (GUID) – versehenen Teilnehmer werden in einer Baumstruktur angeschlossen und können sowohl über Punkt-zu-Punkt-Verbindungen als auch in einer Punkt-zu-Multipunkt-Verbindung erreicht werden. Die Arbitrierung des Busses wird durch einen Root vorgenommen, der nach dem Bus-Reset – eine Nachricht, die nach dem Initialisieren der Geräte oder beim Anschluss eines neuen Gerätes ausgesandt wird – bestimmt wird.
Ähnlich wie bei Bluetooth können die Daten in einem asynchronen, gesicherten Verbindungsmodus oder aber in einer isochronen, ungesicherten Verbindung übermittelt werden. Damit eröffnet sich auch hier die Möglichkeit, hochratige Multimediadienste

innerhalb des Netzwerks anzubieten. Dazu wird nach einem Zeitmultiplexverfahren der Bus in mehrere Abschnitte eingeteilt, wobei ein Anteil von höchstens 80% zur Übertragung von isochronen Daten reserviert werden kann [Anderson1999]. Die Zuteilung erfolgt durch einen ausgewählten Teilnehmer, der bei Ausfall dynamisch gewechselt wird. Der verbleibende Zeitschlitz wird zur Übertragung von asynchronen Nachrichten verwendet.
Neben Verbesserungen, die im Laufe der Zeit zu Nachträgen des Standards geführt haben, wurde die Notwendigkeit einer größeren Bandbreite in [IEEE1394B-2002] abgedeckt. Neben dem in [IEEE1394-1995] aufgeführten Kabel wurden weitere physikalische Medien wie *Polymer Optical Fiber* (POF) oder aber *Glass Optical Fiber* (GOF) eingeführt. Damit kann über IEEE1394 eine maximale Datenrate von 3,2 GBit/s erreicht werden.
In [IEEE1394-1995] werden nur die unteren Übertragungsschichten definiert. Für die Anwendungsseite gibt es bereits Normen zur Übermittlung von Daten beziehungsweise Befehlen zwischen Audio-/Videogeräten [IEC61883], die die Audio- und Videodatenformate sowie Formate für die Steuerung von Multimediageräten festlegen.
Vergleichbar mit Bluetooth ist auch in diesem Standard das Auffinden und Bekanntmachen von neu angeschlossenen Geräten bereits im Protokoll vorgesehen, die sofort identifiziert werden und bei der Anwendung vorgegebene Aktionen auslösen.

3.4.1.4 Ethernet

Ethernet wurde durch die Norm IEEE802.3 mit seinen Zugriffsmethoden und Übertragungsraten spezifiziert. Zunächst war IEEE802.3 noch auf Koaxialkabel mit einer Übertragungsrate von 10 MBit/s beschränkt. Mittlerweile wurde das Gigabit Ethernet eingeführt, das einen maximalen Datendurchsatz von 1 Gigabit/s zulässt.
Während dieses System ursprünglich nur in der Bürokommunikation genutzt wurde, hat sich die Verwendung im Laufe der Zeit auch auf die Automatisierungstechnik ausgebreitet. Der hierarchische Aufbau eines Automatisierungssystems [Furrer2000] in Leitebene, Steuerungsebene und Feldebene siedelte Ethernet nur in der Leitebene an. Bei den heutigen Lösungen werden jedoch bereits Teile der Feldebene durch Ethernet verbunden, um mehrere Segmente miteinander zu koppeln. Dies findet direkt in der Steuerungsebene statt, ohne Verletzung der Echtzeitfähigkeit. Nach [Furrer2000] gibt es bereits Bestrebungen, die Bussysteme der Feldebene ebenfalls durch Ethernet zu ersetzen und unter Verwendung von Middleware miteinander zu koppeln.
Die breite Anerkennung des Ethernet im industriellen Sektor ist besonders auf seine weite Verbreitung in Verbindung mit TCP/UDP-IP zurückzuführen. Dennoch müssen gewisse Vorkehrungen getroffen werden, da Ethernet nicht von sich aus echtzeitfähig ist. Aufgrund des verwendeten Zugriffsverfahrens[10] CSMA/CD kann keine deterministische Aussage über Antwortzeiten beim Zugriff auf den Bus gemacht werden. Damit müssen in den verwendeten übergeordneten Protokollen Mechanismen vorgesehen werden, die die Echtzeitfähigkeit mit Einschränkungen garantieren können.

[10] Siehe dazu [IEEE802.3].

Ähnliche Voraussetzungen wie in der Automatisierungstechnik gelten auch für die Multimediaanwendungen. Dort steht die Übertragung isochroner Daten im Vordergrund, die ebenfalls mit Ethernet mit Hilfe von Zusatzprotokollen durchgeführt werden kann. Damit stellt Ethernet eine Grundlage für die vielfältigen Anwendungsarten zur Verfügung.

3.4.1.5 Wireless LAN

In einem ähnlichen Zusammenhang wie Ethernet ist auch Wireless LAN (WLAN) zu sehen. Dort gilt es ebenfalls, paketorientierte Daten zu übermitteln, wobei die Daten drahtlos zwischen den Kommunikationsteilnehmern versendet werden.
WLAN stellt also den drahtlosen Ersatz zu Ethernet dar. Es ist in den Standardisierungsvorschriften auf der gleichen Ebene angesiedelt und zwar unter IEEE 802.11. In [IEEE802.11] werden die physikalischen Eigenschaften des Übertragungsmediums (Sendefrequenz bei 2,4 GHz, gleiches Frequenzband wie bei Bluetooth) sowie die Zugriffsverfahren beschrieben.
War die Übertragungsrate anfangs auf 2 MBit/s beschränkt, ist sie im neuen Standard, dem IEEE 802.11b, bereits mit 11 MBit/s angegeben.[11] Prinzipiell werden zwei Arten der Netzwerkbildung unterschieden: Eine Peer-to-Peer Verbindung ohne Vermittler oder aber ein Übergang zu einem anderen Netzwerk in Form von Bridges, den so genannten Access Points. Für den Übergang Ethernet auf WLAN gibt es bereits mehrere kommerzielle Lösungen, die in der Heimautomatisierung kostengünstig eingesetzt werden können.

3.4.1.6 Digital Enhanced Cordless Telecommunication

Digital Enhanced Cordless Telecommunication (DECT) ist nach [ETSI300-1] standardisiert. Die Hauptaufgabe dieser Spezifikation besteht in der drahtlosen Übermittlung von digital kodierter Sprache. Auch dieses Kommunikationsprotokoll ist geschichtet aufgebaut und stellt neben einer Spezifikation der physikalischen Übertragungsparameter und Kodierung eine Medium Access Control sowie eine Vermittlungsschicht zur Verfügung. DECT arbeitet mit einer Sendefrequenz zwischen 1880 MHz – 1900 MHz[12] bei Einteilung dieses Bereichs in zehn Frequenzkanäle (FDMA) mit einem Kanalabstand von 1,728 MHz. Zusätzlich werden Zeitrahmen vorgegeben (TDMA), die aus je 24 Slots (12 Uplink, 12 Downlink) zu 480 Bits bei einer Symbolrate von 1152 kBit/s (damit 10 ms je Zyklus) bestehen. Dabei können Zeitrahmen auch zusammengelegt bzw. nur teilweise genutzt werden [ETSI300-2].
In der *Medium Access Control Layer* (MACL) werden die Datenrahmen zur Übertragung von Steuer- und Nutzdaten definiert. Für eine Datenverbindung stehen bei einem verbindungslosen Service 32 kBit/s je Zeitscheibe zur Verfügung [ETSI300-4]. Der Zugriff auf das Datenübertragungsmedium ist zentral geregelt und wird von einer

[11] Von der ETSI gibt es bereits Spezifikation für eine Übertragung im 5 GHz Band mit Datenraten von 20 MBit/s. Vgl. dazu [EN300652]. Eine weitere Spezifikation ist zur Zeit noch nicht vollständig (Stand 28.10.02): HIPERLAN/2 mit 54 Mbit/s.

[12] Die genaue Mittenfrequenz eines Kanals ergibt sich aus: Fc = F0 – c * 1,728 MHz (F0 = 1897.344 MHz;c = 0...9) [ETS300-2].

Basisstation (Fixed Part) übernommen, die mehrere Mobilteile (Portable Part) verwalten kann.
Auf Grund der sensiblen Daten – es wurde ursprünglich zur Übermittlung von Telefongesprächen innerhalb eines Nahbereichs entworfen – wurden Algorithmen zur Verschlüsselung der gesendeten Daten und der Authentifizierung der beteiligten Kommunikationspartner eingebunden [ETSI300-7]. Die höheren Protokollschichten übernehmen die Kapselung der Anwendungsdaten. Diese werden durch so genannte Profile definiert, die die Art der Anwendung und den Dateninhalt näher spezifizieren.
Mittlerweile wurde mit der Fortführung und Anpassung von DECT ein neuer Standard geschaffen, der nun neben reinen Audiodaten auch für die Übertragung von Multimediadaten [ETSIDPRS] geeignet ist. Weitere firmenspezifische Lösungen zur Übermittlung von Daten liegen ebenfalls vor. Dabei sind die übertragenen Daten nicht weiter spezifiziert, d.h. es wird keine Vorgabe über den Inhalt des gesendeten Datenstroms getroffen. Daher ist es nur möglich, für den speziellen Einzelfall ein Datenmodell zu entwerfen. In dieser Arbeit wird DECT nur für den Transport von übergeordneten Protokollen verwendet.

3.4.1.7 Bluetooth

Während DECT sich anfangs ausschließlich zum Ziel setzte, in der Telefonie eingesetzt zu werden, wurde Bluetooth weitgehend universell einsetzbar gestaltet. Die Hauptanwendung zielte auf den Einsatz zur einfachen Verbindung von mobilen Geräten unter Berücksichtigung eines geringen Konfigurationsaufwands und des geringen Stromverbrauchs. Die Nettoübertragungsrate beträgt im asymmetrischen Betrieb maximal 723,2 kBit/s bei 57,6 kBit/s für den Rückkanal. Im symmetrischen Vollduplexbetrieb beträgt die maximale Übertragungsrate 433,9 kBit/s [Williams2001]. Dadurch ist Bluetooth auch zur Übertragung von Sprache und in begrenztem Umfang sogar zur Übertragung von Videodaten geeignet. In [Williams2001] werden die unteren Schichten spezifiziert, wobei - ähnlich wie bei DECT - eine weitere Spezifikation zur Beschreibung von Profilen existiert. Dort werden die Anwendungsfälle (Sprachübertragung, Interkom-Dienste, Schnittstellenersatz, Protokollkapselung von TCP/IP etc.) beschrieben.
Ein großer Vorteil des Bluetooth gegenüber DECT sind die Mechanismen für das so genannte Service Discovery, das dem Auffinden von anderen Bluetooth-Teilnehmern dient, wobei bereits in diesem Status die Art der angebotenen Dienste näher beschrieben werden kann. Damit ergibt sich die Möglichkeit zur ad hoc Bildung von Kleinnetzwerken, die in einem Piconet[13] sieben Slaves und einen Master zusammenschließen. Diese Netze können auf Scatternets[14] erweitert werden, bei denen ein Teilnehmer als Brücke zu dem anderen Netzwerk dient.

[13] *Piconet* ist eine Bezeichnung aus [Williams2001] für den Zusammenschluss von acht Teilnehmern nach der oben beschriebenen Struktur.

[14] Im so genannten *Scatternet* werden mehrere Piconets zusammengeschlossen. Dabei kann ein Slave gleichzeitig mit mehreren Piconets verbunden sein. Ein Master kann jedoch nur innerhalb eine Piconets sichtbar sein, da er die Hopping-Sequenz für das Frequenzsprungverfahren vorgibt, die einmalig für jedes

Die Datenübertragung kann in verschiedenen Modi erfolgen, wobei ein asymmetrischer, ein symmetrischer sowie ein isochroner und asynchroner Betrieb möglich ist.

3.4.2 Analyse und Kategorisierung der Bussysteme

Die vorgestellten Systeme erlauben keinen unmittelbaren Vergleich untereinander, dennoch soll in der nachfolgenden Tabelle kurz eine Einstufung anhand der wichtigsten Kriterien (physikalische Schnittstelle, Bandbreite, Kommunikationsarten, Einsatzgebiet) unternommen werden.

Tabelle 3.1: Übersicht über einige Bussysteme

Bezeichnung	Übertragungs-medien	Übertragungsrate	Max. Leitungslänge/ Reichweite	Verbindungsarten
EIB	TP, PL, RF	9,6 kBit/s	1000 m (TP)	asynchron
LonWorks	TP, PL, RF, IR	5,4 kBit/s (PL) bis 1,5 MBit/s (TP)	2700 m (TP)	asynchron
IEEE 1394-1995 IEEE 1394B-2002	STP GOF, POF,UTP-5	400 MBit/s bis 3,2 GBit/s	4,5 m max. 16 Hops max. 100 m[15]	asynchron, isochron
IEEE 802.3	Koax, TP	bis 1 GBit/s	max. 2000 m[16]	asynchron, paketorientiert
IEEE 802.11b	RF	11 MBit/s	100 m (100 mW)	asynchron, paketorientiert
Bluetooth	RF	768,2 kBit/s[17] 64 kBit/s (isochron)	150 m (100 mW)	asynchron, isochron
DECT DECT DPRS	RF RF	32 kBit/s[18] (isochron) max. 736 kBit/s[19]	300 m (250 mW)	asynchron, isochron

In der Tabelle lässt sich gut erkennen, dass außer EIB und LonWorks alle aufgeführten Bussysteme in der Lage sind, Multimediadaten, also ungesicherte isochrone Datenströme, zu übertragen. Bei den paketorientierten Protokollen sind dabei durchaus zusätzliche Maßnahmen in den höheren Protokollschichten vorzusehen. Durch die maximal erreichbare Datenrate eignen sich manche nur bedingt für einen Videodatenstrom, können

Piconet ist. Ansonsten wäre keine ungestörte Übertragung innerhalb eines Piconets möglich. Siehe dazu [Williams2001].

[15] 100 m bei UTP-5 und Modus S100 (100 MBit/s); 100 m bei GOF und S1600 – 50µm MMF (830 - 860 nm Wellenlänge) nach [IEEE1394-1995b]

[16] nach [IEEE802.3] : 10BASE5 : 500 m; 10BASE-FB : 2000 m.

[17] Rückkanal 57,6 kBit/s, asymmetrisch, Vollduplex.

[18] je Zeitrahmen

[19] 23 Slots zusammengefasst. Rückkanal 32 kBit/s, asymmetrisch, Vollduplex.

jedoch in der Regel gut Audiodaten übertragen. Aus den systemimmanenten Eigenschaften ergeben sich später die zu definierenden Dienste.
Die von den Systemen zu übertragenden Daten können auf unterster Kommunikationsebene in einer sehr starken Vereinfachung von ihrem zu Grunde liegenden Datenmodell entkoppelt werden. Dieser Schritt ist notwendig, um die in allen Systemen vorhandenen Gemeinsamkeiten herauszuarbeiten, damit ein möglichst einheitliches Entwurfskonzept erstellt werden kann.
Der Hauptunterschied zwischen den diversen Datenformaten lässt sich auf ihre zeitlichen Beschränkungen und die Datensicherheit zurückführen. Bei vielen Daten sind Schwankungen in der Übertragungszeit durchaus erlaubt, ohne dass die übermittelte Information ungültig wird. Als Beispiel dafür kann ein *Address Resolution Protocol* (ARP) Request auf Ethernet-Basis angesehen werden. Das ARP ist für die Adressauflösung innerhalb von IP zuständig und dient der Zuweisung von IP-Adressen und der Auflösung zwischen der IP-Adresse und der Adresse des untergeordneten Protokolls. Das IP ist nicht zwingend auf Ethernet angewiesen, dennoch hat sich Ethernet bei LANs durchgesetzt. Soll die zugehörige Ethernet-MAC-Adresse für eine bestimmte IP gefunden werden, so wird zunächst ein (Ethernet-) Broadcast-Telegramm mit der MAC-Zieladresse FF:FF:FF:FF:FF:FF und der gesuchten IP-Adresse ausgesendet. Dieses Telegramm wird von allen Empfängern ausgewertet und bei Übereinstimmung der eigenen mit der mitgesendeten IP- die dazugehörige MAC-Adresse zurückgesendet. Auch wenn ARP in den unteren Kommunikationsschichten anzusiedeln ist, zeigt dieses Beispiel ein wechselseitiges, bestätigtes Protokoll.

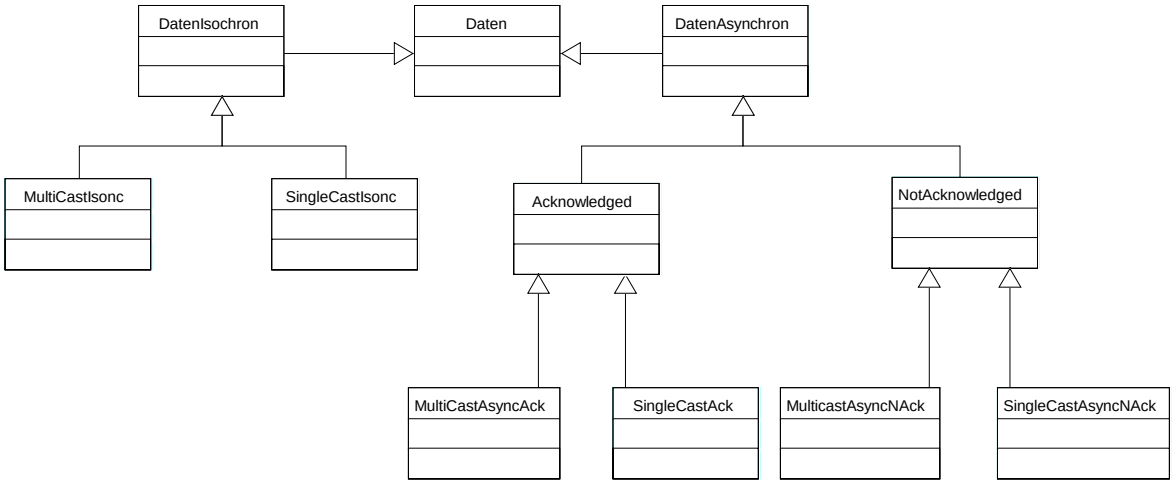

Abbildung 3.1: Paketformate in der Bustechnik

Die verwendete Einteilung der Bussysteme nach Abb. 3.1 zeigt nicht explizit die Verwendung von Broadcast-Nachrichten, da diese auf vergleichbare Methoden wie die Multicast-Telegramme aufsetzen. Ansonsten findet eine Einteilung in zwei Hauptgruppen statt: in die isochronen und die asynchronen Nachrichten. Bei den isochronen Nachrichten ist in den hier vorgestellten Bussystemen keine bestätigte Verbindung vorgesehen und

fehlerhafte Telegramme werden über Redundanz rekonstruiert oder aber verworfen, ohne eine Wiederholung anzufordern.
Bei der Datenübermittlung hat sich eine Vielzahl von Buszugriffsmethoden etabliert, die ganz entscheidend Einfluss auf die Verfügbarkeit eines Mediums haben. Die einfachste Zugriffsmethode ist der Master/Slave-Betrieb, bei dem ein Master zentral die Zugriffserlaubnis für den Start der Datenübertragung verteilt. Eine Abwandlung davon ist der Multi-Master-Betrieb, bei dem mehrere bevorzugte Busteilnehmer die Busarbitrierung vornehmen können.
Durch das verwendete Bussystem lassen sich bereits Rückschlüsse auf seine Echtzeitfähigkeit bei bestimmten Anwendungen ziehen. Nachfolgend soll dies für die bereits aufgezählten Bussysteme erläutert werden.

3.4.2.1 EIB

Beim EIB/KNX wird *Carrier Sense Multiple Access / Collision Avoidance* (CSMA/CA) für die Busarbitrierung verwendet. Zu diesem Zweck ist bei der Twisted Pair Variante ein dominanter Zustand auf dem Bus definiert. Eine logische Null setzt sich bei gleichzeitiger Überlagerung mit einer logischen Eins am Bus durch. Durch den Vergleich zwischen den gesendeten und den empfangenen Daten kann eine Kollision erkannt werden. In diesem Fall zieht sich der Knoten mit dem festgestellten Übertragungsfehler zurück und gestattet dem Teilnehmer, der eine logische Null gesendet hat, mit der Datenversendung fortzufahren. Diese Kollisionserkennung wird während des gesamten Telegramms beibehalten.[20]
Nach dem Versenden der Quelladresse der Nachricht sollte bei einem richtig konfigurierten EIB die Arbitrierung vollzogen sein – jede physikalische Adresse bezeichnet eindeutig ein Gerät in der gesamten Installation. Bei der Anbindung durch Powerline kann nicht mehr alleine durch die Quelladresse entschieden werden, zu welchem Bereich ein Teilnehmer gehört, da sich unter Umständen mehrere Installationen vermischen können. Daher wird dort zusätzlich noch eine *Domain Address* zu Hilfe genommen, um den ungestörten Betrieb mehrerer EIB-Installation zu garantieren. Logisch entspricht dies einer Erweiterung des Adressbereichs.
Die beim EIB festgelegten Zeitrahmen regeln nur die zeitliche Abfolge bestimmter Telegrammtypen. So werden minimal einzuhaltende Zeiten zwischen den einzelnen Telegrammen spezifiziert, wobei dies weitgehend den Buszugriff (Busfreizeiten) betrifft. Dadurch können keine Aussagen über die Belegzeit des Busses gemacht werden, womit sich der EIB nicht zur Übertragung isochroner Daten eignet.

[20] Diese Art der Kollisionserkennung funktioniert bei Powerline nicht mehr, da dort die Informationen frequenzmoduliert werden und eine Überlagerung von zwei Bitkodierungen eine Schwebung ergibt, die nicht durch den Empfänger erkannt wird. Daher werden dort andere Methoden eingesetzt, um Kollisionen von vornherein zu vermeiden.

Da der EIB zur Sensor/Aktor-Kommunikation entwickelt wurde, ist auch das Datenmodell mit dem eines Feldbussystems vergleichbar [21]. Der Zustand der Sensoren und Aktoren wird in Systemvariablen abgebildet, die verteilt in jedem Busankoppler verwaltet werden. Dadurch kann ein Verbund von mehreren Busteilnehmern die gleiche Variable verwalten. Dabei ist im Regelfall sichergestellt, dass alle Teilnehmer den gleichen Zustand für die jeweilige Variable vermerken. Nur im Fehlerfall, der beispielsweise durch den Ausfall eines beteiligten Teilnehmers oder aber durch eine Störung der Kommunikation hervorgerufen wird, kann diese Datenkonsistenz verloren gehen. Durch dieses Systemverhalten lassen sich die Zustände als zeitinhärente Objekte ansehen, die sich nur in der Größe der gespeicherten Information und deren Bedeutung unterscheiden.

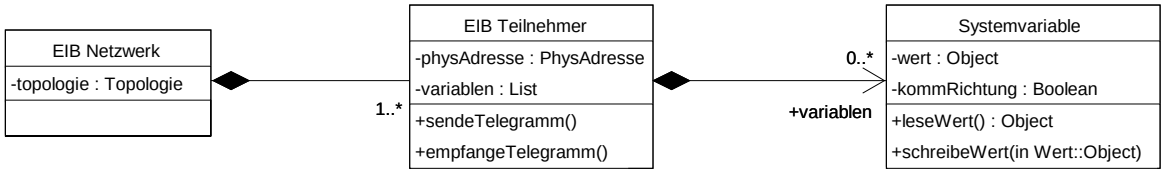

Abbildung 3.2: Strukturansicht des EIB

Diese Darstellung erfasst nur sehr grob die tatsächlichen Verhältnisse. Die Kommunikation ist vielschichtig und die obige Ansicht zeigt nur die Verhältnisse im Betrieb, schließt jedoch die Zwischenzustände aus, die während der Konfiguration erreicht werden können. Dennoch lassen sich die Grundzüge des Systems gut abbilden. Im Betrieb werden die Inhalte der Systemvariablen mehrfach gespeichert, wobei der Zustand einmalig sein sollte, so dass für die Abbildung des Systems die einzelnen Teilnehmer zugunsten des Systemzustands vernachlässigt werden können.

3.4.2.2 IEEE 1394

Das Haupteinsatzgebiet des IEEE 1394 ist eindeutig in der Übertragung von isochronen Daten zu sehen. Das bereits beschriebene Verfahren zur zentralen Zuteilung von Buszugriffsrechten erlaubt eine feste Zusicherung von Datenraten. Da in der Spezifikation nur die unteren Schichten (Physical Layer, Link Layer, Transaction Layer) – verglichen mit dem ISO/OSI-Referenzmodell bis zum Network/Transport Layer – beschrieben werden, können für Anwendungen spezielle Datenformate verwendet werden. Damit ist bei der Beschreibung des Bussystems die Unterscheidung zwischen isochron und asynchron in diesem Kontext ausreichend.

3.4.2.3 IEEE 802.3 und IEEE 802.11

Da beide Systeme auf den gleichen Ebenen arbeiten, sollen sie hier kurz zusammengefasst werden. Aufbauend auf diesen Kommunikationsschichten lassen sich vielfältige Protokolle verwenden. Im Zusammenhang mit Netzwerken ist wohl am weitesten das TCP/IP verbreitet. Auf der Basis von TCP/IP können wiederum Anwendungsprotokolle verwendet

[21] CAN arbeitet nach einem sehr ähnlichen Prinzip. Dabei werden Identifier zur Adressierung von Kommunikationsobjekten eingesetzt [CAN1991].

werden, wie *Post Office Protocol* (POP), *Simple Mail Transfer Protocol* (SMTP), *Simple Network Management Protocol* (SNMP), *Hypertext Transfer Protocol* (HTTP) oder aber *File Transfer Protocol* (FTP).

3.4.2.4 Bluetooth

Der Buszugriff wird bei Bluetooth durch einen Master gesteuert. Gelangen zwei oder mehr Geräte in Empfangsreichweite, wird ein Gerät als Master ernannt und legt daraufhin die Hopping Sequenz für das Frequenzsprungverfahren fest. Ein Frequenzkanal wird dabei im Time Division Duplex genutzt, wobei zwischen zwei Nachrichtentypen unterschieden wird: *Synchronous Connection-Oriented Link* (SCL) oder aber *Asynchronous Connection-Less Link* (ACL). Die synchronen Nachrichten arbeiten nach einem ähnlichen Prinzip wie die bei IEEE 1394. Dort werden Zeitschlitze zur Übertragung reserviert, um eine gesicherte Übertragungsrate einzuhalten. Es werden keine Prüfsummen für die Fehlererkennung verwendet, jedoch wird mit *Forward Error Correction Code* (FEC) 1/3 bzw. 2/3 übertragen, ohne dass beim Auftreten eines Fehlers eine Wiederholung stattfindet. Der Nachrichtenfluss ist dabei auf den Austausch von Daten zwischen dem Master und dem Slave beschränkt.
Die asynchrone Übertragung verwendet sechs bestätigte Rahmenformate und ein unbestätigtes Rahmenformat unterschiedlicher Länge, die durch einen *Cyclic Redundancy Checksum* (CRC) gesichert und im Fehlerfall wiederholt werden.
Für die höheren Kommunikationsschichten sind Profile definiert worden, die bestimmten Anwendungen gerecht werden. Dazu zählen Headsets für die drahtlose Sprachanbindung, serieller Schnittstellenersatz oder Bluetooth TCP/IP-Übergänge. Das verwendete Datenmodell wird dann durch das jeweilige Profil festgelegt.

3.4.2.5 DECT

Beim DECT Data Link Layer wird eine Einteilung in *C-Plane-* und *U-Plane-Services* vorgenommen. Die C-Plane-Services dienen hauptsächlich dem Austausch von Kontrollinformationen. Über die Services der U-Plane lassen sich anwendungsspezifische Daten übertragen. So ist dort ein Rahmen für die Übermittlung von Sprachdaten definiert (LU1 - *TRansparent UnProtected service* (TRUP)). Damit können Sprachdaten bei einer Datenrate von 32 kBit/s übertragen werden, jedoch nur mit einer Fehlererkennung und einer begrenzten Möglichkeit zur Fehlerkorrektur. Für asynchrone Daten werden andere Dienste angeboten, die auch eine Segmentierung über mehrere Zeitscheiben zulassen und für asynchrone Nachrichten besser geeignet sind, da diese bestätigt ausgeführt werden.
Die Implementierung der LU-Dienste ist optional und die Art und Anzahl der Dienste hängen von der jeweiligen Anwendung ab. Damit ergibt sich bei DECT die Möglichkeit der isochronen und asynchronen Übertragung. Mit der Schaffung des *DECT Packet Radio Service* (DPRS) wurde die Aufteilung des Up- und Downlinks in zwei gleichlange Zeitrahmen zu je zwölf Zeitscheiben aufgelöst und zusätzlich neue Dienste auf der U-Plane eingeführt. In der Gesamtheit der DECT-Spezifikation sind alle Voraussetzungen für das

Erstellen von Medienübergängen geschaffen. Es sind Formate (FU10) für eine Relay-Funktion vorgesehen, die Verbindungen von verschiedenen Netzwerkprotokollen über DECT erlauben (ISO/IEC 8802-3, Ethernet; ISO/IEC 8802-5, Token-Ring; Point-to-Point protocol[22] (PPP), V.24). In der nachstehenden Grafik ist dieser Sachverhalt dargestellt.

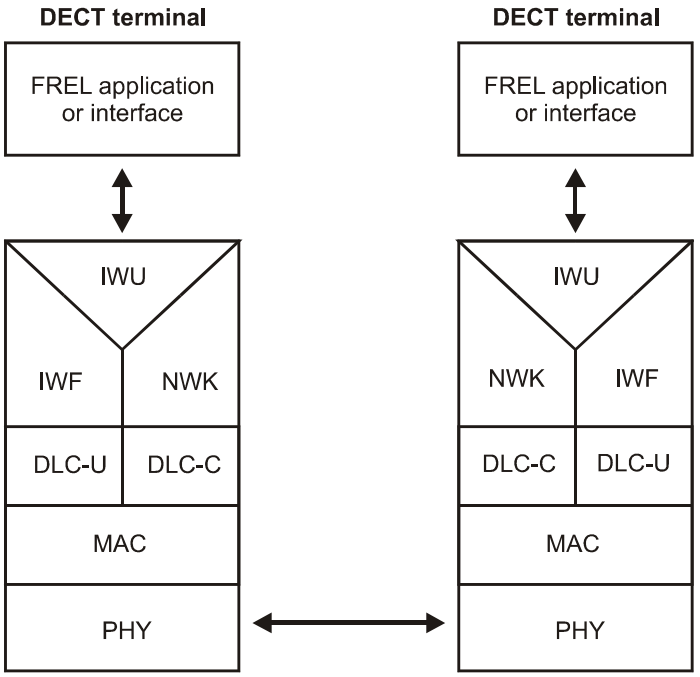

Abbildung 3.3: Konfiguration für den Frame Relay Service (verbindungslose und verbindungsorientierte Dienste)[23]

[ETSIDPRS] ergänzt und ersetzt hauptsächlich Teile der Spezifikation ETSI 300 175-2 – ETSI 300 175-4. Dies zeigt sich sehr deutlich an den Frame-Relay-Diensten, die in den vorherigen Spezifikationen bereits angedacht waren. Das in Abbildung 3.3 gezeigte Prinzip ist auch auf andere Protokolle übertragbar. Im Regelfall bezieht sich die Umsetzung jedoch auf Anpassung und Tunnelung der unteren Kommunikationsschichten.

[22] Nach RFC 1662, RFC 1618, RFC 1973, RFC 2364.
[23] Aus [ETSIDPRS].

4. Erstellung des Gesamtkonzepts

Bei der Ankopplung mehrerer Systeme ergeben sich prinzipiell unterschiedliche Lösungsansätze, die hier im Einzelnen gezeigt werden sollen.

4.1 Einstufiges System

Die Verbindung der unterschiedlichen Bussysteme erfordert eine Anpassung der verschiedenen Anforderungen durch Definition einer einheitlichen Kommunikationsbasis. In der Regel kann diese Aufgabe logisch und physikalisch zentral gelöst werden. Dabei werden alle Bussysteme an einem Punkt zusammengefasst. Dies setzt einen Knoten voraus, der alle angeschlossenen Busse sowohl physikalisch, als auch in der Datenrepräsentation zu vereinheitlichen vermag.
Einfache Systemübergänge – beispielsweise eine WLAN-Ethernet Bridge - verfolgen diesen Ansatz. Dabei wird eine je nach Systemmodell, in dem genannten Beispiel handelt es sich um eine Layer 2 Bridge, mehr oder weniger direkte Abbildung der Dateninhalte vorgenommen. Dieser Ansatz lässt sich in der Praxis sehr gut umsetzen, wenn die beteiligten Bussysteme über ein ähnliches Kommunikationsmodell verfügen. Bei der oben genannten WLAN-Ethernet Bridge ist das der Fall; IEEE802.3 und IEEE802.11 umfassen den gleichen Bereich des ISO/OSI Referenzmodells[24]: Teile des Physical und des Data Link Layers. Der Dateninhalt kann daher transparent weitergegeben werden, ohne auf die Besonderheiten der unteren Schichten eingehen zu müssen.

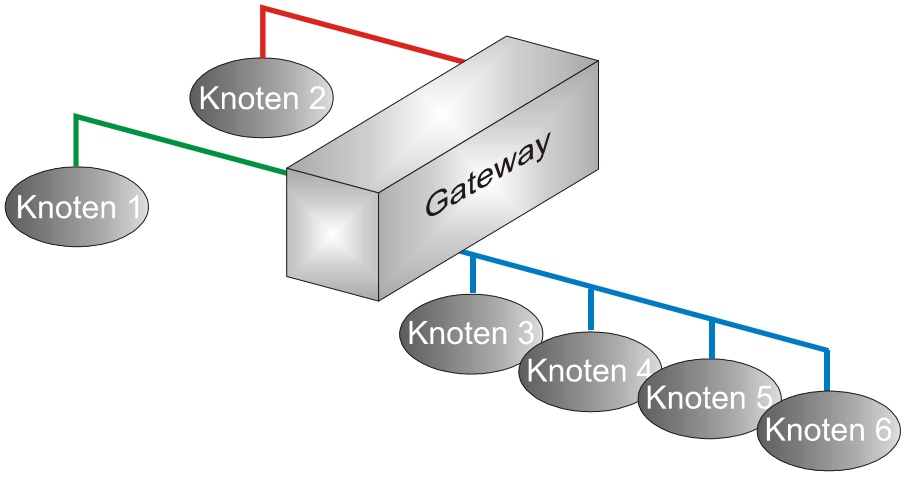

Abbildung 4.1: Gateway mit unterschiedlichen Bussystemen

Ein anderer Sachverhalt ergibt sich jedoch bei einem Übergang zwischen mehreren Systemen, der so aufgebaut ist, dass sich eine Sterntopologie ergibt, wie oben abgebildet. Das oben genannte Vorgehen kann auf Grund der Unterschiedlichkeit der Bussysteme

[24] Streng genommen ist ein solcher Vergleich nur bedingt möglich, da das ISO/OSI Referenzmodell zeitlich später definiert wurde und damit die Einordnung von IEEE802.3 und IEEE802.11 nur mit Einschränkungen erfolgen kann.

dann nicht mehr verwendet werden, da die Anforderungen an die Einzelsysteme, bedingt durch ihre unterschiedlichen Dateninhalte und –formate, zu verschieden sind. Dadurch ergibt sich bei der Entwicklung eines solchen Systems der Zwang, eine gestufte Softwarearchitektur zu verwenden, die in mehreren Ebenen eine Überführung auf ein gemeinsames Kommunikationsprotokoll vornimmt und damit bei einem System mit n Anschlüssen alle Kombinationen von 1:(n-1) Übergängen berücksichtigt. Es ist dann nicht mehr möglich und sinnvoll viele 1:1 Übergänge zu bilden, da dies in einer sehr starren Systemumgebung endet und damit Erweiterungen kaum noch zulässt und zudem hohe Systemanforderungen an das Zielsystem stellt. Damit lässt sich ein einstufiges System in der Sterntopologie nur bei sehr wenigen Knotenzugangspunkten realisieren, da dann die Übergänge auch kostengünstig realisiert werden können. In der nachfolgend abgebildeten Grafik (Abb. 4-2a) ist ein System mit vier angeschlossenen Busssystemen dargestellt, bei dem bei einem solchen Vorgehen bereits sechs Übergänge geschaffen werden müssen. Die Anzahl der Verbindungen ergibt sich aus:

$$n = \frac{m(m-1)}{2}.$$ Gleichung 4-1

m Anzahl der Eingänge
n Anzahl der Verbindungen

a) **b)**

Abbildung 4.2: Logische Verknüpfung der Bussysteme

Besser geeignet ist eine Struktur, die in Abb. 4-2b zu finden ist. Dort wird durch die Bildung eines einheitlichen, internen Kommunikationskanals die Vielzahl der direkten Übergänge vermieden. Gleichzeitig muss der gesamte Nachrichtenaustausch durch den internen Kanal geleitet werden. Dadurch verringert sich die Anzahl der zu bildenden Übergänge um

$$k = \frac{m(m-3)}{2} \text{ für m > 2}$$ Gleichung 4-2

auf *m* interne Verbindungen.
Der Nachteil dieses Systems liegt in der zweifachen Konvertierung der Nachrichteninhalte. Die Daten müssen beim Erhalt zunächst in ein vom Kommunikationsbus verständliches Datenformat umgewandelt werden, um dann durch Adressierung an den Zielübergang weitergeleitet zu werden. Dort erfolgt eine weitere Umwandlung und Anpassung an das

entsprechende Bussystem. Dieser zusätzliche Schritt der Zwischenkonvertierung reduziert die Systemleistung, wobei jedoch die Transparenz bei der Systementwicklung erhöht wird.
Durch die Systemstruktur innerhalb des Übergangsknotens müssen die zur Verfügung gestellten Daten als geschlossene Objekte dargestellt werden, die sowohl den Zustand der erhaltenen Daten beschreiben, als auch eine Beeinflussung durch im Objekt definierte Zugriffsmethoden erlauben. Die Objekte tauschen dann Nachrichten über fest definierte Schnittstellen aus. Dabei können die Verknüpfungen zwischen den Objekten fest erfolgen, da das Hinzufügen von zusätzlichen Subsystemen durch den weitgehend festgelegten physikalischen Aufbau unter Verwendung nur eines Gerätes kaum möglich ist. Neben der festgelegten Kommunikation können auch die Struktur und die Zugangsmethode für abfragbare Systemdaten bereits fest integriert werden. Die Überführung der businternen Daten auf die Objekte muss in den unteren Kommunikationsschichten erfolgen, damit die Vereinheitlichung auf höherer Systemebene stattfinden kann.

4.1.1 Auslastungsabschätzung der Einzelstufen

Das gezeigte System stellt einen Summierpunkt für die anfallenden Nachrichten dar. Damit ist die Auslastung direkt von dem Datenaufkommen an allen Zugangspunkten abhängig. Die anfallende Datenmenge hängt von den bestehenden Verknüpfungen zwischen den Subsystemen ab und kann somit (je nach ausgewählten Dienstmerkmalen) schwanken. Eine Gleichzeitigkeit bei der Nachrichtenübermittlung innerhalb des internen Kommunikationssystems kann nicht erreicht werden.

4.1.2 Ausfallbetrachtung

Die Wahrscheinlichkeit des Auftretens von Fehlern in einem solchen System soll hier nicht näher betrachtet werden, da dieses einstufige System nicht den Anforderungen an die Flexibilität genügt, die in dieser Arbeit angestrebt werden. Der Ausfall von Busteilnehmern am Subsystem fällt dabei nicht ins Gewicht, solange dadurch nicht fälschlicherweise eine Vielzahl von Nachrichten erzeugt wird, die vom Systemübergang ausgewertet werden muss. Ein bloßes Fehlen eines Teilnehmers durch Ausfall wirkt sich nicht weiter auf den Übergang aus und kann damit vernachlässigt werden.
Weiterhin können die Schnittstellen zu den Untersystemen versagen, was zur Folge hat, dass die Nachrichten nicht mehr weitergeleitet werden können. Das vom Untersystem erstellte Abbild wird damit ungültig, wobei zwischen den anfallenden Daten zu unterscheiden ist. Isochrone Datenströme werden ohnehin nicht als Systemzustand gespeichert, sondern direkt an Dienstnehmer weitergeleitet.
Als größter Fehlerfall wäre damit der Totalausfall des Übergangsknotens zu nennen, der jegliche Kommunikation unterbindet und sogar die aufgetretene Störung in die Untersysteme einbringen kann, womit das ganze System gestört wäre.

4.2 Mehrstufiges System

Das in der Hausautomatisierung verwendete Konzept der dezentralen Verwaltung und Steuerung von Geräten kann auch in der Bildung von Übergängen zwischen den

Einzelsystemen eingesetzt werden. Sein Vorteil liegt eindeutig in der Ausfallsicherheit des Gesamtsystems. Der Ausfall eines Übergangs beeinträchtigt im Regelfall nicht das Gesamtsystem. Der Nachteil liegt jedoch in der Verwaltung und Konfiguration sowie in den anfallenden Kosten für die Einzelkomponenten.
Das vorgestellte einstufige System zeigt aber auch Systemeigenschaften, die auf ein mehrstufiges System übertragen werden können. Dazu gehört eindeutig die vorher entwickelte Bussystemstruktur, die die Anzahl der Systemübergänge reduziert. Dies erfordert jedoch einen zusätzlichen Kommunikationsaufwand und eine Konzentration der anfallenden Nachrichten auf ein gemeinsames Bussystem. Der interne Systembus kann bei mehreren Geräten in einen Backbone-Bus überführt werden, der den Nachrichtentransport übernimmt.
Im Regelfall sind in einem Gebäudeautomatisierungssystem sehr unterschiedliche Geräte zu finden, die nicht alle über die Möglichkeit verfügen, ein komplexes Kommunikationsprotokoll zu unterstützen. Damit muss auch bei einem System mit ressourcenbeschränkten Geräten (embedded devices) ein Kompromiss gefunden werden. Dazu können nicht standardisierte Kommunikationsprotokolle verwendet werden, die dann auf einem Gerät mit mehr Rechenkapazität in das übergeordnete Schnittstellenprotokoll überführt werden. Dies bedeutet natürlich eine Reduzierung in den Antwort- und Reaktionszeiten, die jedoch je nach Anwendung vernachlässigt werden können, soweit die Datenübertragungsrate und Verarbeitungsgeschwindigkeit der Systeme nur groß genug gewählt werden.

Tabelle 4.1: Dienste und die zugehörigen Datentransferraten

Dienst/Anwendungsart	Erforderliche Übertragungsrate	Anwendung
Inhalte der unteren Ebene (Sensordaten)	< 1MBit/s	Übertragung von Zuständen, Abfrage von Zuständen
Inhalte aus dem Internet	> 128 kBit/s	Einfache Dienste: E-Mail (SMTP, POP, IMAP), HTTP Kommunikation
Inhalte aus dem Internet	> 768 kBit/s	Dateitransfer (FTP), Peer-to-Peer Dienste
Audio	Sprache, Telefonqualität: 8-32 kBit/s [25] Sprache, Telekonferenzen: 48-64 kBit/s Musik, CD Audio: 128 kBit/s [26]	Telefon, Interkom-Anlagen Konferenzen Musikdatenübertragung
Video	Videokonferenz (H.320): 64 kBit/s Video, gute Qualität: 15 MBit/s [27] Video, hohe Qualität: 20-40 MBit/s [28]	Videostreaming, DVB

[25] Aus [Halsall1996], S. 561: 8 ksamples/s, 8 Bit/sample.
[26] Aus [Halsall1996], S. 561: 44,1 ksamples/s, 16 Bit/sample.
[27] Aus [Halsall1996], S. 561: MPEG2 Kodierung bei einer Auflösung von 720 x 576, 12 Bit Farbtiefe, 25 Frames/s (PAL).

In der Tabelle 4.1 sind die häufigsten Anwendungen innerhalb der Gebäudevernetzung eingetragen, wobei darin durchaus Elemente enthalten sind, die in Zweckbauten Verwendung finden. So beispielsweise die Videokonferenz, die man bei entsprechenden Voraussetzungen dieser Dienste auch im privaten Bereich nutzen kann. Denkbar wäre das zum Beispiel bei einer Gegensprechanlage mit Videobild zur Kontrolle der Besucher.
Für die Übertragung der Sensordaten wird im Vergleich zu den anderen Dateninhalten eine relativ hohe Datenrate benötigt. Die in Tabelle 4.1 dargestellten Werte zeigen jedoch nur einen Ausschnitt aus der Gesamtheit der möglichen Dienste und Anwendungen. So gibt es durchaus Anwendungen, die mit etwas anderen Anforderungen in den gleichen Bereich fallen und dennoch mit einer geringeren Übertragungsrate auskommen. Für die Sensor-/Aktordaten gilt dies insoweit, als dass es bereits Bussysteme mit weitaus geringeren Transferraten gibt, die in der Gebäudeautomatisierung eingesetzt werden können. Eine hohe Übertragungsrate kann durchaus sinnvoll sein, nämlich dann, wenn eine hohe Anzahl von Teilnehmern auf das gleiche Bussegment zugreift. Dies kann in der Gebäudeverwaltung für die Anzeige von Prozessdaten vieler Knoten notwendig werden. Gleichzeitig bestimmt jedoch die Leitungsausdehnung die maximale Übertragungsrate, d.h. bei zunehmender Übertragungsrate wird eine größere Bandbreite und ein größerer Dekodier-/Kodieraufwand notwendig, um die gleiche Leitungslänge beibehalten zu können. Dies lässt sich sowohl im Telekommunikationssektor (vgl. ISDN und xDSL) als auch in der Industrieautomatisierung (CAN) beobachten. Ein Ausweg ist der Umstieg auf ein anderes Übertragungsmedium, beispielsweise der Einsatz von Lichtwellenleitern, die eine sehr hohe Bandbreite bei vergleichsweise geringer Dämpfung aufweisen [29].
Aus diesen Überlegungen ergibt sich ein gestaffeltes, hierarchisches Kommunikations-system in Anlehnung des aus der Automatisierungstechnik bekannten Modells für Automatisierungsebenen (vgl. [Furrer 2000]). Eine solche Hierarchie muss in Anlehnung an einen geringen Konfigurationsaufwand jedoch möglichst flach gehalten werden. Vorstellbar wäre eine Aufteilung nach folgender Darstellung:

[28] Aus [Halsall1996], S. 561: MPEG-3 Kodierung bei einer Auflösung von 1920 x 1080, 12 Bit Farbtiefe, 30 Frames/s.

[29] Nach [IEEE1394B-2002] werden Multimode Fasern vorgeschlagen: Polymer Optical Fiber and Glass Optical Fiber (GOF). Mit GOF soll bei einer Datenübertragungsrate von 1600 MBit/s noch eine Reichweite von 100 m erreicht werden. In der Kommunikationstechnik wird ein anderes Verfahren eingesetzt, das Dense Wavelength Division Multiplex, bei dem über eine Single-Mode Faser Licht mit mehr als 16 unterschiedlichen Wellenlängen gleichzeitig übermittelt wird.

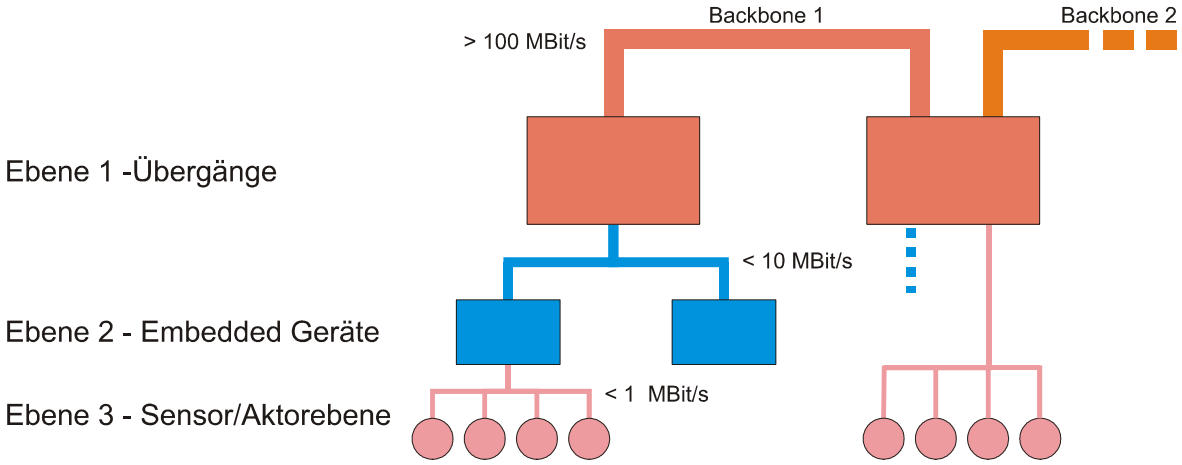

Abbildung 4.3: Hierarchie des mehrstufigen Systems

Die in Abb. 4.3 gezeigte Hierarchie sollte drei Ebenen aus den genannten Gründen nicht überschreiten. In der Regel sind bereits zwei Ebenen ausreichend, in besonderen Fällen kann es jedoch notwendig sein, eine weitere Ebene einzufügen, um leichter Übergänge erstellen zu können. Auch die Datenübertragungsraten müssen gestaffelt werden, da jeder übergeordnete Datenknoten als Konzentrator wirkt und in der Lage sein muss, die anfallenden Daten auch in der Summe an das darüberliegende System zu übermitteln. In Kap. 3.4 wurden bereits die einzelnen Bussysteme mit ihrer Übertragungsrate vorgestellt, die als Vorlage für die entsprechende Eingliederung in die Ebenenstruktur herangezogen werden kann. Die Einteilung der Übertragungsraten ist dabei an die Verfügbarkeit von Bussystemen mit den geforderten Transferraten und der Verwendung der Ebenen innerhalb des logischen Aufbaus angelehnt.

In der Sensor/Aktor-Ebene hat sich gezeigt, dass bei Automatisierungssystemen in der Feldbustechnik Datentransferraten von <1 MBit/s bei der Verwendung von nur wenigen Knoten und einer Einteilung in getrennte Bussegmente ausreichend ist. In diese Kategorie fallen der CAN, der EIB und auch der LON, die als Systeme für die Verknüpfung von Sensoren und Aktoren für die Steuerung von automatisierten Prozessen verwendet werden. Dies gilt besonders in der Heimautomatisierung, da dort in privaten Anwendungsfeldern (wie zum Beispiel beim Einfamilienhaus) selbst bei komplexen Regelkreisen aus unterschiedlichen Heizungs-/Klimasystemen und Lichtsteuerung selten mehr als 150 Busknoten benötigt werden[30]. Dabei sind die Anforderungen in der Gebäudeautomatisierung bezüglich schneller Antwort- und Reaktionszeiten nicht sehr hoch. Die Regelprozesse haben Zeitkonstanten, die bis in den Bereich von Stunden reichen können.

Die anfallenden Daten werden in den Systemübergängen vorverarbeitet und gelangen nicht in ihrer Gesamtheit auf den höherratigen Bus. Je nach Einrichtung wird nur ein Bruchteil der Daten weitergeleitet. Daher kann nicht direkt auf die Anzahl der anzuschließenden

[30] Das in Neubiberg bei München errichtete *tele*-Haus verwendet mehrere Systeme zur Gewinnung von Wärme für den Heißwasserbedarf. Dabei wird einerseits ein Anteil aus der Erdwärme bezogen, der andere aus Sonnenkollektoren. Die Steuerung der Heizungselemente (Fußboden- und Wandheizung) übernimmt eine Regelung aus den gesendeten Messwerten.

Untersysteme geschlossen werden. Ein Verhältnis von 1:9 erscheint jedoch sinnvoll; damit können – abgesehen von Protokollüberhängen – zehn Untersysteme mit voller Auslastung angeschlossen werden, was im Regelfall bei einem Gebäude ausreichen sollte – der EIB sieht bereits 61249 Geräte am Bus vor [EIBA1999], die über Linien- und Bereichskoppler in Segmente aufgeteilt werden. Im Normalfall wird jedoch nur ein sehr geringer Teil der Bandbreite ausgelastet sein.
Für die Verteilung der Datentransferrate der nächsten beiden Ebenen sind andere Anforderungen zu stellen. Hier geht es nicht mehr primär um die Übermittlung einfacher Senor-/Aktordaten, sondern vielmehr um die Übertragung von Diensten und deren Inhalte.
Aus der Tabelle 4.1 lässt sich leicht ablesen, dass die Aufteilung in drei Ebenen nicht unbedingt notwendig ist, da sowohl die Übertragung der Sensor-/Aktordaten als auch die der Sprachdaten und der Dienste wie *Hypertext Transport Protocol* (HTTP), *Simple Mail Transport Protocol* (SMTP) oder *Post Office Protocol* (POP) noch in der Ebene 2 aus Abb. 4.3 abgewickelt werden kann. Natürlich lässt sich die dort gezeigte Struktur auch horizontal erweitern, um so Bussysteme mit gleichwertigen Eigenschaften miteinander zu koppeln, wie dies in Abb. 4.3 in Ebene 1 eingetragen ist. Bei den Kopplungselementen auf oberster Ebene sind die Systemanforderungen an die Übertragungsrate anzupassen.
Ebene 1 trägt die Datenströme für die hochratigen Dienste, beispielsweise die Videoanwendungen aus Tab. 4.1 mit 15-40 MBit/s. Dabei können die Daten zentral von einem Punkt im System mittels Multicast-Nachrichten übertragen werden, um mehrere Dienstnehmer zu unterstützen, ohne jeweils eine Punkt-zu-Punkt-Verbindung aufbauen zu müssen. Paketorientierte Übertragungsprotokolle - wie beispielsweise Ethernet - eignen sich im Allgemeinen weniger, da dort keine isochrone Datenübermittlung vorgesehen ist, können jedoch bei Verwendung von Multicast-Adressen in Verbindung mit übergeordneten Protokollen verwendet werden. Dennoch ist der Einsatz von Protokollen, die isochrone Datenrahmen vorsehen, vorzuziehen.

4.2.1 Konzeptionelles Vorgehen

In dem gebildeten Netzwerk werden zur Vereinheitlichung alle Systemressourcen als Dienste modelliert. Der Verbund von Diensten wird erst möglich, wenn die Verknüpfung unabhängig von dem jeweiligen Busmedium betrachtet werden kann. Einen wichtigen Beitrag dazu leistet Middleware.

4.2.1.1 Schnittstellenentkopplung durch Verwendung von Middleware

Middleware gibt es für verschiedene Bereiche und eine eindeutige Definition kann kaum gegeben werden. In [Aiken2000], einem Bericht über einen Workshop, wird es so ausgedrückt : „The Workshop participants agreed on the existence of middleware, but quickly made it clear that the definition of middleware was dependent on the subjective perspective of those trying to define it.“. Der einzig auffindbare Konsens bei der Definition von Middleware ist die Übereinkunft, dass dadurch die untere und oberste Schicht eines Dreischichtenmodells (von verteilten Anwendungen) miteinander verbunden werden. Beispiele dafür sind Dienste oberhalb des IP-Protokollstacks (Netzwerk), Protokolle zur

geräteunabhängigen Kommunikation (Drucker), die dynamische Generierung von HTML-Seiten durch Active oder Java Server Pages (Webserver-Anwendungen). Auch bei der Verwendung von Ressourcen in einem verteilten System wird die Organisation und Kommunikation oftmals über Middleware abgewickelt. Fernaufrufprotokolle wie etwa *Common Object Request Broker Architecture* (CORBA) oder *Remote Method Invocation* (RMI in Java) zählen auch zur Middleware. Daher wird in diesem Zusammenhang der Begriff Middleware folgendermaßen definiert:

Middleware wird zur Abstrahierung der Schnittstellen zwischen Dienstnehmern und Dienstanbietern verwendet. Das umfasst sowohl Methoden zur Registrierung von Diensten (Verzeichnisdienst), zum Auffinden eines Dienstes, zur Benachrichtigung über Ereignisse und zur Nutzung eines Dienstes in einem verteilten System. Für alle Funktionen sind weitere, tieferliegende Kommunikationsprotokolle notwendig, die nicht mehr zur Middleware gehören. Alle Zugangsschnittstellen werden durch APIs beschrieben und sind allen Teilnehmern innerhalb des Verfügungsraums der Middleware bekannt. Das bezieht sich lediglich auf den Rahmen, der zur Verfügung gestellt wird. Für die Definition der umgesetzten Dienstschnittstellen sind weitere Spezifikationen notwendig. Es gibt auch Ausnahmen, bei denen die implementierten Schnittstellen Teil der Spezifikation sind.

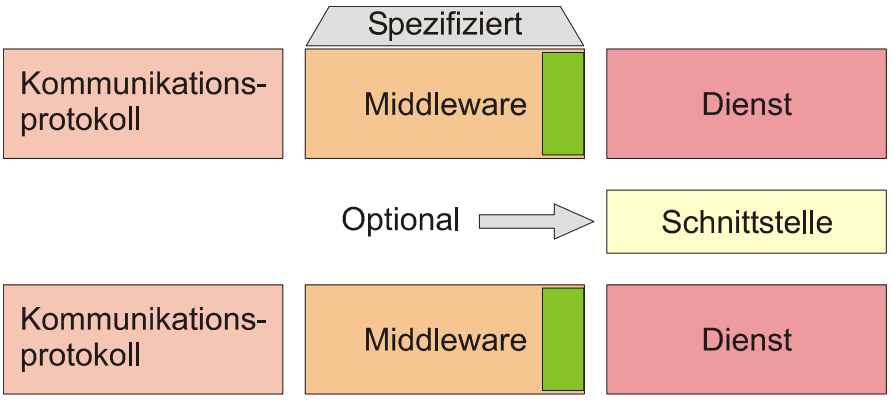

Abbildung 4.4: Definitionsumfang von Middleware

In diesem Punkt unterscheiden sich die am Markt auffindbaren Middleware-Lösungen. Bei einem stark festgelegten System sind die Dienst-Dienst-Schnittstellen klar definiert. Damit ist eine Kommunikation der Dienstanbieter mit den Dienstnehmern ohne Einschränkungen möglich, auch wenn die implementierten Funktionen von unterschiedlichen Herstellern stammen. Das setzt jedoch eine zentrale Verwaltungsinstanz voraus, die diese Spezifikation übernimmt und Neuerungen in das System einarbeitet. Nachfolgend sollen vier in Betracht kommende Middleware-Lösungen vorgestellt werden.

- *Home Audio/Video interoperability* (HAVi)
  Ende 1999 wurde unter Leitung von acht Firmen[31] der Unterhaltungsbranche ein Kon-

[31] Dazu gehören: Grundig AG, Hitachi AG, Matsushita Electric Industrial Co., Ltd. (Panasonic), Royal Philips Electronics, Sharp Corporation, Sony Corporation, Thomson Multimedia, Toshiba Corporation. Seit

sortium mit dem Ziel gegründet, Multimediageräte unterschiedlicher Hersteller unter Verwendung von IEEE 1394 miteinander verbinden zu können. Dabei können die angeschlossenen Geräte Funktionen innerhalb des Netzwerkes auslösen und auch Dienste anderer auf der eigenen Plattform ausführen. Dazu gehört eine Definition zur Darstellung von Benutzeroberflächen und zur Interaktion mit Benutzern.
Bei diesem Funktionsumfang wurden vier Kategorien von Geräten entwickelt, die sowohl nicht konforme Protokolle als auch eingeschränkte Ressourcen berücksichtigen. Im Einzelnen sind dies *Legacy A/V Devices* (LAV), die zwar über eine IEEE 1394 Schnittstelle verfügen, jedoch nicht HAVi-konform sind, *Basic A/V Devices* (BAV), die nur Teile der Spezifikation berücksichtigen und Programmcode (*Device Control Module*, DCM) auf *Full A/V Devices* ausführen können, die über eine Laufzeitumgebung verfügen und HAVi konform sind und letztendlich *Immediate A/V Devices* (IAV), die zwar HAVi konform sind, aber keine Möglichkeit zur dynamischen Bindung von Programmcode bieten. Das besondere an HAVi ist, dass die Implementierung des notwenigen HAVi Stacks in Java vorgeschrieben wird.

- *Java Intelligent Network Infrastructure* (Jini)
  Jini wurde erstmals am 25. Januar 1999 der Öffentlichkeit im Rahmen der Worldwide Analyst Conference von Sun Microsystems in San Francisco (Californien) präsentiert [Bader2000]. Die Aufgabe von Jini ist die Abstrahierung von Fähigkeiten einer Funktion oder einer Funktionsgruppe durch deren Abbildung auf Dienste. Dabei kann es sich um Funktionen von Geräten handeln, oder aber auch um Aufgaben, die in einem bestimmten Zusammenhang gelöst werden. Vorzugsweise befinden sich die Funktionen oder Funktionsgruppen in einer verteilten Umgebung. Das soll nicht heißen, dass die Funktionen von Jini auf Verteiltheit beschränkt sind; es ist durchaus vorstellbar, Jini auch in einer singulären Ausführungsumgebung zu verwenden.
  Die Gesamtheit der Dienste von allen Mitgliedern eines Verbunds (Netzwerks) wird über einen Verzeichnisdienst verwaltet. Der Ort dieses Dienstes muss ebenfalls nicht bekannt sein. Es werden nur Methoden beschrieben, wie dieser gefunden und verwendet wird. Wie der Name bereits besagt, handelt es sich dabei selbst um einen Dienst. Die Funktionalität wird ausschließlich durch Schnittstellen bereitgestellt, wobei die Art und Weise, wie diese Dienste implementiert worden sind, unerheblich ist.
- *Universal Plug and Play* (UPnP)
  UPnP wurde ebenfalls im Jahr 1999 vorgestellt und gilt als Konkurrenzprodukt zu Suns Jini. Tatsächlich gibt es starke Ähnlichkeiten zwischen den Zielsetzungen der beiden Systeme. Beide sollen Geräten unterschiedlicher Art eine Kommunikation über ein gemeinsames Protokoll ermöglichen, wobei die Kommunikationspartner Informationen austauschen können. Die Unterschiede werden jedoch deutlich, wenn man die Implementierung betrachtet. Bei Jini wird als Kommunikationsprotokoll RMI

der Gründung haben sich mehrere Firmen angeschlossen, so dass das HAVi Konsortium aus 21 Firmen/Partnern besteht.

eingesetzt und Java zwingend als Programmiersprache vorgegeben. Hingegen wurde der Schwerpunkt bei UPnP auf die gemeinsamen (Standard-)Protokolle gelegt. UPnP verwendet als Basis TCP/UDP-IP zur Kommunikation. Darauf bauen Discovery-Protokolle zur Dienstanmeldung und Dienstfindung auf. Dazu wird HTTP in abgewandelter Form (Unicast und Multicast HTTP) verwendet und zur Beschreibung der Daten wird auf XML zurückgegriffen. Weiterhin werden die Dienstschnittstellen einzelner Geräte durch das UPnP-Forum – der Zusammenschluss mehrerer Firmen (zur Zeit 533) – festgelegt.

Der Vorteil von UPnP liegt in der Verfügbarkeit der benötigten Protokollstacks auf unterschiedlichen Plattformen. Die Dienste sind jedoch plattformgebunden, da die Implementierung an das Zielsystem gebunden ist.

- *Open Services Gateway initiative* (OSGi)

  OSGi wurde 1999 durch 15 Firmen gegründet und besteht mittlerweile aus mehr als 60 Mitgliedsfirmen. Das Hauptanliegen von OSGi besteht darin, eine offene Schnittstelle zur Auslieferung von Diensten über ein *Wide Area Network* (WAN) an LANs und Geräte zu definieren. An einem OSGi konformen System können zur Laufzeit Dienste installiert, entfernt, gestartet und gestoppt werden. OSGi ist plattformunabhängig, da seine Implementierung und Schnittstellen auf Java basieren. Die Dienstpakete werden in so genannten Bundles vom OSGi-Framework verwaltet, dessen APIs in der Spezifikation festgelegt sind. Auch wenn OSGi sich weitgehend auf die Anwendungsebene bezieht, ist es als Bindeglied zwischen den Geräten und deren Dienstimplementierung als Middleware anzusehen.

4.2.1.2 Konzeption und Systemmodell

Die oben vorgestellten Abstraktionsebenen eignen sich sehr gut, um die Datenmodelle der Bussysteme von den Diensten zu entkoppeln. Dies ist jedoch nur ein Teil des Systemkonzeptes, das nicht ausschließlich den Einsatz von Middleware zur Umsetzung der Systemübergänge verwendet.

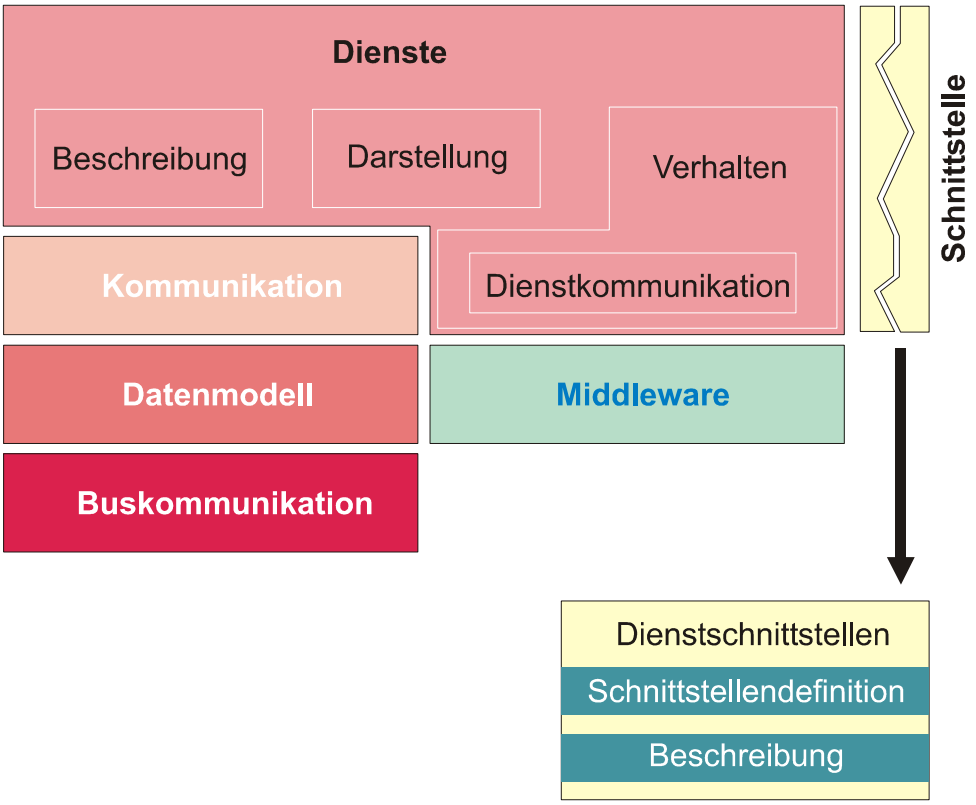

Abbildung 4.5: Konzept zur Entkopplung von Systemen und Diensten

Mit dem oben dargestellten Modell lassen sich alle Bussysteme in das Gesamtsystem integrieren. Kernpunkt des Modells ist der Dienst, der sich durch mehrere Parameter auszeichnet (Beschreibung, Darstellung, Verhalten). Die Nutzung der Dienste untereinander erfolgt über die Schnittstellen, die ebenfalls näher beschrieben werden müssen. Dies stellt eine besondere Herausforderung beim Einsatz von Middleware dar. Erst wenn die Dienste ihre Schnittstellen zur Nutzung beschreiben können, ist eine gegenseitige Verwendung möglich.

Die Kommunikation der Dienste mit den angeschlossenen Bussen wird ebenfalls durch ein geschichtetes Modell beschrieben. Auf unterster Ebene ist die Buskommunikation abzuwickeln. In dieser Ebene werden alle empfangenen Telegramme entgegengenommen und der vorhandene Protokollstack wird – wenn nötig – vervollständigt. Dazu gehört die Filterung der Telegramme, denn bereits in dieser Schicht muss eine Vorauswahl getroffen werden, um die Telegrammanzahl möglichst gering zu halten. Eine genaue Auswertung kann jedoch erst in der darüber liegenden Schicht erfolgen, da die Filterung der Telegramme Wissen über den Informationsgehalt der empfangenen Nachrichten erfordert, das erst durch das Datenmodell festgelegt wird. Bei einigen Bussystemen können zwar während der Buskommunikation schon gewisse Nachrichtentypen verworfen werden, andere müssen jedoch transparent weitergeleitet werden. Das Datenmodell übernimmt die Kapselung der Daten in Objekte und bildet somit die erste Stufe zur Abstrahierung der Daten in Richtung Dienst. Der Dienst nutzt das Datenmodell zur Abbildung seiner Funktionen. Dabei kann, wie in Abb. 4.5 vorgesehen, eine weitere Kommunikationsschicht verwendet werden, die einen Zugriff auf das Datenmodell zulässt (KomTyp 3, Abb. 4.6).

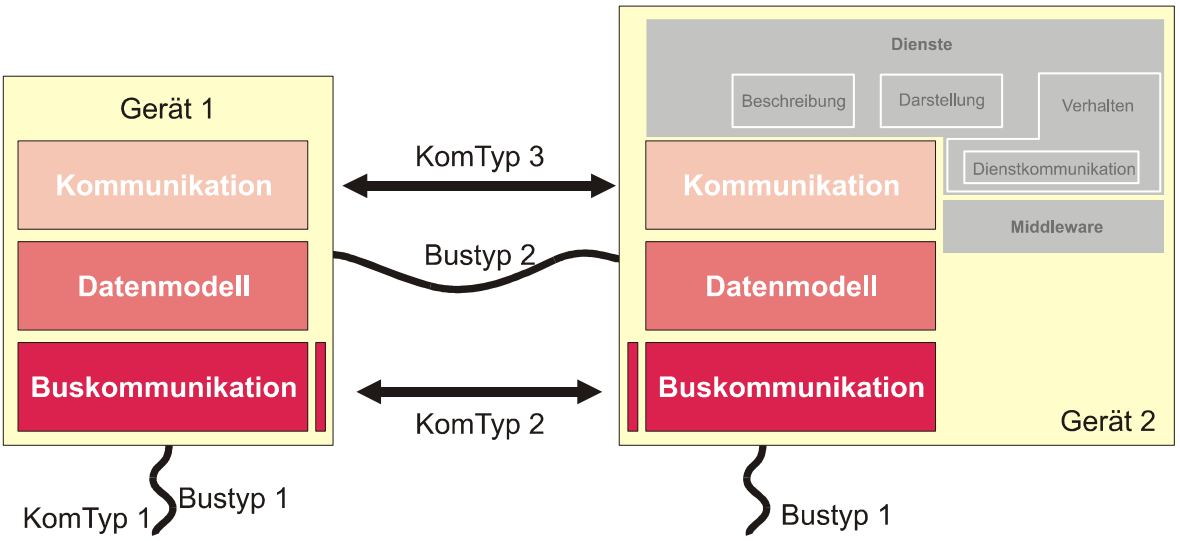

Abbildung 4.6: Austausch von Nachrichten

Dadurch wird eine weitere Schnittstelle zwischen dem Dienst und dem Datenmodell erstellt, die bereits einen nicht dienstorientierten Übergang zwischen zwei Datenmodellen erlaubt und somit eine weitere Kommunikationsmöglichkeit zur Tunnelung der Daten zwischen zwei gleichartigen Bussystemen ermöglicht. Dieser Zwischenschritt ist optional, kann jedoch beim Einsatz von Geräten unterschiedlicher Leistungskategorien erforderlich sein. Eine weitere Schnittstelle lässt sich auf unterer Ebene einsetzen, deren Umsetzung wohl am naheliegendsten ist (KomTyp 2).
Durch die in Abb. 4.6 gezeigte Struktur können zwei Bussysteme gleichen Typs in das Dienstgefüge integriert werden, auch wenn ein Gerät die Dienstimplementierung nicht direkt unterstützen kann. In besonderen Fällen kann die Kommunikationsschicht für den Dienst vollkommen transparent sein. Dieser Sonderfall tritt beispielsweise bei der Verwendung von RMI ein. RMI ist im Sprachumfang von Java enthalten (seit der Version 1.1) und definiert Methoden zur Verwaltung von verteilten Objekten. Die Kommunikation findet dabei über das *RMI Wire Protocol*[32] statt, das zur Zeit für TCP/IP implementiert ist. Durch eine zusätzliche API innerhalb von RMI kann die Transportschicht vollkommen ausgetauscht werden. Damit ist RMI auch auf IEEE 1394 umsetzbar. Die RMI-Spezifikation beinhaltet sowohl Methoden zur Verwaltung von verteilten Objekten inklusive Garbage Collection, als auch Fernaufrufe, die für den Programmierer bei der Definition von Signaturen vollkommen transparent zu lokalen Signaturen sind.
In dem aufgestellten Modell lassen sich damit vier Kommunikationstypen lokalisieren:

- *Kommunikation Typ 1* (KomTyp 1)
  Dieser Typus entspricht den Kommunikationsprotokollen der Bussysteme und wird nur von Geräten innerhalb des Bussystems verstanden. Diese Ausprägung ist das Ausgangsmaterial für die weitere Verarbeitung.

[32] S. [SUNRMI]

- *Kommunikation Typ 2* (KomTyp 2)
  Bei der Verbindung von Bussegmenten, die eine Vergrößerung des Ausdehnung des Bussystems zum Ziel hat, wird diese Methode oft eingesetzt. Einige automatisierungstechnische Lösungen verwenden dieses Konzept, um mit einfachen Mitteln und zumeist auch ohne Konfigurationsaufwand den Wirkungsbereich eines Systems zu vergrößern. Dabei werden in den Segmenten Geräte eingebracht, die dann unter Verwendung von gesicherten Punkt-zu-Punkt-Nachrichten den Verkehr nur spiegeln und weiterleiten. Im weitesten Sinn kann man auch Bridges auf der physikalischen Ebene des ISO/OSI-Modells dazu zählen.
- *Kommunikation Typ 3* (KomTyp 3)
  Die vom Bussystem erhaltenen Daten haben in dieser Ebene bereits eine Interpretation und eine Abbildung auf interne Datenstrukturen erfahren. Damit setzt eine Kommunikation auf dieser Ebene – bis auf die beschriebene Ausnahme (RMI) – auf Schnittstellen des Datenmodells auf.
- *Kommunikation Typ 4* (KomTyp 4)
  Nachrichten diesen Typs enthalten keine Information über den Aufbau oder die buseigenen Datenstrukturen. Sie dienen ausschließlich der Dienst/Dienst-Kommunikation und sind zu einem hohen Grad abstrahiert.

Diese Aufteilung in vier Kommunikationstypen ist sehr flexibel und berücksichtigt alle Varianten der Kommunikation und Abstraktionsgrade. Der kumulierte Aufwand bei der Verarbeitung der Ursprungsdaten (KomTyp 1) steigt mit zunehmender Typzahl. Dementsprechend fließen Latenzzeiten ein, die nur durch den Verzicht auf bestimmte Schichten vermieden werden können. Denkbar ist der Wegfall einer Kommunikationsschicht, um gleich auf die Dienstebene überzugehen. Im Einzelfall sind die Vor- und Nachteile bei der Implementierung abzuwägen.

4.2.2 Definition von Diensten

Bei der Definition eines systemübergreifenden Konzepts lassen sich Dienste aus unterschiedlichen Blickwinkeln definieren. Dabei stehen einerseits die Bedürfnisse der Anwender, andererseits die technischen Möglichkeiten im Vordergrund.
Der Begriff eines Dienstes umfasst mehrere Aspekte, die sich durch den verwendeten Kontext ändern können. Daher soll der Dienst innerhalb eine Automatisierungssystems durch folgende Eigenschaften beschrieben werden:

- Verhalten
  In einem Dienst wird die Aufgabe klar gefasst. Dazu gehört eine „Blackbox“-Beschreibung, also eine Verknüpfung von Ereignissen und Aktionen, die durch geeignete Mittel beschrieben wird.
- Schnittstellen
  Der Dienst verfügt für die Verarbeitung und Ausgabe über definierte Schnittstellen, die im Kapitel 4.2.2.2 genauer beschrieben werden sollen. Nur durch die Definition der Schnittstellen können Dienste untereinander kommunizieren. Um die

verschiedensten Dienste miteinander koppeln zu können, muss eine geeignete, universelle Beschreibungssprache verwendet werden.

- Darstellung
  Neben den Systemparametern, zu denen das Verhalten und die Schnittstellen gehören, soll auch eine Darstellung des Dienstes erarbeitet werden, soweit dies möglich ist. Mit der Darstellung befasst sich ein weiterer Teil des Dienstes, der Benutzereingaben entgegennimmt und über geeignete Schnittstellen in den Zustand eines Dienstes eingreift. Es wird jedoch auch eine Vielzahl von Diensten geben, die keine direkte Interaktion mit dem Benutzer erfordern.

4.2.2.1 Dienste aus der Sicht des Anwenders

Die bisher existierenden Umsetzungen orientieren sich stark an den technischen Möglichkeiten, verlieren dabei jedoch einen wichtigen Aspekt aus den Augen: den Anwender. Bei allen Möglichkeiten, die die Verwendung von Bussystemen bieten, wird zumeist nur das Machbare, jedoch nicht das dem Anwender Zumutbare im Blickfeld gehalten. Innerhalb des Forschungsprojekts *tele*-Haus hat sich durch die öffentliche Ausstellung der Ergebnisse gezeigt, dass die Benutzer von der Komplexität der Systeme oftmals überfordert sind. Daher soll, bevor auf die technische Definition von Diensten eingegangen wird, ein Blick aus der Perspektive des Anwenders gewagt werden.
In den vorangegangenen Kapitel wurden die einsetzbaren Medien zur Automatisierung eines Gebäudes vorgestellt. Leider sind die für den Anwender zugänglichen Möglichkeiten eines Bussystems stark von den eingesetzten Geräten abhängig. So nützt eine 10 GBit/s Infrastruktur innerhalb eines Gebäudes wenig, wenn der Benutzer nur mit PCs arbeitet, die mit 10 MBit/s an die Ethernet-Leitung angebunden sind. Dieses Beispiel ist zwar trivial, da es sich nur auf physikalische Parameter bezieht, jedoch stellvertretend für Fälle, in denen der Fokus auf die Anwendungsmerkmale eines Geräts gelegt wird.
Die Dienste sind eng an die Buseigenschaften angelehnt. Weiterhin werden die an den Bus angekoppelten Geräte nur einen Bruchteil der realisierbaren Dienste nutzen können; zum einen, weil die Geräte durch ihre Systemeigenschaften nur eine begrenzte Kapazität zur Verfügung stellen, und zum anderen, weil es nicht sinnvoll ist, bestimmte Dienste miteinander zu kombinieren.

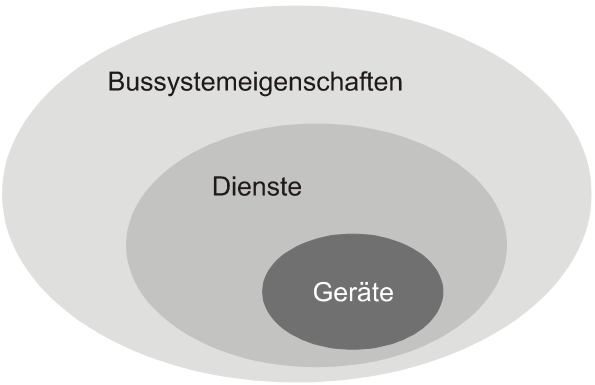

Abbildung 4.7: Dienstangebot und Buseigenschaften

Aus der Menge der Dienste ist wiederum nur ein Teil für den Benutzer im alltäglichen Umgang sinnvoll. Zu den gebräuchlichsten Anwendungen zählen:

- Telefonie/Fax
  Nach einer Hochrechnung des statistischen Bundesamts [StatBund2001] besaßen im Jahre 2001 rund 96% aller Haushalte ein stationäres Telefon. Die Ausstattung mit Mobiltelefonen betrug dabei immer noch nahezu 56%. Ein Faxgerät war in 16% der Fälle anzutreffen, ohne dabei auf die Faxfunktion von PCs (56% der Haushalte) einzugehen. Weiterhin sind 43% der Haushalte mit einem Anrufbeantworter ausgestattet. Daraus lässt sich ableiten, dass die Integration von Sprachdiensten in ein Gebäudeautomatisierungssystem einen hohen Akzeptanzgrad erzielt.
- Gegensprechanlage
  Gegensprechanlagen sind auf Grund ihres vergleichbaren hohen Preises und den erforderlichen Umbaumaßnahmen nicht flächendeckend anzutreffen. Dennoch ist der Gebrauch von Audio- und Videofunktionen zu Überwachungszwecken ein sehr nützliches Mittel, ohne dass die Benutzung ein hohes technisches Wissen verlangt.
- Heizung/ Klima/Lüftung
  Aus den Erfahrungen des Forschungsprojekts *tele*-Haus und den Messeauftritten[33] lässt sich schließen, dass dieser Bereich bei den meisten Bewohnern noch konventionell durch eine hartverdrahtete Elektroinstallation gelöst wird, was im Wesentlichen zwei Gründe hat: die zur Verfügung stehende Technik wird - verglichen mit einer konventionellen Installation - als zu teuer empfunden und der Gewinn durch die Einrichtung erscheint den meisten Bewohnern nicht greifbar. Aus der Verwendung eines Bussystems zur Verknüpfung von Sensoren und Aktoren lassen sich neben der Visualisierung und Steuerung eine Vielzahl von Zusatzfunktionen erschließen.
- Audio-/Videogeräte
  Auch auf Grund der mangelnden Einigungsfähigkeit der Hersteller auf ein gemeinsames Format zur Übertragung von kopiergeschützten Video- und Audioformaten fühlt sich der Kunde verunsichert und greift auf weit verbreitete Komponentensysteme zurück, die über Analogverbindungen miteinander gekoppelt werden. Die bereits existierenden Lösungen zur digitalen Kopplung sind (noch) verhältnismäßig teuer oder aber inkompatibel zu den Produkten anderer Hersteller. Durch die Schaffung von Übergängen können hier potenzielle Hemmschwellen abgebaut werden und so ein Weg zur Vereinheitlichung gefunden werden. Die Anwendungen sind weit gefächert: Übertragung von Filmen innerhalb eines Gebäudes an mehrere Punkte (Streaming), Anzeige von wichtigen Informationen auf Geräten mit Display (Fernseher, PDA, Web Pad), Abspeicherung von Bildern einer im Eingangsbereich angebrachten Kamera auf einem digitalen Videorekorder.

[33] Dies ist natürlich ein recht subjektiver Eindruck, dennoch hier eine Liste der Messeauftritte: Light + Building 2000, Frankfurt, 19.-23.04.2000; Heim + Handwerk München 24.11.-2.12.2001, e/home 2002, Berlin, 29.-31.08.2002.

- E-Mail / HTTP
  Durch die weitreichende Ausstattung von PCs in der Arbeitswelt sind elektronische Nachrichten im Arbeitsumfeld weitgehend akzeptiert. Mit den sinkenden Kosten von PCs und der Kommunikationsdienste erweiterte sich der Einflussbereich von elektronischen Nachrichten auch in das private Umfeld.
  Das Gleiche gilt für den Bezug von Information jeder Art aus dem Internet unter Verwendung von Standard Internetbrowsern mittels HTTP. Zusammen mit den Videodiensten sind auch elektronische Fernsehprogramme für den Benutzer von Interesse.

4.2.2.2 Dienstdefinitionen

Zur Definition von sinnvollen Diensten innerhalb eines Gebäudes muss eine funktionelle Sichtweise angewendet werden, die nach Bildung von bestimmten Kategorien dann weiter verfeinert werden kann. Dies entspricht dem Vorgehen nach der Dienstausrichtung an den technischen Rahmenbedingungen.
Die Beschreibung der Dienste wird in diesem Kapitel mit Hilfe von UML erfolgen. Dabei wurde die Spezifikation in der Version 1.3 verwendet. In der Zwischenzeit wurde bereits die Version 2.0 von der *Object Management Group* (OMG) herausgegeben, die von dem verwendeten Modellierungstool Poseidon[34] zur Erstellung der Modelle jedoch nicht berücksichtigt wird.
Von den verschiedenen Darstellungstypen, die von UML angeboten werden, wird hier nur von den Klassendiagrammen Gebrauch gemacht, die die Struktur der Dienste am ehesten vermitteln können. Dazu werden Stereotypen neu definiert, die nicht in der Spezifikation existieren. Obwohl dies zunächst als Bruch mit der Spezifikation erscheint, besitzt es die Flexibilität, die durch die semantische und syntaktische Struktur von UML unterstützt wird. Stereotypen sind genau ein solches Mittel, das eine Erweiterung erlaubt, ohne die Spezifikation zu verletzen.
Die Dienste werden daher hier als Klassen innerhalb eines Klassendiagramms dargestellt. Eventuell benötigte Zusatzkonstrukte durch Klassen werden durch selbstdefinierte Stereotypen erweitert. In der nachfolgenden Tabelle sind die wichtigsten Stereotypen zusammengefasst.

Tabelle 4.2: Zusätzliche Stereotypen zur Beschreibung der Dienste

Stereotyp	Beschreibung	Bezug
<<*service*>>	Kennzeichnet einen Dienst, der innerhalb des Systems genutzt werden kann.	Klasse
<<*storage*>>	Nichtflüchtiger Speicher zur Hinterlegung von Daten	Klasse

[34] Dies ist ein in Java programmiertes Softwaremodellierungstool der Firma Gentleware. In der aktuellsten Ausgabe (Version 1.5) wird nur die UML Version 1.3, jedoch auch Teile der Version 1.4 umgesetzt.

Die Eigenschaften der Dienste werden durch Attribute der Klassen gekennzeichnet. Die Sichtbarkeit der Attribute ist dabei zur Kapselung der Eigenschaften als *private* festgelegt. Um auf die Eigenschaften zugreifen zu können, werden daher Operationen vorausgesetzt, die über eine erweiterte Sichtbarkeit gegenüber den Attributen verfügen und dadurch die Kapselung besser unterstützen. Zu den Attributen, die nur die Eigenschaften der Dienste beschreiben, werden noch Operationen hinzugefügt, die das Verhalten des Dienstes und seine Schnittstellen besser darstellen können. Es sei an dieser Stelle noch einmal darauf hingewiesen, dass die nachfolgenden Betrachtungen keine Darstellung einer Software-Implementierung sind, sondern lediglich als Vorlage dazu dienen. Man kann daher in diesem Fall auch von einer Meta-Modellierung sprechen. Die eigentliche Umsetzung der Dienste in Software-Klassen ist dementsprechend eine Instanz dieser hier aufgestellten Modelle.

Basisdienst

Um für alle Dienste, die im Netzwerk angeboten werden, eine einheitliche Grundschnittstelle zu schaffen, muss zusätzlich zu dem eigentlich von dem Gerät angebotenen Dienst ein Basisdienst zur Verfügung gestellt werden. Dieser Basisdienst erlaubt Auskünfte über den angebotenen Dienst und einige Grundfunktionen. Die Eigenschaften dieses Basisdienstes werden durch Vererbung an die implementierenden Dienstgruppen weitergegeben. Der gleiche Sachverhalt kann auch durch eine Komposition gelöst werden, wird jedoch nachfolgend als Vererbung modelliert, da dies die Darstellung und die spätere Umsetzung vereinfacht.

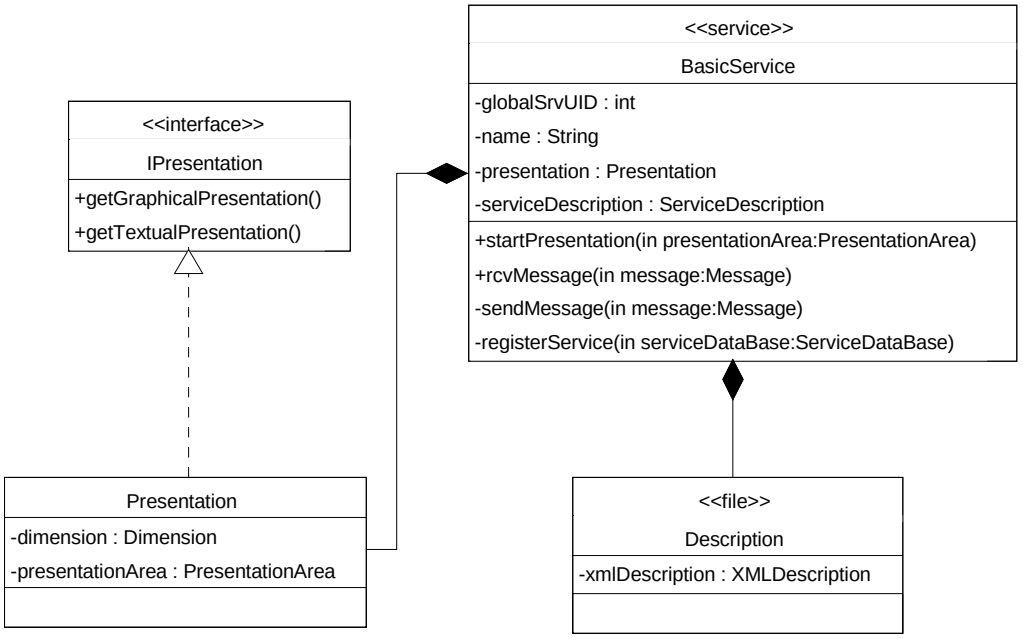

Abbildung 4.8: Basisdienst

Der Basisdienst ist ein elementarer Baustein des Systems. Er soll nur die notwendigsten Schnittstellen für die von ihm abgeleiteten Dienste bereitstellen. Die in einer XML-Datei

mitgelieferte Funktionsbeschreibung findet ihre Widerspiegelung in dem Attribut *serviceDescription*, das bei der Anmeldung des Services an das System über die Methode *registerService* an die Serviceverwaltung weitergegeben wird. Sie ist nur notwendig, wenn die Dienst-Dienst-Schnittstellen nicht von der Middleware vorgegeben werden. Sie kann jedoch optional mitgeliefert werden, um eine allgemeine Beschreibung der Schnittstelle zu erlangen, wobei diese bei der Verwendung des Dienstes nicht benötigt wird.
Es wurde bereits festgelegt, dass jeder Dienst auch für die Darstellung der eigenen Funktion zuständig ist. Die graphische Repräsentation wird vom Dienst selbst geliefert, der Dienstnehmer stellt nur eine Umgebung zur Darstellung bereit. Diese Umgebung ist in Abb. 4.8 als Klasse *Presentation* umgesetzt. Der Zugriff auf die Standardoperationen zur Anzeige wird noch einmal in einem Interface gekapselt. Es ist daher nur als Interface anzusehen, da die Implementierung der Anzeige nicht im Basisdienst erfolgt, sondern erst in den daraus abgeleiteten, erweiterten Diensten. Die Schnittstellenbeschreibung sieht zwei Methoden vor: eine für die graphische, eine andere für die textliche Darstellung. Dabei ist auch hier die Implementierung noch nicht festgelegt. Es ist denkbar als textliche Variante XML-[35] oder HTML-Seiten zurückzugeben, die dann in einem Client graphisch aufgearbeitet werden.
Die Kommunikation der Dienste untereinander ist hier stark vereinfacht durch die Methoden *sendMessage* und *receiveMessage* vertreten. Die dabei ausgetauschten Nachrichten sind dienstspezifisch und können nur bei Kenntnis des Nachrichteninhalts ausgewertet werden. Daher müssen die Nachrichten einer ganz besonderen Betrachtung unterzogen werden, damit dieses spezielle Wissen der internen Nachrichtenstruktur auf ein universelles Konzept übertragen werden kann. In einer Software-Umsetzung sind diese Nachrichten auf einzelne Methodenaufrufe abzubilden.

Videodienst
Dieser Dienst fasst mehrere Anwendungen zusammen, die über einen kontinuierlichen oder einen diskontinuierlichen Datenstrom auf Anfrage basieren.

[35] In der Verbindung mit der *Extensible Stylesheet Language* (XSL) können diese Informationen auch in einem Internetbrowser angezeigt werden.

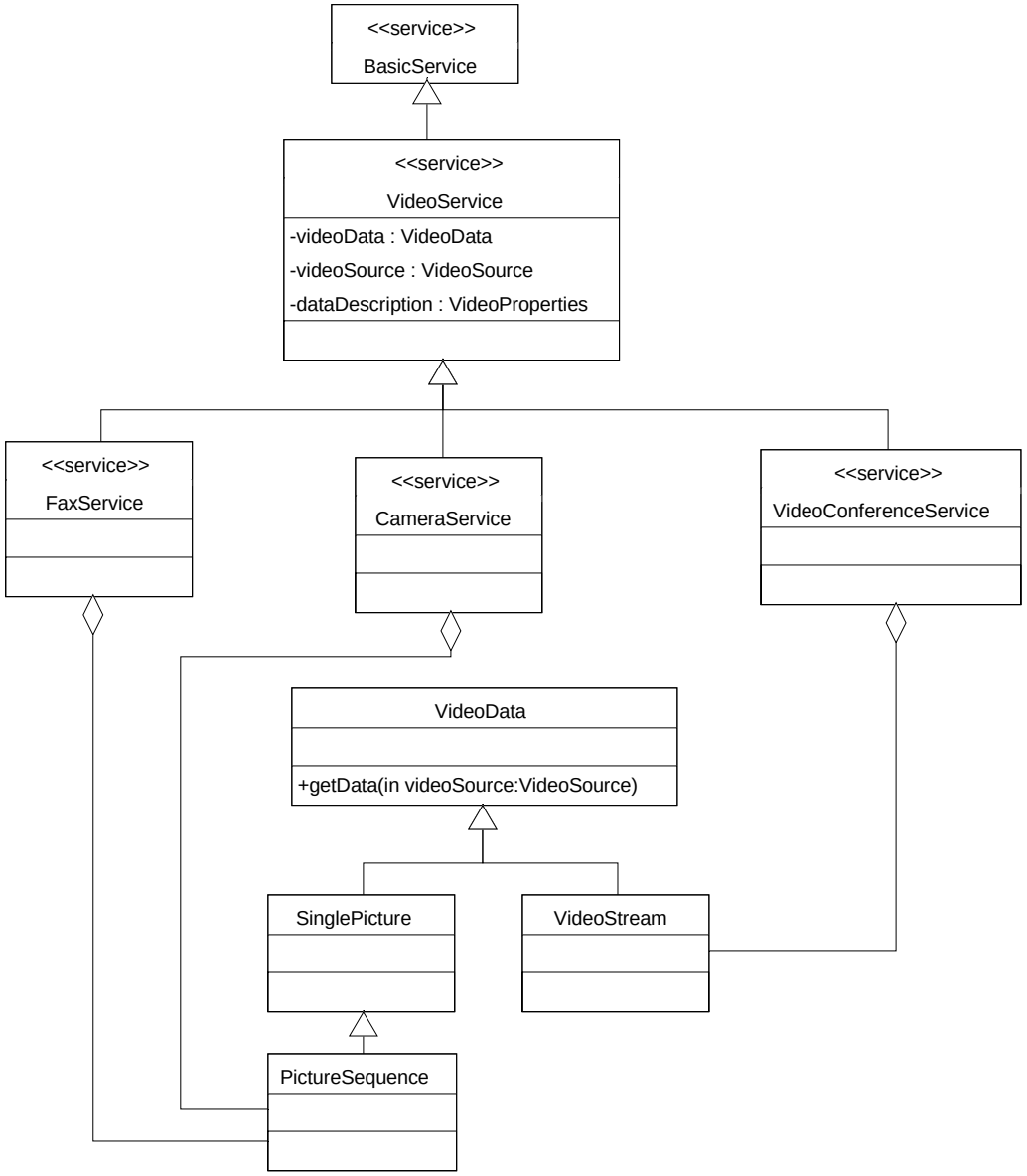

Abbildung 4.9: Videoservice

Die Art der empfangenen Daten kann je nach Anwendung variieren. Zu den möglichen Anwendungen gehören Videokonferenzen (*VideoConferenceService*), Gegensprechanlagen, Fax-Dienste (*FaxService*) und Einzelbildaufnahmen einer Überwachungskamera (*CameraService*). Der *VideoService* besitzt neben den eigentlichen Bildinformationsdaten auch eine Beschreibung der Bildeigenschaften (*VideoProperties*), die sich je nach Art der Anwendung voneinander unterscheiden.
Die Angabe einer Quelle (*VideoSource*) ist nur für den Dienstanbieter von Wichtigkeit. Er vermerkt dort die Quelle der Daten, die für den Dienstnehmer irrelevant ist, weil er allein an der Funktion des Dienstes interessiert ist. Zur Anzeige der Daten wird das im Basisdienst vorhandene Objekt zur Darstellung verwendet (*Presentation*)

Audioservice

Zur Anforderung von Audiodaten gibt es den Audiodienst, der es dem Dienstnehmer ermöglicht, den Audiodatenstrom zu erhalten. Der Aufbau ähnelt im Wesentlichen dem des Videoservices, auch wenn an dieser Stelle eine weitere Unterscheidung zwischen diversen Verbindungsarten entfällt.

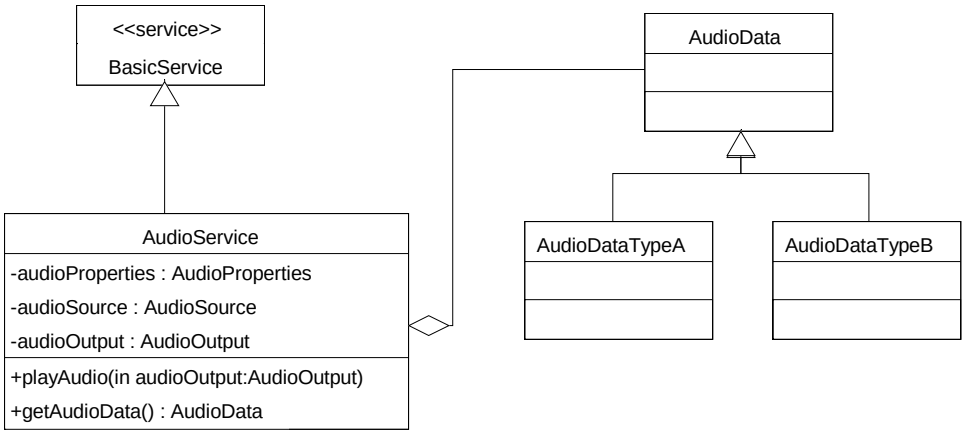

Abbildung 4.10: Dienst zur Abfrage von Audiodaten

Die Eigenschaften können hier ebenfalls über ein Attribut (*AudioProperties*) abgefragt werden. Die Ausgabe erfolgt über die Methode *playAudio*, in deren Parameter das Ziel der Ausgabe angegeben wird. Es kann jedoch auf die Audiodaten mit Hilfe der Methode *getAudioData* direkt zugegriffen werden.

Steuerungsservice

Zur Steuerung und Kontrolle der Heizung/Klima/Lüftung wird der Steuerungs- und Überwachungsservice eingesetzt. Er ist für mehrere Bussysteme umsetzbar, sofern die Datenmodelle miteinander vereinbar sind. Da die meisten Systeme primitive Datentypen für die Speicherung der Systemzustände einsetzen, genügt hier eine Einigung auf diese grundlegenden Datentypen.

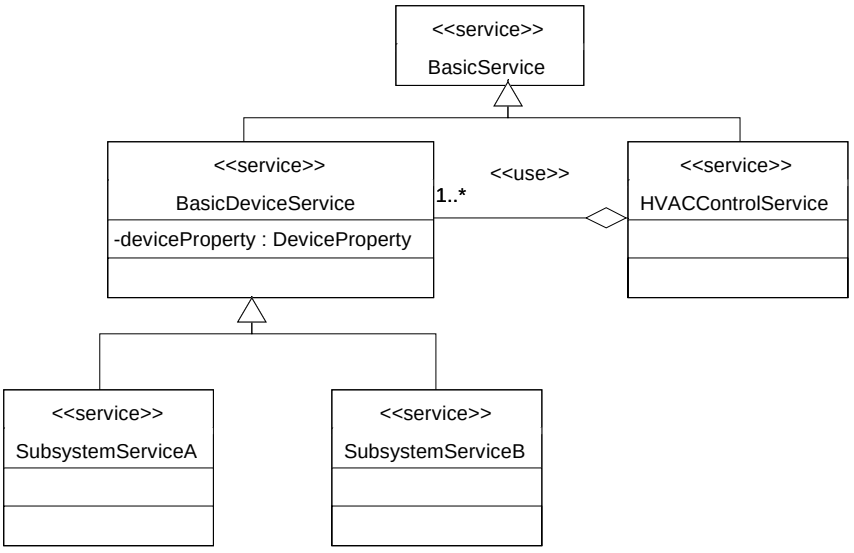

Abbildung 4.11: Dienst zur Steuerung und Wartung der HKL-Technik

In der obigen Darstellung ergeben sich zwei Arten der Systemeinbindung, die sich in ihren Abbildungseigenschaften voneinander unterscheiden. Bei der gerätebezogenen Abbildung kann jedes Gerät des Subsystems als eigener Service umgesetzt werden. Der Aufbau der Untersystemdienste ist hier in Form von Vererbung dargestellt, was die Einigung auf ein gemeinsames Datenmodell beziehungsweise geeignete Operationen für den Zugriff auf die Systemzustände ausdrückt. Bei einer Vielzahl von Geräten wird das Gesamtsystem jedoch unter Umständen mit der Verwaltung und Anmeldung seitens der Dienste des Typs *BasicDeviceService* überfordert, so dass eine andere Darstellungsform zweckmäßiger ist.
Für die reine (graphische) Steuerung ist der Dienst *HVACControlService* besser geeignet, da er alle Geräte zu einer Einheit zusammenfasst und deren Funktionalität exportiert. Er greift dabei auf die einzelnen Geräte zu, ohne dass die Dienste im Netzwerk verfügbar gemacht werden müssen; die Kontrolle kann auch lokal erfolgen.

Informationsservice
Die Verwaltung von Informationen jeder Art innerhalb eines Hauses kann ebenfalls durch einen Dienst zugänglich gemacht werden. Zu den möglichen Informationsbeständen zählen elektronische Fernsehprogrammübersichten, Daten der Terminverwaltung, Notizen oder aber aktuelle Nachrichten. Auch Benutzerprofile können dort gespeichert werden und für andere Dienste als Datenbasis dienen.
Als ein mögliches Anwendungsbeispiel kann die Hinterlegung von persönlichen Umgebungsprofilen für die Steuerung des Klimas und der Beleuchtung innerhalb eines Raumes über die Nutzung des *HVACControlService* dienen. Auch die Benutzerprofile für die Anzeige verschiedener Dienste können dort hinterlegt werden. Die im System vorhandenen Dienste lassen sich damit sehr gut individualisieren.
Durch die Offenlegung der Dienstschnittstellen und die Integration der graphischen Anzeige durch die Implementierung der Basisdienste werden die Daten innerhalb des Systems leicht zugänglich.

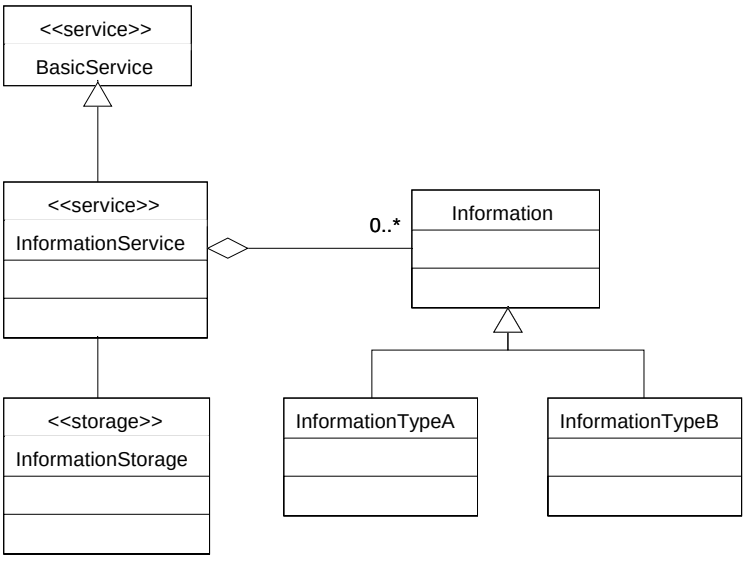

Abbildung 4.12: Service zur Verwaltung von dezentralen Informationen

Die Informationen unterscheiden sich nicht nur in ihrem Inhalt, sondern auch in ihrer inneren Struktur. Dieser Zusammenhang zeigt sich in Abb. 4.12 durch die Klassen *InformationTypeA* bzw. *InformationTypeB*. Die Daten werden aus dem *InformationStorage* gelesen und dann in Instanzen vom Typ *Information* gespeichert.

4.2.3 Subsystemverknüpfung

Die Integration der Dienste innerhalb des Systems erfolgt durch die Verwendung eines Verzeichnisdienstes und konfigurierbarer/selbstkonfigurierender Verknüpfungskomponenten. Die im vorangegangenen Kapitel vorgestellten Dienstdefinitionen können verwendet werden, um nur einen Dienst zu nutzen, oder aber die Resultate mehrerer Dienste miteinander zu verweben.
Es gibt vier mögliche Fälle der Kommunikation zwischen einem Dienstanbieter und einem Dienstnehmer, die an dieser Stelle zu unterscheiden sind:

a) Gemeinsame Middleware, Dienstschnittstellen beiderseits bekannt
   Der Austausch der Nachrichten erfolgt über die Middleware. Dabei kennt der Dienstnehmer die Schnittstellen des Dienstanbieters im Netzwerk und verwendet die Funktionen über die bekannten Dienstsignaturen. Dadurch können vorgefertigte Verknüpfungen zwischen den beteiligten Kommunikationspartnern etabliert werden. Das setzt voraus, dass die Dienstschnittstellen durch eine übergeordnete Instanz spezifiziert werden. Dies erlaubt keine Erstellung eigener Formate für den Nachrichtenaustausch, sofern sie nicht in die Spezifikation aufgenommen werden. Der Nachteil dieser Kommunikationsart liegt in der starren Bindung zwischen Umsetzung des Dienstes und der Schnittstellendefinition.
b) Gemeinsame Middleware, Grunddienst festgelegt
   Dieser Fall kann nur durch zusätzliche Systeminformationen beherrscht werden. Mit Hilfe von Zusatzinformationen, die unter Verwendung eines Basisdienstes abfragbar sind, können die Eigenschaften der Dienstschnittstelle festgestellt werden und dann die Verknüpfungen nachträglich gebildet werden. Die Verknüpfung muss dann auf Grund festgelegter Regeln erfolgen, da eine intuitive Erstellung durch Software nur schwer nachzubilden ist, oder aber ein Eingriff vom Benutzer nötig ist, was in der Regel vermieden werden soll.
c) Gemeinsame Middleware, Dienstschnittstellen nicht bekannt
   In der Middleware sollte immer ein Basisdienst vorhanden sein, der das Auffinden von Diensten ermöglicht. Ist das nicht der Fall, erbringt der Einsatz von Middleware in diesem System keinen Vorteil, da dies eine im Voraus festgelegte, teilnehmerbezogene – im Gegensatz zur dienstbezogenen – Konfiguration erfordert, die die Teilnehmerinteraktion regelt.
d) Keine Middleware
   Für diesen Fall können Übergänge über Kommunikationskanäle des Typs KomTyp 1-3 gebildet werden (vgl. Abb. 4.6). Dies entspricht der Umsetzung eines Bussystems in die dienstorientierte Umgebung. Damit ist eine ad hoc Verbindung nicht möglich. Unterscheiden sich die verwendeten Bussysteme des Dienstnehmers und des

Dienstanbieters, sind zwei getrennte Umsetzungen zu erstellen, die dann eine Kommunikation ermöglichen. Ansonsten gilt das in Kap. 4.2.1.2 Gesagte.

Die Fälle a) und b) werden nachfolgend weiter erläutert. Für den Fall c) kann eine Anpassung mittels spezieller Konfigurationssoftware erfolgen, was prinzipiell jedoch nicht sinnvoll erscheint. Fall d) ist bereits in den vorherigen Kapiteln beschrieben und gilt damit als Basis für das weitere Vorgehen.
Im Fall a) sind keine weiteren Vorkehrungen zu treffen. Da die Dienstschnittstellen bekannt sind und die Kommunikationspartner feststehen, kann über die Verwendung der Middleware der Dienst im System lokalisiert und anschließend eingesetzt werden.

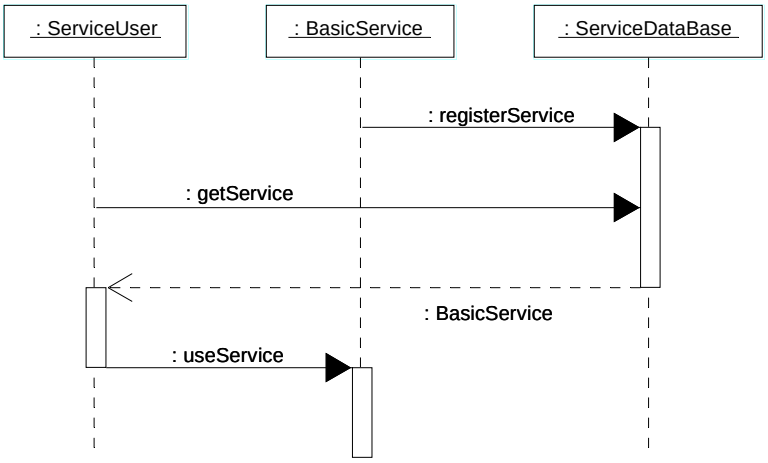

Abbildung 4.13: Dienstkommunikation bei bekannten Schnittstellen

Dieser Vorgang ist noch einmal in Abb. 4.13 dargestellt. Alle im System vorhandenen Dienste melden sich bei der *ServiceDataBase* an, die diese verwaltet. *BasicService* kann durch jeden der vorgestellten Dienste ausgetauscht werden. Sucht ein Dienstnehmer einen Dienst, so fragt er über eine bekannte Methode bei der *ServiceDataBase* an und erhält den Aufenthaltsort des gesuchten Dienstes. Da die Schnittstellen zwischen Dienstnehmer (*ServiceUser*) und Dienstanbieter bekannt sind, kann der Dienstnehmer sofort den Dienst nutzen.
Anders sieht dies im Fall b), wenn die Schnittstellen zwischen den beiden Kommunikationspartnern nicht im Voraus bekannt sind. Für diesen Fall muss das System erweitert werden.

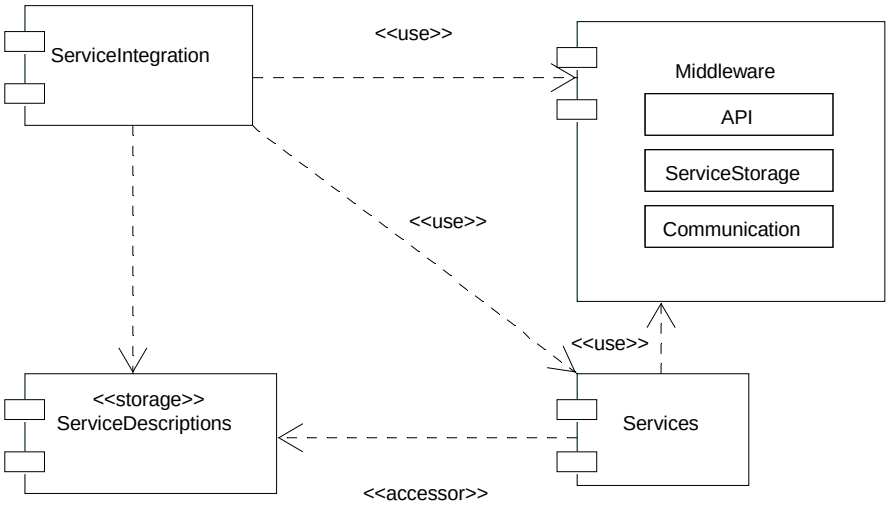

Abbildung 4.14: Komponenten der Dienstumgebung

In der Abbildung 4.14 wird die Einbettung der geschaffenen Dienste im Gesamtzusammenhang gezeigt. Die Dienste greifen auf eine Datenbasis zur Beschreibung der Schnittstellen zu (*ServiceDescriptions*). Diese Beschreibung liegt als XML-Datei vor. Wahlweise kann diese Information auch über den *InformationService* bezogen werden, oder aber dynamisch aus den Diensten generiert werden[36].
Die Middleware stellt APIs zur Verfügung, die für die Dienst/Dienst-Kommunikation benötigt werden. Dazu zählen alle Methoden zur Anmeldung, zum Auffinden und zum Abmelden eines Dienstes. Die Middleware selbst setzt intern auf eine Kommunikationsschicht auf, von der der Benutzer jedoch nichts bemerkt. Optional kann diese austauschbar sein, um an verschiedene Übertragungsmedien angepasst zu werden.
Die Komponente *ServiceIntegration* repräsentiert alle Verknüpfungs-Teilkomponenten innerhalb des Systems, die die angebotenen Dienste nutzen oder auch erweitern. Eine Erweiterung kann durch die Verbindung von einem oder mehreren Diensten erfolgen, um einen neuen Dienst zu erschaffen. Sinnvoll ist dies bei denjenigen Videodiensten, die nur in Kombination mit einem Audiodienst verwendet werden können, beispielsweise bei einer Videokonferenz. Um eine Kopplung von Audio und Video zu erreichen, kann ein weiterer Dienst erstellt werden, der die beiden anderen bündelt, oder aber der Videodienst bedient sich gleichzeitig eines Audiodienstes.
Eine weitere Verknüpfung ist die Zusammenführung von *HVACControlServices* mit der Videoüberwachung. So kann eine videogestützte Personenerkennung das gespeicherte Profil des Benutzers über den *InformationService* abfragen und daraufhin die klimatischen Raumbedingungen anpassen.

[36] Bei Java kann der von einer Klasse/Interface erzeugte Byte Code zur Laufzeit untersucht werden, um Informationen über die Struktur zu bekommen. So können die Operationen inklusive der erforderlichen Daten- und Objekttypen aus dem Byte Code gewonnen werden. Auch damit kann eine Beschreibungsstruktur zusammengesetzt werden, die dann dem Dienstanbieter übergeben wird.

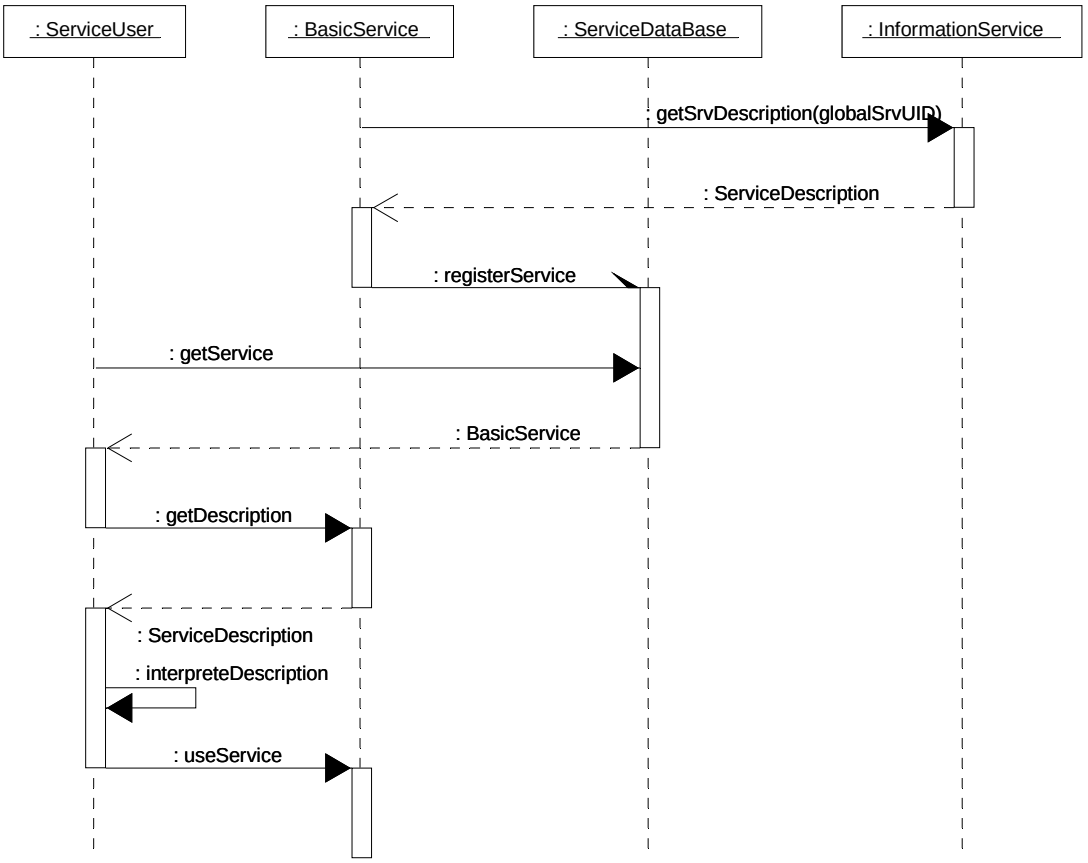

Abbildung 4.15: Dienstverknüpfungsprozess im Sequenzdiagramm

Das in dieser Darstellung gezeigte Vorgehen beschreibt den Vorgang der Dienstnutzung für Fall b). Bevor der Dienstanbieter sich anmeldet, bezieht er seine Dienst- und Schnittstellenbeschreibung. Die nachfolgenden Schritte sind wie die in Abb. 4.14, jedoch wird vor der Nutzung des Dienstes die Beschreibung angefordert. Mit dem Erhalt der Beschreibung erfolgt die Interpretation der darin enthaltenen Daten. Dies kann entweder über minimale Benutzereingriffe erfolgen oder aber durch die Aufstellung von Verknüpfungsregeln, aus deren Anwendung auf die Beschreibung die Dienstnutzung erfolgt.

4.2.4 Einsatz von Agenten

Die Verwendung von Diensten durch Menschen lässt sich durch die oben gezeigten Methoden erreichen. Durch die Vielzahl der Kombinationsmöglichkeiten erscheint es jedoch erstrebenswert, die Menge der Nutzungsmöglichkeiten einzuschränken bzw. den Anwender bei der Auswahl zu unterstützen.
Eine solche Funktionskomponente muss alle Dienste eines Systems sammeln und aufbereiten können, so dass der Anwender sich nicht selbst mit dem System auseinandersetzen muss. Zu diesem Zweck werden in dieser Arbeit Agenten eingesetzt, die stellvertretend für einen Benutzer Aktionen ausführen.

4.2.4.1 Definition eines Agenten

Agenten treten in der Softwareentwicklung an vielen Stellen hervor, wobei es sich in den meisten Fällen nicht wirklich um Agenten handelt, sondern um sich wiederholende Anwendungsmuster einer starren Umgebung. Die Definitionen eines Agenten unterscheiden sich teilweise sehr. So werden sie innerhalb eines IBM Produkts[37] zur Vorsortierung von E-Mails eingesetzt. Dieser Agent untersucht die Nachrichten unter Verwendung eines Klassifizierungsalgorithmus, der anhand der Häufigkeit der in der Nachricht vorkommenden Wörter Vorschläge für die Ablage in bestimmte Nachrichtenordner vergibt. Laut [Segal2000] wird damit nach einer Lernphase eine Treffsicherheit von mehr als 70% erreicht.
In [Foner1993] wird der Einsatz eines Agenten namens *Julia* im Bereich der künstlichen Intelligenz untersucht. Dieser Agent wird zur Suche von Informationen verwendet, die durch textliche Eingaben abgefragt werden können. Ein interessanter Aspekt ist neben der Tatsache, ausformulierte Sätze zur Abfrage verwenden zu können, dass der Agent Antworten zu Fragen des eigenen Befindens geben kann und sich ähnlich wie eine reale Person verhält.
Die in [Nöhmeier1998] erarbeiteten Agenten dienen der Informationssammlung innerhalb des Internets. Sie verfügen nicht über soziale Eigenschaften, so wie die in [Foner1993], dennoch sind sie an ihre Umgebung anpassbar und führen autonom diverse Aufgaben durch.
Fasst man die Ergebnisse aus [Foner1993], [d'Inverno2001] und [Murch2000] zusammen, so zeichnet sich ein Agent durch die folgenden Charakteristiken aus:

- Autonomie
  Ein Agent muss in der Lage sein, selbsttätig eine Aufgabe auszuführen und dabei selbst seinen inneren Zustand bewerten und verändern. Bei *Julia* zeigt sich dieses Verhalten dadurch, dass die Antworten auf Fragen nicht ausschließlich durch eine zufällige Wahl aus einer festen Datenbasis gewonnen werden, sondern, dass sich auch der Verlauf einer Unterhaltung im Ergebnis widerspiegelt.
- Soziale Fähigkeiten
  Um seine Aufgabe erfüllen zu können, kommuniziert der Agent entweder mit anderen Agenten (Multi-Agent Systeme) oder aber mit Menschen. Im Arbeitsfeld der künstlichen Intelligenz gibt es bereits eigene Sprachdialekte für die Kommunikation von Agenten– bestehend aus Nachrichtenformaten und Protokoll [Finin1997].
- Personalisierbarkeit
  Nach [Foner193] muss ein Agent an seine Nutzer anpassbar sein. Nicht alle Nutzer wollen die Aufgaben in der gleichen Art und Weise erledigt haben. Im einfachsten Fall kann dies durch Änderung der Programmierung erreicht werden.

[37] Teil des Programmpakets Lotus Notes, SwiftFile. S. [Segal2000]

4.2.4.2 Aufbau des Agenten

Für dieses System können Agenten in zwei unterschiedlichen Anwendungen eingesetzt werden. Einerseits kann ein Agent innerhalb des System verweilen und bei Feststellung eines Dienstwunsches mit dem Anwender in Verbindung treten. Dieser Zusammenhang ist in Abb. 4.16 dargestellt.

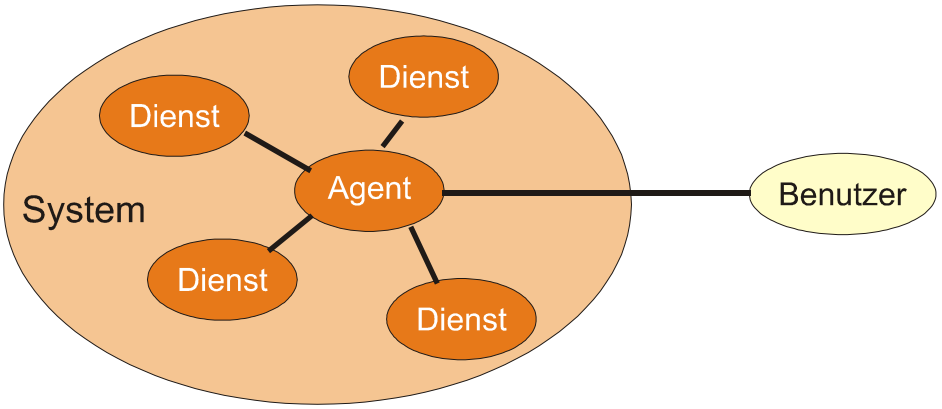

Abbildung 4.16: Agent zur Vereinfachung der Dienstnutzung

Eine andere Variante setzt einen sehr universellen Agenten voraus, der vom Benutzer in das System eingebracht wird, dort alle Informationen zusammenträgt, auswertet, dem Benutzer anzeigt und als Stellvertreter bei der Dienstnutzung handelt.

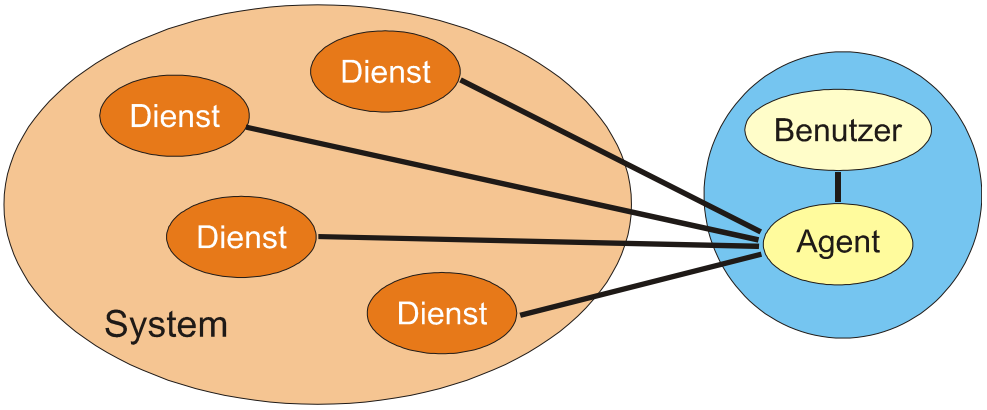

Abbildung 4.17: Agent außerhalb des Systems

Die in Abb. 4.17 dargestellte Variante erfordert einen sehr komplexen Agenten, da dort keine Festlegungen über die Art der Dienste gemacht werden können. Der Agent muss sich jeder Dienstumgebung anpassen können und die gefunden Informationen graphisch oder in textlicher Darstellung umsetzen können. In Kombination mit einem mobilen Gerät ist dies die flexibelste Art der Dienstnutzung. Wird ein Funksystem, wie beispielsweise Bluetooth zur Systemankopplung eingesetzt, kann der Nutzer bereits eine unterstützte Dienstschnittstelle bei der Anbindung seines Gerätes mitgeteilt bekommen.
Diese Umsetzung setzt jedoch voraus, dass jeder Nutzer über einen Agenten verfügt. Dagegen spricht auch, dass die Rechenleistung des mobilen Gerätes der Komplexität des

Agenten nicht gerecht wird. In diesem Fall kann jedoch der Agent auch kurzfristig auf das System übertragen werden, um dort Rechenkapazitäten auszunutzen[38].

4.3 Vergleich und Auswahl

Die beiden Systeme besitzen sehr unterschiedliche Voraussetzungen. Das einstufige System ist nur in sehr begrenzten Umgebungen einsetzbar, also bei Vorhandensein weniger Bussysteme, die alle zentral verwaltet werden können.
Dennoch ist es in der Flexibilität dem mehrstufigen System unterlegen, da hier mehrere Systemaspekte berücksichtigt werden können, darunter auch die Verteilung von Systemressourcen auf mehrere Teilnehmer und Einbindung unterschiedlichster Bussysteme, ohne die Ausfallsicherheit außer acht zu lassen. Durch die Einteilung der Kommunikationstypen gestaltet sich die Integration neuer Bussysteme einfach, da hier auch Protokolle berücksichtigt werden können, die nicht direkt mit dem System interagieren können, sofern ein anderer Teilnehmer in der Lage ist die Protokollübersetzung zu übernehmen. Durch die Staffelung des System in unterschiedliche Übertragungsratenkategorien wird eine Überlastung des übergeordneten Systems bei richtiger Abstufung vermieden. Durch die Integration von Middleware in das Gesamtkonzept kann das System selbstheilend ausgelegt werden, je nach Grad der Unterstützung der Middleware.
Auf Grund der oben genannten Eigenschaften beider Systeme wird für die Realisierung daher der mehrstufige Ansatz ausgewählt.

[38] Dazu muss er jedoch wiederum die Umgebung kennen, um zu wissen auf welche Art und Weise er Rechenzeit auslagern kann. Damit ist diese Methode nicht auf jede beliebige Umgebung anwendbar.

5. Realisierung

Die Grundlagen für die Umsetzung des Gesamtsystems wurden in der vorangegangenen Kapiteln gelegt und können nun umgesetzt werden. Als Testplattform dient das Forschungshaus *tele*-Haus.

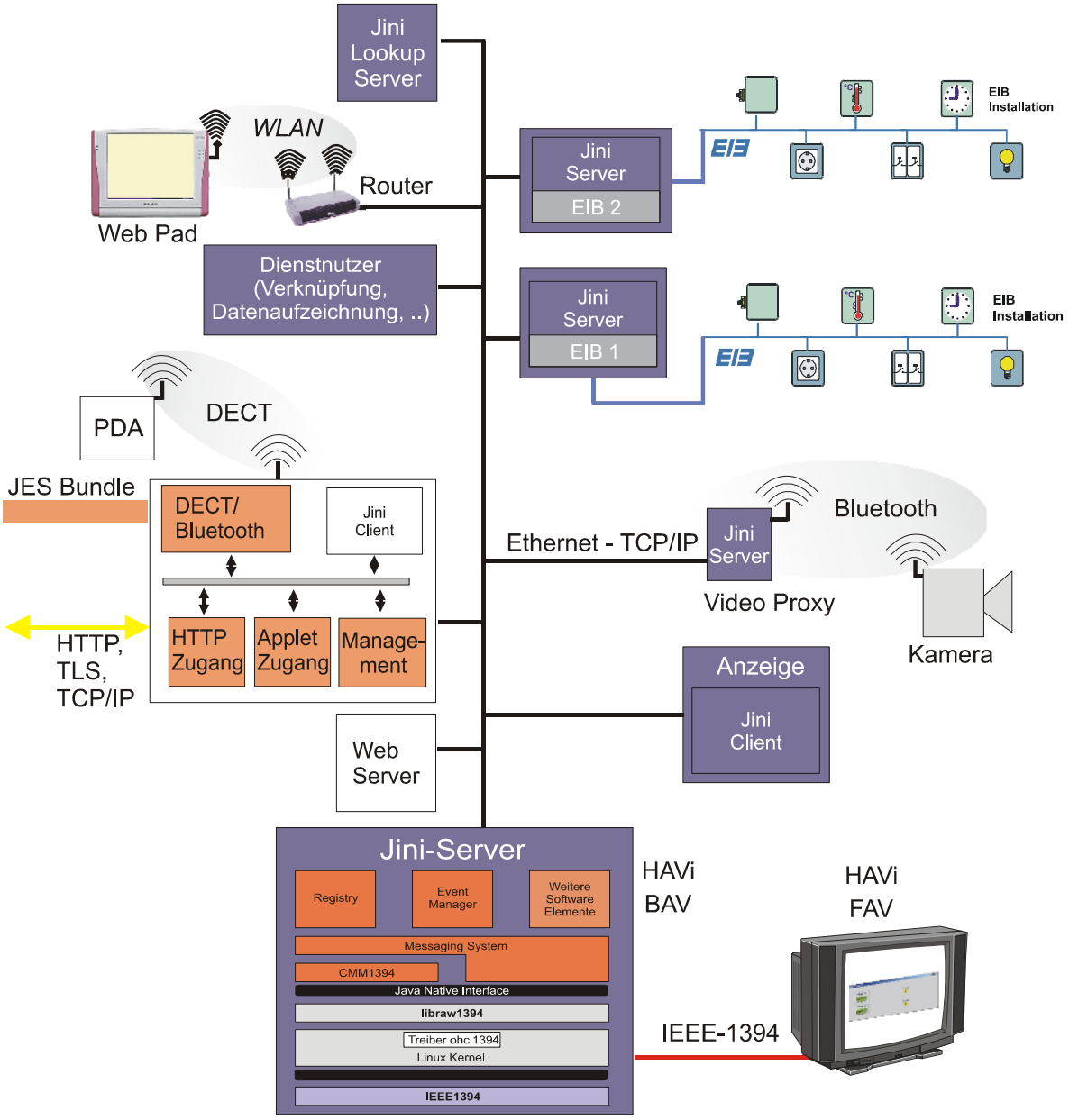

Abbildung 5.1: Gesamtsystem im Überblick[39]

Das gezeigte System verwendet aus der in Kap. 4.2.1 vorgestellten Struktur die Ebene 1 und die Ebene 3. Die Ebene 2 kommt nur indirekt über die Einbeziehung von ISDN vor, das jedoch nur für die transparente Datenweiterleitung zweier Netzwerksysteme verwendet

[39] Alle am Ethernet angeschlossenen Komponenten bis auf den Web Server und den Router enthalten Middleware (Jini).

wird (zum Beispiel für die Einwahl in das hausinterne Netzwerk). Als Zentralbusse (Backbone) wurden Ethernet und IEEE 1394 eingesetzt.

5.1 Gerätebeschreibung

Auf Basis der Abb. 5.1 sollen alle hier verwendeten Geräte und deren Funktion kurz beschrieben werden. Dabei sollen im weiteren Verlauf die Funktionen soweit als möglich in das Gesamtkonzept integriert werden.

5.1.1 EIB-Geräte

Das Haus verfügt über zwei durch Linienkoppler miteinander verbundene EIB-Installationen (Sensor-/Aktorebene, vgl. Kap. 4.2.1.2), die in die Dienstumgebung des Systems eingebettet werden sollen. Dazu wird als Vermittlungsknoten zu dem höherratigen Bussystem (Ethernet) ein PC eingesetzt. Dieser kann gegebenenfalls – also bei Nutzung eines Übergangs mit weniger Ressourcen – durch ein KomTyp 3 oder KomTyp 2 Protokoll an einen Dienstanbieter der höchsten Ebene gekoppelt werden. Zu den Bereichen, die über den EIB gesteuert werden, gehören:

- Beleuchtung
  Die Beleuchtung kann sowohl innerhalb des Hauses als auch im Außenbereich gesteuert werden.
- Temperatursteuerung
  Das *tele*-Haus verfügt über eine dezentrale Lüftung mit Wärmetauschern und ein Wand- bzw. Bodenheizungsssystem. Die Steuerventile der Kreisläufe können ebenfalls über den EIB angesprochen werden.
- Anwesenheitssensorik
  Das Haus verwendet mehrere Systeme zur Überwachung der Personenpräsenz innerhalb von Räumen. Ein sehr einfaches System verwendet einen PIR-Sensor für die Erfassung von Bewegungen; dies entspricht den heute käuflichen Bewegungsmeldern. Ein anderer Sensor verwendet zwar ebenfalls einen PIR, erreicht jedoch durch die Anordnung mehrerer Sensoren und durch ein spezielles Linsensystem eine Auflösung von wenigen Zentimetern. Durch eine nachgeschaltete Auswerteelektronik werden Störungen unterdrückt. Die Präsenz einer Person kann an den angeschlossenen EIB übermittelt werden.
- Verbrauchswerte
  Alle Verbrauchswerte, d.h. Stromverbrauch, Wasserverbrauch und Brauchwassernutzung des Hauses werden erfasst und können über den EIB abgefragt werden.

5.1.2 Videokamera

Die Videokamera ist ebenfalls eine Entwicklung[40] aus dem Projekt *tele*-Haus und wird über Bluetooth angesteuert. Als Basis dazu dient ein VideoProxy, der die Aufgabe der

[40] An der Entwicklung der Kamera waren hauptsächlich die Firmen IFAM GmbH und Binder Elektronik beteiligt.

Umsetzung des Bluetooth-Protokolls auf die Dienstebene übernimmt. Der Aufbau der Kamera ist in Abb. 5.3 zu erkennen. Es handelt sich um einen Mikrosystemstapel, der mehrere stapelbare Ebenen der Größe 40x40 mm verwendet. Der Kamerastapel besteht aus drei Ebenen, die mit Rahmen voneinander getrennt sind. Die mit Bauteilen bestückten Ebenen werden über am Rand angebrachte Durchführungen durch den gesamten Stapel geleitet und können so miteinander verbunden werden.

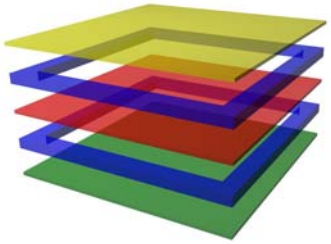

Abbildung 5.2 Stapelbausweise des Kameramoduls

Abb. 5.2 zeigt den schematischen Aufbau des Stapels. Beim Kameramodul wurde eine Ebene zur Bluetooth-Kopplung und Datenübertragung (Abb. 5.3 links oben), eine Ebene zur Signalverarbeitung mit einem DSP (Abb. 5.3 rechts oben) und eine Ebene zur Aufbringung des CCD Sensors (Mitte unten, ohne angebrachtes Linsensystem) verwendet.

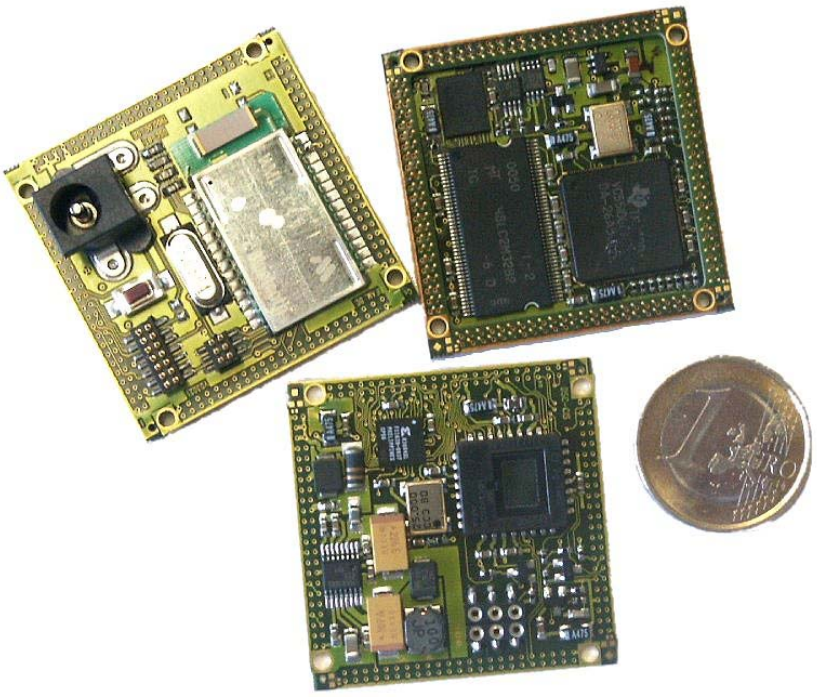

Abbildung 5.3 Kamera Mikrosystemstapel

Das DSP Modul komprimiert die Bilddaten des CCD-Sensors und übermittelt diese auf Anfrage als JPEG-Graphiken über das Bluetooth-Modul.

5.1.3 Fernseher

Die Einbindung des Fernsehers in das System erfolgt über eine Set-top Box. Eine Set-top Box ist ein Embedded PC für Multimediaanwendungen. Er verfügt über spezielle Schnittstellen (IEEE 1394, TV Out, PCMCIA-Steckplätze, DVD-ROM Laufwerk) und wird in Verbindung mit einem Ausgabegerät (LCD, Fernseher) eingesetzt.
Der Fernseher wird ausschließlich zur Ausgabe des Bildes der Set-top Box verwendet und besitzt keine Laufzeitumgebung. Dieser Schritt war notwendig, da ein Fernseher mit den gewünschten Eigenschaften – IEEE 1394 Anschluss und HAVi-konform – zum Zeitpunkt der Ausarbeitung des Konzepts noch nicht verfügbar war. Mittlerweile gibt es ein Produkt, das jedoch ca. 12.000 US-Dollar kostet und damit weit oberhalb eines annehmbaren Anschaffungspreises liegt. Dadurch ergibt sich folgende Struktur:

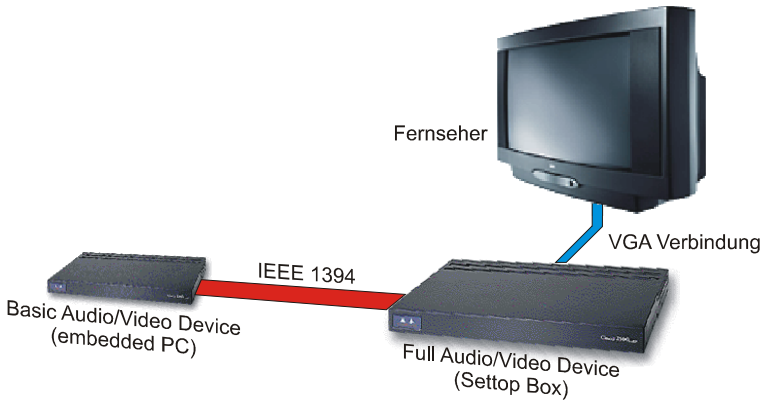

Abbildung 5.4: Verwendung der Set-top Box

Die Set-top Box wird in dieser Anordnung über IEEE 1394 mit dem BAV verbunden. Damit sind die Voraussetzungen für die Umsetzung von HAVi geschaffen.

5.1.4 Web Pad

Für die mobile Anzeige und Steuerung steht im *tele*-Haus ein Web Pad zur Verfügung, das eine Anzeigefläche und eine Ausführungsumgebung für Java Programme sowie einen Internetbrowser zur Verfügung stellt. Das Web Pad lässt sich ohne zusätzliche Eingabegeräte über einen integrierten Touch Screen bedienen. Der enthaltene Akku erlaubt eine autonome Verwendung bei gleichzeitiger Anbindung an das System über eine PCMCIA Wireless LAN Einsteckkarte. Die Anbindung an das System wurde über einen Wireless LAN Access Point erreicht, der als Router eingesetzt wurde und damit für das Funksystem ein eigenständiges Netzwerksegment darstellt.

5.1.5 PDA

Das kleinste Gerät mit einer aktiven, einfarbigen Anzeigefläche ist der im System verwendete PDA. Es wurde ein Palm Vx verwendet, der mit einem Dragonball Prozessor und installiertem Palm OS ausgeliefert wird. Die Hauptanwendungen sind in einem nichtflüchtigen ROM untergebracht (2 MByte), während der RAM-Bereich (8 MByte) gegebenenfalls als Programmspeicher zur Ablage weiterer Anwendungen verwendet

werden kann. Das RAM wird durch einen Lithium-Ionen-Akku gepuffert, der auch zur Stromversorgung des Geräts im eingeschalteten Betrieb dient.

Abbildung 5.5: Palm Vx

Der berührungsempfindliche Bereich erstreckt sich auf die komplette sichtbare Anzeigefläche. Ein Bereich ist für die laufenden Anwendungen, einer für Tasten und zwei weitere Bereiche sind für die Schrifterkennung reserviert. Die Eingabe wird wie beim Web Pad über einen beigelegten Spezialstift abgewickelt.

5.2 Bussystemankopplung

Damit das in dieser Arbeit erstellte und umgesetzte Systemkonzept auf andere Systeme übertragbar ist, muss auch die Umsetzung des Modells möglichst wenig hardwarespezifisch ausgeführt werden. Erst dann lässt das Konzept eine Vielzahl von Geräten und Diensten zu, und ist gleichzeitig noch auf verschiedenen Zielplattformen einsetzbar. Zu diesem Zweck wurde ein Großteil der Implementierung in Java ausgeführt. Anwendungen, die ausschließlich auf Schnittstellen zugreifen, die durch Java bzw. die JVM spezifiziert sind – dazu gehören auch die graphischen Benutzeroberflächen –, wurden vollständig in Java realisiert. In diesem Kontext steht Java den anderen, direktausführenden Sprachen nicht nach und der Performance-Unterschied ist minimal. Durch die geräteunabhängige Sprachdefinition kann jedoch eine hardwareunabhängige Implementierung nur bis zu einem gewissen Grad durchgeführt werden. Es sind durchaus Mechanismen in Java vorgesehen, um auch über die Grenzen der *Java Virtual Machine* (JVM) hinaus operieren zu können (über das *Java Native Interface* - JNI[41]), dabei bindet man sich jedoch wieder an das eingesetzte Betriebssystem. Dies ist insbesondere beim Zugriff auf bestimmte Gerätetreiber des Betriebssystems notwendig. Die Standardschnittstellen eines PCs werden durch Java bereits unterstützt. Dazu gehören die graphische Ausgabe, Textausgabe, Netzwerkschnittstelle und serielle/parallele Schnittstelle. Anwendungen, die andere Betriebssystemressourcen verwenden wollen, müssen dies über die Einbindung eigener Treiber bzw. Schnittstellen machen.

[41] Vgl. auch [Liang1999].

5.2.1 Ethernet

Zusammen mit IEEE 1394 bildet das Ethernet Netzwerk das Rückgrat des Systems. Dabei wird Ethernet durch TCP/UDP-IP erweitert. TCP/UDP-IP kann vielseitig eingesetzt werden und trägt in diesem System Daten der Middleware und unterstützt Javas RMI.
Es wurden bereits einige verfügbare Middleware-Lösungen vorgestellt, von denen für die weiteren Anwendungen Jini eingesetzt wird. Jini besitzt folgende Hauptmerkmale:

- Dienstorientierung
  Alle Ressourcen innerhalb eines Jini-Netzwerks werden in Diensten zusammengefasst, die in die Gruppierungen einzuordnen sind, die in Kap. 4.2.2 aufgestellt wurden.
- Netzwerkorientiert
  Alle Jini-fähigen Geräte verfügen über Kommunikationsmechanismen zur verteilten Objekt- und Ereignisverwaltung. Diese findet mit Hilfe von Remote Method Invocation statt.
- Dienstverwaltung
  Alle Dienste eines Jini-Netzwerks werden durch den Lookup Service bekanntgegeben und ihr Dienstort an einer oder mehrerer Stellen vermerkt. Die Verwaltung der Dienste übernimmt eine oder mehrere Instanzen des Lookup Servers.

Die Dienstkommunikation von Jini wird mittels RMI abgewickelt. RMI ist ein zentraler Bestandteil der Jini-Implementierung, da sowohl die Dienstdefinition, als auch die Dienstanmeldung und Dienstnutzung auf RMI angewiesen sind.

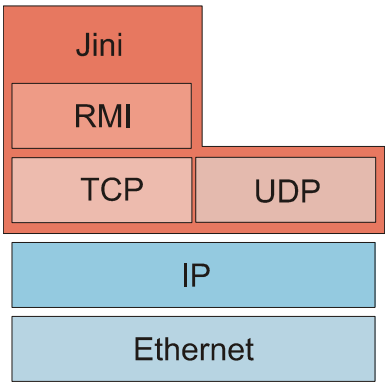

Abbildung 5.6: Kommunikationsstruktur von Jini

Nicht alle Protokolle von Jini nutzen RMI. So wird für das Auffinden des Lookup Servers ein UDP-IP Multicast-Telegramm versendet.
RMI verwendet für den Datentransport in der Standardimplementierung TCP. Zur Objektkommunikation wird client- und serverseitig das aus Remote Procedure Call bekannte Verfahren der Stubs und Skeletons eingesetzt, die als Stellvertreter den Datentransport und die Parameter an die darunterliegenden Schichten für das Marshalling und Unmarshalling übernehmen. Bevor die Parameter eines RMI-Aufrufs übertragen werden, findet eine Serialisierung statt, also die Umformung der Daten in einen Datenstrom. Dieser Vorgang

wird *Marshalling* genannt. Der Empfänger der Daten gewinnt aus dem Datenstrom wieder Objekte bzw. Referenzen. Dieser Vorgang wird als *Unmarshalling* bezeichnet
Die Funktionsweise von Jini soll an einem einfachen Beispiel erläutert werden: Ein Jini-fähiger Lautsprecher und ein digitales Abspielgerät sind innerhalb eines Jini-Netzwerkes miteinander verbunden. Diese zwei Komponenten entsprechen einer klassischen Client/Server- Anwendung. Die Daten des Abspielgeräts können auf dem Lautsprecher ausgegeben werden. Diese Funktion konnte bisher auch über eine feste Verdrahtung gelöst werden. Ist jedoch das Abspielgerät mobil und wird damit zum ersten Mal in der Jini-Umgebung integriert, so kann keine feste Verknüpfung vorgesehen werden, da die Kommunikationswege noch nicht bekannt sind.

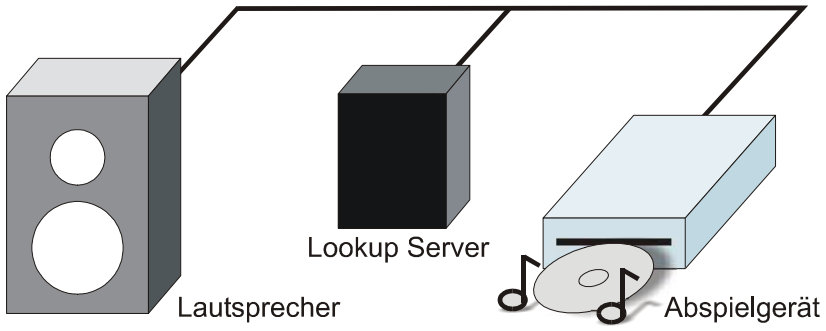

Abbildung 5.7: Eine mögliche Jini-Konfiguration

Neben dem Jini-fähigen Lautsprecher und dem Abspielgerät befindet sich noch eine weitere Komponente in Abb. 5.7 – der Lookup Server. Der Lookup Server übernimmt die Verwaltung der Dienste. Der gesamte Vorgang besteht aus folgenden Schritten (Zahlen in Klammern verweisen auf Abb. 5.8):

- Discover (1)
  Wird ein Gerät in das Netzwerk eingebracht, das einen Dienst zur Verfügung stellt, so sucht es als erstes den Lookup Server über ein Multicast-Telegramm. Dabei ist der Lookup Server selbst als Dienst (Lookup Service - 2) im Netzwerk verfügbar.
- Join (3)
  Der Dienstanbieter meldet dann seinen Dienst beim Lookup Server an. Zusammen mit dem Aufenthaltsort des Dienstes (genau genommen ist dies eine Remote Reference) können Eigenschaften angegeben werden, die eine Suche nach diesen Kriterien erlauben. Damit ist der Dienst von allen Teilnehmern des Netzwerks auffindbar und nutzbar.
- Discover (4)
  Der Dienstnehmer sucht den Lookup Service.
- Lookup (6)
  Mit Hilfe des Lookup Services sucht der Dienstnehmer den Dienst unter Angabe der gewünschten Eigenschaften oder aber durch gezielte Angabe des gesuchten Services.
- Receive (7) / Use (8)
  Mit der vom Lookup Service erhaltenen Referenz wird der Dienst genutzt.

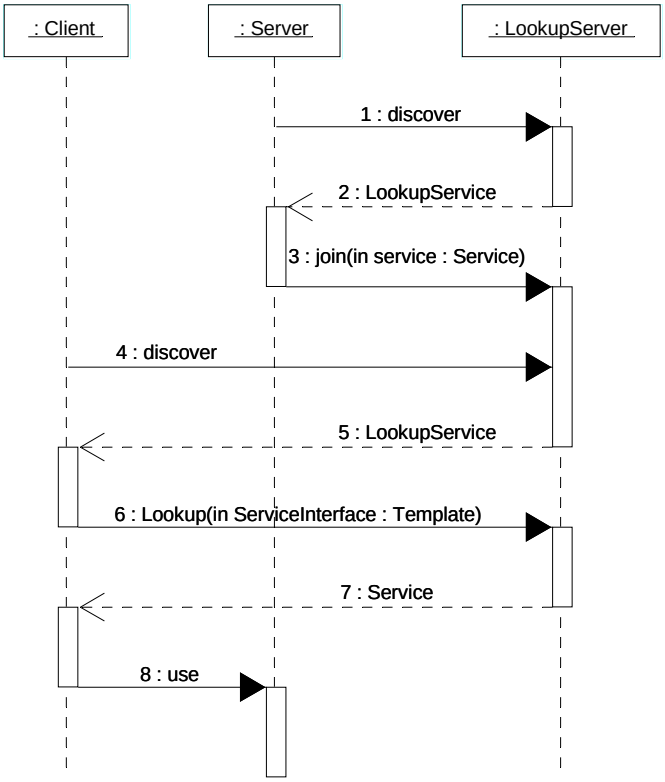

Abbildung 5.8: Jini-Protokoll für Client/Server/LookupServer

Die Dienste, die in das Netzwerk eingebracht werden, können frei definiert werden. Es gibt keine Vorgabe über den Aufbau der Dienstschnittstellen. Die Implementierung der Dienste ist nur an wenige Randbedingungen gebunden. Dienste werden als Referenzen auf Objekte mit (selbst-)definierten Schnittstellen umgesetzt.

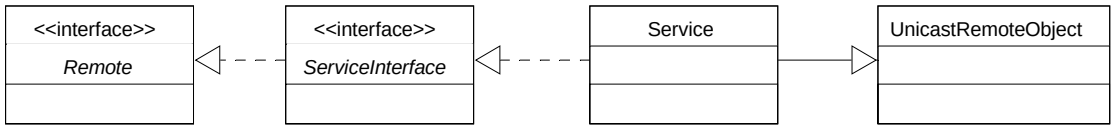

Abbildung 5.9: Dienstimplementierung bei Jini

Die Klasse mit der Bezeichnung *Service* ist das eigentliche Dienstobjekt, das beim Lookup Server angemeldet wird. Das *ServiceInterface* beschreibt die Methoden des exportierten Dienstes und das Interface *Remote* dient nur als Markierung innerhalb von Java zur Kennzeichnung von RMI-Methoden, ohne Operationen zu definieren. Das *UnicastRemote-Object* übernimmt alle Funktionen für den RMI-Export (Anlegen der TCP-Verbindungen, Lesen und Schreiben des Objekts, Erzeugung der Stubs etc.). Bei der Verwendung von RMI werden niemals die Schnittstellen von Klassen direkt sichtbar. Die Fernaufrufe beziehen sich auf die implementierten Schnittstellen und nicht auf die Klassen-implementierung, was eine strikte Trennung zwischen Schnittstelle und Implementierung einer Klasse erlaubt. Dementsprechend sind bei Jini alle Dienste über Interfaces erreichbar;

jede Referenz auf ein Objekt lässt nur Rückschlüsse auf die verwendete Schnittstelle zu, ohne jedoch weitere Operationen des Objekts offenzulegen, die nicht in der Schnittstelle enthalten sind.

5.2.1.1 Basisdienst

Nach der Vorgabe aus Kap. 4.2.2 soll nun der Basisdienst für das Jini-Netzwerk festgelegt werden. Es wurde bereits ein Muster des Basisdienstes erstellt, das mehrere allgemeine Operationen umfasst, die grundlegende Funktionen eines Dienstes zugänglich machen. Diese werden im nachfolgenden Klassendiagramm abgebildet:

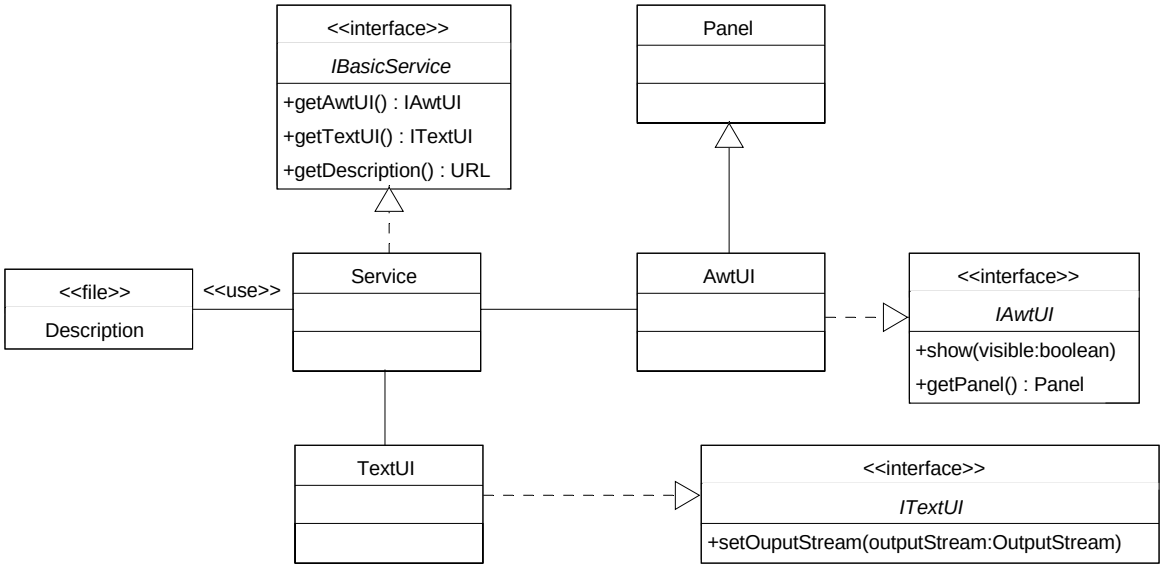

Abbildung 5.10: Der Basisdienst in Jini

Der Basisdienst wird im Wesentlichen durch Schnittstellen definiert. Die Implementierung erfolgt durch die abgeleiteten Dienste. Daher wurde in der obigen Abbildung eine Klasse namens *Service* eingeführt, die hier das prinzipielle Vorgehen veranschaulichen soll. Das Interface *IBasicService* bietet drei Methoden an: zwei interaktive Schnittstellen und einen Verweis auf die Dienstbeschreibung. Die interaktiven Schnittstellenobjekte sind hier durch *IAwtUI* und *ITextUI* gegeben. Sie bieten Methoden zur graphischen oder textlichen Ausgabe von Informationen an. Bei der graphischen Variante wird ein Standardelement aus dem Java *Abstract Window Toolkit* (AWT) zurückgeliefert, das so auch in jede Anzeige integrierbar ist. Die Methode *getDescription* hat hier einen besonderen Stellenwert. Sind die Dienste dem Nutzer bekannt, werden keine weiteren Beschreibungen benötigt. Der Zugriff erfolgt über festgelegte Algorithmen innerhalb des Dienstnehmers. Wird jedoch der Dienst von einem Teilnehmer verwendet, der den Zweck der Methoden nicht im Voraus weiß, so kann er alle notwendigen Informationen über die Beschreibung des Dienstes erhalten.
Über die Methode *getDescription* erhält er zunächst nur eine URL. Mit dieser URL kann er auf eine XML-Datei zugreifen. Die Struktur der Datei wird wiederum durch eine

Document Type Declaration[42] (DTD) vorgegeben. Die DTD ist eine Art Bauanleitung für eine XML-Datei oder aber anders ausgedrückt eine XML-Datei ist eine Instanz einer DTD.

```
<?xml version='1.0' encoding='UTF-8'?>
<!ELEMENT Service (ServiceID, ServiceName, ServiceVersion, ServiceVendor, ServiceDescription, ServiceInterface)>
<!ELEMENT ServiceInterface (Operation+)>
<!ELEMENT Operation (OperationName, OperationDescription, OperationParameter*, OperationReturnValue)>
<!ATTLIST Operation
                        AccessDirection (Consume|Produce) #REQUIRED
                        System (true|false) "true"
                        Monitoring (true|false) "false"
                        Human (true|false) "false">
<!ELEMENT OperationParameter (OperationParameterType, OperationParameterDescription)>
[…]
<!ENTITY BasicServiceInterface
[…]
    <Operation AccessDirection='Produce' Human='true'>
      <OperationName>getAwtUI</OperationName>
      <OperationDescription>Get a graphical User Interface. </OperationDescription>
      <OperationReturnValue>
        <ParameterType>AwtUI</ParameterType>
        <ParameterDescription>Graphical presentation of this service</ParameterDescription>
      </OperationReturnValue>
    </Operation>

    <Operation AccessDirection='Produce' Human='true'>
      <OperationName>getDescription</OperationName>
      <OperationDescription>Description of all methods and parameters in XML</OperationDescription>
      <OperationReturnValue>
        <ParameterType>URL</ParameterType>
        <ParameterDescription>URL to description of this service</ParameterDescription>
      </OperationReturnValue>
    </Operation>">
```

Abbildung 5.11: DTD für alle Dienste im System[43]

Im oberen Teil der Abb. 5.11 erkennt man die Strukturdefinition eines Dienstes. Jede Dienstbeschreibung enthält allgemeine Informationen wie die ServiceID, den Namen des Herstellers, eine kurze Beschreibung und eine Beschreibung jeder Methode inklusive der Parameter und Rückgabewerte. Ein besonders wichtiger Abschnitt ist die Attributdefinition der Methode. Unter anderem wird dort der Informationsfluss der Methode beschrieben; *Produce* bedeutet die Verarbeitung und Ausgabe von Daten, *Consume* die Aufnahme von Daten. Durch den Zusatz #REQUIRED wird die Anwesenheit dieses Attributs in der XML-Datei zwingend vorgeschrieben. Daneben findet eine Einteilung dieser Methode in Benutzerkategorien statt, die Auskunft über den Verwendungszweck gibt. Es wurden vier Kategorien verwendet, die in der nachfolgenden Tabelle zusammengefasst werden.

[42] Dies ist ein Standard des World Wide Web Consortium (W3C), dass neben DTD viele andere Standards hervorgebracht hat. Zu einigen zählen DTD, HTML, XML, XLS, die sich alle mit Metasprachen/Markup sprachen beschäftigen.

[43] Alle Elemente, deren Definition hier nicht aufgeführt sind wurden der Übersichtlichkeit halber weggelassen und sind wie folgt festgelegt: „<!ELEMENT name (#PCDATA)>“.

Tabelle 5.1: Kategorisierung der Methoden innerhalb der Dienste

Kategorie	Beschreibung
System	Alle Methoden sind vom Typ *System*. Sie können bei genauem Wissen über die Folgen der Zustandsänderung des Dienstes eingesetzt werden.
Monitoring	Mit den ein- bzw. ausgegebenen Daten kann eine Überwachungsfunktion ausgeführt werden. Ein Beispiel dafür ist die Aufzeichnung von Verbrauchswerten des EIB.
Human	Im weiteren Verlauf wird die Auswertung der Dienstbeschreibung ihren besonderen Nutzen bei der Verwendung von Agenten zeigen. Diese Methoden sind direkt nutzbar, meistens ohne den Zustand des Dienstanbieters vorher ändern zu müssen. Alle Methoden, die eine Anzeige für einen Bediener zurückgeben, zählen zu dieser Kategorie.

Der letzte Abschnitt der DTD definiert eine ENTITY, also eine Schablone, die in einem XML-Dokument über Referenzen eingefügt werden kann. Jeder Dienst muss diese Referenz in seiner XML-Beschreibung aufnehmen und die Basisdienste anbieten. Wie oben bereits erwähnt, gibt es keinen selbständigen Basisdienst, da er immer durch andere Dienste implementiert wird. Daher gibt es auch keine Instanz dieser DTD für den Basisdienst.
Da die Beschreibungsdatei über eine URL indiziert wird, gibt es keine Notwendigkeit, diese am Dienstort zu hinterlegen.

5.2.1.2 Informationsdienst

Aus der Vielzahl der angebotenen Dienste ergibt sich auch die Notwendigkeit der Datenspeicherung von Informationen. Diese Aufgabe kann von einem oder mehreren Diensten erledigt werden.
Ein Dienst wird zum Verwalten von Dateien jeglicher Art verwendet. Diese können über entsprechende Methoden kategorisiert und eingeordnet werden. Es können sowohl Informationen hinzugefügt, als auch gelöscht werden. Für diese Aufgabe sind nur wenige Methoden zuständig.

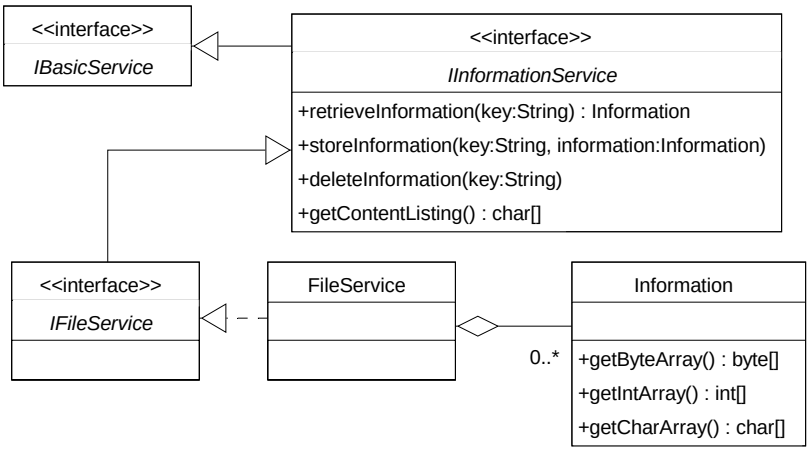

Abbildung 5.12: Dateidienst

Für diesen Dienst wird zunächst die allgemeine Schnittstelle *IInformationService* definiert, die nur vier Zugriffsmethoden enthält. Jede Information wird zusammen mit einem Schlüssel gespeichert, der dann auch als Zugriffsmuster gilt. Der *FileService* ist eine Spezialisierung der Information, der dazu dient, Dateien zu speichern. Die Methoden implementieren dieses Verhalten. Der Schlüssel ist in diesem Fall eine URL, über die die Daten abgefragt werden können. Dabei spiegelt die URL ein virtuelles Dateisystem wider, das auf die lokale Datenbasis des *FileServices* abgeglichen werden muss. Eine Übersicht der gespeicherten Dateien erhält man durch *getContentListing*. Die eigentlichen Daten können in unterschiedlichen Datenformaten abgefragt werden. Für diesen Dienst soll hier nur ein Ausschnitt aus der XML-Datei gezeigt werden, die die Dienstschnittstellen beschreibt.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Service SYSTEM "ServiceDescription.dtd">
<Service>
   <ServiceID>0x2</ServiceID>
   <ServiceName>FileInformationService</ServiceName>
   <ServiceVersion>v 1.0</ServiceVersion>
   <ServiceVendor>MST</ServiceVendor>
   <ServiceDescription>Low level Service providing general operations</ServiceDescription>
   <ServiceInterface>
      <!-- import basic service -->
      &BasicServiceInterface;
      <!-- define additional service -->
      <Operation AccessDirection='Produce' Monitoring='true'>
         <OperationName>retrieveInformation</OperationName>
         <OperationDescription>Get a file accessible through key (URL)</OperationDescription>
         <OperationParameter>
            <ParameterType>URL</ParameterType>
            <ParameterDescription> URL to information</ParameterDescription>
         </OperationParameter>
         <OperationReturnValue>
            <ParameterType>Information</ParameterType>
            <ParameterDescription>contains requested information</ParameterDescription>
         </OperationReturnValue>
      </Operation>
      …
   </ServiceInterface>
</Service>
```

Abbildung 5.13: Ausschnitt aus der Schnittstellenbeschreibungsdatei

Es wurden in Abb. 5.13 nicht alle Methoden berücksichtigt, dennoch veranschaulicht dieser Ausschnitt die Vorgehensweise.
Eine weitere Spezialisierung ist der Aufzeichnungs- und der E-Mail-Dienst. Der Aufzeichnungsdienst bietet eine zusätzliche Methode (*trackLogData*) an, die eine periodische Abfrage von anderen Diensten ermöglicht und das Ergebnis dann anschließend speichert. Dieser Mechanismus sollte in der Regel durch eine ereignisgesteuerte Verknüpfung erfolgen, d.h. der Dienstnehmer ruft bei Bedarf den Informationsdienst zum Speichern von Daten auf.

Bei Aufruf von *trackLogData* wird periodisch ein Abfrage der Quelle (*source*) ausgelöst, wobei die Intervallzeit übergeben wird. Dabei müssen die in *source* aufzurufende Methode und der Rückgabetyp bekannt sein, damit die Daten anschließend unter dem angegebenen Schlüssel gespeichert werden können.

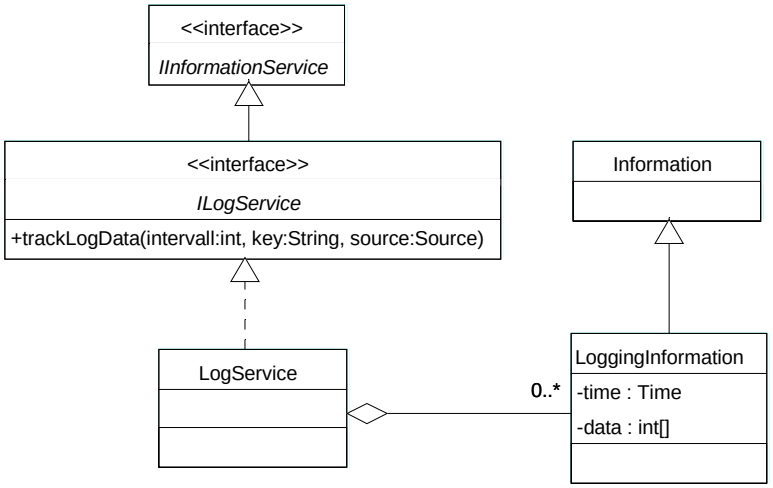

Abbildung 5.14: Aufzeichnungsdienst, vereinfacht

In Abb. 5.14 wurden Vereinfachungen vorgenommen. Die mitgeschriebene Information wird in diesem Beispiel als Integer-Array gespeichert. Beschränkt man sich nur auf die primitiven Datentypen, so müssten hier weitere Attribute eingeführt werden. Damit wird der Zugriff auf die Information auch wieder schlüssig, da in *Information* die entsprechenden Zugriffsfunktionen vorgesehen sind.

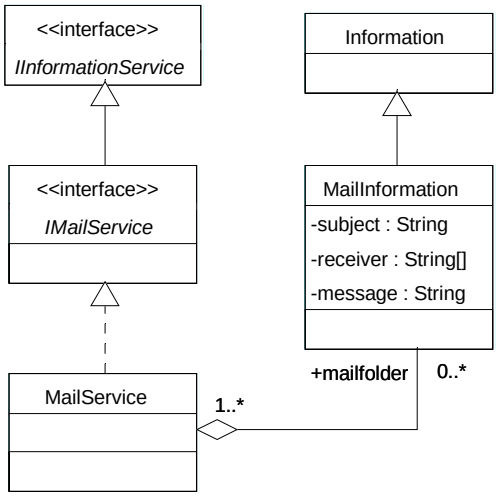

Abbildung 5.15: E-Mail-Dienst

Der E-Mail-Dienst überschreibt lediglich die vorhandenen Methoden. Er kann dabei verschiedene Nachrichtenordner verwalten, die wiederum mehrere Nachrichten speichern können. In der jeweiligen E-Mail-Information sind jeweils Angaben über den Empfänger und den Betreff sowie die Nachricht selbst enthalten.

Dieser Dienst ist besonders nützlich, um kurze Mitteilungen an den Anwender zu schicken, sei es, um Warnmeldungen weiterzugeben oder auch Hinweise anderer Art, beispielsweise in Verbindung mit einem selbstkonfigurierendem Jini E-Mail-Client.

5.2.1.3 Lokalisierungsdienst

Die Vielseitigkeit der Dienste erlaubt nun auch eine flexiblere Gestaltung der Benutzerinteraktion. Bei vielen Konzepten ist der Benutzer an festgelegte Standorte gebunden, um Informationen über das System zu erhalten. Durch die Nutzung von mobilen Geräten lässt sich die Bindung an bestimmte Dienstorte nicht nur bei der Nutzung aufheben, sondern auch bei der Darstellung der Dienste. Man kann die Dienstnutzung sogar noch weiter einschränken, indem man personenbezogene Daten berücksichtigt. Durch die Personalisierung des Datenzugriffs kann das Wissen des Anwenders in das System eingebracht werden. Dadurch werden nur die Dienste und Informationen angezeigt, die auch im Interesse des Benutzers liegen. Neben der Verwaltung der benutzerspezifischen Daten, die durch einen Informationsdienst gehandhabt wird, ist auch der Aufenthaltsort der Person von Belang. Da im Regelfall die Gebäude nicht umgebaut werden können bzw. sollen oder der Aufwand zu kostspielig ist, wird hier eine Lösung vorgestellt, die ohne große Umbaumaßnahmen auskommt.
Der Aufenthaltsort einer Person lässt sich passiv zumeist nur recht aufwendig feststellen. Dazu kommen verschiedene Messmethoden zum Einsatz: Bewegungsmelder über passive Infrarotempfänger, Kamerasysteme mit Auswertesoftware, Mikrowellensender/-empfänger oder Schall.
Einfacher wird es hingegen, wenn die Person selbst ein aktives, kodiertes Signal aussendet, das dann ausgewertet wird. Auch hier sind viele Arten von Signalen möglich. Dabei muss ein Kompromiss zwischen der Genauigkeit des Verfahrens, der Reichweite, dem Energiebedarf und den Bauteilkosten gefunden werden. Die Reichweite ist ein besonderer Punkt. Vorstellbar ist die Verwendung eines Minisenders, der an einer Person angebracht wird, beispielsweise am PDA. Über ein Triangulationsverfahren wird ihr Aufenthaltstort bestimmt. Eine wesentlich günstigere Variante, wenn auch nicht so genau, ist die Verwendung eines Infrarotsenders, der periodisch ein Signal sendet, dass dann aufbereitet weitergeleitet wird. Dadurch ergeben sich mehrere Vorteile: Die Reichweite des IR-Senders ist begrenzt und der Aufenthaltsort einer Person lässt sich eindeutig auf einen Raum innerhalb des Gebäudes zurückführen. Damit können die verfügbaren Dienste des Gesamtsystems auf die des Raumes reduziert werden. Wird an einer oder mehreren Stellen innerhalb eines jeden Raumes ein IR-Empfänger angebracht, so benötigt man lediglich einen kleinen Sender, der mit sehr wenig Bauteilaufwand produziert werden kann.

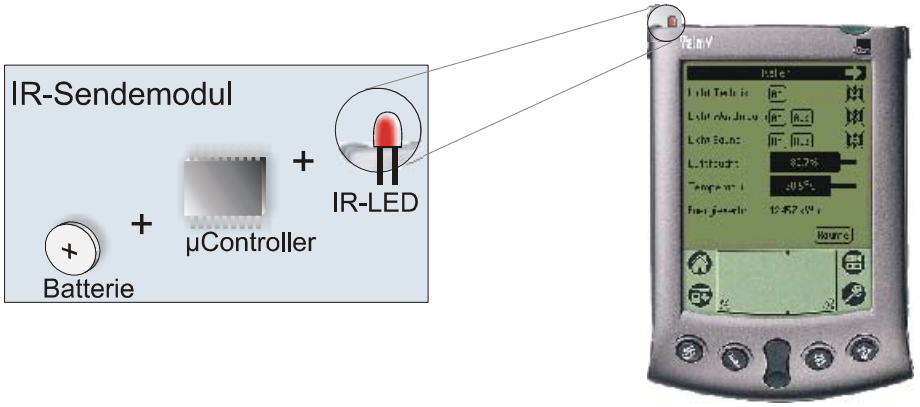

Abbildung 5.16: Aufbau des Senders

Der Aufbau des Senders besteht aus wenigen Bauteilen und wurde energiesparend umgesetzt. Der IR-Empfänger ist direkt an den EIB angeschlossen und verarbeitet die Signale. Zur Verbesserung der Übertragung wird das Signal auf einem 38kHz Trägersignal amplitudenmoduliert. Der Empfänger dekodiert es, überprüft die Datenintegrität und versendet ein Telegramm mit der Personenidentifikationsnummer. Die Zuordnung zu einem Raum erfolgt über die Gruppenadresse des EIB-Telegramms. Der Empfänger wird ohne zusätzliche Stromversorgung betrieben und versorgt sich aus dem EIB.
Zu diesem Aufbau wurde ein Dienst implementiert, der einige Hauptinformation zur Verfügung stellt.

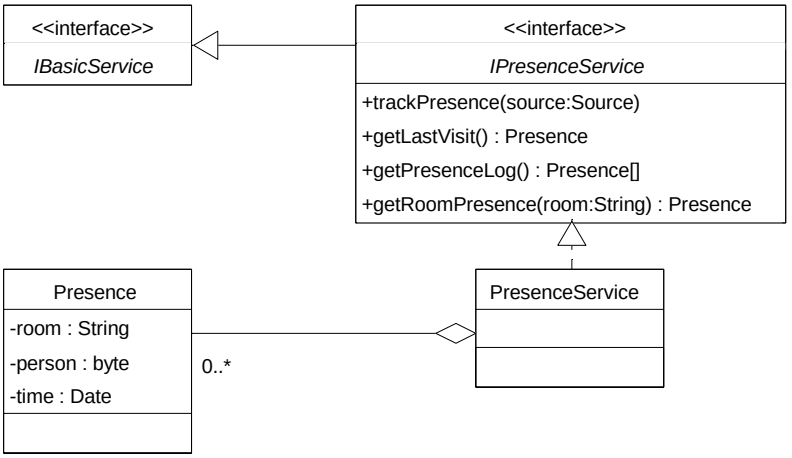

Abbildung 5.17: Anwesenheitsdienst

Betritt eine Person einen Raum, so kann auch eine Benachrichtigung an einen Dienstnehmer verschickt werden, indem sich der Dienstnehmer über *trackPresence* anmeldet. Die weiteren Methoden dienen nur zur Abfrage gezielter Informationen über die Anwesenheit in bestimmten Räumen (*getRoomPresence*) oder aber einer Verlaufsliste.

5.2.1.4 Visualisierungsdienst

Zur Anzeige wichtiger Informationen oder auch von Teildiensten wurde dieser Dienst eingeführt, der eine Schnittstelle zur Informationsdarstellung anbietet.

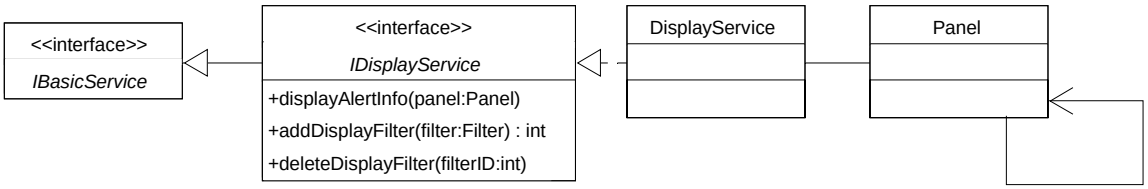

Abbildung 5.18: Anzeige der verfügbaren Dienste innerhalb des Systems

Der Dienst nutzt die Basisschnittstelle zur Erlangung der Dienstansichten. Alle darstellbaren Dienste werden in Kategorien eingeordnet, aus denen der Anwender eine Auswahl treffen kann. Nach Auswahl der anzuzeigenden Elemente werden automatisch deren graphische Darstellung und damit auch die erforderlichen Klassen zur Steuerung entsprechend der Umsetzung des Basisdienstes geladen. Die Anordnung der Elemente erfolgt durch einen Layout Manager, so dass kein Eingriff des Benutzers notwendig ist. Durch die Funktion *addDisplayFilter* kann die Anzeige durch Filterung auf bestimmte Dienste eingeschränkt werden. Der Rückgabewert der Funktion wird zur Entfernung des Filters als *filterID* benötigt.

5.2.1.5 Auslastungsanalyse

Aus der Verwendung mehrerer Zwischenschichten für Informationen der Anwendungsebene ergibt sich ein wachsendes Datenvolumen. Da Jini als Middleware eingesetzt wurde, müssen hier die Übertragungsmechanismen betrachtet werden, die bei der Nutzung eines Dienstes zum Einsatz kommen. Jini unterstützt den Anwender hauptsächlich beim Auffinden von Diensten innerhalb eines Netzwerkes und stellt zudem Dienste zur Verfügung. Abgesehen von UDP Multicast-Nachrichten, die der Lokalisierung des Lookup Dienstes dienen, werden bei Jini nach Erhalt einer Dienstreferenz die Methodenaufrufe über RMI ausgeführt. RMI übernimmt dabei den Verbindungsaufbau, die Umformung der Daten und die Entgegennahme der Rückgabewerte. Alle RMI-Aufrufe werden durch [SUNRMI] beschrieben. Im Wesentlichen lässt sich die Kommunikation zwischen zwei JVMs durch RMI auf den Aufbau einer TCP/IP-Verbindung zurückführen, die zur Übermittlung des RMI-Protokolls verwendet wird. Alle Parameter müssen serialisierbar sein oder werden in ein besonderes Format – dies ist bei primitiven Datentypen der Fall – umgewandelt. Danach werden die Objekte in einen Datenstrom überführt (serialisiert) und innerhalb des RMI-Protokolls eingebettet.
Daher kann die Dienstnutzung auf den Aufruf einer Funktion zurückgeführt werden. Nachfolgend soll hier beispielhaft das Datenaufkommen beim Aufruf eines Objektes gezeigt werden. [SUNRMI] spezifiziert das RMI Wire Protocol, das hier nur kurz beschrieben werden soll.
Die Kommunikation wird in der Spezifikation aus der Sicht des Clients beschrieben. Ausgehende Nachrichten können in zwei Formaten vorliegen, als RMI Wire Protocol Anfrage oder gekapselt in einer HTTP-Anfrage. Die Kapselung innerhalb von HTTP dient der Überwindung von Firewalls, die nur bestimmte TCP-Ports zulassen. Über eine

Common Gateway Interface (CGI) Skript wird dann die HTTP-Anfrage an einen HTTP RMI-Server weitergereicht, der die Antwort als HTTP-response ausgibt. Der Aufbau einer Nachricht ohne Verwendung von HTTP sieht wie folgt aus:

Header Message(s)

Die Nachrichtenelemente können der folgenden Tabelle entnommen werden:

Tabelle 5.2: Element des RMI Wire Protocols

Element	Inhalt	
Header	0x4a 0x52 0x4d 0x49 Version Protocol	
Version	0x00 0x02	
Protocol	StreamProtocol SingleOpProtocol MultiplexProtocol	0x4b 0x4c 0x4d
Message	Call Ping DgcAck	0x50 CallData 0x52 0x54 UniqueIdentifier
UniqueIdentifier	*Number Time Count*	
CallData	ObjectNumber UniqueIdentifier Operation Hash $Argument_{opt}$.	
Argument	*Object* *Primitive*	

Die Formate der kursiv gedruckten Bezeichnungen sind von der Transportschicht abhängig und werden in der Spezifikation nicht erläutert. Die Transportschicht für RMI ist jedoch Teil der JVM und es ist kein Sourcecode verfügbar, so dass sich das Protokoll einer genaueren Untersuchung weitgehend entzieht.
In der Tabelle ist zu erkennen, dass innerhalb des RMI-Protokolls unterschiedliche Verbindungstypen unterstützt werden. So können auch Multiplex-Verbindungen aufgebaut werden, die über eine Verbindung Anfragen an unterschiedliche Server-Objekte ermöglichen, so dass nicht zu jedem Server-Objekt eine eigene Verbindung aufgebaut werden muss. Im Regelfall wird das StreamProtocol verwendet.
Für die Antwort einer Anfrage eines Clients ergeben sich drei Möglichkeiten für den Server:

- er bestätigt und überträgt die Rückgabewerte
- er teilt dem Client mit, dass dieses Protokoll nicht unterstützt wird
- er überträgt eine HTTP-response mit den obigen Inhalten.

Auch hier soll das HTTP-Format nicht weiter erläutert werden, da es nur in Sonderfällen eingesetzt werden muss. Eine gängige Antwort sieht wie folgt aus:

ProtocolAck Returns

Die wichtigsten Elemente können der nachfolgenden Tabelle entnommen werden:

Tabelle 5.3: Elemente der RMI Rückgabedaten

Elementname	Inhalt	
ProtocolAck	0x4e	
Returns	Return Returns Return	
Return	ReturnData PingAck	0x51 ReturnValue 0x53
ReturnValue	0x01 UniqueIdentifier *Value*$_{opt}$ 0x02 UniqueIdentifier *Exception*	

Um die Auslastung näher untersuchen zu können, müssen zunächst einige Vereinfachungen getroffen werden. Die nachfolgenden Betrachtungen beziehen sich ausschließlich auf den Betrieb des Systems, das heißt, alle Dienstreferenzen sind bekannt oder werden über einen Lookup Server zur Verfügung gestellt. Die Datenaufkommen bei den Join- und Discovery-Prozessen sind zu vernachlässigen, da sie nur beim Hinzukommen oder Entfernen von Diensten auftreten. Ein Aspekt, der jedoch in den laufenden Betrieb eingeht, sind die Leases. *Leasing* ist ein Konzept, das in Jini dazu verwendet wird, um Teilnehmerausfälle erkennen zu können. Ein Dienst muss über Leases anzeigen, dass er immer noch verfügbar ist. Dies gilt für die Anmeldung beim Lookup Service, bei der Verwendung von Remote Events und bei Nutzungsbeziehungen jeder Art zwischen Dienstnehmer und Dienstanbieter. Meldet sich ein Dienst beim Lookup Service an (Join), so muss er diese Anmeldung periodisch erneuern, da er sonst nach Verfall des durch den Lease festgelegte Zeitintervalls aus dem Lookup Server entfernt wird. Die Lease-Anforderung erfolgt auf folgende Weise: Der Dienst fordert beim Lookup Server einen Lease für eine bestimmte Zeit an; es liegt jedoch beim Lookup Server festzulegen, wie lange der Lease gilt. Damit stellt das vom Dienst vorgeschlagene Zeitintervall nur eine Obergrenze dar, der Lookup Server kann dieses Intervall nach eigenem Ermessen verkürzen. Gewährt der Lookup Server dem Dienst den Lease, so gibt er ihm das Zeitintervall zurück, nach dem der Dienst den Lease wieder erneuern muss.
Der gleiche Mechanismus ist auch auf die anderen Nutzungsbeziehungen anwendbar. Damit gehört das Erneuern des Leases zum festen Bestandteil von Jini. Die Lease-Dauer, die vom Lookup Services gewährt wird, kann nach [Oaks2001] von der Netzwerkauslastung abhängen. Daraus sind keine Voraussagen möglich. Bei Messungen wurden im unbelasteten Fall (Netzwerkauslastung <<1%) Zeiten von 5 Minuten festgestellt[44].
Zusammenfassend tragen folgende Elemente zu der Auslastung des Netzwerks bei:

[44] Die Implementierung des Lookup Services wird von Sun Microsystems ohne Sourcecode ausgeliefert, so dass sich der genaue Algorithmus zur Bestimmung der Lease-Dauer einer genaueren Untersuchung entzieht.

- Service Lookup (Discover)
  Bevor sich ein Dienst einer Jini-Gemeinschaft anschließen kann, muss ein Lookup Service gefunden werden. Dies geschieht (auch[45]) durch Multicast-Telegramme
- Service Join
  Jeder Dienst meldet sich unter Verwendung des Lookup Services beim Lookup Server an.
- Service Use
  Nach dem Discover wird der Lookup Service von Dienstnehmer benutzt, um eine Dienstreferenz zu bekommen. Dies ist vergleichbar mit dem Join, da auch hier eine Referenz auf den Lookup Service zur Anmeldung verwendet wird und letztlich auf einen RMI-Aufruf zurückgeführt werden kann.
- Leases
  Leases werden, wie oben ausgeführt, bei mehreren Dienstverhältnissen angewandt.
- RemoteEvents
  Ein RemoteEvent ist ein Callback-Verfahren, bei dem sich ein Ereignisinteressent bei einer Ereignisquelle anmeldet, die ihn dann über eine festgelegte Schnittstelle benachrichtigt.
- Dynamisches Nachladen von Byte Code
  Ein sehr wichtiger Punkt bei der Verwendung von RMI ist das Nachladen von Byte Code. Können Klassenbeschreibungen lokal nicht gefunden werden, so besteht die Möglichkeit diese vom Server-Objekt nachladen zu lassen. Dieser Vorgang ist für den Anwender nicht sichtbar und wird von der JVM erledigt. Über diesen Mechanismus werden auch die RMI-Stubs der Server-Objekte übermittelt.
- Aufrufe der *Distributed Garbage Collection* (DGC)
  Auch Fernreferenzen werden bei der Garbage Collection nicht ausgelassen. So verwaltet ein Server-Objekt intern alle Client-Fernreferenzen. Sobald keine Referenz mehr vorhanden ist, kann das Objekt aus dem Speicher beseitigt werden.
  Für die Kommunikation zwischen den JVMs werden ebenfalls Fernaufrufe verwendet, die auch Leases beinhalten.
- Sonstige Daten
  Alle Daten, die nicht Jini oder RMI betreffen, fallen in diese Kategorie.

Durch diese unterschiedlichen Nachrichten, deren Aufbau sehr von den verwendeten (RMI) Schnittstellen abhängt und die nicht durch offengelegte Spezifikationen beschrieben sind, können in diesem komplexen System nur recht allgemeine Aussagen zur Auslastung gemacht werden. Daher soll exemplarisch das Datenaufkommen eines RMI-Aufrufs näher untersucht werden. Dazu wurde eine Client/Server-Anwendung erstellt. Die Aufgabe des Clients besteht darin, Informationen an den Server zu senden, der diese umwandelt und an den Client zurücksendet. Der Server stellt dabei die folgende Schnittstelle zur Verfügung:

[45] Wie bereits erwähnt, werden mehrere Mechanismen zum Auffinden des Lookup Services angeboten. Einer davon wird über Multicast-Telegramme abgewickelt.

```
public interface Hello extends Remote
{
    String sayHello(int i, String txt) throws RemoteException;
}
```

Die Implementierung des Aufrufs innerhalb des Server-Objekts sieht wie folgt aus:

```
public String sayHello(int i, String txt)
{
      return "Hello World! "+ i + " " + txt;
}
```

Der Aufruf des Clients sah dabei wie folgt aus:

```
     helloServ.sayHello(0x12345678,"Text1");
```

```
Client to Server (7/61 bytes)
(1) Header Version Protokoll
0000  4A 52 4D 49 00 02 4B                                    JRMI..K
Server to Client (21/75 bytes)
(2) Acknowledge, EndpointIdentifier (host + port)
0000  4E 00 0E 31 39 32 2E 31  36 38 2E 31 36 31 2E 36  N..192.168.161.6
0010  35 00 00 07 2B                                    5...+
Client to Server (471/525 bytes)
(3) EnpointIdentifier
0000  00 0E 31 39 32 2E 31 36  38 2E 31 36 31 2E 36 35  ..192.168.161.65
0010  00 00 00 00
[…] (Garbage Collect method call)
Server to Client (333/387 bytes)
[…] (Garbage Collect method return)
Client to Server (1/55 bytes)
Ping
0000  52                                                R
Server to Client (1/55 bytes)
Ping Acknowledge
0000  53                                                S
Client to Server (53/107 bytes)
(4) Call Time (in ms seit 1970) Operation[46] Hash Argument (0x12345678:int)
Argument ("Text1":String)
0000  50 AC ED 00 05 77 26 00  00 00 00 00 00 00 00 00  P....w&.........
0010  2F 66 84 00 00 00 F2 50  F8 15 5F 80 00 FF FF FF  /f.....P.._.....
0020  FF 8E 0B E7 63 1A A0 1E  CB 12 34 56 78 74 00 05  ....c.....4Vxt..
0030  54 65 78 74 31                                    Text1
Server to Client (53/107 bytes)
(5) ReturnValue Time String
0000  51 AC ED 00 05 77 0F 01  00 2F 66 84 00 00 00 F2  Q....w.../f.....
0010  50 F8 15 5F 80 04 74 00  1C 48 65 6C 6C 6F 20 57  P.._..t..Hello W
0020  6F 72 6C 64 21 20 33 30  35 34 31 39 38 39 36 20  orld! 305419896
0030  54 65 78 74 31                                    Text1
Gesamt Ethernet: 1738 Bytes
```

Abbildung 5.19: Auszug RMI Client/Server-Kommunikation

Der TCP/IP-Verkehr zwischen dem Client und dem Server wurde aufgezeichnet. Nicht dargestellt ist hier der Bootstrap-Mechanismus zum Auffinden der Remote-Referenz über die Registry und das Nachladen des Byte Codes des Stubs. Beide Vorgänge finden nur bei der Initialisierung statt.

[46] Eigentlich ist *Operation* ein Hash-Wert der Methode und *Hash* der Hash-Wert des Stubs. In Java 2 wurden jedoch Skeletons abgeschafft, womit auch kein Stub Hash mehr notwendig war. Damit wird *Operation* zu –1 (0xFFFFFFFF) gesetzt und *Hash* enthält jetzt den Methoden-Hash-Wert.

Etwas deutlicher wird Zusammenspiel zwischen Client und Server bei Verwendung des *StreamProtocols* mit dem nachfolgenden Sequenzdiagramm:

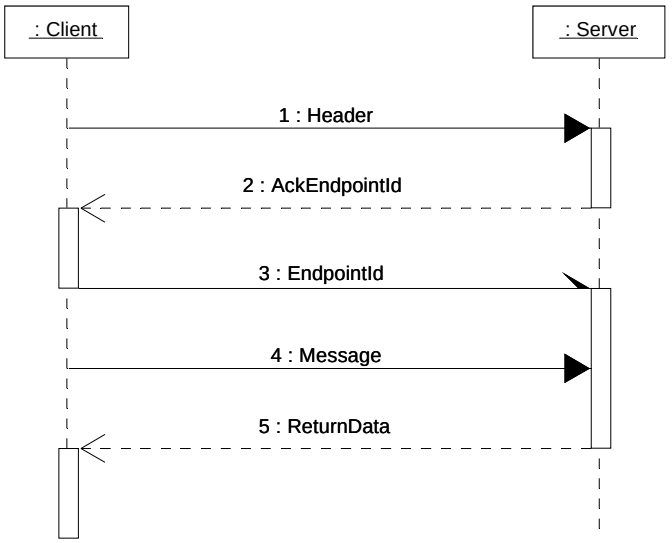

Abbildung 5.20: Nachrichtensequenz bei einer RMI Anfrage

Nach dem Versenden des Headers (1) bestätigt der Server mit einem *Acknowledge* und sendet gleichzeitig den *EndpointIdentifier* (2) des Clients[47]. Danach sendet der Client seinen *EndpointIdentifier* (3), der angibt, auf welcher Verbindung er Daten entgegennehmen kann. Erst dann folgt der Methodenaufruf (4), der vom Server mit dem Rückgabewert (5) beantwortet wird. In der dargestellten Aufzeichnung finden sich noch zwei Besonderheiten: Während der Rückgabe des *EndpointIdentifiers* (3) wurde noch zusätzlich eine Distributed Garbage Collection ausgeführt. Des Weiteren sieht man das Senden von Pings. Innerhalb der DGC werden komplette Objekte serialisiert.
Die erste Längenangabe (Abb. 5.19) in Bytes ist die Länge des Inhalts einer TCP/IP-Nachricht. Es kommen Daten des TCP-Protokollkopfes und des Ethernet-Protokolls hinzu, so dass sich eine Gesamtlänge ergibt, die als zweite Angabe hinzugefügt wurde. Am Ende wurden alle Ethernet-Pakete unter Einbeziehung der Bestätigungstelegramme zusammengezählt. Von den 1738 übertragenden Bytes sind nur 484 Bytes für die Übermittlung des Befehls, wie in Abb. 5.19 dargestellt, notwendig. Dabei wurden ein primitiver Datentyp (32 Bit Integer) und ein String (5 Byte UTF-8), sowie als Antwort ein String (27 Bytes UTF-8), zwischen dem Client und dem Server ausgetauscht.
Obwohl das Verhältnis zwischen Nutzdaten und Protokollüberhang bei weniger als 10 % liegt, sind die Übertragungszeiten vernachlässigbar. Aufzeichnungen eines Ethernet Capture Tools zeigten für die Übertragung der 484 Bytes (theoretisch 38,7 µs) eine Gesamtdauer von 20 ms, die in keinem Verhältnis zu Übertragungszeit stehen. Dort stehen

[47] Der Nutzen davon wird erst ersichtlich, wenn man berücksichtigt, dass die Sicherheitseinstellungen der JVM des Clients es nicht zulassen, seinen Host-Namen zu erfahren. Dies ist bei Applets mit Sicherheitsbeschränkungen der Fall, die dennoch Sockets zu dem Server aufbauen können, von dem das Applet heruntergeladen wurde.

die Einflüsse von Switches, Routern und die Verarbeitungszeiten der Rechner im Vordergrund.

Tabelle 5.4: Datenaufkommen auf Ethernet Ebene (Zeiten bezogen auf 100 MBit/s[48])

Aktion	Datenaufkommen	Häufigkeit	Übertragungs-dauer
Service Lookup	Unbekannt	Selten	Unbekannt
Service Join	Abhängig vom angemeldeten Dienst	Selten	Unbekannt
Service Nutzung	Abhängig von der Art und Zahl der Aufrufparameter, bei Parametern, die primitive Datentypen sind: < 1 kByte	Nicht allgemein bestimmbar	< 80 µs
Leases	532[49] Bytes	Jini Lookup Server variabel Ansonsten abhängig vom Dienstanbieter	42 µs
RemoteEvents	> 703[50] Bytes	Abhängig vom Dienstanbieter	> 56 µs
Dynamisches Nachladen von Byte Code	Abhängig vom Stub, bei wenigen Operationen (<5): < 2kByte	Nicht allgemein bestimmbar	Unter den gennanten Bedingungen: < 160 µs
Aufrufe der DGC	1112 Byte	Abhängig von der Anzahl der zu verwaltenden Objekte. Da die DGC asynchron ausgeführt wird, ist keine Vorhersage zu treffen.[51]	89 µs

In der Tab. 5.4 sind alle Messergebnisse eines Protokollierungsvorgangs dargestellt. Nicht alle Parameter der aktiven Dienstumgebung können erfasst werden. Die einzigen konstanten Werte innerhalb der Tabelle sind die Aufrufe der DGC zwischen den JVMs und die Lease-Erneuerung. Alle anderen Ereignisse können nur bei Kenntnis des angebotenen Dienstes bestimmt werden. Nicht relevant ist dies ohnehin für den Discovery- und den Join-Prozess. Es hat sich gezeigt, dass der Lookup Service periodisch Multicast-Telegramme aussendet, die jedoch von ihrem Datenumfang sehr gering sind und –

[48] Die Telegrammlängen beinhalten nicht die Preamble, den Start Delimiter und die Prüfsumme des Ethernet Rahmens.

[49] Diese Länge wurde mit einem Ethernet-Protokollanalysewerkzeug gemessen. Die Größe einer Lease-Anforderung ist konstant, da der RMI-Aufruf neben der *ServiceID* des Services nur zwei Long Werte enthält.

[50] Hier konnte nur eine untere Grenze angegeben werden, da die Parameter eines Remote Events von der Anwendung abhängen.

[51] Bei der Verwendung von zwei JVM wurden minimale Zeiten von 26ms gemessen, die größte Zeit betrug 268s.

verglichen mit der Übertragungsgeschwindigkeit – in großen zeitlichen Abständen[52] auftreten. Der Join-Prozess findet nur gelegentlich statt, also beim Hinzufügen eines neuen Dienstes, und trägt damit auch nicht zur Grundlast bei. Hauptträger der Grundlast sind Leases, Remote Events und die DGC. Die Leases sind zwar anwendungsabhängig, sollten jedoch immer im Minutenbereich gewählt werden. Die Hauptaufgabe der Leases ist die Aufrechterhaltung der Funktion des Systems durch Zusicherung der Ressourcen. Diese Funktion ist auch im Minutenbereich noch gewährleistet.
Für einen sehr einfachen Dienst, bestehend aus mindestens einer Funktion zur Anmeldung eines Remote Events gelten folgende Kommunikationsbeziehungen:

- Lease zwischen Dienstanbieter und Lookup Server
  Jeder Dienst muss seinen Lease zum Lookup Server erneuern, um im Dienstverzeichnis verfügbar zu bleiben
- Lease zwischen Dienstanbieter und Dienstnehmer zur Verwaltung des Remote Events
  Meldet sich ein Dienstnehmer zur Benachrichtigung durch einen Dienstanbieter an, so wird durch einen Lease gewährleistet, dass der Dienstnehmer noch verfügbar ist.
- Remote Event
  Das Remote Event wird durch den Dienstanbieter bei Erreichen eines bestimmten internen Zustands ausgelöst.

Daraus ergibt sich aus den in Tab. 5.4 aufgelisteten Werten ein Datenaufkommen von 1767 Bytes zuzüglich 36 Bytes für den Ethernet-Header. Unter der Voraussetzung, dass alle drei Ereignisse unmittelbar hintereinander auftreten, ergibt sich daraus eine Busbelegzeit von 144 µs zuzüglich der Interpacket-Gap[53] von 2,88 µs, also 146,88 µs. Erst bei mehr als 3400 Aufrufen hintereinander ist die Hälfte der Bandbreite ausgenutzt. In der Praxis werden weniger Daten genügen, da die Kollisionen die Teilnehmer veranlassen, größere Wartezeiten bis zur nächsten Busbelegung zu wählen [Furrer2000]. Dennoch sollten bei der Implementierung der Dienste folgende Richtlinien beachtet werden:

- Aufrufparameter von Diensten sollten, wenn möglich, primitive Datentypen sein. Diese werden offensichtlich in binärer Form in das RMI Wire Protocol eingebunden. Objekte hingegen werden rekursiv[54] serialisiert, womit der Zustand eines kompletten Objekts übermittelt wird. Eine Ausnahme bilden Strings, die in UTF-8 übermittelt werden und damit kaum zusätzlichen Platz beanspruchen (vgl. Abb. 5.18).
- Es sollten möglichst große Lease-Intervalle ausgewählt werden, oder aber die Intervalle an die Busauslastung angepasst werden.
- Die Anmeldung bei Ereignisquellen für die Zuteilung von Ereignisbenachrichtigungen ist der ständigen Abfrage von Diensten vorzuziehen. Dadurch werden innerhalb des Systems nur die Änderungen übertragen. Der Zusatzaufwand durch die

[52] 34 Bytes im Abstand von mehreren Sekunden.
[53] Aus [Furrer2000],für Fast Ethernet: 0,96µs.
[54] Sind innerhalb des Objektes Attribute mit Objektereferenzen enthalten, die nicht als transient gekennzeichnet sind, werden auch diese und ihre Attribute serialisiert.

Anmeldung und die Steigerung der Grundlast durch die Verwendung von Leases sind vernachlässigbar.
- Die Busübergänge sollten eine Vorfilterung vornehmen und keine irrelevanten Daten „tunneln“.

Beachtet man diese Vorgaben, ist die Busauslastung im Betrieb unkritisch und erlaubt auch den Transport von synchronen Daten. Durch die Art der angebotenen Dienste des Gebäudeautomatisierungssystems, die asynchrone Daten in großen zeitlichen Abständen liefern – so die Daten des HKL-Bussystems – werden die Systemgrenzen kaum erreicht werden. Damit entfällt weitgehend ein Ressource Management, da nicht mehr vorhandene Dienste durch das Leasing Konzept bereits beseitigt werden und RemoteEvents zur Verringerung der Grundlast durch Übertragung von Zustandsänderungen beitragen.

5.2.2 IEEE 1394

Obwohl die Hauptaufgabe von IEEE 1394 die Übermittlung isochroner Daten ist, wurde es in dieser Arbeit nicht dazu verwendet. Die Übermittlung isochroner Daten ergibt in diesem System zwar durchaus einen Mehrwert, für die Umsetzung eines beispielhaften Dienstes genügt jedoch hier die Verwendung des asynchronen Verbindungsmodus. Für weitere Arbeiten wäre die Vertiefung und Anwendung des hier vorgestellten Konzepts zur Dienstkopplung sehr interessant. In diesem System könnten zum Beispiel Daten der Videokamera auf den Fernseher übertragen werden. Auch die Kopplung eines Videorekorders oder Videoabspielgeräts mit dem PDA ist ein interessantes Beispiel für die weiteren Möglichkeiten.
Für die Integration des IEEE 1394 in das Konzept wird auch hier Middleware in Form von HAVi verwendet. Die Grundeigenschaften von HAVi wurden in Kap. 4.2.1.1 gezeigt und sollen hier nun noch weiter ausgeführt werden.

5.2.2.1 Eingliederung in das Systemmodell

Der Einsatz von IEEE 1394 gleicht in diesem Fall einer horizontalen Erweiterung des Systems, also dem Fortführen des Backbones, dennoch ist die eigentliche Aufgabe vielschichtiger. Obwohl dies strukturell der Weiterführung der Ebene 1 gleichkommt, findet hier ein Wechsel des physikalischen Übertragungsmediums, des Übertragungsprotokolls und zusätzlich auch der Middleware durch die Verwendung von HAVi statt. Es wird daher zunächst nur die Funktion innerhalb des IEEE 1394 Netzwerks betrachtet.
Jini und HAVi warten mit einigen Gemeinsamkeiten auf, die auf den ersten Blick offensichtlich sind. Beide stellen eine Dienstumgebung und Kommunikationsstruktur zur Verfügung. Die angebotenen Dienste lassen durchaus Parallelen zu. Genau wie Jini ist auch HAVi vollständig in Java implementiert. Die Definition der API ist durch die *Interface Definition Language* (IDL) in der Spezifikation angegeben.

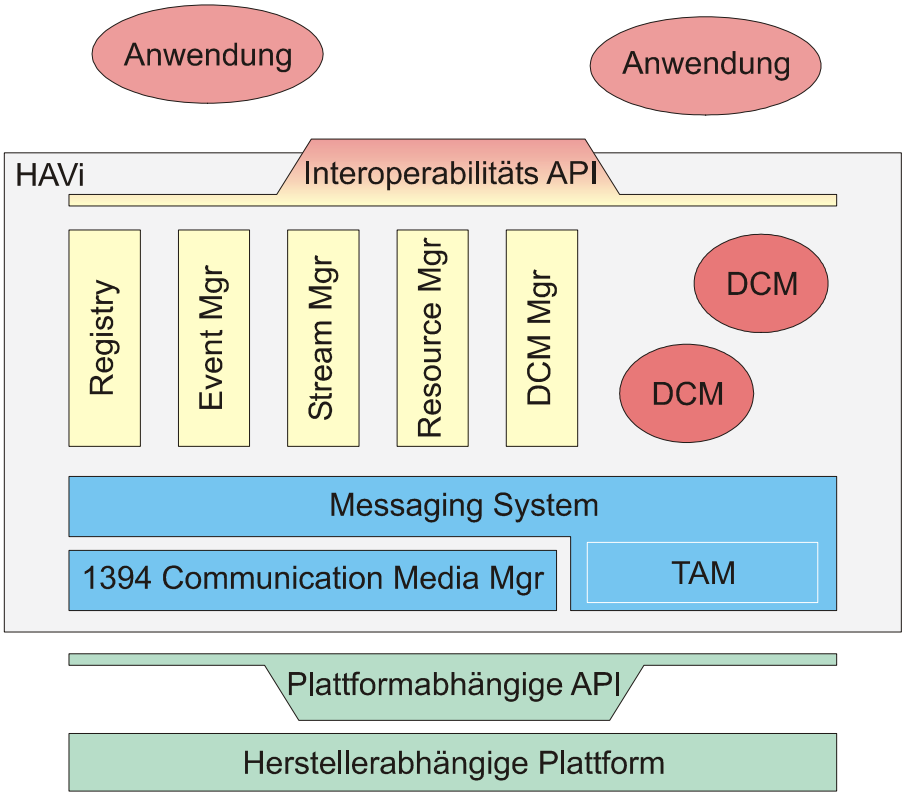

Abbildung 5.21: HAVi-Architektur

Die Architektur von HAVi sieht mehrere Dienstkomponenten vor, die alle nur auf einem FAV/IAV ausgeführt werden können. Die unterste Schicht ist herstellerabhängig. Der *1394 Communication Media Manager* liegt bereits innerhalb der JVM und ist damit herstellerunabhängig. Die Anbindung an IEEE 1394 muss über das Java Native Interface erfolgen. Das *Messaging System* verteilt alle Nachrichten innerhalb des Systems und gibt sie bei Bedarf an den *Communication Media Manager* weiter. Direkte Parallelen zu Jini sind die *Registry*, die alle DCMs verwaltet, und die *Software-Elements*. Ein *Software-Element* ist eine zentrale Komponente, die über Nachrichten mit anderen Software-Elementen kommuniziert. Verglichen mit Jini sind dies die implementierenden Dienstobjekte, wobei es auch eine freie Schnittstellendefinition gibt. Die Nachrichten-schnittstellen der oben dargestellten Komponenten sind durch die Spezifikation festgelegt. Ähnlich wie bei Jini durch die Service ID, sind auch hier die Software-Elemente durch *Software-Element ID*s (SEID) gekennzeichnet, die einmalig im gesamten System sind. Dies wird durch die Einbettung des IEEE 1394 GUID erreicht.
Der *Event Manager* ist für die Verwaltung von Ereignissen zuständig – auch in Jini gibt es eine Ereignisbehandlung. Komponenten, die in Jini keine Entsprechung finden, sind der *Stream Manager* und der *Resource Manager*. Der *Stream Manager* ist für die Lastverteilung und Anforderung der isochronen Video- und Audiodaten zuständig. Der *Resource Manager* dagegen verwaltet Geräteressourcen.
Die *Interoperability API* legt die Schnittstellen zum System fest. Eine sehr wichtige Schnittstelle zur Anzeige von Benutzeroberflächen ist die so genannte *Level 1 User Interaction* (UI), durch welche die Schnittstellen zwischen Benutzer und System definiert

sind. Die Umsetzung der *Level 1 UI* erfolgt durch das *Data Driven Interaction* (DDI) Protokoll. Jedes anzeigefähige FAV/IAV muss einen *DDI Controller* zur Verfügung stellen, der ein *DDI Target* steuert. So können Benutzereingaben und Rückmeldungen des Targets über dieses Protokoll verarbeitet und angezeigt werden. Die Einzelheiten der Nutzung von DDI werden auch im weiteren Verlauf der vorliegenden Arbeit noch beschrieben.
Durch die Parallelen ließen sich auch über HAVi mehrere Bussysteme miteinander verbinden. Der vorgeschlagene Systemaufbau kann dadurch beibehalten werden. Es bleibt lediglich zu klären, wie die Bussysteme in HAVi einzuordnen sind.

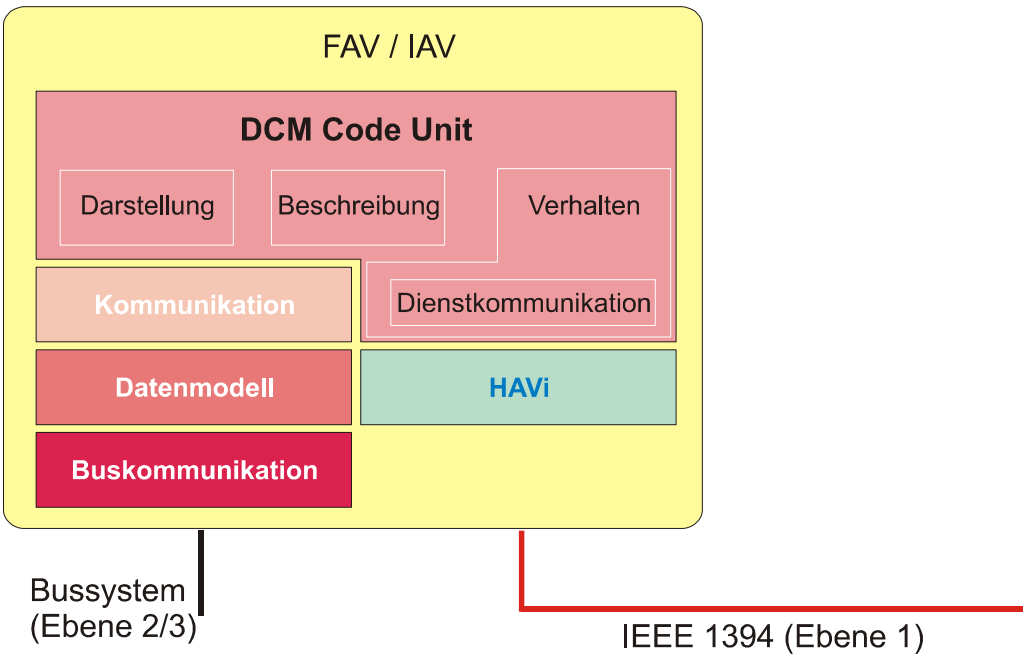

Abbildung 5.22: Einbettung von HAVi in das Systemkonzept

In diesem Aufbau sieht man sofort, dass sich HAVi ebenfalls in das Gefüge aus dienstorientiertem Netzwerk und Busankopplung einbetten lässt. Die einzelnen Systemübergänge müssen immer noch nach dem in Abb. 4.5 dargestellten Aufbau strukturiert werden, also zunächst die Busanbindung, dann die Abbildung auf ein Datenmodell und eventuell noch ein Zwischenprotokoll zur Kommunikation mit dem Dienst, wie es sich in den folgenden Betrachtungen zeigen wird. Zunächst einmal müssen die Grundvoraussetzungen geschaffen werden.

5.2.2.2 Aufbau des HAVi-Stacks

Die Arbeiten an einem Systemübergang zu HAVi begannen im Jahr 2001[55]. Die ersten Versuche einen kommerziellen HAVi-Stack zu erhalten, gestalteten sich ein Jahr nach der Ausgabe der ersten Spezifikation als schwierig und kostspielig – die marktüblichen Preise lagen zu diesem Zeitpunkt bei etwa 10.000 US-Dollar. Eine Zwischenlösung war eine

[55] Die Vorbereitung fanden Anfang des Jahres statt. Es wurde zunächst versucht ein kommerzieller HAVi-Stack einzusetzen. S. dazu [Liebl2001]

Evaluierungslizenz[56] für 3 Monate. Da dies jedoch nur eine befristete Lösung war, wurde ein eigener HAVi-Stack entwickelt, der jedoch kein Streaming unterstützt. Der am Lehrstuhl für Messsystem- und Sensortechnik der TU München entwickelte HAVi-Stack verfügt über die Basiskomponenten, also einen

- *1394 Communication Media Manager*
  Seine Aufgabe ist die Kommunikation mit dem Busmedium, wobei bisher nur die asynchrone Kommunikation unterstützt wird. Durch eine entsprechende Erweiterung des JNI kann auch diese Funktionalität noch nachträglich erweitert werden.
- *MessagingSystem*
  Zur internen und externen Nachrichtenverwaltung wurde das MessagingSystem implementiert.
- *Registry*
  Die Registry übernimmt die oben beschriebenen Aufgaben. Sie verwaltet die im Netzwerk verfügbaren DCMs
- *EventManager*
  Die Ereignisbehandlung wird durch dieses Software-Element übernommen. Alle Software-Elemente können sich hier eintragen, um über Systemereignisse informiert zu werden. Ein sehr wichtiges Ereignis ist die Entfernung oder das Hinzukommen von Geräten, das durch einen Bus Reset gemeldet wird.
- *DDI Controller*
  Zum Austausch von Information und Daten wird das *Data Driven Interaction Protocol* verwendet
- *DDI Target*
  Das Ziel ist das zu steuernde Gerät oder der zu steuernde Funktionsblock (DCM, FCM). Es kann sich dabei um einen Videorekorder oder aber um ein Gateway handeln.

Wie bereits erörtert wurde, ist der eigentliche HAVi-Stack hardwareunabhängig in Java vorgesehen. Dennoch gibt es eine Schnittstelle zum ausführenden System, in der Regel – jedoch nicht zwingend – zum Betriebssystem. Da die Implementierung auf einem PC entwickelt wurde, musste zur Entwicklung der IEEE 1394 Gerätetreiber ein Betriebssystem ausgesucht werden. Das markgängigste PC-Betriebssystem war zu der Zeit Windows in unterschiedlichen Auslegungen (Windows 98, Windows NT, Windows 2000). Windows 98 schied wegen seines umständlichen und zu den Nachfolgern weitgehend inkompatiblen Treiberkonzepts aus. Windows NT war bereits durch Windows 2000 abgelöst worden. Das *Windows Driver Model* (WDM), das in Windows 2000 eingesetzt wird, sieht bereits so genannte Mini Driver für USB und IEEE 1394 Controller vor.
Linux wird überwiegend für Server-Anwendungen herangezogen, kann jedoch auf Grund seiner feingliedrigen Zusammensetzung modular den eigenen Bedürfnissen angepasst

[56] Von der Firma TwonkyVision, Sitz in Berlin.

werden und ist zudem frei verfügbar. Es fallen also keine Lizenzgebühren an, wie es bei verschiedenen embedded Betriebssystemen der Fall ist (VxWorks, Windows CE, Embedded NT,...). Zudem lässt es sich so einrichten, dass keine beweglichen Speichermedien benötigt werden (nichtflüchtiger Speicher: Flash Speicher, DiskOnChip, gepuffertes RAM etc.).
Im Kernel der Version 2.2.4 war die Unterstützung von IEEE 1394 noch nicht stabil, so dass ein Patch eingesetzt wurde. Der Treiber spricht die verwendete IEEE 1394 PCI-Einsteckkarte über das *Open Host Controller Interface* (OHCI)[57] an.
Das OHCI ist eine Abbildung des Link Layers von 1394 zur Anbindung an ein Zielsystem und stellt sowohl asynchrone als auch isochrone Datenübertragungsmodi zur Verfügung. Zusätzlich werden noch Verfahren für den hochratigen Datentransfer an höhere Schichten beschrieben (*Direct Memory Access*). Es findet bei den meisten IEEE 1394 Controllern Verwendung und ist bereits im ASIC integriert. Angesprochen wird das OHCI über einen mehrgeteilten Adressraum, von dem ein Teil [ISOIEC13213] standardisiert ist. Über diesen Adressraum wird der Controller angesprochen, so dass über diese festgelegte Schnittstelle Daten empfangen, versendet, DMA Transfers eingerichtet und allgemeine Statusinformationen abgefragt werden können.
Die Treiberkomponenten des neuen Kernels (Version 2.4) sind stabil, so dass sie ohne ein Patch eingesetzt werden können. Zum Ansprechen wurde eine Bibliothek mit dem Namen libraw1394 eingesetzt, die die asynchrone Kommunikation unterstützt. Darauf wurde ein JNI für den 1394 *Communication Media Manager* aufgebaut, so dass der Zugriff auf IEEE 1394 innerhalb der JVM erfolgen kann.

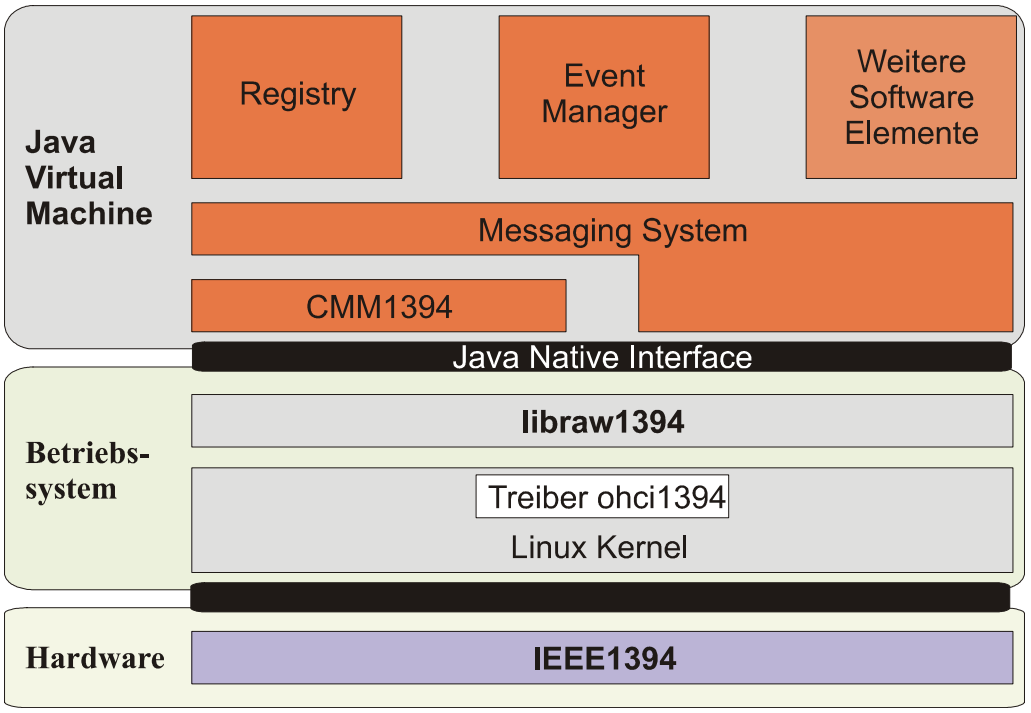

Abbildung 5.23: HAVi-Stack unter Linux

[57] S. [Anderson1999].

Das *Messaging System* ist durch das in der Spezifikation definierte *Transport Adaptation Module* (TAM), eine vom Transportmedium abhängige Zwischenschicht, ebenfalls an der IEEE 1394 Kommunikation beteiligt. Alle Nachrichten, die HAVi-spezifisch sind, werden von dem *Messaging System* verarbeitet. Allgemeine Informationen, wie Topologiewechsel[58] des Netzwerkes oder aber ein Bus Reset, unterliegen dem *Communication Media Manager*.
Zwischen der libraw1394 und dem Betriebssystem befindet sich der eigentliche Gerätetreiber (ohci1394), der über das OHCI den IEEE 1394 Nachrichtenverkehr koordiniert.

5.2.2.3 Übergang von EIB-HAVi

Eine wichtige Anwendung bei einem Bussystem in der untersten Ebene, besonders für den Einsatz in der Heimautomatisierung, ist die Meldung von systemrelevanten Daten auf einem Anzeigegerät. Diese Aufgabe kann von einem FAV/IAV in HAVi übernommen werden. Daher soll im Folgenden ein Dienst innerhalb von HAVi erstellt werden, mit dem es möglich ist, Daten des EIB anzeigen zu lassen und den Zustand von Objekten über HAVi verändern zu können.
Die zentrale Komponente zur Umsetzung von EIB auf HAVi ist ein *Device Control Module* (DCM). Ein DCM abstrahiert die Funktionen eines Geräts und ist über festgelegte Schnittstellen in die HAVi-Architektur eingebettet. Es wird zwischen mehreren Arten von DCMs unterschieden, die sich in ihrem Herkunftsort (uploadable, embedded), in ihrer Ausführbarkeit (Java Byte Code oder native) und in ihrer Funktionalität (nur HAVi API, erweiterte API) kategorisieren lassen.
Unter der Prämisse einer breiten Anwendbarkeit wurden in dieser Arbeit nur DCMs verwendet, die in Java Byte Code auf einem BAV als eine herunterladbare DCM Code Unit zur Verfügung stehen. Prinzipiell spricht nichts gegen ein embedded DCM, ein herunterladbares DCM erscheint jedoch mehr mit dem Gedanken von Jini vereinbar. Bei Jini verbleibt der Dienstanbieter als Software-Objekt an Ort und Stelle und es wird nur eine Dienstschnittstelle geladen. Bei HAVi wird ein ausführbarer Programmcode geladen und gestartet. Der in HAVi geladene Programmcode ist jedoch vergleichbar mit einem Stub des Dienstanbieters in Jini (RMI), da das DCM nur die Kommunikation mit dem anderen Quellgerät des DCMs unternimmt, die eigentliche Funktionalität jedoch im Gerät verbleibt. DCMs werden sowohl von FAV, BAV und optional auch von IAV angeboten. Eine besondere Rolle bei der Lokalisierung der DCMs kommt dabei dem *Self Describing Device* (SDD) Datenbereich zu. Dieser ist im ROM-Bereich nach [ISOIEC13213] eines jeden IEEE 1394 Geräts zu finden. Dort sind unterschiedlichste Informationen über das Gerät hinterlegt. HAVi hat einen besonderen Bereich für allgemeinen Informationen über HAVi-konforme Geräte definiert. In diesem wird festgelegt, um welche Gerätekategorie es sich handelt (FAV, IAV, BAV) und es kann dort eine DCM Code Unit hinterlegt werden.

[58] Dies bezieht sich auf die Topologie-Map, die Abbildung der Netzwerktopologie bei IEEE1394. Bei einem Bus Reset wird automatisch die neue Topologie ermittelt.

Nach der Spezifikation muss jedes BAV entweder eine Code Unit oder aber eine URL auf eine Code Unit in diesem Bereich angeben. Damit wird sichergestellt, dass jedes BAV über ein FAV steuerbar ist[59]. Die DCM Code Units werden in *Java Archive* (Jar) Dateien gespeichert. Jar-Dateien sind im Zip-Format gepackte Verzeichnisse und Dateien. Die Verzeichnisstruktur spiegelt dabei die Einordnung der Klassen in Packages wider. Neben dem gespeicherten Byte Code können auch andere Ressourcen gespeichert werden (Bilddateien). Ein fester Bestandteil der Jar-Datei ist das Manifest File, das Zusatzinformationen verschiedenster Art enthalten kann (SH1 Hash Code der Klassendateien zur Sicherheitsüberprüfung, Einstiegspunkt einer Anwendung etc.). Die Classloader[60] einer jeden JVM sind in der Lage, diese Dateien zu entpacken und daraus den Byte Code der Klassen zu laden.
Ein DCM kann wiederum in mehrere *Functional Component Modules* (FCM) unterteilt sein. Während ein DCM die Abstrahierung des gesteuerten Geräts ist, stellt ein FCM eine funktionale Komponente des Gerätes dar.
Durch die Architektur von HAVi ergeben sich mehrere Vorgehensweisen bei der Ankopplung eines Bussystems an HAVi. Diese sollen nachfolgend anhand der Systemanforderungen gestaffelt aufgeführt werden:

a) Übergang BAV, Anzeige FAV, keine standardisierte Kommunikation
   Der Übergang soll nur ein Bussystem anbinden. Die Kommunikation zwischen dem BAV und dem FAV zur Steuerung wird nicht durch HAVi abgedeckt. Dort sind nur die Mechanismen für den Systemstart, also das Laden und Auffinden des DCMs vorgegeben. Damit bleibt weiterhin die Möglichkeit, das DCM auf dem BAV zu hinterlegen, dann vom FAV herunterladen zu lassen, das dann die Anzeige übernimmt. Zur Anbindung mehrerer Bussysteme kann das DCM auch in FCMs unterteilt werden, die dann jedes Bussystem einzeln abbilden.
b) Übergang BAV, Anzeige FAV, DDI-Kommunikation
   Da die Spezifikation keine Aussage darüber macht, wie das Protokoll zur Kommunikation des heruntergeladenen DCMs mit dem BAV zu gestalten ist, kann auch DDI verwendet werden. Dies setzt jedoch auf dem BAV einen HAVi-Stack voraus. Der Vorteil ist, dass sich das BAV nach außen wie ein BAV verhält, jedoch den anderen Teilnehmern keine Möglichkeit bietet, dort Code auszuführen. Dabei wird gleichzeitig ein herstellerunabhängiges Protokoll verwendet.
c) Übergang und Anzeige FAV
   Die Abbildung des Datenmodells des Bussystems und die Umsetzung auf HAVi sowie die Anzeige erfolgen im gleichen Gerät. Dieser Ansatz erfordert die größten Systemressourcen, da auch noch eine Aufbereitung der Daten zur graphischen Anzeige eingebunden werden muss.

[59] BAVs verfügen nicht über eine Laufzeitumgebung und benötigen auch nicht den vollen HAVi-Stack, daher müssen sie über ein DCM gesteuert werden.
[60] Classloader sind ein Teil der JVM und sorgen für das Auffinden und Laden von Java Byte Code. Dabei findet auch eine Integritätsüberprüfung statt.

Der hier gewählte Ansatz entspricht dem Vorgehen aus b) aus zwei Gründen: Es wurde ein BAV gewählt, da dort die Systemanforderungen definitionsgemäß am geringsten sind. Damit ist auch ein Übergang über ein embedded System denkbar. Dennoch wurde bei der Systemimplementierung auf die Verwendung eines eigenen Protokolls zur Kommunikation zwischen dem DCM auf dem FAV und dem BAV bewusst verzichtet, da dadurch die Entwicklung des HAVi-Stacks nicht nur auf die lokalen Komponenten beschränkt wurde, sondern während der Entwicklung auch die HAVi-IEEE 1394 Kommunikation getestet werden konnte. Daher wurde für die Benutzerinteraktion die Level 1 UI verwendet, also ein Nachrichtenaustausch über das DDI-Protokoll. Eine Aufteilung in FCMs – je Busübergang ein FCM – ist hier nicht sinnvoll, da eine Zentralisierung des Systems durch Bündelung mehrerer Bussysteme an einem Punkt aus den in Kap. 4.1 genannten Gründen vermieden werden soll.
Auf dem FAV muss ein DDI Controller vorhanden sein, der im Stack auch berücksichtigt wurde. Während DDI Controller eindeutig auf einem FAV/IAV angesiedelt werden, kann ein DDI Target durch ein DCM repräsentiert werden[61].
Das DCM enthielt ein DDI Target, das die Ereignisse, also die Benutzereingaben an das BAV weitergeleitet und Antworten des BAV dem DDI Controller übergeben hat. Damit ergibt sich der folgende Aufbau:

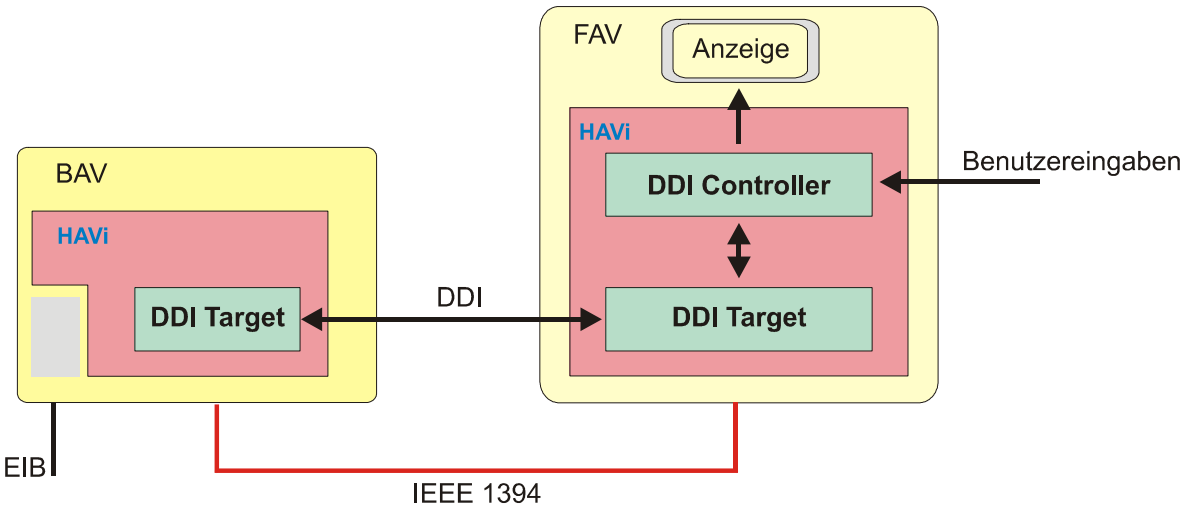

Abbildung 5.24: Benutzerinteraktion des BAV/EIB-Gateways

Dieser zu Entwicklungszwecken gewählte Aufbau kann im nächsten Schritt so umgearbeitet werden, dass aus dem BAV ein FAV wird und damit über den DDI Controller eines anderen FAVs gesteuert wird. Dazu würde das DCM innerhalb des ehemaligen BAV ausgeführt werden und es würde ein DDI Target implementieren. Dies entspricht einer der vorgeschlagenen Architekturen aus [HAVi2001][62]. Die Überführung in

[61] Auf S. 101 in Kap. 4.1 von [HAVi2001] heißt es: „[…] it is possible for any application (not only a UI-controller) to act as a DDI Controller and for any application (not only a DCM) to act as a DDI Target." Damit wird nicht ausgeschlossen, dass jede beliebige Anwendung als DDI Target verwendet werden kann.
[62] Kap. 2.6.2, S. 19 in [HAVi2001].

ein BAV ohne DDI-Protokoll, also auch ohne Systemkomponenten des HAVi-Stacks kann bei embedded Anwendungen eingesetzt werden
Mit dem systemunabhängigen DDI-Protokoll werden Nachrichten und Elemente definiert, die eine Anzeige von graphischen Benutzeroberflächen und die Entgegennahme von Benutzereingaben erlaubt. Der DDI Controller übernimmt dabei die Aufgabe der Darstellung und der Weitergabe der Benutzerkommandos. Er erhält dazu eine Beschreibung der graphischen Oberfläche durch HAVi-Objekte, die *DDI Elements*. Damit eine Kommunikation zustande kommt, muss sich der DDI Controller beim DDI Target anmelden. Nach der Anmeldung kann der Controller die DDI-Elemente abfragen und darstellen. Dieser Vorgang ist nachfolgend skizziert:

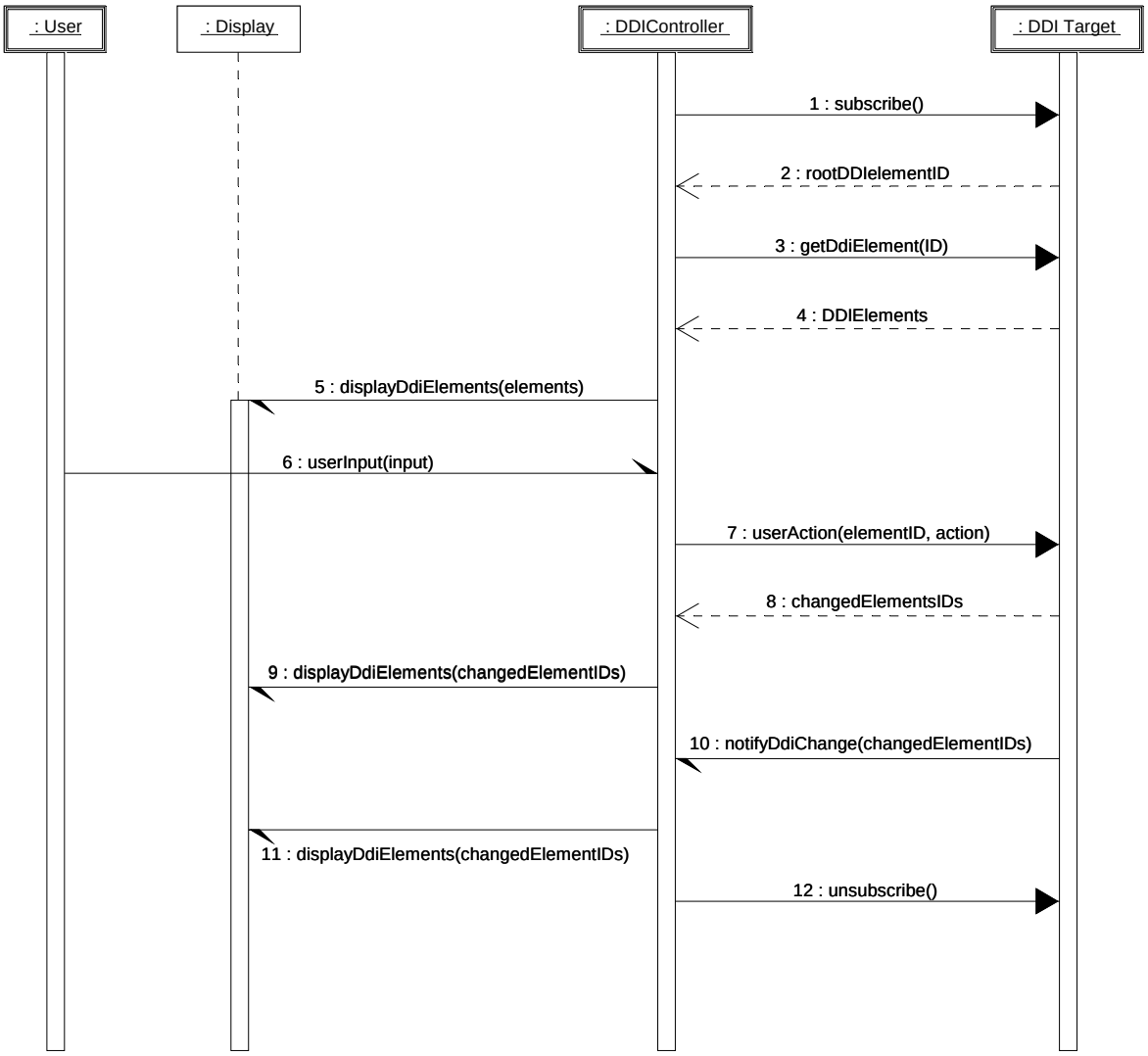

Abbildung 5.25: Nachrichtenablauf zwischen DDI Controller und DDI Target

Das in Abb. 5.25 dargestellte Verhalten wird mit dem entwickelten HAVi-Stack verwirklicht. Die DDI-Elemente sind in einer Hierarchie angeordnet, die durch die gruppierenden Elemente DDI Panel und DDI Group erreicht wird. Innerhalb dieser

Elemente sind alle zugehörigen DDI-Elemente eingeordnet, wie beispielsweise Icons, Buttons, Text oder Bilder.
Mit Hilfe des oben dargestellten Protokolls kann das Aussehen der Anzeige vollständig vom DDI Target bestimmt werden. Das Aussehen kann als Folge von Benutzereingaben (6 in Abb. 5.25) als auch durch Anregung des DDI Targets (10 in Abb. 5.25) neu aufgebaut werden.

Tabelle 5.5: Vergleich der Anforderungen und der Umsetzung des HAVi-Stacks

Element	FAV	IAV	BAV	FAV Impl.[63]	BAV Impl.[63]
Java Runtime	✓[64]			✓	✓
Application Module	✓	✓[65]		✓	✓
DDI Controller	✓	✓		✓	
Resource Manager	✓	✓			
Stream Manager	✓	✓			
DCM Manager	✓	✓		✓	
Registry	✓	✓		✓	✓
Event Manager	✓	✓		✓	✓
Messaging System	✓	✓		✓	✓
1394 Com. Media Manager	✓	✓		✓	✓
SDD Data	✓	✓	✓	✓	✓
DCM	✓	✓	✓	✓[66]	✓

Tab. 5.5 zeigt die Gegenüberstellung der Komponenten, die in der Spezifikation vorgeschrieben werden, und die Umsetzung des FAV/BAV für die hier vorliegende Arbeit. Dabei erkennt man gut die Nutzung des BAV von Komponenten, die eigentlich einem FAV vorbehalten sind. Auf der Basis des DCM und des DDI-Protokolls wird nun die EIB-Anbindung entwickelt. Das zugehörige Datenmodell des EIB wurde bereits in Kap. 3 vorgestellt. Eine Schnittstellenbeschreibung, wie sie im Systemmodell vorgesehen ist, entfällt hier, da die Kommunikation über das in der Spezifikation definierte DDI-Protokoll erfolgt.

[63] Basierend auf selbst erstellten HAVi-Stack
[64] Soweit nicht anders angegeben: vorhanden
[65] Soweit nicht anders angegeben: optional
[66] Optional, jedoch DCM von BAV.

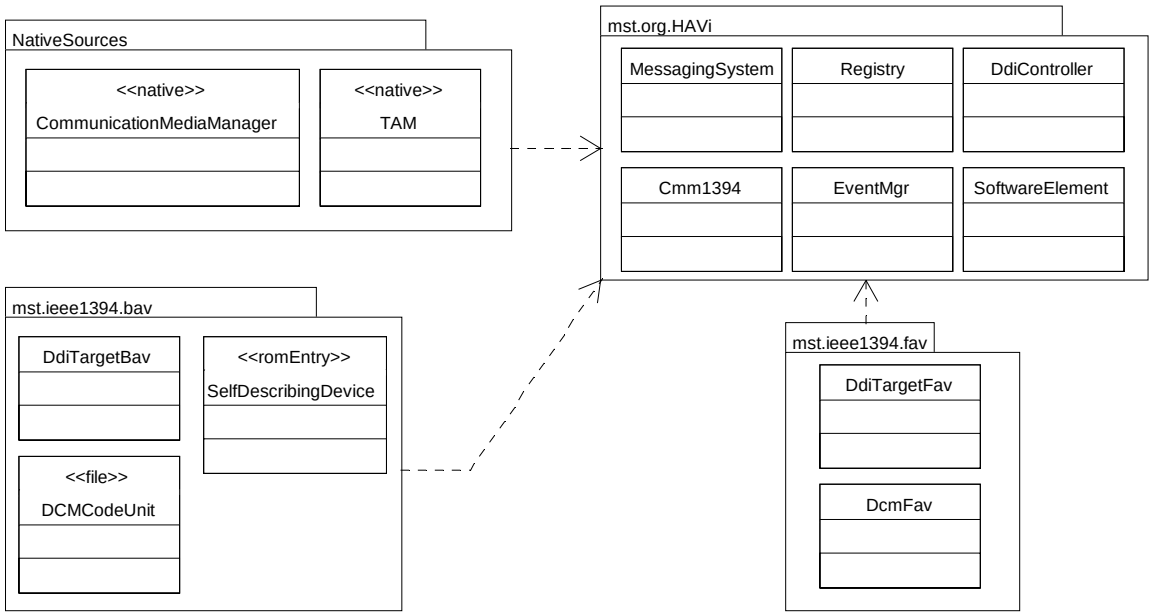

Abbildung 5.26: Paketeinteilung des Java HAVi-Stacks

Abb. 5.26 zeigt die sehr grobe Paketeinteilung des Stacks. In *NativeSources* werden alle Module für die Treiberanbindung und die JNI-Implementierung für den CMM und das TAM abgelegt. Die Hauptanteil des Stacks ist im Paket *mst.org.HAVi* zu finden, wobei hier noch weitere Pakete vorhanden sind, die aus Übersichtlichkeitsgründen weggelassen worden sind. Dazu zählen Konstanten aus *mst.org.havi.constants*, Hilfsklassen aus *mst.org.havi.types* und *mst.org.havi.system*. Im Paket *mst.ieee1394.bav* sind die verwendeten Hauptklassen des BAV, in *mst.ieee1394.fav* die für das FAV untergebracht; auch hier wurde wieder eine Vereinfachung vorgenommen.

5.2.2.4 Erstellung der Kommunikationsschichten für den EIB

Für die Anbindung des EIB stehen je nach Anwendung unterschiedliche Verfahren zur Verfügung. Es kann eine BCU verwendet werden, die bereits alle Schichten des EIB-Protokollstacks enthält. Das ist bei Anwendungen mit einer geringen Anzahl von Kommunikationsobjekten sinnvoll, weil hier die Verwaltung der Kommunikationsobjekte bereits von der BCU übernommen werden kann. Da jedoch der Speicherplatz zur Verwaltung der Kommunikationsobjekte je nach BCU-Typ kaum mehr als 7 Objekte (2 Byte Float, BCU1) fassen kann, ist eine Verwendung von BCUs nur bei kleinen Sensor- und Aktoranwendungen sinnvoll. Für die Verwaltung einer Vielzahl von Kommunikationsobjekten muss das Prozessabbild anders aufgebaut werden. Für diesen Zweck kann ein *Twisted Pair UART* (TPUART), ein ASIC, der die physikalische Schicht und Teile der Sicherungsschicht des EIB integriert, eingesetzt werden. Alle Telegramme des EIB können so abgehört und verarbeitet werden. Ein weiterer Vorteil des TPUART ist, dass die Verarbeitung der Telegramme für die unbestätigte Gruppenkommunikation nicht zeitkritisch ist. Dies gilt unter der Voraussetzung, dass der Host Controller, der den TPUART steuert, nicht als aktives Element, also als eigenständiges Kommunikations-

objekt angesehen werden soll. Ansonsten müssten an den TPUART gerichtete Telegramme über ein *Immediate Acknowledge*[67] bestätigt werden.
Für die Umsetzung der EIB-Kommunikationsobjekte in ein Datenmodell, das in das Systemkonzept integriert werden kann, wurde in dieser Arbeit ein TPUART verwendet. Der Vorteil des TPUART bei dieser Anwendung[68] lässt sich auf die serielle Verbindung zwischen Host Controller und TPUART zurückführen. Es wird ein einfaches serielles Protokoll verwendet[69], das sich über die so genannte CommAPI[70] in Java einbinden lässt, obwohl in Java keine Garantie der Verarbeitungszeiten eines Befehlszyklus gegeben werden kann. Da die HAVi-Spezifikation eine Implementierung in Java vorsieht, liegt es nahe, auch die EIB-Anbindung in Java zu implementieren.
Die Buskommunikation wird auf unterster Ebene durch eine spezialisierte Bussystemzugangsklasse verarbeitet. Dies trägt dem Umstand Rechnung, dass neben der Umsetzung mit einem TPUART auch eine BCU1, BCU2 oder ein EIB-Modem verwendet werden kann. Bisher implementiert wurde der TPUART, das EIB-Modem[71] und BCU1-Protokoll, wobei beim BCU1-Protokoll ein *native* Treiber[72] über das JNI aus den bereits genannten Gründen angepasst werden musste. Über die physikalische Zugriffsschicht wird eine Pufferstruktur gelegt, die die Verwendung der gleichen Schnittstelle durch mehrere Anwendungen erlaubt. Darauf baut die Verwaltung des Prozessabbildes auf, indem der Status von vorkonfigurierten Kommunikationsobjekten verfolgt wird. Die Telegrammfilterung findet erst in der Verwaltung, also in dem Datenmodell statt. Dort werden die Empfangsadressen mit dem eingegangenen Telegramm verglichen und die Information an das Datenobjekt weitergegeben. Eine an diesem Objekt interessierte Anwendung kann sich direkt bei dem Objekt anmelden und wird im Falle einer Änderung direkt informiert. Die von der Verwaltung geführten Objekte stellen eine abstrahierte Ansicht der Kommunikationsobjekte des EIB dar und sind aus Anwendungssicht reine Informationsträger. Zu diesem Zweck findet auch eine Konvertierung des Datenformats statt. Durch den *EIB Interworking Standard* (EIS) sind 10 Grunddatentypen definiert, die teilweise wiederum in Untertypen gegegliedert sind. Ein EIB-Objekt zur Übermittlung von Schaltinformationen ist durch den EIS Type 1 beschrieben, alle skalierten Werte (Temperatur, Druck, Windrichtung) werden durch EIS Type 6 (Scaling) beschrieben. Innerhalb der Objektverwaltung werden alle diese Datentypen auf Java-spezifische

[67] Das *Immediate Acknowledge* findet sich in [EIB1999] wieder. Durch die Sicherungsschicht muss auch bei der Gruppenkommunikation wenigstens ein Teilnehmer den Empfang bestätigen.
[68] Da zur Erstellung des Prozessabbildes nur Gruppenkommunikation (Multicast) verwendet wird, kann das mit Standardmethoden in Java gelöst werden.
[69] RS232 mit TTL-Pegeln; 8E1, also 8 Datenbits, Even Parity und ein Stop Bit. Übertragung bei 19200 Bit/s.
[70] Die CommAPI definiert Schnittstellen und Klassen für den Zugriff auf serielle und parallele Schnittstellen.
[71] Das EIB-Modem ist eine Entwicklung des Lehrstuhls für Messsystem- und Sensortechnik und besteht aus einer BCU1 und einem Mikrocontroller, der das asynchrone, hardware-gesteuerte Handshake Protokoll der PEI in eine Zweidrahtverbindung (RxD, TxD) übersetzt und damit ähnlich wie ein TPUART angesprochen werden kann - mit dem Vorteil einer vollständig implementierten Data Link Layer.
[72] Dieser wurde von der Technischen Universität Wien für Linux entwickelt. Es erfolgte eine Anpassung an die JVM und kleinere Änderungen im Treiberaufbau.

primitive Datentypen umgewandelt. Da Java nur vorzeichenbehaftete Datentypen anbietet, sind vorzeichenlose EIB-Datentypen durch einen höherwertigen Typ darzustellen, um ein Type-Casting zu vermeiden. So wird ein 8 Bit Unsigned Integer des EIB (Wertebereich 0..256) auf ein 16 Bit vorzeichenbehafteten Short Wert[73] (Wertebereich -32768..32767) umgewandelt.

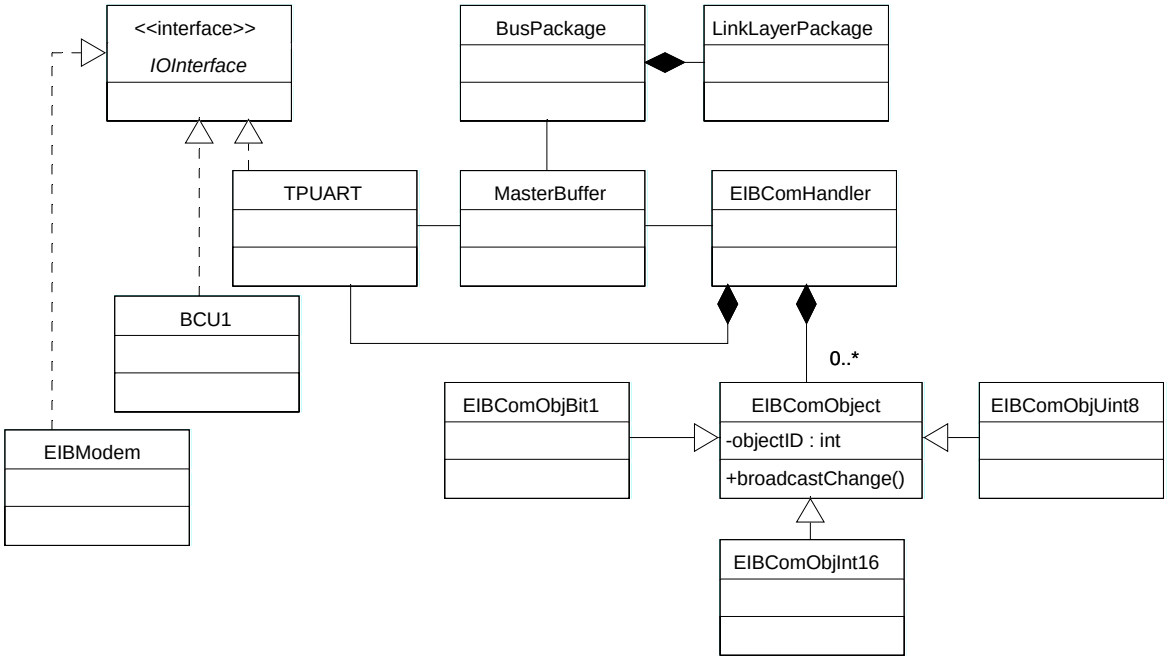

Abbildung 5.27: Umsetzung des Datenmodells für den EIB

Der *EIBComHandler* ist der Zugangspunkt für die Anwendungen, die Daten des EIB-Prozessabbildes abfragen wollen. Damit eine Zuordnung zwischen EIB und Datenobjekt möglich ist, wird jedem *EIBComObject* eine *objectID* vergeben, durch die das Element identifiziert werden kann. Es werden alle Datentypen zur Verfügung gestellt, die zur Steuerung und Überwachung des EIB benötigt werden. Nicht eingezeichnet sind die Verwaltungsstrukturen, die die Filterung der ankommenden Telegramme ermöglichen, sowie die Fähigkeit, die Objektverwaltung auch über RMI ansprechen zu können. Der *EIBComHandler* übernimmt viele Aufgaben, so auch das Auslesen der Konfigurationsdaten und die Erstellung eines Datenobjekts sowie die Eintragung der zugehörigen Gruppenadressen. Die Konfigurationsdatei ist die Verbindung zwischen EIB und Anwendung. Die Anwendung kennt nur Datenobjekte zur Speicherung von elementaren Datentypen, die durch ihre *objectID* erreicht werden können; sie muss sich nicht mehr um busspezifische Gruppenadressen kümmern.
Das *IOInterface* erleichtert die Abstrahierung des Buszuganges von der Datenpufferung. Durch das Interface wird die darunterliegende Kommunikation nebensächlich, da nur mit den *BusPackages* gearbeitet wird.

[73] Der kleinere Datentyp Byte umfasst nur einen Wertebereich von –128..127 und kann daher nicht verwendet werden.

Des Weiteren können mehrere Objektverwaltungen über eine RMI-Schnittstelle miteinander gekoppelt werden. Damit ist ein Übergang durch KomTyp 3 möglich. Es gibt bereits Umsetzungen für TCP/IP über IEEE 1394. Durch Austausch der Transportschicht ist folgender Einsatz denkbar:

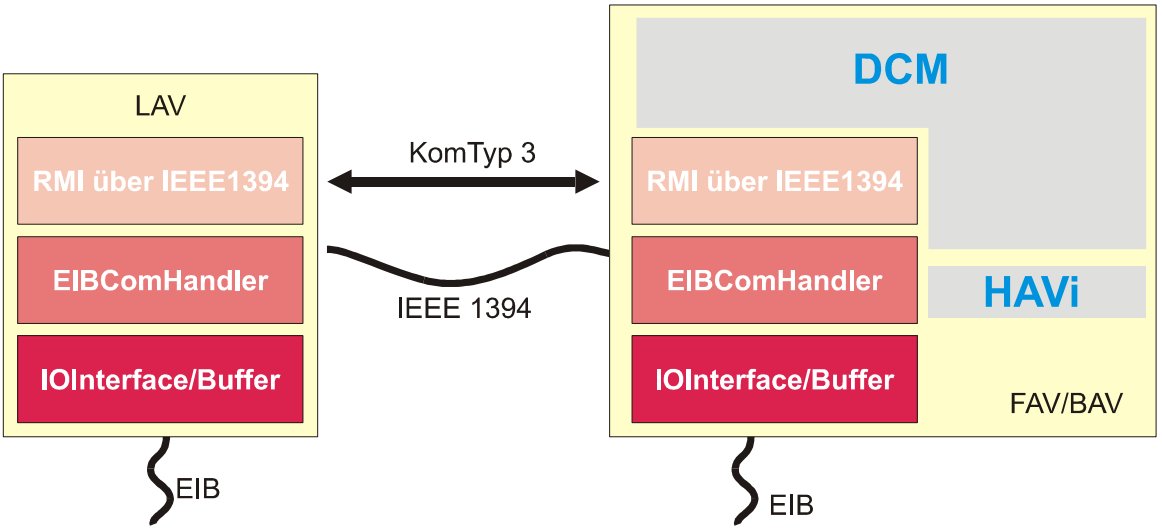

Abbildung 5.28: Systemübergang zwischen zwei EIB Installation durch IEEE 1394

Der Übergang in der linken Hälfte ist an IEEE 1394 und an den EIB angeschlossen. Dieses Gerät ist als LAV zu bezeichnen, da es kein HAVi-Protokoll unterstützt. Auf dem *EIBComHandler* wird RMI über IEEE1394 aufgesetzt, das zur Kommunikation mit einem FAV/BAV eingesetzt wird. Damit sind die Prozessabbilder zweier Systeme miteinander kombinierbar. Bei der Zusammenschaltung beider Verwaltungen werden die Zustände gedoppelt, wobei jede lokale Änderung unverzüglich der anderen Verwaltung mitgeteilt wird. Diese Verbindung kann auf dem FAV/BAV dazu verwendet werden, um gleichzeitig Funktionalität aus beiden Installationen auf einem anderen HAVi-Gerät verfügbar zu machen. Der Vorteil der Verkopplung zweier *EIBComHandler* liegt darin, dass der Zustand lokal abgefragt werden kann und dass das DCM den Unterschied zwischen einem lokalen und einem entfernten Datenobjekt nicht feststellen kann. Für die Verkopplung ist jedoch eine Zusatzinformation notwendig, um den Aufenthaltsort des Datenobjekts innerhalb der Verwaltung feststellen zu können. Diese Information kann ähnlich wie bei der GUID des IEEE 1394 innerhalb der *objectID* gehalten werden.
Die Verbindung des Datenmodells und des DCMs, also dem eigentlichen sichtbaren Dienst innerhalb von HAVi, wird durch Zusatzklassen hergestellt, die die Statusinformationen des EIB nach einer vorgegebenen Anordnung auf DDI-Elemente abbildet. Die Oberfläche wird über ein entwickeltes Konfigurationsprogramm erstellt, im XML-Dateiformat gespeichert und vom DCM wieder eingelesen. Die XML-Datei kann entweder bereits in der DCM Code Unit enthalten sein, oder über den IEEE 1394 von einem beliebigem Gerät innerhalb des Bussystems geladen werden.

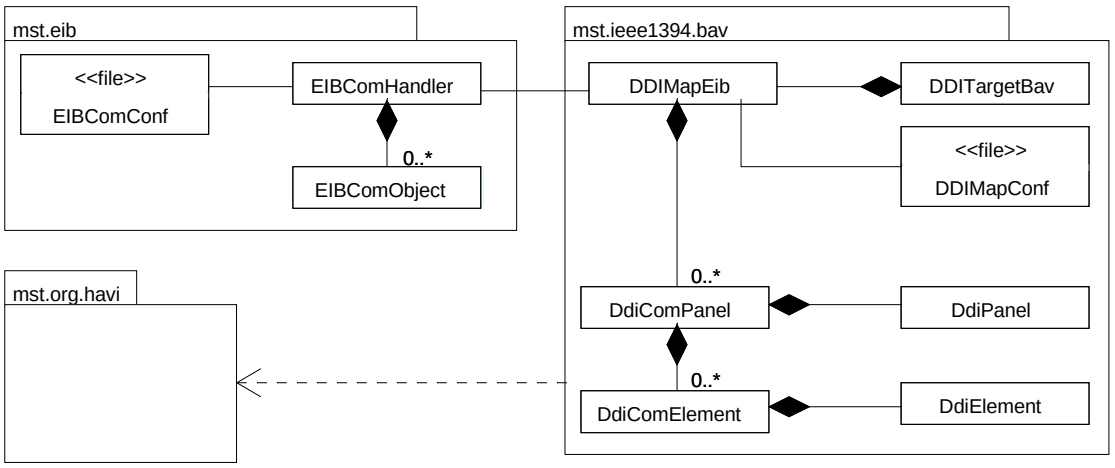

Abbildung 5.29: Einordnung des EIB-Datenmodells in HAVi

Das BAV verwendet die Klassen aus dem Datenmodell des EIB, die in dem Package *mst.eib* zu finden sind. Das *DDITargetBav* besitzt eine Referenz auf die Klasse *DDIMapEib*, die hier symbolisch die Verknüpfung der beiden System darstellt.
Die Datenobjekte des *EIBComHandlers* (*EIBComObject*) können mit einer Hüllklasse zu einem *DdiComPanel* bzw. *DdiComElement* zusammengefasst werden. Das *DdiComPanel* ist für die Darstellung eines graphischen Fensters zuständig und nimmt *DdiComElements* auf. Jede dieser Hüllklassen besitzt dabei eine Entsprechung des HAVi-Modells (*DdiPanel*, *DdiElement*). Dadurch wird der Übergang zwischen EIB und HAVi vervollständigt und eine Abbildung des EIB-Datenmodells auf das BAV respektive den HAVi-Dienst zur Anzeige und Steuerung erreicht.

5.2.2.5 Übergang Jini-HAVi

Der in dieser Arbeit nicht realisierte Übergang zwischen Jini und HAVi soll an dieser Stelle der Vollständigkeit halber vorgestellt werden.
Aus den vorherigen Betrachtungen wird klar, dass ein DCM in HAVi vergleichbar mit einem Dienst unter Jini ist, sieht man von der unterschiedlichen Schnittstellendefinition ab. Während zur Umsetzung des EIB auf HAVi das DDI verwendet wurde, werden beim Übergang von EIB zu Jini die EIB-Geräte in einem Jini-Service abgebildet.
Der Übergang kann nur durch die Abbildung der Jini-Komponenten in ein DCM und der HAVi-Komponenten in einen Jini-Service erfolgen. Der verantwortliche Knoten muss dazu auf der IEEE 1394 Seite als FAV ausgeführt werden. Es ist zwar durchaus möglich, auch ein BAV mit einer DCM Code Unit zu verwenden, dadurch erhöht sich aber der Protokoll-Overhead – ein leistungsschwaches embedded System kann hier ohnehin nicht zur Anwendung kommen, da die Verarbeitung der beiden Middleware-Protokolle Jini und HAVi einen relativ hohen Rechenaufwand erfordert.

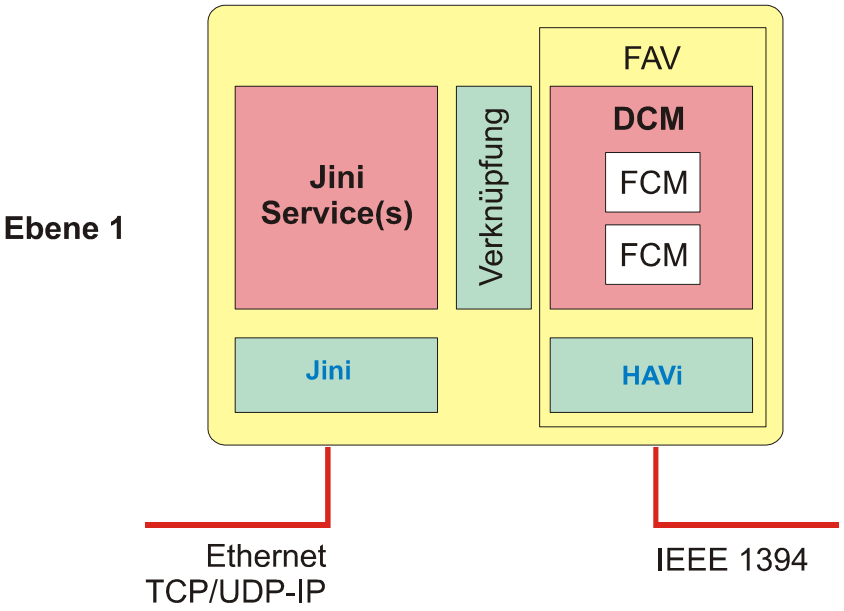

Abbildung 5.30: Verbindung der Dienstebenen Jini und HAVi

Die Zuordnung der einzelnen Jini-Dienste kann auf die FCMs abgebildet werden, so dass der Datenübergang innerhalb von HAVi als ein gesamtes Gerät erscheint, während die Einbettung der Jini-Dienste in die FCMs überführt wird.

5.2.2.6 Auslastungsanalyse

Der Hauptanteil des Datenverkehrs bei dieser HAVi-Implementierung besteht aus DDI-Nachrichten, die zwischen dem BAV und dem FAV ausgetauscht werden. Eine gute Übersicht wurde bereits durch Abb. 5.25 gegeben, in der die Anmeldungssequenz des DDI Controllers und beidseitig ausgelöste Ereignisfolgen beschrieben werden. Für den Betrieb sind nur wenige Nachrichten von Interesse. Die Anmeldung des DDI Controllers und der Austausch der angezeigten Elemente erfolgt nur bei der Initialisierung. Bei einer *UserAction* werden nachfolgend die geänderten Elemente zurückgegeben.
Anders als bei Jini gibt es bei HAVi keinen Grundverkehr, da Geräteausfälle bereits durch IEEE 1394 bemerkt und weitergeleitet werden. Daher sind Konzepte wie Leasing weitgehend überflüssig, wenn auch Software-Ausfälle nicht bemerkt werden. Dennoch spezifiziert HAVi keine Mechanismen zur Erkennung des Ausfalls von Software-Elementen.
Erhält das Messaging System eine Nachricht von einem Software-Element, die für einen anderen Geräteknoten vorgesehen ist, wird diese Anfrage an das TAM weitergeleitet. Das Format einer solchen Anfrage ist nachfolgend dargestellt:

Tabelle 5.6: Nachrichtenformat für einen ausgehenden Operationsaufruf

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	…	Byte 13
0011 rrrr	nnnn nnff	0Rnn nnnn	0000 0000	dddd dddd	...	dddd dddd
FCPHdr	TAMHdr		Reserved	Destination SEID		

Byte 14	…	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27
ssss ssss	...	ssss ssss	0000 0000	0000 0010	NNNN NNNN	0000 0000
Source SEID			ProtType	MsgType	MsgNo	reserved

Byte 28	Byte 29	Byte 30	Byte 31	Byte 32	Byte 33	Byte 34
LLLL LLLL	LLLL LLLL	LLLL LLLL	LLLL LLLL	oooo oooo	oooo oooo	oooo oooo
Message Length				Operation Code		

Byte 35	Byte 36	Byte 37	Byte 38	Byte 39
cccc ccc0	TTTT TTTT	TTTT TTTT	TTTT TTTT	TTTT TTTT
Control Flag	Transaction Id			

Innerhalb von HAVi werden vier Nachrichtentypen verwendet: Eine Anforderung (*MsgReliable*) und deren zwei Bestätigungen sowie eine unbestätigte Nachricht (*MsgSimple MsgType 0x01*). Die Bestätigung einer Nachricht (*MsgReliable*), angegeben durch den *MsgType* 0x02, erfolgt mit einem *Acknowledge* (*MsgReliableAck 0x03*) oder einem *NoAcknowledge* (*MsgReliableNoAck 0x04*) Telegramm. Die hier nicht aufgeführten Bestätigungen stimmen bis zum Byte 27 im Aufbau überein. Das *Acknowledge* endet nach Byte 27, das *NoAcknowledge* enthält noch weitere Informationen über den Grund der negativen Quittierung.
Der FCPHdr enthält den *Command and Transaction Set* (CTS), der in [HAVi2001] neu spezifiziert wurde. Dadurch können andere IEC61883-konforme Geräte erkennen, dass es sich um eine HAVi-Nachricht handelt. Der TAM header enthält Informationen über die Fragmentierung, und zwar über die Art[74] (Kennzeichnung „ff“ in TAMHdr Tab. 5.6) und die Nummer des Fragments. Ein sehr wichtiges Element ist die *DestSEID*, in der der GUID enthalten ist und damit die Zieladresse des Teilnehmers. Außerdem enthält die SEID die Nummer des Ziel Software-Elements auf dem entfernten Teilnehmer und wird vom Messaging System zur Weiterleitung benötigt. Der *Operation Code* gibt Auskunft über die Art der Operation und ist vergleichbar mit dem *Operation* Hash bei Jini. Die *TransactionId* wird verwendet, um die Antworten mehrerer Anfragen trennen zu können und wird vom Empfänger in der Antwort wieder vermerkt.
Um die Nachrichtenlänge einer Nachricht zu veranschaulichen, wurde in der nachfolgenden Abbildung ein Mitschnitt der Kommunikation zwischen BAV und FAV,

[74] Dies sind: Simple Fragment Packet, Begin of Message Packet, Continuation Of message Packet, End of message Packet.

den DDI Targets[75] dargestellt. Die in Klammern gesetzten Zahlen vor den Befehlen beziehen sich auf Abb. 5.25.

```
DDI Target FAV (48 Bytes)
(1) Subscribe (CTS, DestSEID (GUID, Software-Element Nr.),  SourceSEID (GUID,
Software-Element Nr.), MsgType Reliable, MsgNo:1, OperationCode:subscribe)
30 08 00 00 00 01 b7 00 00 00 8d ba 01 01 00 01 b7 00 00 00 8d b5 01 01 00 02
01 00 00 00 00 10 00 0a 00 00 00 00 00 01 00 14 00 00 00 00 00 00

DDI Target BAV (28 Bytes)
Acknowledge (CTS, DestSEID, SourceSEID, MsgType Ack Reliable, MsgNo: 1)
30 a0 00 00 00 01 b7 00 00 00 8d b5 01 01 00 01 b7 00 00 00 8d ba 01 01 00 03
01 00

DDI Target BAV (52 Bytes)
(2) DDI ElementID – PanelID (Msg Type Simple)
30 a4 00 00 00 01 b7 00 00 00 8d b5 01 01 00 01 b7 00 00 00 8d ba 01 01 00 01
36 00 00 00 00 14 00 0a 00 01 00 00 00 01 00 0a 00 00 00 00 00 00 75 30 00 01

DDI Target FAV (44 Bytes)
(3) GetDdiPanel (TransactionID:2, DDIElementHandle: 0x7530, DDIElementType:
DDIPanel)
30 0c 00 00 00 01 b7 00 00 00 8d ba 01 01 00 01 b7 00 00 00 8d b5 01 01 00 02
02 00 00 00 00 0c 00 0a 03 00 00 00 00 02 75 30 00 01

DDI Target BAV (28 Bytes)
Acknowledge
30 a8 00 00 00 01 b7 00 00 00 8d b5 01 01 00 01 b7 00 00 00 8d ba 01 01 00 03
02 00

DDI Target BAV (364 Bytes)
(4) DdiElements(Länge (330 Bytes), Operation Code (GetDdiPanel), TransactionID:
2, DDIElementHandle: 0x7530, Panel Name: “Chose Panel”, ChoiceElement1:
“Namenlos 4”)
30 ac 00 00 00 01 b7 00 00 00 8d b5 01 01 00 01 b7 00 00 00 8d ba 01 01 00 01
37 00 00 00 01 4a 00 0a 03 01 00 00 00 02 00 0a 00 00 00 00 00 00 75 30 00 01
00 00 00 0d 00 43 00 68 00 6f 00 6f 00 73 00 65 00 20 00 50 00 61 00 6e 00 65
00 6c 00 00 00 cd 01 2c 00 00 00 00 00 00 00 00 00 01 75 31 00 0d 00 00 00 01
00 00 02 58 00 00 00 00 00 00 00 01 00 0d 75 31 00 0d 00 00 00 00 00 11 00 61
00 76 00 61 00 69 00 6c 00 61 00 62 00 6c 00 65 00 20 00 50 00 61 00 6e 00 65
00 6c 00 73 00 00 00 5a 00 c8 00 00 00 00 00 00 00 00 00 02 00 01 00 00 00 00
00 00 00 00 00 01 00 00 00 04 00 00 00 0a 00 4e 00 61 00 6d 00 65 00 6e 00 6c
00 6f 00 73 00 34 00 00 00 00 00 01 00 03 00 00 00 …
```

Abbildung 5.31: Protokollaufzeichnung des DDI-Protokolls

Alle Daten sind dem TAM des Messaging System entnommen, stellen also den Datenanteil dar, der über ein FCP Paket [IEC61883] auf IEEE 1394 übertragen wird. Zusätzliche Daten für den Zugriff auf das Medium wurden hier nicht weiter berücksichtigt, weil sie für die vorliegenden Untersuchungen nicht weiter relevant sind. Da eine Fragmentierung in [IEC61883] nicht vorgesehen ist, wird dies Aufgabe vom TAM übernommen.
Das zurückgelieferte *RootPanel* enthält zwei Arten von Elementen: Eine *DdiGroup* zur graphischen Gruppierung und vier *DdiChoices*. Die Anfrage für den Subscribe-Befehl

[75] Dieser Nachrichtenverkehr ist vergleichbar mit der Kommunikation eines FAV mit einem FAV unter Verwendung des DDI-Protokolls. In der Implementierung ist es jedoch das DDI Target des FAV, das mit dem DDI Target des BAV kommuniziert.

besteht aus 48 Bytes, die durch das BAV mit einem Acknowledge bestätigt wird. Man erkennt hier gut die Bestätigung durch die fortlaufende Nummerierung der Nachrichten. Beide haben übereinstimmend die Nachrichtennummer 1. Die Nutzdaten mit der Panel ID des *DdiPanels* (0x7530) wird über eine ungesicherte Nachricht (*MsgSimple*) versendet und wird daher nicht vom FAV bestätigt. Nach Erhalt der DDI Panel ID fordert das FAV dessen Inhalt an. Diese Anfrage wird wiederum bestätigt und der Inhalt (*DdiElementList* – eine Liste von DdiElementen) vom BAV durch eine unbestätigte Nachricht übermittelt.
Der hier gezeigte Nachrichtenaustausch verwendet keine Fragmentierung. Dennoch kann es bei anderen Elementen vorkommen, dass der Inhalt größer als die erlaubten 508 Bytes (512 – 4 Bytes aus dem TAM) wird. So müssen bei Elementen vom Typ *DdiButton* die Bilder der Darstellungen als Byte Stream mitübertragen werden. In diesem Fall werden mehrere Nachrichten gesendet, die anschließend durch das TAM zusammengesetzt werden.
Aus den obigen Betrachtungen lässt sich leicht erkennen, dass die Nachrichtengrößen bei der Verwendung des DDI Panels selten den IEEE 1394 auslasten werden. Auch hier sind kurze Nachrichten eher von Nachteil, da zu jeder Nachricht ein Protokollkopf generiert werden muss und die Programmlaufzeiten weitaus größer sind, als die Übertragungszeiten der Nachrichten selbst. Verglichen mit Ethernet steht hier auch eine größere Übertragungsrate zur Verfügung (> 100 MBit/s), auch wenn ein Teil davon für synchrone Daten reserviert sein kann (80% nach [Anderson1999]). Selbst dann dürften die übertragenen Datenmengen des DDI-Protokolls keine Belastung darstellen. Auch wenn eine Arbitrierung innerhalb eines IEEE 1394 Zyklus nicht möglich sein sollte, werden die zu bestätigenden Telegramme im Fehlerfall von der HAVi-Architektur wiederholt.

5.2.3 DECT

Für den Zugriff auf Daten der Heizung/Klima/Lüftung wird der bereits vorgestellte PDA verwendet, der über ein DECT-Funkmodul und ein Proxy an die Dienstumgebung angebunden wird. Der Proxy nimmt Anpassungen des Kommunikationsprotokolls vor, so dass Befehle des PDAs den Dienstanbieter erreichen können. Der PDA wird hier nur als Dienstnehmer eingebunden, nicht als Dienstanbieter.
Der Datenverkehr zwischen dem PDA und dem Proxy erfolgt über so genannte DECT-Modems der Firma Siemens. Dabei handelt es sich um eine Kombination aus DECT Basisband-Controller und einem Mikrocontroller. Der Mikrocontroller übernimmt dabei die Verwaltung der höheren Kommunikationsschichten des proprietären Protokolls, während der Basisband-Controller die Umformung der Datenströme in ein Funksignal und die unteren Schichten des Protokolls übernimmt. Die Module können in mehreren Betriebsmodi verwendet werden, darunter als serieller Schnittstellenersatz und als Punkt-zu-Punkt-Datenverbindung zwischen einem Mobilteil und der Basisstation. Dazu muss ein Modem als Basisstation und das andere als Mobilteil konfiguriert werden. Hier zeigt sich der zentrale Ansatz der DECT-Spezifikationen, der immer eine Basisstation als Vermittler benötigt. Eine Kommunikation zwischen zwei Datenmodulen kann auch in diesem Fall nur durch eine Weiterleitung der Daten durch die Basisstation erfolgen. In der Dokumentation

gibt es leider keinen Hinweis auf das verwendete Protokoll zur Datenübermittlung und die Verwendung der Dienste des Network Layers. Auf Grund der beschriebenen Konfigurationsparameter des DECT-Modems lässt sich vermuten, dass die Standard-verfahren zur Anmeldung eingehalten werden.

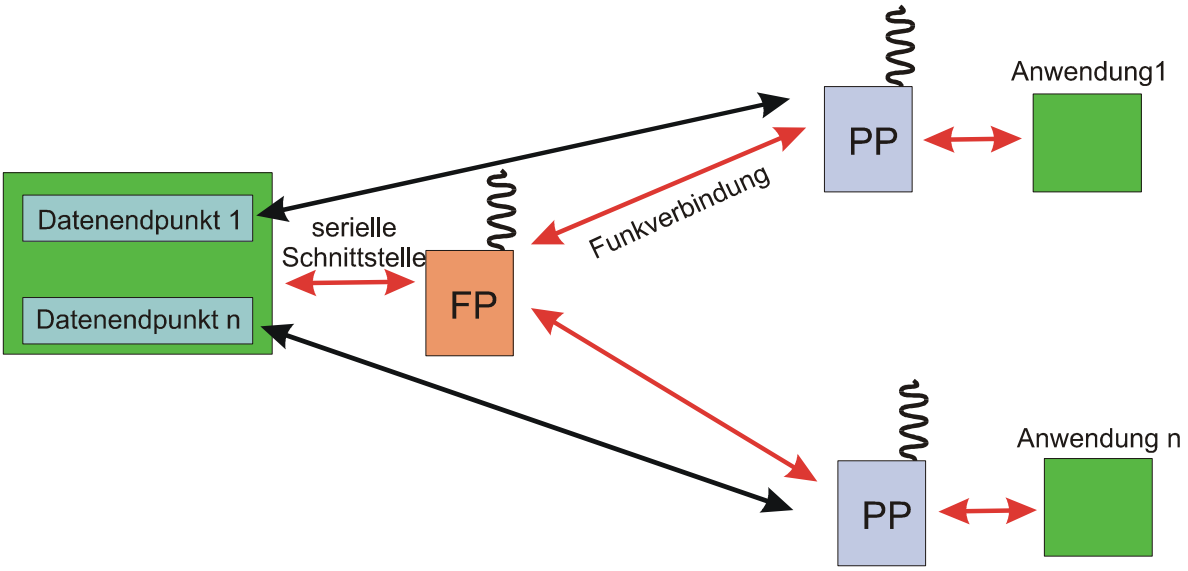

Abbildung 5.32: Verbindung zwischen den DECT-Modems

Die Kommunikation[76] mit den DECT-Modems erfolgt durch ein serielles Protokoll (V.24), dessen Protokollinhalt vom verwendetem Modus abhängt. Das Mobilteil als V24-Ersatz benötigt kein besonderes Protokoll zum Versenden von Daten. Durch die Anmeldung bei der Basisstation ist der Kommunikationspartner bereits festgelegt. Damit werden alle über die serielle Schnittstelle empfangenen Daten an die Basisstation weitergegeben.
Für die Kommunikation mit einem Mobilteil muss über ein besonderes Protokoll die Mobilteilnummer angegeben werden. Im Protokollrahmen werden den Daten die Empfängeradresse und Steuercodes für die Signalleitungen der seriellen Schnittstelle vorangestellt. Von der Basisstation können Datenverbindungen zu vier Mobilteilen gleich-zeitig aufgebaut werden, wobei sich die Datenrate von theoretisch 24,2 kBit/s auf ein Viertel verringert.
Der PDA wird für die Steuerung des EIB-Dienstes eingesetzt. Daher wird auf Seite des PDAs eine Datenstruktur aufgebaut, um die Datenzustände des EIB zu verwalten. Obwohl der PDA lediglich als Anzeige vorgefertigter Graphiken genutzt werden könnte, ist es dennoch von Vorteil, die Erzeugung der Darstellung auf dem PDA vorzunehmen, da dann nur Zustandsänderungen der EIB-Objekte übermittelt werden müssen und der PDA selbst für den Anzeigeaufbau verantwortlich ist.
Die Abfrage der Datenzustände kann entweder über die Dienstimplementierung des EIB oder aber über eine Verbindung KomTyp 3 zu einer nicht direkt angebundenen EIB-Installation erfolgen. Der Proxy verwendet zur Abbildung der Daten eine Kommuni-

[76] In [SIEMENSDECT] ist die genaue Beschreibung der Modi und des Protokolls zu finden.

kationsobjektverwaltung, wie sie in Kap. 5.2.2.4 vorgestellt wurde. Der schematische Aufbau ist im folgenden Diagramm dargestellt.

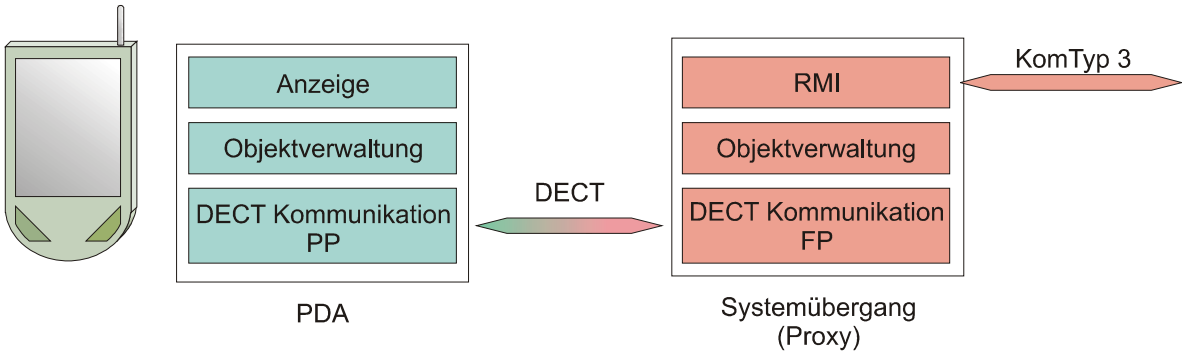

Abbildung 5.33: Kommunikation zwischen PDA und Proxy

Die vorhandene Umsetzung wurde nach Abb. 5.33 vorgenommen. Die Anzeige der EIB-Daten am PDA wird aus einem gespiegelten Zustandsabbild gewonnen. Das lokale Zustandsabbild des PDA wird mit dem auf dem Proxy vorhandenen Abbild synchronisiert. Durch die Nutzung von RMI kann das überwachte Bussystem entweder direkt an den Proxy angekoppelt sein oder aber an einer anderen Stelle im System eingebunden sein.
Die Implementierung der Kommunikation und der Darstellung des PDA wurde in Java programmiert. Auf Grund des begrenzten Speichers wurde eine spezielle JVM verwendet, die *Java 2 Platform Micro Edition* (J2ME). Diese JVM benötigt lediglich zwischen 160 und 512 kByte und richtet sich an „[...]small, resource-constrained, connected devices [...]“ [SUNCLDC]. Innerhalb der Spezifikation werden die Mindestanforderungen der JVM definiert. Im Vergleich zu einer Standard-JVM wurde auf einige Elemente verzichtet, um eine Hardwareunabhängigkeit zu erreichen, die den Einsatz auf möglichst vielen mobilen Geräten erlaubt. Dazu gehört auch das JNI, so dass beim Einsatz dieser *Connected Limited Device Configuration* (CLDC) JVM keine hardwarenahen Schnittstellen angesprochen werden können. Die einzige Verbindung mit der Außenwelt sind HTTP-Verbindungen über Funkschnittstellen. Der in der Referenzimplementierung[77] verfügbare Zugriff auf die serielle Schnittstelle wurde mittlerweile gestrichen. Die JVM bietet selbst keine Möglichkeit zur graphischen Anzeige. Die dazu notwendigen APIs sind Teil von weiteren Spezifikationen (hier der KJavaAPI). Ein neuerer Ansatz sind die *Mobile Information Device Profiles* (MIDP), die neben Kommunikationsprofilen auch die graphische Darstellung beinhalten.

[77] In dieser Umsetzung wurde die KVM (K JVM) zusammen mit der KJavaAPI zur graphischen Anzeige verwendet, die ähnlich dem AWT in einer J2SE ist.

Abbildung 5.34: Darstellung der Steuerung auf dem Palm V

Ein Beispiel für eine solche Darstellung ist in Abb. 5.34 zu sehen. Es wurde eine Einteilung in Räume unternommen, in denen mehrere Elemente angezeigt werden können.

5.2.3.1 Auslastungsanalyse

Da die implementierte Kommunikation der Siemens DECT-Modems werkseitig fehlerhaft war und auch auf Anfrage kein Update verfügbar war, musste noch zusätzlich ein Sicherungsprotokoll eingeführt werden. Alle Datenpakete wurden daher zusätzlich durch *Serial Line Internet Protocol* (SLIP) gekapselt. Auf dieser Basis wurde ein Sicherungsprotokoll aufgebaut. Daraus ergab sich ein zusätzlicher Protokollrahmen, der sowohl beim PP als auch beim FP behandelt werden musste.
Die Kommunikationsendpunkte sind im Regelfall die Kommunikationsobjektverwaltung für den EIB auf der einen Seite und die Anzeige der Daten auf dem PDA auf der anderen. Der Aufwand bei den einzelnen Paketen lässt sich ungefähr an einem Beispiel abschätzen:

Tabelle 5.7: Datenaufkommen der Protokollebenen

Protokoll	Datenlänge gesamtes Paket (in Byte)	Nutzdatenlänge (in Byte)
Anwendungsdaten	10	-
Datensicherung	12	10
SLIP[78]	14	12
Siemens	16	14

[78] Kodierung ist vom Dateninhalt abhängig. Berücksichtigt sind nur Start- und Endezeichen eines Blocks, wobei vorausgesetzt wird, dass kein Steuerzeichen des SLIP-Protokolls im Datenpaket vorkommt.

Man erkennt dabei das Anwachsen der Telegrammlänge in Senderichtung. Die 10 Byte Nutzdaten erweitern sich durch die zusätzlichen Protokolle auf 16 Byte. Das ergibt ein Verhältnis von

$$\frac{Nutzdaten}{Paketlänge} = \frac{10}{16} = 62{,}5\%$$

Setzt man voraus, dass die Datenrate im gleichen Umfang sinkt, erhält man einen Wert von:

62,5 % von 19,2 kBit/s = 12 kBit/s

Dieser theoretische Wert berücksichtigt noch nicht das DECT-Protokoll und die Programmlaufzeiten. Daher weicht diese Betrachtung noch von den in einem Test erreichten Übertragungsraten von 8 kByte/s ab. Selbst bei einer Übertragungsrate von 8 kBit/s dauert die Übermittlung eines (Nutz-)Datenpakets von 10 Byte Länge lediglich 10 ms. Damit wirkt sich der Abfall der Nutzdatenrate nur geringfügig auf die Reaktionszeiten aus.

Erst bei größeren Paketlängen wird die Übertragungsrate deutlich höher, wobei die Größe eines Paketes durch das Datenmodul auf 112 Byte beschränkt ist. Der Protokollüberhang hängt von den gesendeten Daten ab, da beim Vorkommen von SLIP-Steuerzeichen im Datenpaket diese in Zwei-Byte-Sequenzen umgesetzt werden, um sie von Steuerzeichen unterscheiden zu können [Romkey1988].

Bisher wurden nur die unteren Schichten des Protokolls beschrieben. Die oben genannten Prozeduren dienen lediglich dazu, einen Duplex-Kommunikationskanal aufzubauen. Dieser Kanal kann dann von einer Anwendung benutzt werden, wie es bei der Kommunikation zwischen der Basisstation und dem PDA der Fall ist. Dazu wurden mehrere Dienste definiert, die sich in zwei Gruppen unterteilen lassen:

- Initialisierung: Laden der Konfigurationsdaten auf den PDA
  Nach Einschalten des PDAs und dem Start des Programms wird mit der Basisstation Kontakt aufgenommen und die Daten der Konfiguration werden geladen. Diese Kommunikation wird von der Basisstation abgewickelt. Mit den Konfigurationsdaten werden die Anzeigedaten übermittelt. Dazu gehören alle anzuzeigenden Objekte, Beschriftungen und Statustexte. Nachdem dieser Vorgang abgeschlossen ist, ist der PDA einsatzbereit und kann zur Anzeige und Steuerung verwendet werden.
- Betrieb: Empfang von Benutzereingaben vom PDA, Senden von Wertänderungen des EIB.
  Bei Empfang eines geänderten Wertes vom EIB für ein bestimmtes Kommunikationsobjekt wird der Zustand intern aktualisiert und an den PDA weitergegeben. Die durchschnittliche Länge eines Telegramms für die Aktualisierung eines Anzeigewertes beträgt ungefähr sieben Bytes. Die Länge einer Benutzeraktion ist gleich lang. In der vorherigen Betrachtung wurde gezeigt, dass die Übertragung nur einen Bruchteil der Zeit einnimmt. Mehr ins Gewicht fällt die Abwicklung der Kommunikationsobjektverwaltung, da der Wert beim PDA erst durch die Bestätigung

des EIB-Telegramms verändert wird. Bis zur Aktualisierung des Wertes auf der Anzeige können daher bis zu 0,5 s vergehen.

Die Gesamtauslastung im Betrieb ist eher gering, da nur Statusänderungen der Objekte übertragen werden müssen, die aus Telegrammen mit einer Länge von weniger als 20 Byte bestehen.

5.2.4 Bluetooth

Ähnlich wie bei der DECT-Übergang wird bei Bluetooth ein Proxy verwendet, der die Kommunikation mit anderen Diensten stellvertretend für das eigentliche Gerät übernimmt. Beispielhaft für andere Geräte soll hier die Umsetzung der Einbindung der Bluetooth-Kamera vorgestellt werden.

Die Kommunikation mit der Kamera erfolgt durch das in [Arai2001] definierte *Basic Imaging Profile*. Dieses Profil basiert auf Basisprofilen der Bluetooth-Spezifikation, so das *Generic Object Exchange*, *Serial Port* und *Generic Access Profile*. Für die Übermittlung von Befehlen wird das OBEX-Protokoll verwendet, das auf dem IrOBEX-Protokoll der *Infrared Data Association* (IrDA) basiert. Die Kamera verwendet nur einige Befehle, die das Abfragen des integrierten Bewegungsmelders und die Aufnahme eines neuen Bildes veranlassen. Da die Kamera keine Möglichkeit zur Ereignismeldung bietet, wird sie durch den Proxy periodisch abgefragt. Der Proxy speichert die aktuellen Bilder auf einem nichtflüchtigen Medium und veranlasst die Umsetzung auf Dienstebene. Die Kommunikation erfolgt über das *Bluetooth Serial Port Profile*, es werden also innerhalb eines (embedded) PCs virtuelle Com Ports zur Verfügung gestellt, die als Kommunikationskanal zur Kamera verwendet werden. Die physikalische Schnittstelle ist ein USB-Bluetooth-Adpater, der mit entsprechenden Gerätetreibern die seriellen Schnittstellen (RFCOMM von Bluetooth) anbietet, die über die Comm API in Java angesprochen werden können.

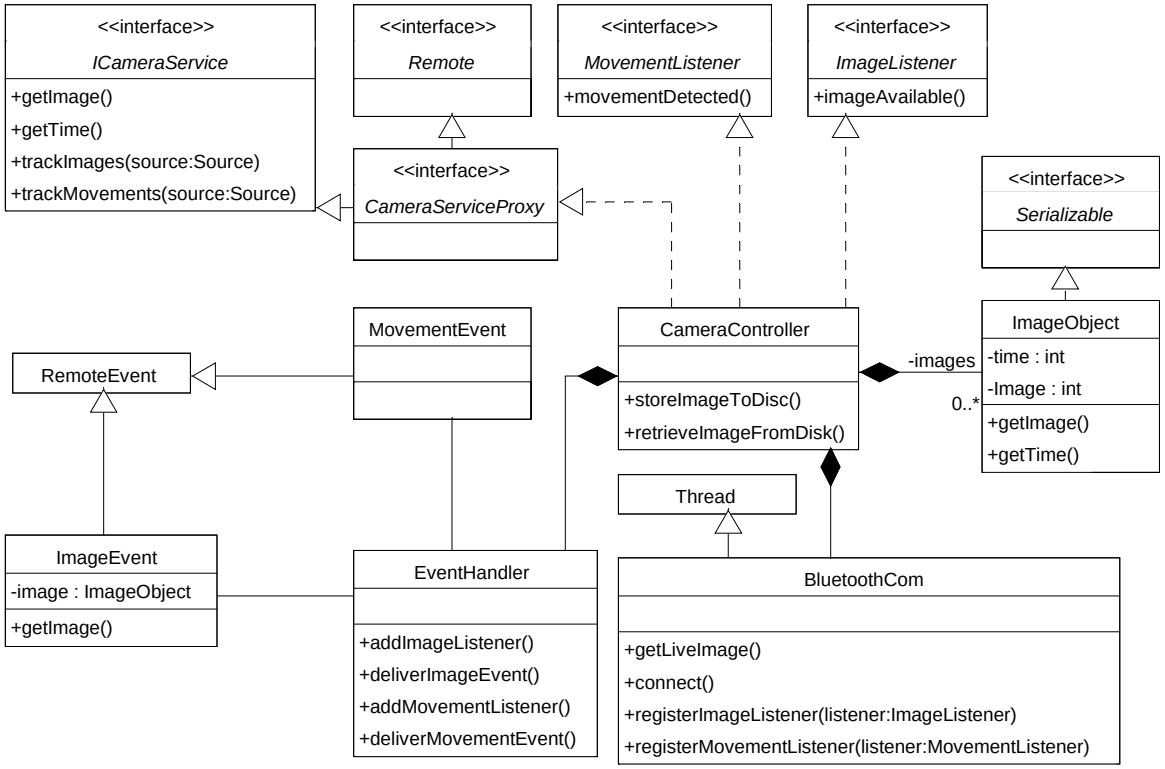

Abbildung 5.35: Struktur des Kamera-Proxys

Die Softwarestruktur des Proxys ist zweigeteilt: Ein Teil ist für die Kommunikation mit der Kamera zuständig, der andere beinhaltet die Dienstimplementierung. Innerhalb eines Threads wird in festgelegten Zeitabständen die Kamera abgefragt. Das OBEX-Protokoll ist verbindungsorientiert, so dass vor der Übermittlung eines Befehls ein Verbindungsaufbau stattfinden muss. Die zurückgelieferte ConnectionID[79] wird für die nachfolgenden Anfragen benötigt.

[79] Die Aufgabe der ConnectionID besteht in der Unterscheidung mehrerer Datenkanäle. Dadurch kann eine physikalische Verbindung mehrere Datenverbindungen tragen (Multiplexing).

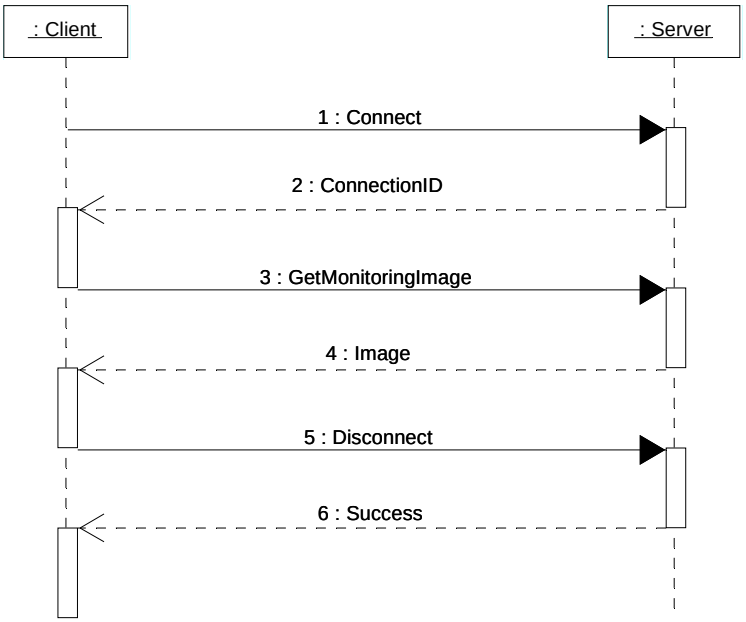

Abbildung 5.36: Nachrichtenabfolge zwischen OBEX-Client und -Server

Aus Abb. 5.36 lässt sich gut der Nachrichtenverlauf erkennen, der notwendig ist, um ein Bild von der Kamera anzufordern. Nach Erhalt der ConnectionID wird ein Befehl zur Abfrage des Bildes gesendet, dass als Antwort bereits Bildinformationen enthält. Die Antwort kann fragmentiert sein, erkennbar über das *Final Bit* im Header des OBEX-Protokolls. Der Client fragt den Server solange ab, bis dieser durch das Setzen des Final Bits das Ende der Anfrage anzeigt. Das verwendete Protokoll ähnelt sehr stark dem HTTP, was sich auch in den Antwort-Codes[80] widerspiegelt. Das in der Antwort enthaltene Bild wird als JPEG-Datei auf dem lokalen Dateisystem abgelegt und wird in einer Liste des *CameraControllers* eingetragen. Die restlichen Schnittstellen beziehen sich auf die Dienstimplementierung der Dienstschnittstelle (*ICameraService*) zur Anmeldung als *RemoteEventListener* (*MovementEvent*, *ImageEvent*) zur Abfrage von Bildern (*getImage*) und der aktuellen Zeit (*getTime*). Aus Platzgründen nicht explizit dargestellt ist die Erweiterung der Basisdienstschnittstelle durch *ICameraService* und die sich daraus ergebende Dienstschnittstellenbeschreibung durch eine XML-Datei.

5.2.5 EIB

Die Einbindung des EIB als Bussystem für die Steuerung und Überwachung der HKL-Technik wurde mit der in Kap. 5.2.2.4 beschriebenen Softwarestruktur erreicht. Die dort erstellte Daten- und Kommunikationsstruktur ermöglicht eine vielseitige Verwendung innerhalb des Gesamtsystems. Für die Einbettung in das Dienstgefüge wurden zwei unterschiedliche Ansätze untersucht: Eine Dienstumsetzung der Kommunikationsobjekt-verwaltung als Einheit und eine Umsetzung eines einzelnen EIB Gerätes in einen Dienst.

[80] Die Entsprechung des HTTP OK Headers mit dem Code 200 ist bei OBEX 0x20, HTTP BAD REQUEST 400 entspricht 0x40.

Es hat sich dabei gezeigt, dass die Kommunikationsobjektverwaltung als Einheit nur mit sehr detailliertem Systemwissen sinnvoll genutzt werden kann. Dennoch ist die Anpassung der Kommunikationsobjektverwaltung an Jini gut für Systemdiagnosen geeignet, da die Installationsorte der Verwaltungen nicht bekannt sein müssen. Durch einen entsprechenden Lookup sind alle Verwaltungen auffindbar und nutzbar.
Für die Anzeige und Nutzung einiger Geräte des Systems ist jedoch der Einsatz einer Geräteabbildung von Vorteil, da dadurch die Funktionalitäten der Geräte bereits wiedergegeben werden, während die Objektverwaltung nur Kommunikationsobjekte zur Verfügung stellt. Daher wurde aufbauend auf der EIB-Busanbindung ein Dienst für den EIB erstellt, der eine universelle graphische Steuerung und Überwachung von Teilen des Bussystems erlaubt. Dabei wird der Dienst in mehreren Stufen entwickelt:

- Aufbau eines bzw. mehrerer Dienste zur Abbildung der EIB-Geräte
  Eine einfache Abbildung der Kommunikationsobjekte des EIB auf Jini genügt hier nicht, da die Eigenschaften eines Gerätes erst durch das Zusammenspiel der Kommunikationsobjekte bestimmt sind. Daher muss für jede Gerätegruppe, die über gleiche Eigenschaften verfügt, ein eigener Dienst erstellt werden.
- Implementierung des Basisdienstes zur Anzeige
  In dieser Arbeit wurde der Basisdienst insofern implementiert, als dass eine Umsetzung der graphischen Anzeige geschaffen wurde. Die Anzeige verwendet die durch den Basisdienst definierten Schnittstellen.
- Initialisierung der Einzeldienste durch eine übergeordnete Instanz
  Die Dienstgruppen müssen bei der ersten Inbetriebnahme instanziiert und initialisiert werden. Diese Aufgabe übernimmt eine Verwaltungsinstanz.
- Umsetzung eines Dienstnutzers zur Anzeige der Dienste
  Es wurde ein Dienstnutzer erstellt, der eine Anzeigefläche zur Verfügung stellt, innerhalb der sich alle EIB-Geräte darstellen können. Der Dienstnutzer ist nicht vorkonfiguriert und die Anzeige kann zur Laufzeit zusammengestellt werden

Aus der Auslastungsanalyse des Ethernet hat sich gezeigt, dass eine Verwendung von primitiven Datentypen und Strings vorteilhaft ist, da diese Datentypen ein geringes Datenaufkommen nach sich ziehen. Beruhend auf diesem Ansatz wurden die Grunddienste der EIB-Geräte umgesetzt.

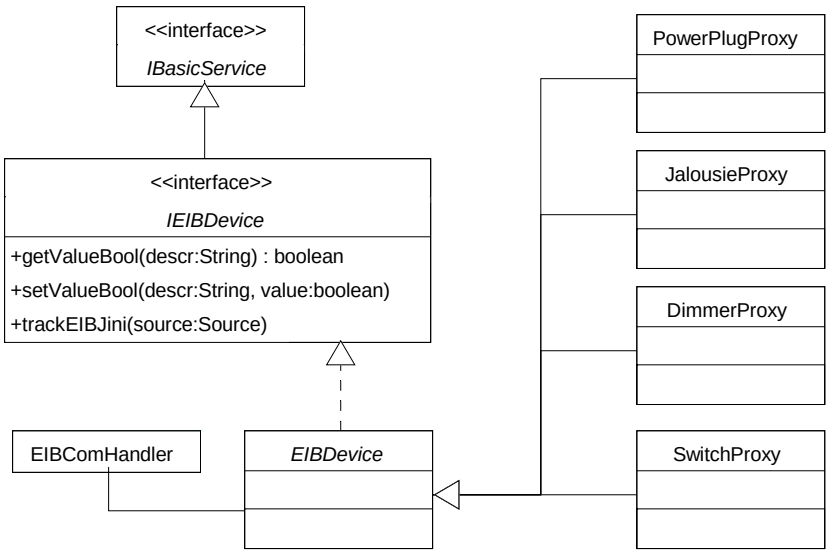

Abbildung 5.37: EIB Geräte als Jini Dienste

Als Basisklasse der Geräte dient die abstrakte[81] Klasse *EIBDevice*, die Dienstschnittstelle *IEIBDevice* implementiert. Dabei wurden nicht alle Methoden aufgezeigt, die in der Schnittstelle *IBasicService* enthalten sind. Für sechs der acht in Java vorhandenen primitiven Datentypen wurden Operationen zum Setzen und Abfragen der Werte vorgesehen. Besitzt ein Gerät mehrere Objekte gleichen Datentyps, können diese durch eine Kennung (*descr)* unterschieden werden. Diese Art der Implementierung bietet den Vorteil, dass alle Geräte über die gleiche Schnittstelle verfügen. Durch diese Implementierung müssen auch bei Erweiterungen – beispielsweise beim Hinzufügen neuer Geräte – keine neuen Schnittstellen eingeführt werden. Die Objekte innerhalb des Gerätes sind nur mit entsprechenden Kennzeichnungen auszustatten, die innerhalb des Systems eindeutig sein dürfen, jedoch nicht müssen – sofern sie die gleiche Funktion besitzen. Die Funktion *trackEIBJini* dient der Anmeldung zur Benachrichtigung über Objektänderungen innerhalb der Kommunikationsobjektverwaltung (*EIBComHandler*).
Die meisten Geräte aus Abb. 5.37 sind Standardkomponenten des EIB. Der *PowerPlugProxy* ist die Umsetzung der am Lehrstuhl entwickelten intelligenten Steckdose[82], die über diesen Dienst gesteuert und abgefragt werden kann.

[81] Von einer abstrakten Klasse können keine Instanzen erzeugt werden. Sie ist nur ein formelles Konstrukt, um ein gemeinsames Grundverhalten an abgeleitete Klassen weiterzugeben.
[82] Vgl. [Müller2001]

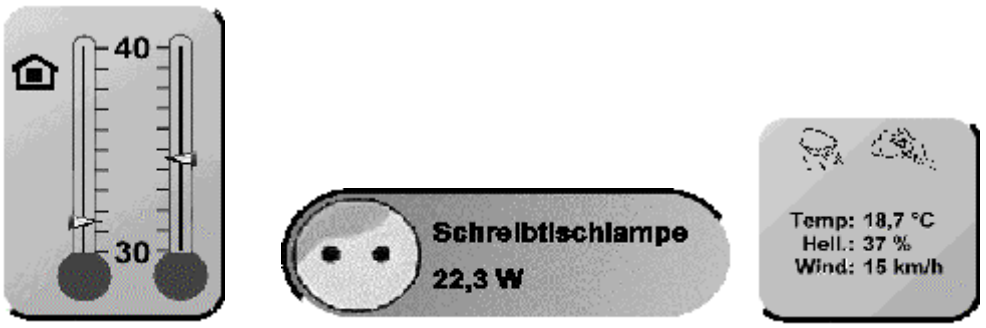

Abbildung 5.38: Darstellungen einiger EIB-Dienste

Da jedes Gerät über die *IBasicService*-Schnittstelle verfügt, gibt es die Möglichkeit, über die Methode *getAWTUI* eine Referenz auf ein Objekt zur Erstellung eines anzeigefähigen Elements zu erhalten. Dabei ist für den Dienstnehmer kein Wissen über die Kommunikation mit dem EIB-Gerät notwendig. Sie wird bereits von dem zurückgelieferten *Panel* übernommen.

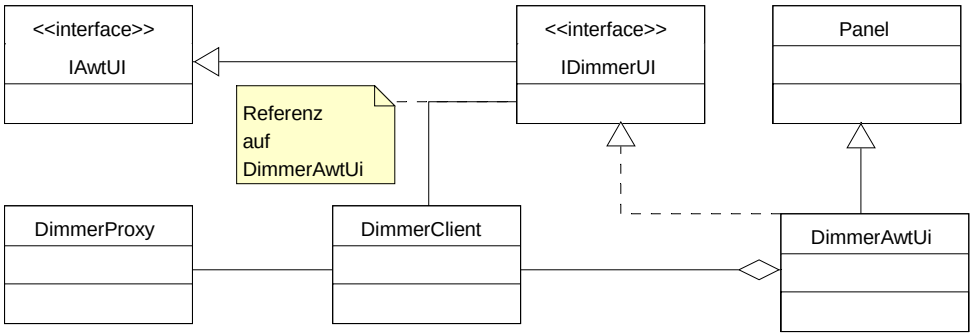

Abbildung 5.39: Umsetzung des Basisdienstes zur graphischen Anzeige

Der *DimmerClient* übernimmt den Austausch der Leases zur Aufrechterhaltung der Dienstbeziehung und die Entgegennahme der Remote Events. Innerhalb der Remote Events werden neue Werte des EIB-Gerätes übermittelt, damit eine ständige Abfrage vermieden wird. Die Schnittstelle *IDimmerUI* entkoppelt die Darstellung von der Kommunikation. Dadurch lassen sich unterschiedliche Darstellungen mit der gleichen *DimmerClient* Klasse umsetzen. Alle notwendigen Zusatzklassen und graphischen Elemente werden von *DimmerAwtUi* nachgeladen.
Durch diese Struktur lassen sich sehr leicht einzelne Geräte darstellen. Der Dienstnehmer muss nur ein graphisches Element vom Typ *Panel* aufnehmen können, das die Basis einer graphischen Komponentensammlung in Java ist.
Die Generierung der EIB-Proxys erfolgt durch eine zentrale Klasse, die aus einer Konfigurationsdatei alle Parameter für die Kopplung mit dem EIB sowie die graphischen Grundinformationen liest. Diese Klasse ist verantwortlich für die Erstellung einer Kommunikationsobjektverwaltung, deren Instanz von allen EIB-Proxys geteilt wird; sie melden sich nur für bestimmte Objekte innerhalb der Verwaltung an. Die EIB-Proxys übernehmen selbständig die Anmeldung bei den gefundenen Lookup Servern.

Nach der Anmeldung sind die Dienste im gesamten System zugänglich und können sehr einfach zur Anzeige und Steuerung eingesetzt werden.

5.3 Service-Agenten

Die verfügbaren Dienste des Systems können durch feste Implementierung unter Verwendung der bekannten Dienstschnittstellen miteinander verknüpft werden. Das Vorgehen dazu wurde bereits in Kap. 4.2.3 besprochen. Bei Einbringung eines neuen Dienstes müssen jedoch immer dazu passende Dienstnutzer existieren. Da in Jini die Dienstschnittstellen nicht standardisiert sind, ist im Allgemeinen jedoch davon auszugehen, dass das nur der Fall ist, wenn die Dienste vom gleichen Hersteller angeboten werden. Selbst wenn die Dienstschnittstellen bekannt sind (so wie bei UPnP) wird nur die Möglichkeit gegeben, einen kompatiblen Dienst aufzufinden, dennoch muss der Anwender noch immer eine Auswahl des für ihn in Frage kommenden Dienstes treffen. Daher kann die Middleware nur im Auffinden der Dienste helfen und unter Umständen die Nutzbarkeit durch Festlegung der Schnittstellen garantieren, damit ist noch keine Unterstützung des Anwenders gegeben.
In dieser Arbeit wird der Einsatz von Agenten für die Erstellung von Dienstbeziehungen erörtert. Dabei werden die Mechanismen zur Dienstnutzung, die bereits durch die Middleware gegeben sind, insofern erweitert, als dass auch die Verknüpfung von Dienstnehmern und Dienstanbietern automatisiert wird. Der durch die Dienste bestehende Pool aus Ressourcen, wird durch die Erstellung einer weiteren Komponente, den Software-Agenten, parametriert. Eine derart universelle Parametrierung durchbricht unter Umständen den schnittstellenorientierten Ansatz der Middleware. Wird der Agent nur als Vermittler eingesetzt, um einen Dienstnehmer beim Dienstanbieter anzumelden, bleibt die Schnittstellenkonsistenz erhalten. Verarbeitet der Agent hingegen Daten des Dienstanbieters und gibt sie erst dann an den Dienstnutzer weiter, wird eine prozedurale Vorgehensweise angewendet, wie sie bei funktionsorientierten Sprachen der Fall ist.
Der Agent ist – wie in Kap. 4.2.4 gezeigt – entweder im System bereits vorhanden, oder er wird vom Anwender in das System eingebracht.
Für das Projekt *tele*-Haus wurden mehrere Szenarien aufgestellt, die das Zusammenspiel der Systemkomponenten und –dienste beschreiben. Der Einsatz des Service-Agenten soll hier anhand eines dieser Szenarien näher erläutert werden.

5.3.1 Szenario einer Dienstverknüpfung

Der PDA eines Benutzers ist mit der in Kap. 5.2.1.3 vorgestellten Hardware zur Erfassung des Aufenthaltsortes ausgestattet. Betritt er damit einen Raum, werden auf dem PDA nur noch die Geräte dargestellt, die sich in diesem Raum befinden.
An diesem Vorgang sind zwei Dienste beteiligt: der Lokalisierungsdienst und die Dienstimplementierung eines oder mehrerer EIB-Geräte. Diese beiden Dienste müssen durch den Einsatz des Service-Agenten miteinander verknüpft werden.

5.3.2 Agenten

Die Funktionsweise des Service-Agenten beruht auf einem Konzept aus Java, das dort unter dem Begriff *Reflection* zusammengefasst wird. Die Struktur eines Objekts in Java wird selbst durch ein Objekt beschrieben, das eine Instanz der Klasse *Class* ist. *Class* stellt Funktionen zum Abfragen der implementierten Schnittstellen, Elternklassen sowie Methoden zur Verfügung. Die Methoden sind wiederum Instanzen der Klasse *Method*, die die Rückgabe- und Übergabeparameter beschreibt.

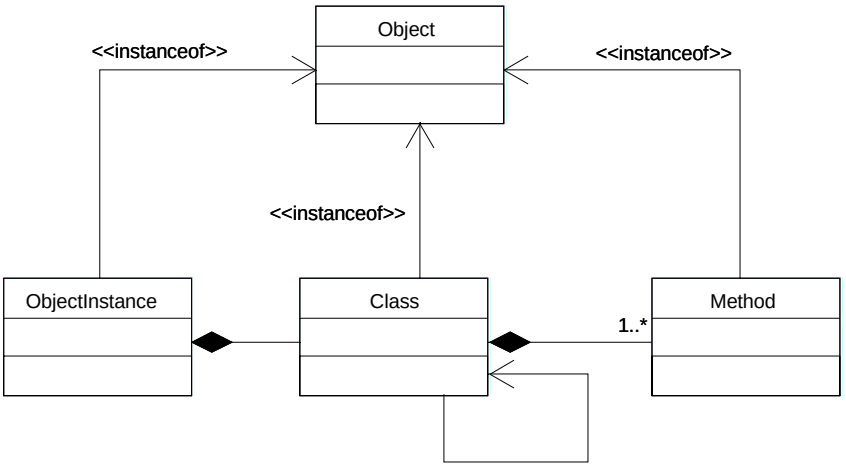

Abbildung 5.40: *Reflection* in Java

Mit diesem Konzept können Klassen geladen werden, Instanzen erzeugt und sogar Methoden aufgerufen werden, deren Schnittstellen zuvor nicht bekannt sind und erst zur Laufzeit des Programms erfragt werden können.
Der Service-Agent erstellt die Dienstverknüpfungen, indem er mit einer vorgegebenen Konfiguration eine bestimmte Befehlsreihenfolge auf die Dienstanbieter und Dienstnehmer anwendet. Diese Befehlsreihenfolge wird in der vorhandenen Version aus einer Datei entnommen. Er kennt dabei nicht die beteiligten Dienstschnittstellen, und zwar in dem Sinne, dass ihm keine Klassendateien vorliegen. Damit lassen sich auch neue Dienste durch den Service-Agenten parametrieren. Dabei wird vorausgesetzt, dass alle Dienstnehmer gleichzeitig Dienstanbieter im Sinne von Jini sind. Nur so können sie im Netzwerk gefunden werden.
Durch die Verteiltheit der Dienste ergeben sich besondere Anforderungen bei der Verwendung des Service-Agenten. Um die Operationen ausführen zu können, muss er die Beschreibungsdateien (Class files) der Dienste und der verwendeten Objekte besitzen. Da sich der Service-Agent weder am Ort des Dienstnehmers noch am Ort des Dienstanbieters befindet[83], muss diese Datei nachträglich geladen werden. Die Dienstauffindung erfolgt über Jini, so dass der Agent bereits die serialisierten Stubs der Fernobjekte besitzt. Die zugehörige Klassenbeschreibung, die er benötigt um ein Objekt aus dem Datenstrom zu

[83] Der Ort ist hier gleichzusetzen mit gleicher Netzwerkadresse. Durch die Trennung sind die Methoden zum lokalen Laden von Klassendateien nicht mehr wirksam.

erzeugen, wird über den *RMIClassLoader* nachgeladen. Der *RMIClassLoader* ist eine besondere Ausführung eines Java ClassLoaders, der nicht die lokalen Beschreibungsdatei liest[84], sondern mit Hilfe von Referenzen (URL), die er über das RMI Wire Protocol mitgeteilt bekommt (vgl. Kap. 5.2.1.5 Nachladen von Stubs), die notwendige Beschreibungsdatei nachzuladen. Beim Aufruf von Methoden mittels *Reflection* findet eine Typenprüfung statt. An dieser Stelle ist zu berücksichtigen, dass zwei Klassen identisch sind, wenn die Klassenbezeichnung[85] und der ClassLoader übereinstimmen. Für die Parameterübergabe eines Aufrufs müssen alle Parameter vom gleichen Typ sein.

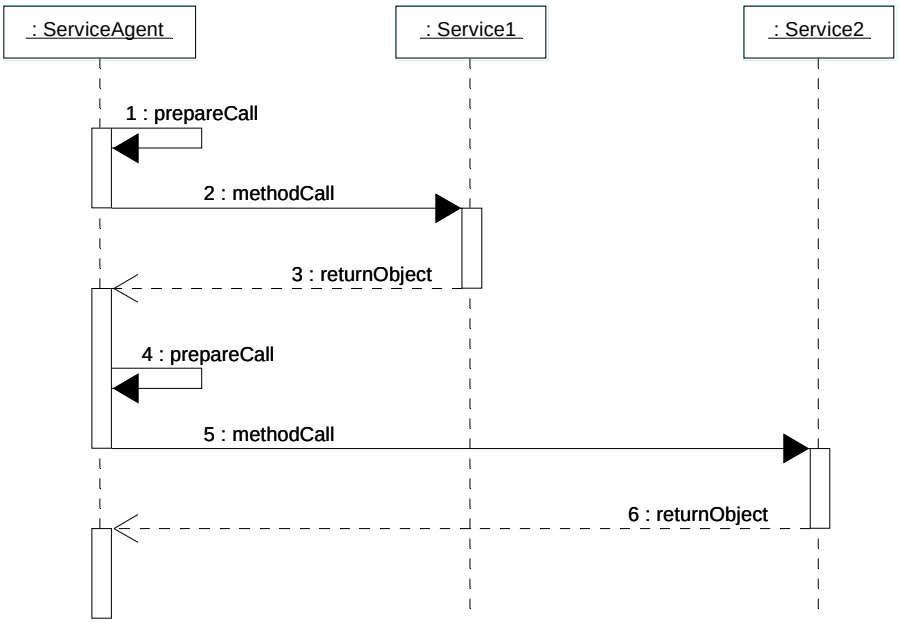

Abbildung 5.41: Verknüpfung von Dienstanbieter und Dienstnehmer durch Service-Agenten

In Abb. 5.41 ist der Ablauf der Verknüpfung dargestellt. Bevor ein Methodenaufruf stattfinden kann, werden die Dienstreferenzen über den Jini Lookup Service ermittelt. Nach Erhalt der Dienstreferenzen lädt der zugehörige *ClassLoader* bereits die Klassenbeschreibungen. Selbst wenn die Klassendateien identisch sind, reicht eine unterschiedliche URL aus, um zwei verschiedene *ClassLoader* der gleichen Klasse zuzuordnen. Um das zu umgehen gibt es zwei Möglichkeiten: beide Dienste verwenden die gleiche URL für die RMI-Codebase, oder aber der Agent sorgt dafür, dass die Klassendateien bereits vor dem Aufruf eines Dienstes geladen werden. Dies kann durch explizites Anweisen zum Laden der Klasseninformationen erreicht werden. Ist die Klasseninformation bereits vorhanden, wird der *ClassLoader* nicht versuchen, diese über die vom Dienst-Stub gelieferte URL nachzuladen. Mit Hilfe der vorhandenen Beschreibungsdateien kann der Service-Agent die benötigten Klassen bereits laden. Es können als Parameter auch Remote Stubs übergeben

[84] Durch eine falsche Parametrierung kann erreicht werden, dass der *RMIClassLoader* nicht vewendet wird. Die JVM hat eine festgelegte Suchreihenfolge, mit der sie die Verfügbarkeit der Klassendateien prüft.

[85] Es handelt sich um den „fully qualified name“ [Halloway2001], der aus dem Klassennamen und dem Package-Namen besteht.

werden, so dass die beiden Dienste auch direkt miteinander verknüpft werden können. Der Service-Agent ist zusätzlich in der Lage, Ergebnisse aus Operationen eines Dienstes zwischenzuspeichern, um diese dann bei Bedarf weitergeben zu können.
Bei der Verwendung des Agenten auf das vorgestellte Szenario wird die Schnittstellenkonsistenz beibehalten. Der Dienstanbieter ist hier der Lokalisierungsdienst, der beim Erkennen einer Person in einem Raum ein Ereignis generieren soll, dass dann an den Anzeigedienst weitergeleitet wird, ohne dass der Service-Agent als Vermittler auftritt. Dazu ruft der Service-Agent zunächst den Lokalisierungsdienst mit dem Remote Stub des Anzeigedienstes auf (*trackPresence*). Beim Eintreten eines Ereignisses wird dann direkt der Anzeigedienst aufgerufen (*addDisplayFilter*), so dass nur alle Dienste mit dem zu dem Raum passendem Diensteintrag[86] angezeigt werden.
Noch nutzt der Agent nicht alle Möglichkeiten, die durch die Dienstbeschreibungen gegeben sind. Mit diesen Beschreibungen kann ein Agent entwickelt werden, der auf Basis der Dienstinformationen selbständig Verknüpfungen vornimmt oder vorschlägt. Dazu dienen auch die Angaben über die Datenflussrichtung innerhalb der XML-Datei (consume/produce). Zur Zeit ist der Agent noch auf das System festgelegt, er könnte jedoch auch, wie in Kap. 4.2.3 vorgeschlagen, Teil eines nicht zum System gehörenden Gerätes sein. Sehr hilfreich wäre diese Art Agent auch zur Systemdiagnose, der als spezialisierte Aufgabe Fehlermeldungen aufsucht und analysiert.

5.4 Externer Zugang

5.4.1 OSGi

OSGi stellt in dieser Arbeit ein Bindeglied zur Konfiguration des Systems dar. Wie alle implementierten Dienste, basiert auch diese Umsetzung auf Java.
Das Framework von OSGi ermöglicht die Installation und Verwaltung vieler unterschiedlicher Funktionen auf einer Plattform. In dieser Arbeit ist es auf einem Zugangsknoten installiert, der von Außen über HTTP erreichbar ist. Damit können die installierten Bundles auf dem Zugangsknoten mit einem einfachen Internetbrowser verwaltet werden.
Alle Bundles stellen Java Klassensammlungen dar, die in einer Jar Datei abgelegt werden. Die Bundles können dann innerhalb des Frameworks installiert werden und Schnittstellen (Dienstschnittstellen) exportieren oder andere Dienste des Systems verwenden. Zur Grundausstattung des *Java Embedded Servers 2.0* (JES) von Sun Microsystems, einer Referenzimplementierung des OSGi, zählen unter anderem ein HTTP-Dienst, der die Anmeldung von *Servlets* erlaubt, zwei Authentifizierungsdienste (Basic and Digest Authentication nach [IETFRFC2617] und ein Aufzeichnungsdienst. Servlets werden zur dynamischen Generierung von HTTP-Antworten mit beliebigen Inhalten verwendet. Auch

[86] Der Diensteintrag - *ServiceEntry* - wird bei der Anmeldung an den Lookup Server übermittelt und kann als Suchkriterium verwendet werden. Er wird hier als Anzeigefilter eingesetzt, so dass nur die Dienste mit den entsprechenden *ServiceEntries* angezeigt werden.

sie sind in Java definiert und bieten Schnittstellen zur Annahme einer HTTP-Request, deren Auswertung und Beantwortung aller von HTTP unterstützten Response Formate.
Die Verwaltung der Bundles ist sehr einfach über ein bereits integriertes Web Interface möglich, das alle Funktionen zum Installieren, Deinstallieren, Starten und Stoppen der Dienste anbietet. Daneben können Informationen über den Zustand des Bundles, der genutzten Dienste, Versions- und Herstellerangaben abgefragt werden. In dieser Arbeit sollen die Einsatzmöglichkeiten von OSGi anhand von zwei Anwendungen aufgezeigt werden.
Beide Bundles nutzen den HTTP-Service und melden sich als Servlet an. Das erste Bundle ist in der Lage, alle zur Zeit aktiven Dienste des Jini Netzwerks aufzulisten. Dabei werden die Dienste mit ihrer Bezeichnung und den Standardeinträgen (ServiceEntries) angezeigt. Dazu muss lediglich eine HTTP-Anfrage an das Gateway gerichtet werden. Durch die Basic Authentication nach [IETFRFC2617] wird der Benutzer nach einem Passwort und Benutzernamen gefragt. Kann sich der Anwender ausweisen, wird als Antwort die Liste der Jini Dienste zurückgegeben. Diese Liste dient nur als Anhaltspunkt über den Zustand des Systems. Für eine detaillierte Analyse muss noch ein Bundle entwickelt werden, das den Zugriff auf die Dienste erlaubt. Mit dem Service-Agenten kann durch die Verwendung einfacher Anweisungslisten dieses Ziel bereits erreicht werden. Der Benutzer ruft dazu eine HTML-Seite mit dem Jini Dienstzugangs-Bundle auf, das einfach den Service-Agenten verwendet, um vorkonfigurierte Befehlsfolgen zur Abfrage von Diensten ausführt. Auch die nachträgliche Änderung der Dienstverknüpfungen ist durch einen Austausch der Agentenverknüpfungsregeln leicht integrierbar.
Das zweite Bundle dient als Zugangspunkt zur Fernabfrage, die dann jedoch nicht über das Framework ausgeführt wird. Unter einer URL ist damit ein HTML-Dokument erreichbar, in das ein Java Applet eingebunden ist.
Durch die dynamische Verwaltung der Bundles können zusätzliche Pakete ohne lokalen Eingriff installiert werden. Damit ist das System erweiterbar und die vorhandenen Dienste des Systems können auch extern genutzt werden.

5.4.2 RMI über TCP/IP

Das im vorherigen Kapitel erwähnte Bundle zur Fernabfrage verwendet ein Java Applet zur Kommunikation. Mit diesem Applet werden die Funktionen des EIB gesteuert. Dazu wird auf dem Zugangsknoten eine Server-Software in Java installiert, die alle Anfragen des Applets entgegennimmt und an den EIB weiterleitet. Andererseits werden die Zustandsänderungen des EIB zurück an den Client übergeben.
Die Interaktion mit dem EIB wird mit den bereits vorgestellten Datenzugriffsmodellen erreicht. Die Verbindung zwischen Applikation und EIB Datenmodell kann dabei durch die Verwendung der lokalen Schnittstelle mittels RMI-Aufrufe erfolgen. Die Server-Software übernimmt damit einerseits die Kommunikation mit dem EIB, andererseits ist sie für alle Verbindungen zu dem Applet zuständig. Die Client/Server Kommunikation wird ebenfalls über RMI abgewickelt, damit sind alle Methodenaufrufe für die Anwendung des Clients transparent. Durch die Netzwerkstruktur ergeben sich jedoch einige

Besonderheiten. Die meisten Netzwerke sind durch Firewalls vor unbefugten Zugriffen geschützt. Dies wird durch die Begrenzung auf wenige TCP/UDP-IP-Ports und durch die Richtungsabhängigkeit einer Verbindung erreicht. Auch im *tele*-Haus wird das interne Netz durch eine Firewall vom Internet abgetrennt. Da jedoch die Konfiguration der Serverseite an die Bedürfnisse für den Fernzugriff angepasst werden kann, stellt dies kein besonderes Problem dar. Erst die Firewall auf der Clientseite erfordert zusätzliche Mechanismen, die eine freie Kommunikation mit dem Server erlauben.

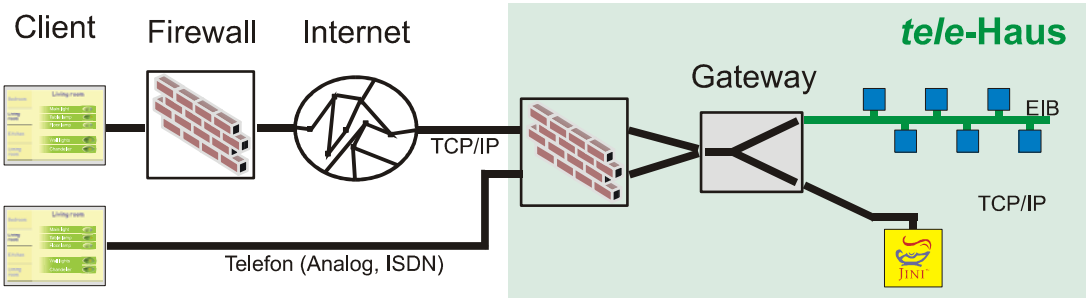

Abbildung 5.42: Konfigurationen von Client/Server Verbindungen

Die oben dargestellten Anordnungen berücksichtigen zwei der häufigsten Fälle beim Zugriff auf Daten des Systems. Entweder ist der Client an das Internet angeschlossen oder es wird eine Einwahlverbindung aufgebaut, über die dann Daten ausgetauscht werden. Der Unterschied ergibt sich in der Anzahl der zu überwindenden Firewalls. Bei der Einwahl entfällt die clientseitige Firewall. In diesem Fall können beidseitig Verbindungen auf- und abgebaut werden.

In der Regel sind Firewalls so konfiguriert, dass nur ausgehende Verbindungen möglich sind. Diese können zusätzlich noch auf bestimmte Ports beschränkt sein (beispielsweise nur HTTP auf Port 80/8080). Eingehende Anfragen werden hingegen blockiert, sofern sie nicht an Zugangsports für Standarddienste (E-Mail, SMTP) gerichtet sind. Für den Fernzugriff wurden diese Beschränkung serverseitig aufgeweicht, so dass zwei zusätzliche Ports für RMI-Aufrufe im System zur Verfügung stehen. Ergeben sich Änderungen am EIB, so müssen diese jedoch an den Client weitergegeben werden. Der Verbindungsaufbau kann bei einer clientseitigen Firewall aus den oben genannten Gründen nicht vom Server aus initiiert werden. Eine Umkonfiguration der Einstellungen ist nicht möglich, da diese durch den Internet Provider vorgegeben werden.

Eine Möglichkeit, diese Beschränkung zu umgehen, ist ein einmaliger Aufbau einer TCP/IP- Verbindung, die dann offengehalten wird und so für den Transport von Daten des Servers an den Client dient. Dazu muss der Client den Aufbau von sich aus initiieren. Der Server lässt die Verbindung dann bestehen. Da die Standardimplementierung der Transportschicht für RMI-Aufrufe jedoch die TCP Verbindungen nach einer gewissen Zeit wieder schließt, muss auf eine eigene Transportschicht zurückgegriffen werden, die das obige Verhalten umsetzt. Eine andere Möglichkeit ist die periodische Abfrage des Servers,

also keine ereignisgesteuerte Aktualisierung. Der Nachteil liegt in den zusätzlichen Verzögerungen durch den Auf- und Abbau der Verbindung[87].
Das Applet selbst verwendet die Daten der EIB-Kommunikationsobjektverwaltung und stellt diese dann innerhalb des Browsers dar. Alle Informationen, die zur Darstellung notwendig sind, werden in einer Jar-Datei abgelegt, so dass nach dem Laden der Datei das Applet gestartet werden kann, ohne weitere Dateien nachladen zu müssen. Der Anzeigebereich unterstützt aktive und passive graphische Elemente. Alle aktiven Elemente sind einem oder mehreren Kommunikationsobjekten zugeordnet. So besitzt ein Dimmer vier Kommunikationsobjekte, die zur Steuerung und Anzeige verwendet werden. Zu den nicht steuerbaren Elementen gehören Text, Rechtecke, Kreise und Linien, die innerhalb des Anzeigebereichs frei platziert werden können.

5.5 Darstellung

5.5.1 HAVi am Beispiel FAV

Die Darstellung ist bei HAVi ein zentrales Element – nur über die Anzeige lassen sich die Komponenten bei Multimediageräten steuern. Daher soll hier kurz die Umsetzung der Darstellung des HAVi-EIB Übergangs gezeigt werden.
Das DDI-Protokoll, das zur Kommunikation zwischen BAV und FAV eingesetzt wird, ist im Wesentlichen auf folgende graphische Elemente beschränkt:

- Text element
  Dadurch wird ein einfacher Text angezeigt. Es hat keine Eingabefunktion und ist auch nicht durch den Benutzer veränderbar.
- Button element
  Innerhalb des Buttons können Graphiken und Text dargestellt werden. Er kann auch interaktiv eingesetzt werden, um DDI Ereignisse auszulösen.
- Choice element
  Dieses Element hat zwei Ausführungsformen. Die Spezifikation lässt eine Liste zu, in der nur eine Auswahl getroffen werden kann oder eine Liste mit mehreren gewählten Einträgen.
- Show range
  Ein einfacher Schieberegler, der einen Wert darstellt.
- Set Range
  Ein Schieberegler zum Setzen eines Wertes. Neben dem Choice element ist dies das einzige Element, das durch einen Benutzer direkt beeinflusst werden kann.

Die Spezifikation weist eindeutig daraufhin, dass die Anzeige dieser Elemente ausschließlich vom DDI Controller vorgenommen wird, der in der Art der Darstellung frei

[87] Ein Fortschritt bei HTTP/1.1 ist die Wiederverwendung von bestehenden TCP-Verbindungen für die Übermittlung neuer Anfragen. Es hat sich gezeigt, dass der zusätzliche TCP Protokollaufwand bei einem ständigen Auf- und Abbau größer ist, als beim Halten der Verbindung.

ist. Er kann entscheiden, ob er ein Show range element beispielsweise als linearen Schieberegler oder aber in Zeigerdarstellung, vergleichbar mit einem Zeigermess-instrument darstellt.
Alle hier umgesetzten Elemente wurde aus den in Java verfügbaren AWT Klassen oder deren Kombination gebildet.

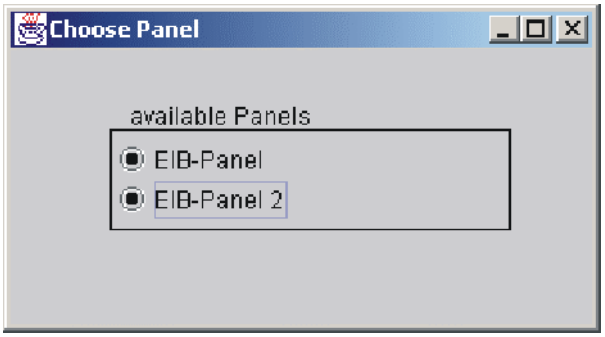

Abbildung 5.43: HAVi DDI Elemente des FAV

In der obigen Abbildung sind zwei *non organizational* und zwei *organizational elements* zu sehen. Das DDI-Panel und das Group Panel sind *organizational elements*. Sie dienen der Gruppierung von Objekten. Die anderen beiden Elemente sind zwei Choice elements, die zur Auswahl eines weiteren Panels dienen.

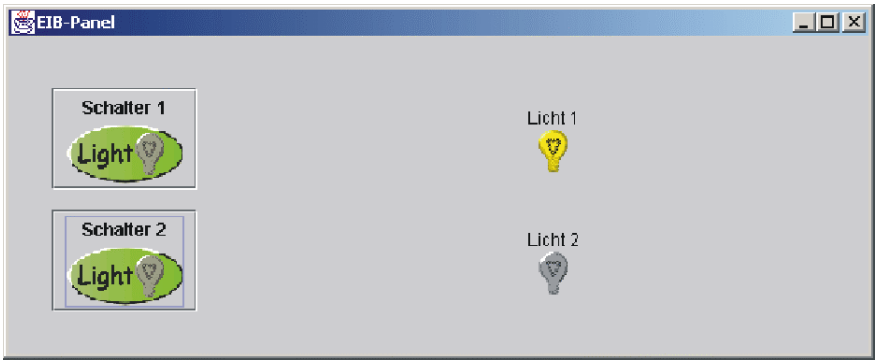

Abbildung 5.44: Steuerung von EIB Geräten über DDI Buttons

Hier wurden mehrere Elemente verwendet, darunter ein Button und ein Text element. Der Button ändert seinen Zustand erst durch die Rückmeldung des BAV. Bei Änderung des Zustands eines EIB-Geräts werden die dazugehörigen Buttons und Graphiken aktualisiert.

5.5.2 Applet mit Web Pad

Obwohl die Software für den Fernzugriff mit einer leicht abgewandelten RMI-Transportschicht arbeitet, sind die Komponenten zur Anzeige und Steuerung identisch. Durch die gewählte Klassen- und Objektstruktur der Software sind nur wenige Modifikationen notwendig gewesen, um eine Steuerung des EIB über das Web Pad zu ermöglichen.

Die einzige Voraussetzung des Web Pads zur Ausführung der Software ist ein Internetbrowser mit installierter JVM. Durch die Verwendung einer Wireless LAN Karte zur Kopplung an das hausinterne Netzwerk konnte das Web Pad auf alle integrierten Kommunikationsobjektverwaltungen zugreifen. Die graphische Oberfläche wird nicht, wie beim Jini Client, automatisch generiert sondern mit Hilfe eines Konfigurationsprogramms eingerichtet.

Abbildung 5.45: Abbildung der Anzeige auf einem Web Pad

Genau wie bei der Software für den Fernzugriff wird auch hier die Eingabe von Befehlen durch aktive Elemente ermöglicht. Zur Eingabe von Befehlen wird ein Touch Panel verwendet, es sind also neben dem Web Pad keine weiteren Zeigeinstrumente (Maus o.ä.) notwendig.

5.6 Sicherheitsaspekte

Die Verknüpfung mehrerer Systeme stellt immer einen Kompromiss aus einer Vielzahl von sich teilweise widersprechenden Anforderungen dar. Gegen die systemweite Verfügbarkeit einzelner Teilnehmer spricht zum einen das zusätzliche Datenaufkommen auf den Backbone-Bussen, zum anderen stellt dies ein potenzielles Sicherheitsrisiko dar, da die Überwindung der Sicherheitsvorkehrungen eines Bussystems das gesamte System gefährdet. Um einen Zugriff auf systemsensible Daten durch Unbefugte zu verhindern, müssen auf das jeweilige Bussystem angepasste Sicherheitsvorkehrungen getroffen werden.

5.6.1 RMI und Jini

RMI verwendet in der derzeitigen Implementierung von Sun Microsystems TCP/IP als Transportprotokoll. TCP/IP dient nur der Sicherung einer Transportverbindung, beherbergt jedoch keinerlei Prozeduren zur Verhinderung eines unberechtigten Datenzugriffs; dafür sind die Anwendungen verantwortlich.

Durch die Zunahme der Verwendung von TCP/IP wurde die Verschlüsselung von Daten immer vorrangiger. Besonders in firmeninternen Netzwerken oder bei der Übertragung von Daten über das Internet. Speziell für TCP/IP gibt es mehrere Verfahren, von denen sich jedoch zwei durchzusetzen scheinen: IPSEC und *Secure Sockets Layer* (SSL)[88]. IPSEC wird überwiegend zur Tunnelung von Daten über das Internet verwendet. Damit ist es möglich zwei abgeschlossene und gesicherte Netzwerke über das Internet zusammenzuschalten. SSL hingegen gehört zur Standardausstattung aller gängigen Internetbrowser und wird mitunter auch für Zahlungstransaktionen verwendet. SSL unterstützt für die Verschlüsselung mehrere Verschlüsselungsalgorithmen, die beim Beginn einer Sitzung ausgehandelt werden. Nach Austausch der Public Keys eines asymmetrischen Verschlüsselungsverfahren werden die eigentlichen symmetrischen Sitzungsschlüssel ausgetauscht, die dann für das vereinbarte Protokoll verwendet werden. SSL ist unter den folgenden Bedingungen als sicher anzusehen, wenn

- die verwendeten Zertifikate, die benötigt werden um den öffentlichen Schlüssel des asymmetrischen Verschlüsselungsverfahrens zu authentifizieren, noch gültig sind, das heißt die privaten Schlüssel des Zertifikats nicht kompromittiert worden sind.
- eine ausreichende Schlüssellänge verwendet wird. Diese sollte je nach Algorithmus mindestens bei 128 Bit liegen (IDEA, RC4). Bei bestimmten Angriffsmethoden wird die Kommunikation gestört, um zu erreichen, dass die Kommunikationspartner eine Verschlüsselung niedrigerer Stärke verwenden.
- Zertifikate sollten nur über eine ungesicherte Verbindung angenommen werden, wenn die Kommunikation nicht durch Angreifer gefährdet ist. Gefälschte Zertifikate erlauben die Entschlüsselung der gesamten Kommunikationsbeziehung zwischen Client und Server (Session).

Werden diese Grundsätze berücksichtigt sind die meisten klassischen Angriffsmethoden wirkungslos. Eine Brute-Force Attack ist bei ausreichend langem Schlüssel mit den heutigen Mitteln nicht in einer vertretbaren Zeit durchführbar[89].
Um RMI sichern zu können, müssen alle Kommunikationspartner die Transportschicht auf SSL umstellen. Dazu müssen den jeweiligen Diensten die Zertifikate mitgeliefert werden, da sonst keine gültige SSL-Session aufgebaut werden kann. Die Zertifikate müssen am Dienstort lokal gespeichert werden und vor einem unbefugten Austausch gesichert werden. Die Anforderung von Zertifikaten über das lokale Netz dürfte in der Regel unbedenklich sein, sofern es gegen äußere Zugriffe geschützt ist. Dennoch ist es sicherer die Zertifikate direkt am Dienstort zu installieren.
Werden die SSL-Sockets so konfiguriert, dass nur bestimmte Protokolle erlaubt sind (beispielsweise 3DES mit 168 Bit, DES oder aber IDEA mit 128 Bit), so ist die Dienstnutzung innerhalb des Systems gesichert. Für die Zertifikatverwaltung sind zwei

[88] Siehe [Freier1996].

[89] Übliche Angriffe nutzen Schwächen von der Verschlüsselungsalgorithmen aus. So sind bei bestimmten Algorithmen nicht alle Schlüssel gleich sicher. Meistens verhelfen jedoch fehlerhafte Implementierungen der Protokolle zu Sicherheitslücken.

Möglichkeiten denkbar: Jeder Dienstanbieter erhält das Zertifikat und den zugehörigen privaten Schlüssel – er ist als Server anzusehen – oder es wird ein Root Zertifikat ausgestellt, mit dem dann Unterzertifikate an die Dienstanbieter ausgestellt werden. Dann muss nur das Unterzertifikat (mit zugehörigem privaten Schlüssel) bei dem Dienstanbieter installiert werden. In diesem Fall reicht für einen Client die Kenntnis des Root Zertifikats, um die Echtheit des Unterzertifikats zu prüfen.
SSL schützt nur die TCP-Verbindungen, jedoch keine UDP-Verbindungen, die zur Bekanntmachung neuer Lookup Server verwendet werden. Dies stellt jedoch kein Problem dar. Über die UDP-Nachrichten geben die Lookup Server ihre TCP-Verbindungsparameter bekannt (IP-Adresse und TCP-Port). Selbst wenn diese durch einen Angreifer gefälscht werden, verfügt er nicht über den nötigen privaten Schlüssel zum im System verfügbaren Zertifikat. So nützt ihm der Besitz des Zertifikats nichts, wenn er den zugehörigen privaten Schlüssel nicht besitzt.

5.6.2 HAVi

IEEE 1394 ist vergleichbar mit TCP/IP nur zum Datentransport bestimmt und bringt von sich aus keine Sicherungsmaßnahmen mit. Die Anwendungsdaten müssen daher, wenn erforderlich, über ein Anwendungsprotokoll gesichert werden. HAVi hat als eine Sicherungsmaßnahme die Software-Elemente in *trusted* und *untrusted* eingeteilt. Alle System Software-Elemente sind vertrauenswürdig. Die Spezifikation überlässt es weitgehend den Herstellern, zusätzlich integrierte Software-Elemente als *trusted* oder *untrusted* zu kennzeichnen. Als Richtlinie wird vorgegeben, dass vorinstallierte Module nach Prüfung durch den Hersteller als vertrauenswürdig eingeordnet werden können. Nachträglich eingebrachte Software-Komponenten sollen durch herstellerspezifische Verfahren auf ihre Vertrauenswürdigkeit geprüft werden. Diese Einteilung verhindert das Entgegennehmen von Nachrichten, wenn die Quelle, gekennzeichnet durch die SEID, nicht vertrauenswürdig ist. So ist beispielsweise nur bestimmten System-Elementen der Zugriff auf den Communication Media Manager gestattet.
Die Einteilung dient jedoch weniger der Vereitelung von bewusstem Missbrauch, sondern vielmehr, um durch die Reduktion der Kommunikationsmöglichkeiten bestimmter Module untereinander ein stabileres Gesamtsystem zu erhalten[90]. In dieser Arbeit wurden keine zusätzlichen Maßnahmen eingesetzt, um einen Schutz gegen unbefugten Missbrauch zu erhalten. Es ist durchaus denkbar vor der Installation eines DCMs eine Prüfung des Jar Files mit einer digitalen Signatur zu unternehmen. Digitale Signaturen werden von Jar Dateien durchaus unterstützt. Mit einer Zertifikatsstruktur wie sie vorher beschrieben worden ist, kann auch hier ein Missbrauch der Ausführungsumgebung (FAV) verhindert

[90] Auf S.23 in [HAVi2001] heißt es genau: “Protection of a device (and the home network) from hostile or flawed applications is the responsibility of the vendor of the device. Such protection is particularly crucial for FAV devices since HAVi specifies an open programming environment for FAVs and arbitrary bytecode applications may be introduced to FAVs”. Damit wird die Verantwortung an den Hersteller abgegeben, durchaus im Bewusstsein, dass die dynamische Struktur und Ausführungsumgebung einen Missbrauch erlaubt.

werden. Eine Sicherung des Datenverkehrs an sich würde jedoch ein zusätzliches Protokoll erfordern, so wie es bei RMI der Fall ist. Ohne diese Sicherung sind alle Arten eines Datenangriffs umsetzbar.
Bei den am (HAVi-)System beteiligten Komponenten ist es jedoch fragwürdig, ob ein solcher Aufwand zum Einbringen falscher Informationen zur Steuerung des EIB auf einem relativ komplexen Protokoll ausgeführt werden sollte. Da der EIB ungesichert ist, kann ein Angreifer mit viel einfacheren Mitteln direkt dort ansetzen.

5.6.3 DECT

Alle Daten werden durch das in [ETSI300-7] definierte Protokoll vor Zugriffen von außerhalb geschützt. Das Protokoll beschreibt den Mechanismus, der die Daten verschlüsselt und nach der Übertragung entschlüsselt. Dazu dient ein *Key Stream Generator* (KSG), der durch bestimmte Sicherheitsschlüssel konfiguriert wird. Dieser KSG erzeugt einen kontinuierlichen Bit-Datenstrom, der mit den zu versendenden Daten verschlüsselt wird. Der Empfänger verfügt über ein gleich konfiguriertes Gegenstück, mit dem die Daten wieder entschlüsselt werden. Die Einstellung des KSG bezieht Anmeldevorgänge der Mobilteile und Feststation sowie Benutzereingaben – in der Regel die Eingabe einer vierstelligen Zahlenkombination – mit ein. Dadurch wird ein hoher Grad an Sicherheit erlangt. Durch ungeschickte Wahl (Verwendung der Werkseinstellung) dieser *Personal Identification Number* (PIN) kann das System jedoch leicht hintergangen werden. Bei Änderung der werkseitigen Einstellung lässt sich dieses Problem jedoch leicht beseitigen. Die Algorithmen zur Verschlüsselung sind jedoch „subject to controlled distribution“ [ETSI300-7]. Dazu gehören die „DECT standard cryptographic algorithms“ und „DECT standard cipher“.
Nicht angemeldete Mobilteile können nicht mit der Basisstation kommunizieren. Daher ist es – mit vertretbarem technischen Aufwand – unwahrscheinlich, dass es zu einem Missbrauch der Steuerungsfunktionalität des DECT Übergangs kommen kann.

5.6.4 Wireless LAN

Wireless LAN definiert bereits in den unteren Protokollschichten Sicherheitsmaßnahmen zur Verschlüsselung von Daten. Das *Wired Equivalent Privacy* (WEP) erlaubt die Verschlüsselung der Daten mit einstellbaren (40 Bit-) Schlüsseln. Mittlerweile wurde die Schlüssellänge auf 128 Bit erweitert, ohne jedoch von allen am Markt verfügbaren Karten unterstützt zu werden. Da Wireless LAN lediglich als Transportmedium zu sehen ist, sollten auch hier zusätzliche Vorkehrungen getroffen werden, um den Dateninhalt vor unbefugten Zugriffen zu schützen. Dazu können die gleichen Protokolle eingesetzt werden, die bei den Sicherungsmaßnahmen der RMI/Jini Dienstbeziehungen genannt worden sind.

5.6.5 Dienstzugriff

Der Zugriff auf Dienste der höheren Ebene ist zur Zeit noch unbeschränkt. Daher kann jeder Dienstnehmer, darunter auch ein extern eingebrachter Service-Agent, Dienste suchen und dann verwenden. Eine Einteilung in mindestens zwei Benutzerkategorien –

Fachpersonal und Anwender (Hauseigentümer) – ist jedoch empfehlenswert. Das Fachpersonal benötigt einen Vollzugriff auf das System, um es einrichten zu können. Dieser Funktionsaufwand ist für einen Hauseigentümer in der Regel nicht notwendig, sofern es sich um selbst-rekonfigurierende Dienste handelt. Die Einteilung kann auch durchaus von der Dienstebene abhängig gemacht werden. Ein Zugriff auf ein Bussystem der unteren Ebene ist auf Grund der fehlenden Werkzeuge von einem Anwender nicht durchführbar. Auf den oberen Ebenen kann ein Anwender jedoch mehr Einfluss nehmen, da dort die existierenden Dienste verwendet werden können, um ungewollte Zustände des Systems zu erreichen. Beispielsweise sollte nur das Fachpersonal einen administrativen Zugang zu dem OSGi-Gateway erlangen, um Bundles zu warten.

5.7 Ausfallbetrachtung

Innerhalb des Systems gibt es mehrere Arten von Fehlerfällen, die sich in ihrer Ursache und ihrem Wirkungsort unterscheiden.

Ein Total- oder Teilausfall, also die Einstellung der Funktion eines Gerätes bewirkt nur einen Fehler bei der Verwendung dieser Ressource. Dieser Fehler wird allen Dienstmerkmalnutzern bekannt gegeben, entweder bereits durch Mechanismen innerhalb des Busprotokolls (IEEE 1394 Bus Reset) oder durch zusätzliche Maßnahmen (beispielsweise Leasing). Damit ist die Fehlerausbreitung auf wenige Teilnehmer beschränkt, ohne die Gesamtfunktionalität des Busses einzubüßen. Die in Jini implementierte Kommunikationsebene 1 des Systemmodells sieht bereits Funktionen zur selbständigen Erholung von solchen Fehlern vor. Der Ausfall eines Lookup Servers kann kurzfristig dazu führen, keine neuen Dienste im System bekannt machen zu können, ohne jedoch die vorhandenen Dienstbeziehungen zu stören. Auch diesem Fall kann man vorbeugen, indem mehrere Lookup Server angeboten werden. HAVi erkennt selbsttätig das Entfernen und Hinzufügen von Geräten, so dass auch hier keine Verletzung der Kommunikationsfähigkeit des Systems auftreten kann. Durch die Verwendung von *ReliableMsg* werden auch Teilnehmer erkannt, die auf Grund eines internen Fehlerzustands nicht mehr erreichbar sind

Ein Kommunikationsfehler, also eine Verletzung des Protokolls kann unter gewissen Voraussetzungen zur Unterbrechung von Kommunikationsbeziehungen innerhalb des gleichen Subsystems führen. Dazu zählen Busüberlastung durch Generierung vieler Telegramme eines Teilnehmers, Fehler im Protokollablauf (senden eines NotAcknowledge auf jede Anfrage) und die Trennung der physikalischen/logischen (Software-Fehler) Verbindungen. Alle hier bearbeiteten Systeme verwenden gesicherte Protokolle zur Übermittlung der Daten[91], so dass diese Fehlerzustände leicht erkennbar sind. Diese Art von Fehlern werden nicht direkt an übergeordnete Schichten weitergereicht, sondern nur im Datenmodell vermerkt. Damit bleibt die Funktion des Systems aufrechterhalten. Sollten dennoch Teile dieser Fehlinformation an das übergeordnete System übergeben werden, so

[91] DECT eigene Sicherungsschicht, Bluetooth OBEX-Protokoll, EIB Immediate Acknowledges bzw. Punkt-zu-Punkt-Verbindungen, IEEE 1394/ HAVi Transport Adaptation Module, Jini TCP/RMI.

ist bedingt durch die Systemauslegung und die Staffelung der Übertragungsraten immer noch ein Betrieb möglich.
Durch falsche Konfiguration entstandene Fehler können große Auswirkungen haben. Vorstellbar ist die Verknüpfung von Komponenten, die zwar die gleichen Datentypen besitzen, aber in der Kombination zu unerwünschten Funktionen führen (Verknüpfung eines Schaltaktors für die Ventilsteuerung der Heizungsregelung mit einem Lichtschalter). Diese Fehlerart ist am schwierigsten zu lokalisieren und benötigt spezielle Kenntnis des Systems und der Parametrierungswerkzeuge. Auch nicht konfigurierte Teilnehmer können leicht Fehlfunktionen nach sich ziehen. Dies ist weniger bei den höheren Anwendungsebenen der Fall, da durch die Vereinfachung der Darstellung und der Konfiguration kaum Fehlzustände auftreten sollten, die dann aber auch einfacher zu lokalisieren sind.

5.8 Effizienzbetrachtung des Gesamtsystems

Kommunikationsbeziehungen des Systems sind über vielfältige Methoden möglich. Dazu wurden im Kap. 4.2 die Kommunikationstypen 1- 4 eingeführt. Innerhalb der Realisierung der Arbeit wurde gezeigt, dass jeder dieser vier Typen in bestimmten Anwendungsfällen sinnvoll eingesetzt werden kann. Der KomTyp 4 stellt die höchsten Anforderungen an das Gerät, damit lässt sich jedoch auch die größtmögliche Integration des dienstorientierten Konzepts erreichen. Dennoch sind die anderen Kommunikationstypen wichtig. So wurde der KomTyp 3 zur Verknüpfung mehrere EIB Installation verwendet. Das lässt sich auch leicht auf andere Bussysteme übertragen. KomTyp 2, also ein einfaches Weiterleiten von Informationen auf ein anderes Medium, kann in begründeten Ausnahmefällen eingesetzt werden. Durch die feine Granulierung der Kommunikationsbeziehungen können alle auftretenden Systemkonfigurationen umgesetzt werden.
Die Auslastungsanalysen haben gezeigt, dass der durch die Verwaltung der Dienste hervorgerufene Busverkehr vernachlässigbar gering ist. Eine Auslastungsanalyse der unteren Bussysteme war nicht notwendig, da dort keine Daten aus einem anderen System übertragen werden. Die Kapazitäten innerhalb dieser Systeme müssen durch die Konfiguration garantiert werden.
Ein Qualitätsmanagement der Ressourcen wie in [Groh1998] ist daher nicht erforderlich. Es handelt sich zwar um ein verteiltes System, aber innerhalb dieses Systems sind keine auffälligen Ungleichheiten der Lastverteilung zu erkennen. Embedded Geräte, die unter Umständen weniger Rechen- und Verarbeitungsleistung stellen können, wurden bereits bei der Systementwicklung durch die Nutzung unterschiedlicher Kommunikationstypen berücksichtigt. Zudem erfolgt eine Lastregelung bereits in der Middleware (vgl. Jini Lookup Server). Erst beim Einsatz von Multimediaanwendungen sollte über ein Ressourcenmanagement nachgedacht werden. Innerhalb eines Gebäudes wird das jedoch bei der zur Verfügung stehenden Bandbreite des Backbones nicht notwendig sein.

6. Konfiguration und Verwaltung

6.1 Zuständigkeiten

Durch die Integration der verschiedenen Systeme ergeben sich viele Aufgaben bei der Einrichtung des Gesamtsystems. Eine grobe Übersicht soll die nachfolgende Abbildung geben:

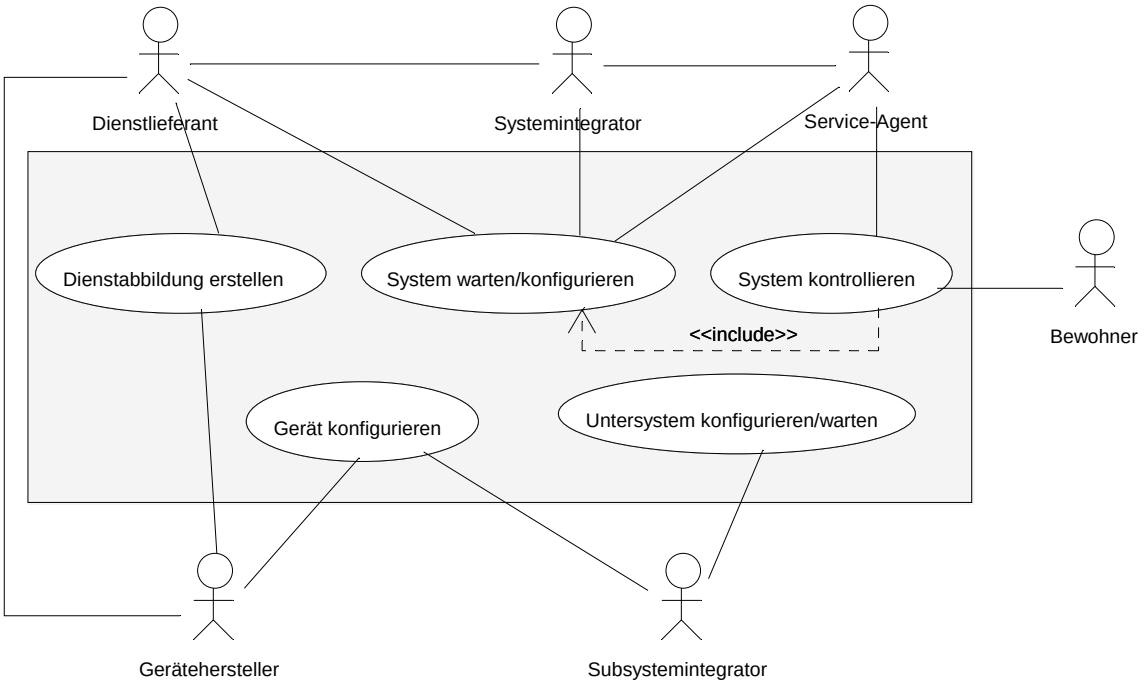

Abbildung 6.1: Aufgaben bei der Konfiguration und Instandhaltung des Systems[92]

Im Anwendungsfalldiagramm aus der obigen Abbildung sind mehrere Rollen eingezeichnet von denen einige von einer Organisation erfüllt werden können. So kann die Konfiguration des Systems durchaus durch die Aufnahme bestimmter Systemfähigkeiten in die Produkte reduziert werden. Ein Beispiel dafür ist ein Hersteller von EIB-Komponenten, der bereits alle Systemanforderungen erfüllt, indem er neben der Implementierung der Geräteeigenschaften auch Komponenten der höheren Systemebenen berücksichtigt. Dazu gehören die Umsetzung der Abbildung des Geräte auf einen entsprechenden Dienst und die Beschreibung der Dienstschnittstellen. Kann diese zusätzliche Dienstleistung nicht vom Hersteller geleistet werden, so kann er sie an einen Dienstlieferanten delegieren, oder aber im Auftrag des Kunden – in diesem Fall des Bewohners – zusätzlich vom Systemintegrator ausgeführt werden. Der Systeintegrator ist für die Koordination der Arbeiten am System zuständig. Er ist nicht nur der Ansprechpartner für Fragen des Bewohners zu der Bedienung, sondern auch

[92] Die selbstverständlich bestehende Beziehung zwischen dem Systemintegrator und dem Subsystemintegrator wurde übersichtshalber nicht eingezeichnet.

verantwortlich für die Durchführung und Beauftragung von Wartungs- und Reparaturaufträgen.
Der Hausbewohner muss sich im Idealfall mit der Konfiguration nicht beschäftigen. Sobald er mit dem Systemintegrator die Systemanforderungen festgelegt hat, werden die weiteren Arbeiten von zusätzlichem Fachpersonal übernommen. Die Beschreibung der Systemanforderungen beinhaltet Festlegungen über die verwendeten Geräte, deren Aufgabe und die Gerätebeziehungen, alles jedoch nur auf funktionaler Ebene, also ohne technisches Hintergrundwissen.
Die Installation eines Gerätes muss von einer Fachkraft vorgenommen werden. Nur in wenigen Fällen kann dies durch den Bewohner erfolgen. Denkbar wäre das Einspielen von Software nach dem Kauf eines PDAs. Dieser Prozess sollte so einfach gestaltet sein, dass keine Zusatzkosten durch Fachpersonal entstehen dürfen. Geräte, die erst mit dem System verknüpft werden müssen, sei es eine Wireless LAN Bridge oder eine EIB-Komponente, erfordern die Anwesenheit von Fachpersonal vor Ort (Subsystemintegrator). Nur bei Komponenten, die eine benutzerfreundliche Installation anbieten, kann dieser Vorgang entfallen. Ein erster Schritt in diese Richtung ist der so genannte Easy-Mode des KNX. Dabei werden einige Konfigurationsschritte (Adressvergabe) durch Automatismen unterstützt, wobei selbst dann die restliche Konfiguration eines Gerätes vom Fachpersonal durchgeführt werden muss.
Nach der Installation des Gerätes erfolgt die Einbindung in die höheren Abstraktionsschichten. Diese Aufgabe unterliegt dem Systemintegrator, der für alle Fragen zuständig ist, die nicht werkseitig gelöst werden. Er kann wiederum einen Dienstlieferanten beauftragen, diese Aufgabe zu lösen, der dann Kontakt mit dem Gerätehersteller aufnehmen kann.
Die nachfolgenden Konfigurationsschritte der Diensteinrichtung und -verknüpfung werden durch die verwendete Middleware vereinfacht. Der Service-Agent kann dem Systemintegrator dann auf Grund der Schnittstellenbeschreibungen Konfigurationsvorschläge geben. Bei HAVi ist keinerlei Konfiguration notwendig. Durch die Spezifikation sind alle Vorgänge zur Inbetriebnahme neu hinzugefügter Geräte bereits abgedeckt. Die Beschreibung der Dienste erfolgt hier durch den Hersteller – in Form der Bedienungsanleitung.
Es ist in naher Zukunft nicht absehbar, dass die Gerätehersteller ein gemeinsames Konzept zur Bildung eines Gesamtsystems unterstützen werden. Daher muss die Integration der einzelnen Komponenten durch zusätzliche Hilfsmittel, beispielsweise wie der hier vorgestellte Service-Agent, vereinfacht werden.
Die Konfiguration der Übergänge und der Einzeldienste war seit Beginn der Arbeiten eine begleitende Komponente bei der Entwicklung. Daher existieren für jede Dienstart Konfigurationswerkzeuge, die nachfolgend beschrieben werden sollen.

6.2 Konfiguration der Systemparameter

6.2.1 EIB

Von der Konnex Association wird bereits ein Werkzeug zur Konfiguration des EIBs angeboten. Dieses Werkzeug stützt sich auf zwei Datenbanken: Eine für die aktuelle Konfiguration des Systems mit den eingestellten Parametern, die andere zur Speicherung der Gerätebeschreibungen. Jedem Gerät wird eine Beschreibungsdatei mitgeliefert, die dann in die Produktdatenbank der Konfigurationssoftware eingefügt werden kann. Mit Hilfe dieser Beschreibung lässt sich das Gerät konfigurieren.
Bisher gab es keinen befriedigenden Ansatz um die Daten dieser Datenbank in irgendeiner Form auszulesen und in anderen Programmen auswerten zu können. Die entwickelte Kommunikationsobjektverwaltung benötigt jedoch wenigstens Auskünfte über die im System befindlichen EIB-Gruppenadressen, um auf dieser Basis gekapselte Objekte für die Repräsentation des Datenmodells erstellen zu können. Da nur die Gruppenkommunikation für die Datenmodellierung ausreichend ist, wurde ein Exportfilter entwickelt, das die zugänglichen Informationen über die Gruppenkommunikation extrahieren und in einer Datei speichern kann. Die fehlenden Informationen, wie der Installationsort, müssen nachträglich eingegeben werden. Dazu wurde ein graphischer Editor entworfen, der die Informationen über Eingabemasken entgegennimmt und diese dann in einer XML-Datei abspeichert.
Das unten dargestellte Beispiel ist ein Ergebnis dieses automatischen Imports; es wurden noch keine Informationen hinzugefügt. Der Importfilter ergänzt die fehlenden Informationen durch festgegebene Vorlagen oder lässt sie offen. Die wichtigsten Parameter wie der Datentyp, die Sende-/Empfangsadressen und die Bezeichnung können bereits auf diese Weise gewonnen werden. Dargestellt ist die detaillierte Ansicht eines EIB Gerätes. Da diese Daten innerhalb eines geschlossenen Konzepts nicht nur für die Gruppenkommunikation, sondern auch für die Konfiguration verwendet werden sollte, wurden bereits Felder vorgesehen, die in dieser Ausbaustufe noch keine Anwendung finden.

```
<Device>
    <EIBData>
      <System>
        <SysInformation>
          <Manufacturer>TU-Muenchen Importfilter</Manufacturer>
          <BCUVersion/>
          <PhyMed/>
          <SystemID/>
        </SysInformation>
        <SysParameter>
          <NLRepeatCnt/>
          <PhysAddr/>
          <DeviceLocation/>
        </SysParameter>
      </System>
      <Application>
        <PEIType/>
        <ApplDateConfigured>Wed Aug 29 16:29:17 GMT+02:00 2001
        </ApplDateConfigured>
        <ApplFlag>not configurable</ApplFlag>
        <BCUApplication>empty</BCUApplication>
      </Application>
      <CommunicationObjects>
        <COFlag>configured</COFlag>
        <CODateConfigured>Wed Aug 29 16:29:17 GMT+02:00 2001
        </CODateConfigured>
        <CommObj>
          <ObjectID>1</ObjectID>
          <COLocation>Eltern</COLocation>
          <DataType>1 Bit</DataType>
          <SendAddress>
            <Address>0x1123</Address>
            <Interval>0</Interval>
          </SendAddress>
        </CommObj>
      </CommunicationObjects>
    </EIBData>
</Device>
```

Abbildung 6.2: Ausschnitt aus einer importierten EIB Konfigurationsdatei

Für die Programmierung des EEPROM-Bereichs der BCU ist die Art und Maskenversion des Controllers wichtig, um die Protokollsequenz für den Programmiervorgang zu kennen. Die *SystemID* ist für die Powerline Komponenten von Bedeutung, da diese den Installationsort abgrenzt. Diese Informationen sind unter dem Punkt *SysInformation* zusammengefasst. Darunter folgen die Produktdaten, also die Daten die eine Parametrierung des Gerätes erst erlauben. Unter *CommunicationObjects* sind alle Informationen über die eingerichteten Kommunikationsobjekte verfügbar. Da einem Objekt mehrere Adressen zugeordnet sein können, kann dem Abschnitt *SendAddress* noch ein weiterer Bereich für die Speicherung der Empfangsgruppenadressen hinzugefügt werden. Der Aufbau der Datei ist durch eine DTD definiert.

6.2.2 Jini

Für die Dienstabbildung der EIB-Geräte auf Jini wurde ebenfalls eine Konfigurationssoftware benötigt, die die EIB-Geräte definiert und deren Eigenschaften beschreibt. Für die beschriebenen Gerätetypen können diese Eigenschaften mit einem

Editor bearbeitet und gespeichert werden. Das Programm greift direkt auf die Daten des oben beschriebenen Programms zurück.

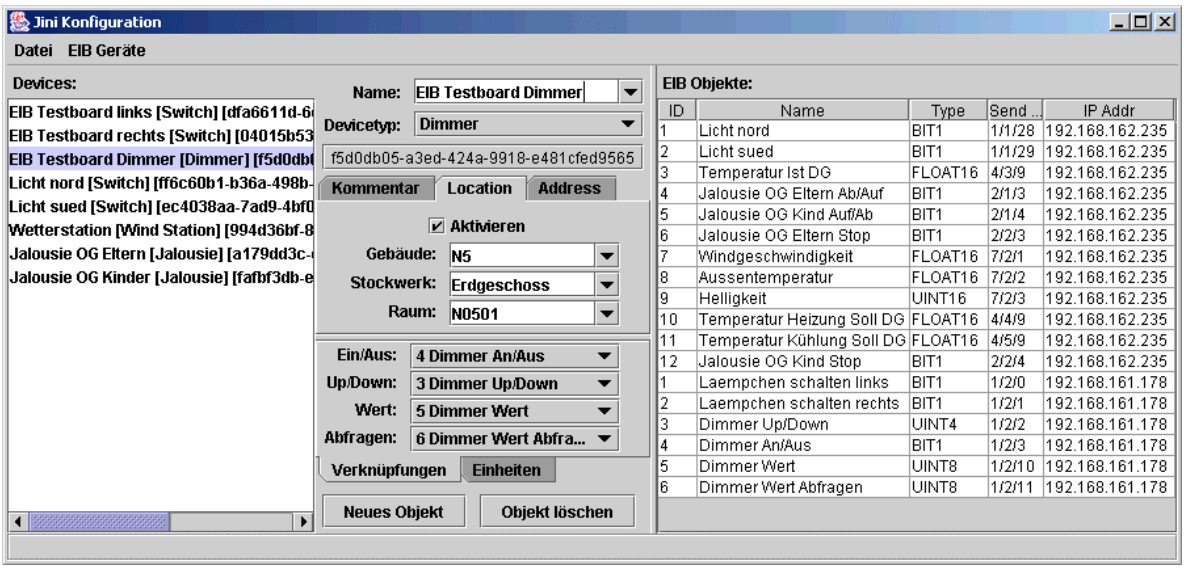

Abbildung 6.3: Ansicht des Jini Editors zur Festlegung der Eigenschaften des EIB

Die rechte Spalte zeigt die Informationen der EIB-Beschreibungsdatei. Dort ist auch bereits die Verknüpfung zum Datenmodell erfolgt (ID). Die linke Spalte zeigt alle bereits eingefügten Geräte und in der mittleren Spalte können die genauen Einstellungen vorgenommen werden. Dazu gehören die zugehörigen Kommunikationsobjekte, kurze Angaben des Dienstes (ServiceEntry) und die Bezeichnung des Gerätes. Ein fester Bestandteil eines Jini-Teilnehmers ist die ServiceID, eine 128 Bit lange Zahl. Diese wird bei der ersten Inbetriebnahme und Anmelden an einen Lookup Server zugewiesen und sollte dann gespeichert werden. Die 128 Bit Zahl wird vom Lookup Server generiert und erhebt den Anspruch weltweit einmalig zu sein. Damit können Dienstbeziehungen durch die Suche eines bestimmten Gerätes hergestellt werden.

Die Zuordnung der Kommunikationsobjekte zu den jeweiligen Gerätefunktionen erfolgt durch Vorauswahl. Nicht in Frage kommende Datentypen werden ausgeschlossen. Es besteht auch die Möglichkeit mehrere Datenbestände über EIB-Installationen gleichzeitig zu laden.

Nach Abschluss der Konfiguration werden diese Daten in einer XML-Datei hinterlegt, so dass die Informationen beim Start des Dienstes zugänglich sind. Mit dem Inhalt dieser Datei können dann die Datenstrukturen initialisiert und die Verbindung zum EIB aufgebaut werden.

Ein Hersteller könnte mit einfachen Mitteln diese Daten bereits mit dem Gerät ausliefern. Zusätzlich zu der sowieso benötigten Produktinformation könnte diese Datei dem Produkt beigefügt werden und somit eine sehr einfache Integration ermöglichen.

6.2.3 Fernzugriff, HAVi und DECT

Zur Darstellung der Steuerungsinformationen auf den beschriebenen Geräten wird ein gemeinsames Werkzeug verwendet. Eine Trennung der Darstellungen und eine Vervielfältigung der Konfigurationswerkzeuge ist kontraproduktiv. Sollte das System in seiner Konfiguration handhabbar bleiben, so ist die Auftrennung der Konfiguration in verschiedene Werkzeuge der Sache nicht dienlich.
Obwohl die Art der Darstellung der verwendeten Anzeige- und Bediengeräte zunächst zu heterogen erscheint, wurde durch geeignete Softwarestrukturen die Möglichkeit einer Wiederverwendung von Klassen gefunden. Das Kernstück sowohl der Bedienung als auch der Konfigurationsanzeige ist ein abstraktes graphisches Element, das alle Basisinformation über die Anzeige in sich trägt. Von dieser Klasse wurden alle nachfolgenden graphischen Elemente abgeleitet und umgesetzt. Sogar die Darstellung des DDI Controller des implementierten FAV verknüpft die DDI-Elemente mit diesen elementaren Klassen.
Damit zeigt sich, dass auch die Konfiguration und die Editierung der graphischen Oberflächen eine gemeinsame Basis haben. Es wurde dennoch bewusst vermieden, diese beiden Aspekte in einem Werkzeug miteinander zu verbinden. Die Konfiguration ist auf der Dienstebene ein einmaliger Vorgang und muss nur bei der Systemeinrichtung, evtl. bei der Systemnachrüstung vorgenommen werden. Für die Steuerung und Bedienung ist es jedoch wichtig Programme zu erhalten, die mit wenig Systemressourcen auskommen. Daher wurde beschlossen, diese beiden Aspekte voneinander zu trennen. Dieser Ansatz wurde beim Softwareentwurf bereits berücksichtigt und führt nicht zur Divergenz der Software Komponenten, da beide Werkzeuge gemeinsame Klassen verwenden.
Der Aufbau der Konfigurationssoftware und der Steuerungssoftware lässt sich auf ein Basiselement zurückführen: Ein Sammelbehälter zur Aufnahme mehrerer graphischer Objekte. Dieser Sammelbehälter besitzt bei allen Systemen die gleichen Grundeigenschaften, er gruppiert Instanzen von abgeleiteten Elementen des abstrakten graphischen Elements. Darauf aufbauend wurden spezialisierte Container entworfen, deren Abweichungen in der Implementierung jedoch marginal waren.

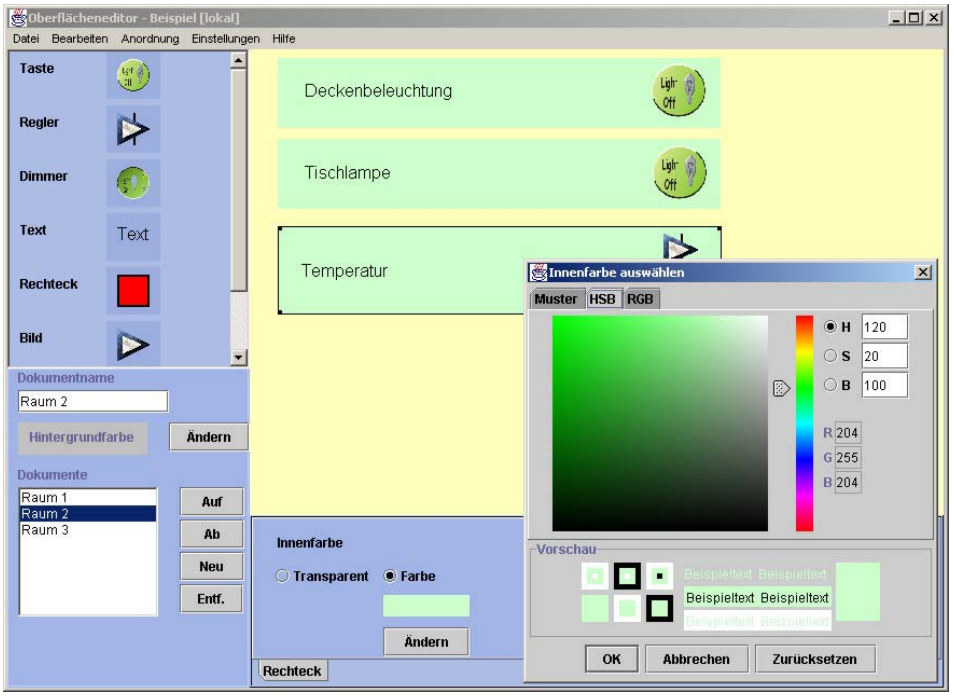

Abbildung 6.4: Konfigurationssoftware im Modus HAVi/Fernzugriff/lokal

Der grundsätzliche Aufbau und Aussehen der Software ändert sich für die Modi HAVi/Fernzugriff kaum. Das Aussehen unterscheidet sich lediglich in der Anzahl und Art der verfügbaren Komponenten. Auf der linken Seite können die graphischen Elemente ausgewählt und in den Editierbereich gezogen werden. Dabei werden mehrere Anzeigebereiche in Form von Dokumenten angeboten. Die Größe der Darstellung spielt keine Rolle: Alle Layout Parameter werden als relative Bezüge zur aktuellen Gesamtgröße gespeichert. Die Steuerungs- und Kontrollsoftware passt die Größe der Darstellung automatisch an den verfügbaren Bereich an und skaliert die graphischen Elemente entsprechend. Damit ist die Konfiguration vollkommen unabhängig von dem später eingesetzten Gerät. Eine geringere Auflösung des Anzeigebereichs wird automatisch angepasst.

Jedes Element verfügt über mehrere einstellbare Eigenschaften, die im unteren Teil des Fensters eingestellt werden können. Die Funktion des Programms ist vergleichbar mit der eines Graphikprogramms. Die Elemente können verschoben, skaliert, kopiert und in Ebenen angeordnet werden. Durch die Verwendung von Applets bzw. durch die definierten DDI-Elemente in HAVi ergibt sich ein hoher Freiheitsgrad bei der Gestaltung der Oberflächen.

Im Vergleich dazu eingeschränkt ist die Konfiguration der Oberfläche für den PDA. Da hier der Anzeigebereich wesentlich kleiner ist, müssen wenige sehr übersichtliche Elemente zur Anzeige verwendet werden, um die Steuerbarkeit beizubehalten. Daher wurde auf passive Elemente zur Ausgestaltung der Anzeige verzichtet.

Da die Steuerung mit dem Eingabestift erfolgt, wurde darauf geachtet möglichst große Flächen für die Eingabe und Auswahl von Elementen zu reservieren. Auch hier lassen sich Parallelen ziehen, auch wenn an dieser Stelle auf die Nutzung gemeinsamer Klassen zur

Darstellung verzichtet werden musste, weil eine andere Graphik-API verwendet wurde. Dennoch ließ sich die Konfiguration des PDAs in das vorhandene Werkzeug einarbeiten. Auch hier wurde eine spezialisierte Container Klasse entworfen, die die Anzeige der erstellten Oberfläche übernahm. Auch der Bereich für die Bereitstellung der graphischen Elemente wurde modular aufgebaut, so dass hier die bei Palm verwendeten Abbildungselemente eingebettet werden konnten.

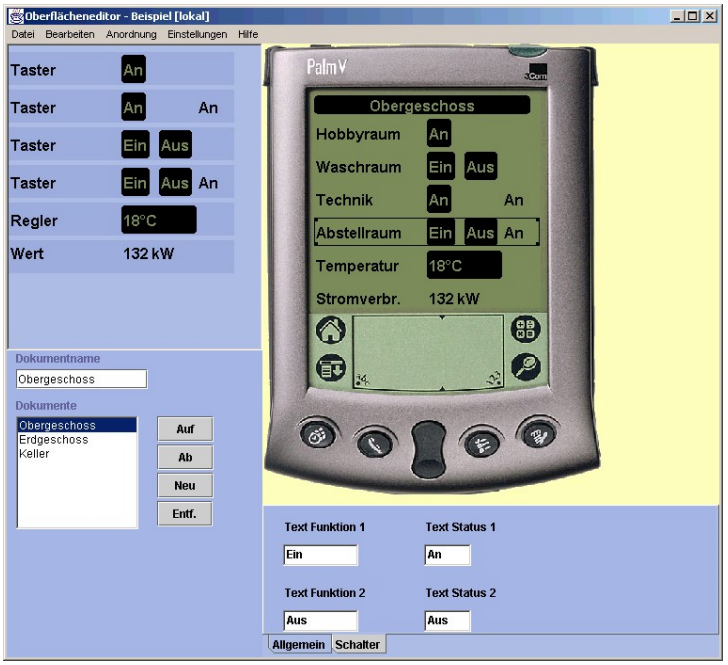

Abbildung 6.5: Konfiguration des PDA – hier der Palm Vx

Das Konzept der Teilung von Organisationseinheiten durch Unterteilung in mehrere Dokumente bleibt erhalten. Anders als bei den anderen Konfigurationsdateien wird beim PDA nicht XML als Datenformat bei der Speicherung verwendet, da er sonst zusätzlich noch die Auswertung von XML-Dokumenten unterstützen müsste.

6.2.4 OSGi

Die in der Arbeit eingesetzten OSGi-Dienste bedürfen keiner Konfiguration. Die implementierte Umsetzung des Fernzugriffs und der Anzeige des Dienststatus erfordert keinerlei Konfigurationsarbeiten innerhalb des OSGi-Frameworks.
Erst durch das Hinzufügen von neuen Diensten zu dem OSGi-Framework kann eine zusätzliche Parametrierung notwendig werden. Diese kann mit Hilfe einer Administrationsoberfläche für einen Internetbrowser integriert werden, wie bereits durch die JES Implementierung von Sun zur Installation und Wartung von Bundles erfolgt.
Hier zeigt sich erst das eigentliche Potenzial von OSGi: Für die Installation neuer Dienste ist es nicht zwingend notwendig vor Ort zu sein. Damit können jederzeit neue Dienste oder Dienstverknüpfungen dem System hinzugefügt werden.

7. Zusammenfassung

Nicht nur im Bereich der industriellen Automatisierungstechnik schreitet die Integration von Bussystemen zur Prozesssteuerung voran. Durch die Verfügbarkeit von neuartigen Komponenten werden automatisierungstechnische Lösungen auch für den Einsatz innerhalb der Gebäudetechnik und dort speziell im Bereich der Heimautomatisierung attraktiv. Bisher wurden die Systemgrenzen dieser Kommunikationssysteme jedoch nicht übersprungen. Dennoch ist die Anzahl der installierten Systeme in den heutigen Haushalten beachtlich: der Internet-Anschluss, die ISDN-Telefonanlage, ein kleines privates (Ethernet-)Netzwerk oder auch IEEE 1394 bei den verwendeten Digitalkameras. Mit nur wenigen zusätzlichen Mitteln kann bereits heute ein Mehrwert durch die Zusammenschaltung der Systeme erzielt werden.

In dieser Arbeit wurden Konzepte zur Verbindung von Bussystemen mit sehr unterschiedlichen Eigenschaften untersucht und umgesetzt. Zur Demonstration des hier aufgestellten Gesamtkonzepts wurde aus jeder für die Heimautomatisierung in Frage kommenden Buskategorie mindestens ein Bussystem in das System eingebunden: für die klassischen Steuerungs- und Kontrollfunktionen der HKL-Technik der EIB, aus dem Bereich der Telefonie DECT und ISDN, Ethernet zur PC-Vernetzung, IEEE 1394 als Einsatzmöglichkeit zur Anbindung von Multimediageräten sowie Wireless LAN und Bluetooth als kabelloser Ersatz für mobile Geräte. Da jedes Bussystem für seinen Aufgabenbereich spezialisiert ist und in bestimmten Marktsegmenten dominiert, ist es kaum vorstellbar, alle Systeme durch ein einziges abzulösen.

Für die Bildung eines Bussystemübergangs wurden hier zwei Konzeptvorschläge entwickelt: ein zentraler und ein dezentral/zentral kombinierter Ansatz. Die Nutzung eines zentralen, einstufigen Konzepts zur Verbindung mehrerer Bussysteme hat sich nur als bedingt tragfähig erwiesen. Das einstufige System verwendet nur ein Gerät zur Untersystemkopplung. Bei einer Vielzahl von unterschiedlichen Bussystemen muss auch die Rechenleistung dieses Geräts entsprechend ausgeprägt sein. Zudem liegt die Verantwortung für die Nachrüstung des Verbindungsknotens beim Hersteller. Diese Aufgabe kann nicht durch einen Gerätehersteller eines Subsystemgeräts durchgeführt werden. Zum einen verfügt er nicht über das notwendige Systemwissen, zum anderen erfordert das eine enge Kooperation zwischen Geräteherstellern und dem Hersteller des Gateways. Auch der Aufbau eines solchen Verbindungsknotens kann nicht derart universell erfolgen, dass alle zukünftigen Erweiterungen bedacht werden können. Auf Grund der geringen Erweiterungsmöglichkeiten und der hohen Fehleranfälligkeit wurde das Konzept daher verworfen.

Ähnlich wie auch in dezentralen Bussystemen wurde anstelle dessen ein System erstellt, das eine autonome Funktionsweise der Untersysteme und gemeinsame Kommunikation unterschiedlichster Bussysteme miteinander vereint. Für die Erstellung eines dezentralen Ansatzes wurde zunächst eine Einteilung der verwendeten Dateninhalte innerhalb der Heimautomatisierung vorgenommen. Dabei hat sich gezeigt, dass sich die Anforderungen der Anwendungen an die Bussysteme in zwei Hauptpunkten unterscheiden: in der

benötigten Übertragungsrate und in der Echtzeitfähigkeit der Übertragung. Besondere Anforderungen an ein Kommunikationsprotokoll stellen die isochronen Datenströme, deren Inhalte innerhalb von festgelegten Zeiträumen verfügbar sein müssen. Im Gegensatz dazu sind die einzuhaltenden Zeitkonstanten bei den Steuerungsbussystemen leicht zu erfüllen. Aus der vorgenommenen Kategorisierung der Bussysteme ergab sich ein maximal dreistufiges, hierarchisches System, dessen Hierarchieebenen durch Vermittlungsknoten miteinander verbunden sind. Dabei muss innerhalb einer Hierarchieebene nicht zwangsweise nur ein Bussystem eingesetzt werden. So wurden auf der obersten Ebene, der Ebene 1, IEEE 1394 und Ethernet benutzt. Damit wurde gezeigt, dass auch auf gleicher Ebene Übergänge eingefügt werden können, wobei die Eigenschaften der beteiligten Bussysteme vergleichbar sein sollten. Darin zeigt sich der dezentral/zentral kombinierte Ansatz des Konzepts. Der Übergangsknoten ist systembedingt eine zentrale Instanz, während die untergeordneten Geräte vollkommen autonom arbeiten können. Auf der obersten Ebene wurden Dienste definiert, die dann systemweit genutzt werden können. Jedem im System verfügbaren Dienst ist ein Basisdienst zugeordnet, der bestimmte Minimalanforderungen eines jeden Dienstanbieters garantiert, darunter Methoden zur Abfrage der Diensteigenschaften und eine Möglichkeit zur visuellen Darstellung des Dienstes. Durch diese Festlegung wird innerhalb des Systems eine Standardschnittstelle angeboten, die zur Dienstverwendung genutzt werden kann. Die zunächst abstrakt beschriebenen Dienste – ein Audio-, Video-, Steuerungs- und Informationsdienst – wurden im weiteren Verlauf der Arbeit für die Bussysteme umgesetzt. Zu diesen Diensten gehören die Integration und Abbildung einzelner Geräte der untersten Ebene, aber auch das Erstellen von Diensten, die nur innerhalb der Ebene 1 arbeiten. Als Bindeglied zwischen den Diensten und der busabhängigen Kommunikation fungierte dabei Middleware, zu deren Funktionsumfang Schnittstellen zum Hinzufügen, Auffinden und Konfigurieren der Diensten gehören. In dieser Arbeit wurden Jini, HAVi, OSGi und RMI eingesetzt, um jeweils die Dienstfunktionen von der Buskommunikation zu entkoppeln.

Neben der Abbildung von Gerätefunktionalitäten auf Dienste wurde auch die Systemverknüpfung durch geeignete Methoden beschrieben und integriert. Dafür wurde ein ebenfalls auf Jini basierender Service-Agent erstellt, der Dienstverknüpfungen auf der Basis von Schnittstellenbeschreibungen herstellen kann. Der Vorteil der Nutzung von Service-Agenten zur Verknüpfung der Dienste und damit auch der Subsysteme liegt in der Möglichkeit der flexiblen Umgestaltung der Dienstverbindungen. Im Gegensatz zu Dienstverknüpfungen, die durch eine feste Software-Implementierung erstellt werden, kann der Service-Agent die Verknüpfungen unter Beibehaltung der Software-Komponenten durch Umkonfiguration verändern.

Der hierarchische Systemaufbau und die Verwendung der Middleware führt zu einem störungsunempfindlichen System. Unregelmäßigkeiten in der Funktionsweise des Systems werden verhindert und durch die Kommunikationsprotokolle erkannt. Mit Hilfe von geeigneten Vorkehrungen, die in dieser Arbeit aus der Kapselung der Daten des niederen Systems und einer Ausfilterung von nicht verwendeten Daten bestehen[93], lassen sich systemweite Auswirkungen von Geräteausfällen wirksam unterdrücken. Die Systemanalyse der Bus- und Anwendungsprotokolle hat gezeigt, dass die Verwaltung und die Verwendung der Dienste bei Einhaltung der hier aufgestellten Regeln im Betrieb nicht zu einer Überlastung der Transportmedien führen können.
Der Aufbau von internen Verbindungsmechanismen wurde auf externe Zugänge erweitert. So ist die Bedienung und Wartung der hier vorgestellten Systemverbindungen mit einer Fernsteuerung und -diagnose gekoppelt. Durch die Verwendung von OSGi kann auf Teilkomponenten des Systems zugegriffen werden, Module können nachträglich gestartet oder neu initialisiert werden. Auch eine Einbindung neuer Komponenten gestaltet sich sehr einfach.
Zum Schutz gegen Missbrauch von Systemfunktionen wurden die Dienste mit Verschlüsselungsprotokollen vor unbefugtem Zugriff geschützt. Dies ist nicht system-durchgängig möglich, weil nicht alle Busprotokolle, insbesondere die der untersten Ebene, Sicherungskonzepte vorsehen.
Zur Einrichtung des Systems wurde ein mehrgliedriges Konzept erarbeitet, das die Aufgabenverteilung zwischen Geräteherstellern, Fachpersonal zur Installation der Geräte und Systemintegratoren regelt. Der vorgestellte Entwurf zur Systemkonfiguration skizziert die Verteilung der Aufgaben unter Geräteherstellern, Subsystemintegratoren, Dienst-lieferanten, Systemintegratoren und Bewohnern. Der Bewohner wird von den Konfigurationsaufgaben befreit, da die Koordinierung der Konfigurationsvorgänge vom Systemintegrator ausgeführt wird. Sofern keine Standardwerkzeuge für die Konfiguration des Systems vorhanden waren, wurden diese innerhalb der Arbeit entwickelt. Alle erstellten Werkzeuge besitzen einen ähnlichen, wenn nicht sogar gleichen Aufbau, was sich auch in der graphischen Darstellung zeigt. Die Konfiguration und die Steuerung wurden bereits beim Systementwurf voneinander getrennt, weil es sich bei der Konfiguration um einen abgeschlossenen Vorgang handelt, der nur beim Hinzufügen eines neuen Geräts wiederholt werden muss. Eine Kombination aus Anzeige und Konfiguration ist daher nicht notwendig.

[93] Ein einfaches Beispiel dazu ist die Abbildung des EIB. Es werden nur der Kommunikationsobjektverwaltung bekannte Objekte am EIB überwacht. Die Datenänderungen werden nicht selbstständig von der Verwaltung an Applikationen weitergegeben, sondern sie müssen sich bei der Verwaltung für bestimmte Objekte anmelden. Broadcast-Telegramme werden gar nicht erst entgegengenommen.

Der größte Teil dieses Systems wurde in Java umgesetzt, darunter auch große Teile der Buskommunikation (HAVi, EIB, DECT, Bluetooth, Jini). Es hat sich gezeigt, dass die Verwendung einer plattformunabhängigen Sprache große Vorteile bietet, da sich die Software ohne Probleme auf den meisten Zielsystemen einsetzen lässt[94].

Das sehr breite Anwendungsspektrum des hier vorgeschlagenen Konzepts zeigt die Möglichkeiten und Potentiale des Systementwurfs anhand einer großen Zahl von unterschiedlichen Bussystemen auf. Dieser Ansatz kann erfolgreich durch die Anwendung des Konzepts auf andere (Bus-)Systeme fortgeführt werden. Zur Einrichtung des Systems muss die Aufgabe des Dienstlieferanten noch von einem spezialisierten Betrieb übernommen werden. Es ist zur Zeit nicht absehbar, dass dies von den Geräteherstellern übernommen wird.
Die Weiterführung des hier erstellten Service-Agenten kann zu einer weiteren Vereinfachung der Konfiguration führen. Durch die Einarbeitung von Regeln für den selbstständigen Aufbau von Dienstverknüpfungen kann der Service-Agent zur halbautomatischen Diagnose und Fehlersuche verwendet werden. Wird der Service-Agent zu einem mobilen Agenten umfunktioniert, der sich auch an systemfremde Umgebungen anpassen kann, könnten Wartungsarbeiten komplett automatisiert oder aber mit der Erstellung eines entsprechenden OSGi-Bundles per Ferndiagnostik ausgeführt werden.

Die Hauptpfeiler dieser Arbeit, bestehend aus einem offenen Systemkonzept zur Integration verschiedenartiger Bus- und Kommunikationssysteme, den hier vorgenommenen Dienstdefinitionen und -umsetzungen, den Konfigurationswerkzeugen und dem Sicherheitskonzept, erlauben die Überwindung der bisher bestehenden Systemgrenzen einzelner Bussysteme innerhalb der Heimautomatisierung.
Die in dieser Arbeit aufgestellten Methoden für ein generelles Heimautomatisierungssystem mit unterschiedlichen Bussystemen wurden prototypisch realisiert und müssen im nächsten Schritt von einem (mittelständischen) Unternehmen übernommen und auf konkrete Projekte angewendet werden. Dazu müssen – abhängig von den Projektanforderungen – neue Komponenten eingebunden, Anpassungen an weitere Bussysteme vorgenommen und die erforderlichen Systemverknüpfungen erstellt werden. Neuentwicklungen sind hierzu in der Regel nicht notwendig. Durch den Abschluss von mehreren Projekten wird ein Grundstock an Buskomponenten, Bussystemanpassungen und vorgefertigten Systemverknüpfungen vorhanden sein, der in weiteren Projekten als Basis dient und damit zu einer drastischen Reduzierung des Aufwands führt. Folglich können die Preise von den Systemintegratoren gesenkt werden und werden damit für den Endkunden attraktiver. Durch den Zuwachs an verschiedenen Busankopplungsbausteinen kann der Systemintegrator eine Vielzahl von Kombinationen umsetzen. Bei Kenntnis der

[94] Die einzige Voraussetzung ist das Vorhandensein einer Java Virtual Machine für das Zielsystem. Einschränkungen ergeben sich bei busabhängigen Gerätetreibern, die dann über ein JNI angebunden werden müssen und Plattformabhängig sind.

Systemkomponenten kann die Einrichtung eines Gebäudes mit einem Bussystem der Ebene 1 und einem der Ebene 3 innerhalb eines Tages abgeschlossen werden. Die erstellten Komponenten können des Weiteren auch an Drittanbieter verkauft werden, wenn keine Kapazitäten mehr für eine Systembetreuung vorhanden sein sollten. Mit einem Lizenzierungssystem werden auch dann noch Wertschöpfungen erreicht.
Durch die plattformunabhängige Verwendung von neuesten Entwicklungen des Software-Engineering kann die Integration neuer Komponenten auch in Zukunft sichergestellt werden. Der Einsatz dieses Systems bringt für den Endkunden – den Bewohner – einen hohen Mehrwert mit sich, der auch zu einer Erhöhung der Nachfrage von Automatisierungskomponenten innerhalb der Heimautomatisierung führen könnte. Davon profitieren sowohl die Gerätehersteller von Buskomponenten als auch der Bewohner. Ein Beispiel für den Nutzen des Systems ist die Anzeige von Alarmen der Haustechnik (Gas- und Brandmelder) auf einem Fernseher oder aber die Kopplung des Einbruchmeldesystems mit der Bluetooth-Kamera. Erst durch die durchgängige, leicht erweiterbare Verbindung der Einzelsysteme können die Vorteile der Teilsysteme dem Endkunden bewusst gemacht werden, wobei seine Privatsphäre durch Einbeziehung des Sicherheitskonzepts gesichert ist und dennoch eine kostengünstige Fernwartung zulässt.

8. Literatur

[Aiken2000] AIKEN, B.; CAREY, M.; CARPENTER, B.; et al.: *Network Policy and Services: A Report of a Workshop on Middleware – RFC2768*. 2000.

[Anderson1999] ANDERSON, Don: *FireWire System Architecture, Second Edition - IEEE 1394a*. 5. Ausgabe, Boston : Addison-Wesley, 1999.

[Arciniegas2001] ARCINIEGAS, Fabio: *XML Developer's Guide – Erstellen effizienter XML-Anwendungen*. Poing : Franzis, 2001.

[Bader2000] BADER, Herbert; HUBER, Walter: *Jini™ – Die intelligente Netzwerkarchitektur in Theorie und Praxis*. München : Addison-Wesley, 2000.

[Breitling2001] BREITLING, Max: *Formale Fehlermodellierung für verteilte reaktive Systeme*. Dissertation, Technische Universität München, 2001.

[d'Inverno2001] D'INVERNO, Mark: *Understanding Agent Systems*. Berlin : Springer, 2001.

[Dietrich2000] DIETRICH, D.; KASTNER, W.; SAUTER, T. (Hrsg.): *EIB – Gebäudebussystem*. Heidelberg : Hüthig, 2000.

[Echelon2001] ECHELON CORP.: *Echelon's LonWorks Products – Fall 2001 Spring 2002 Edition Version A*. 2001.

[Farance2001] FARANCE, Frank: *ISO/IEC JTC 1/SC 25/WG 1 Interconnection of Information Technology Equipment Home Electronic System - Presentation to SC25/WG1 On Standards Approach*. ISO/IEC, 2001.

[Finin1997] FININ, Tim; LABROU, Yannis: KQML as an agent communication language. In: BRADSHAW, Jeffrey (Hrsg.): *Software Agents*. Cambridge : MIT Press, 1997, S. 1-22.

[Foner1993] FONER, Leonard N.: *What's An Agent, Anyway? A Sociological Case Study*. MIT Media Lab, 1993.

[Furrer2000] FURRER, Frank J.: *Ethernet-TCP/IP für die Industrieautomation - Grundlagen und Praxis*. Heidelberg : Hüthig, 2000.

[Gamma1996] GAMMA, E.; Helm, R.; JOHNSON, R.; et al.: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software*. München : Addison-Wesley, 1996.

[Ganesh2002] GANESH, J Pai; DUGAN, Joanne Bechta: *Automatic Synthesis of Dynamic Fault Trees from UML System Models - Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE)*. 2002.

[Gosling2000] GOSLING, James; JOY, Bill; STEELE, Guy; BRACHA, Gilad: *The Java Language Specification - Second Edition*. Boston : Addison-Wesley, 2000.

[Groh1998] GROH, Sascha: *Ein agentenbasiertes, flexibel anpassungsfähiges Ressourcemanagement für verteilte, parallele und kooperative Systeme*. Haar : Verlag Christiane Zaininger, 1998.

[Halloway2001] HALLOWAY, Stuart Dabbs: *Component Development for the Java Platform*. S.l.: Addison-Wesley, 2001.

[Halsall1996] HALSALL, Fred: *Data Communications, Computer Networks and Open Systems*. 4. Ausgabe, Harlow : Addison-Wesley, 1996.

[Kellerer2001] KELLERER, W.; STIES, P.: Signalisierungsplattform zur netzübergreifenden Dienststeuerung in heterogener TIME Infrastruktur. In: KILLAT, Ulrich (Hrsg.); LAMERSDORF, Winfried (Hrsg.): *Kommunikation in Verteilten Systemen (KiVS) - 12.Fachkonferenz der Gesellschaft für Informatik (GI), Fachgruppe Kommunikation und Verteilte Systeme (KuVS) unter Beteiligung des VDE/ITG*. Hamburg 20.-23. Februar 2001, S. 391 – 401.

[Lea2002] LEA, Rodger; GIBBS, Simon; GAUBA, Ravi; BALARAMAN, Ram: *HAVi - Example by Example – Java Programming for Home Entertainment Devices*. London : Prentice Hall PTR, 2002.

[Liang1999] LIANG, Sheng: *The Java Native Interface – Programmer's Guide and Specification*. Reading [Massachusetts] : Addison-Wesley, 1999.

[Liebl2001] LIEBL, Andreas : *Integration des Europäischen Installationsbusses in ein HAVI-Netzwerk*. Diplomarbeit, Technische Universität München, 2001.

[Litwak1999] LITWAK, Kenneth : *Pure Java 2 – A Code-Intensive Premium Reference*. Sams Publishing, 1999.

[Miano1999] MIANO, John: *Compressed Image File Formats – JPEG, PNG, GIF, XBM, BMP - Your guide to graphics files on the Web*. Reading [Massachusetts] : Addison-Wesley, 1999.

[Müller2001] MÜLLER, Wolfgang: *Überwachung elektrischer Hausgeräte durch Leistungsanalyse mit einem vernetzten Sensorknoten*. Dissertation, Technische Universität München, 2001.

[Murch2000] MURCH, Richard; JOHNSON, Tony: *Agententechnologie: Die Einführung – Intelligente Software-Agenten auf der Informationssuche im Internet*. München : Addison-Wesley, 2000.

[Nöhmeier1998] NÖHMEIER, Martina: *Einsatz von Agententechnologie zur Unterstützung von Informationsaustausch in globalen Informationsräumen*. Dissertation, Technische Universität München, 1998.

[Oaks1999] OAKS, Scott; WONG, Henry: *Java Threads*. 2. Ausgabe, Peking : O'Reilly, 1999.

[Oaks2001] OAKS, Scott; WONG, Henry: *Jini in a Nutshell – Deutsche Ausgabe für Jini 1.0 und 1.1*. Köln : O'Reilly, 2001.

[Oestereich1998] OESTEREICH, B.: *Objektorientierte Softwareentwicklung, Analyse und Design mit der Unified Modeling Language*. München : Oldenbourg Verlag, 1998.

[Santifaller1990] SANTIFALLER, Michael: *TCP/IP und NFS in Theorie und Praxis – UNIX in lokalen Netzen*. Bonn : Addison-Wesley, 1990.

[Schneider2001] SCHNEIDER, Friedrich; WERTHSCHULTE, Kay; BINTERNAGEL, Lars; et al.: In: GASSMANN, Oliver; MEIXNER, Hans (Hrsg.) : *Sensors in Intelligent Buildings, Sensors Applications Vol. 2*. Weinheim : Wiley-VCH, 2001, S. 511-558.

[Schneier1996] SCHNEIER, Bruce : *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C*. Bonn : Addison-Wesley, 1996.

[Segal2000] SEGAL, Richard B.; KEPHART , Jeffrey O.: Incremental Learning in SwiftFile. In: *Proceedings of the Seventh International Conference on Machine Learning*. Stanford University, 2000.

[Selke2000] SELKE, Gisbert W.: *Kryptographie – Verfahren, Ziele, Einsatzmöglichkeiten*. Peking : O'Reilly, 2000.

[Škrtić 001] ŠKRTIĆ, Stjepan; WERTHSCHULTE, Kay; SCHNEIDER, Friedrich: Tele-Supervision and Tele-Control of Smart Homes. In: SCHILLING, Klaus; ROTH, H: *Telematics Applications in Automation and Robotics - A Proceedings volume from the IFAC Conference*. S.l. : Elsevier Science, 2001, S. 385-391.

[StatBund2001] STATISTISCHES BUNDESAMT DEUTSCHLAND: *Ausstattung privater Haushalte mit Informationstechnik – Stand 1.Januar 2001*. 2001.

[Vlissides1999] VLISSIDES, John: *Entwurfsmuster anwenden*. München : Addison-Wesley, 1999.

[Weinzierl2001] WEINZIERL, Thomas: *Integriertes Managementkonzept für die Gebäudetechnik*. München : Pflaum, 2001.

[Werthschulte2000] WERTHSCHULTE, Kay; SCHNEIDER, Friedrich: Providing Access to an EIB-Installation Using Internet Browser Technology. In: *Proceedings of the EIB Event*. 2000.

[Werthschulte2001a] WERTHSCHULTE, Kay; SCHNEIDER, Friedrich: Linking Devices to the European Installation Bus. In: *Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference*. Band 2. 2001, S. 827-832.

[Werthschulte2001b] WERTHSCHULTE, Kay; WESTERMEIR, Günter: Secured Data Transmission for Control and Supervision of an EIB-Installation Using Mixed Network Topologies. In: *Proceedings of the EIB Scientific Conference*. 2001.

[Werthschulte2002a] WERTHSCHULTE, Kay; SMARDA, B.: Haussteuerung über Basisstation mit mobilen Bedien- und Anzeigegeräten. In: *Tagungsband des Abschlussseminars Intelligente Hausinstrumentierung IWO-BAY*. Neubiberg b. München, 22.03.2002, S. 187-202.

[Werthschulte2002b] WERTHSCHULTE, Kay: Dual-Funkbuskonzept im Überblick. In: *Tagungsband des Abschlussseminars Intelligente Hausinstrumentierung IWO-BAY*. Neubiberg b. München, 22.03.2002, S. 183-186.

[Werthschulte2002c] WERTHSCHULTE, Kay; SCHNEIDER, Friedrich: Export of Interfaces and Methods for EIB/KNX Devices. In: *Proceedings of the Konnex Scientific Conference*. 2002.

[Zuser2001] ZUSER, Wolfgang; BIFFL, Stefan; GRECHENIG, Thomas; et al.: *Software Engineering mit UML und dem Unified Process*. München : Pearson Education Deutschland, 2001.

Standards und Spezifikationen:

[Anderson2000] ANDERSON, Eric W.; BAKER, Richard; HUNTER, David; et al.: *1394 Open Host Controller Interface Specification Release 1.1*. 2000.

[Arai2001] ARAI, Tatsuo; YAMAMOTO, Ryohei; RANG, Maria; et al. : *Basic Imaging Profile Interoperability Specification – Version 0.95c*. 2001.

[CAN1991] ROBERT BOSCH GMBH: *CAN Specification Version 2.0*. Stuttgart, 1991.

[EIBA1999] EIBA: *EIBA Handbook Series, Release 3.0, Version 1.1*. Brussels : EIBA, 1999.

[EN300652] EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE: *EN 300 652 - Broadband Radio Access Networks (BRAN) - HIgh PErformance Radio Local Area Network (HIPERLAN) - Type 1 - Functional specification*. Sophia Antipolis, 1998.

[ETSI300-1] EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE: *Radio Equipment and Systems (RES); Digital Enhanced Cordless Telecommunications (DECT);Common Interface (CI);Part 1: Overview - ETS 300 175-1*. Zweite Ausgabe, Sophia Antipolis : ETSI, September 1996.

[ETSI300-2] EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE: *Radio Equipment and Systems (RES); Digital Enhanced Cordless Telecommunications (DECT);Common Interface (CI);Part 2: Physical Layer(PHL) - ETS 300 175-2*. Zweite Ausgabe, Sophia Antipolis : ETSI, September 1996.

[ETSI300-3] EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE: *Radio Equipment and Systems (RES); Digital Enhanced Cordless Telecommunications (DECT);Common Interface (CI);Part 3: Medium Access Control (MAC) Layer - ETS 300 175-3*. Zweite Ausgabe, Sophia Antipolis : ETSI, September 1996.

[ETSI300-4] EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE: *Radio Equipment and Systems (RES); Digital Enhanced Cordless Telecommunications (DECT);Common Interface (CI);Part 4: Data Link Control (DLC) Layer - ETS 300 175-4*. Zweite Ausgabe, Sophia Antipolis : ETSI, September 1996.

[ETSI300-5] EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE: *Radio Equipment and Systems (RES); Digital Enhanced Cordless Telecommunications (DECT);Common Interface (CI);Part 5: Network (NWK) layer - ETS 300 175-5*. Dritte Ausgabe, Sophia Antipolis : ETSI, Dezember 1997.

[ETSI300-7] EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE: *Digital Enhanced Cordless Telecommunications (DECT); Common Interface (CI); Part 7: Security features - ETSI EN 300 175-7, V1.5.1*. Sophia Antipolis : ETSI, Februar 2001.

[ETSIDPRS] EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE: *Digital Enhanced Cordless Telecommunications (DECT); DECT Packet Radio Service (DPRS) - ETSI EN 301 649 V1.3.0*. Sophia Antipolis : ETSI, 2002.

[IDA2001] IDA GROUP E.V.: *Interface for Distributed Automation – Architecture Description and Specification Revision 1*. Blomberg : IDA Group, 2001.

[Freier1996] FREIER, Alan O. ; KARLTON, Philip ; KOCHER, Paul C.: *The SSL Protocol Version 3.0 Internet Draft*. 1996.

[IEC61883] IEC: IEC 61883: *Consumer audio/video equipment - Digital interface, Part 1: General*. IEC, 1998.

[IEEE1394-1995] IEEE: *IEEE1394-1995, IEEE Standard for a High Performance Serial Bus*. 1996.

[IEEE1394-1995b] IEEE: *P1394b Draft V 1.3.3 - Standard for a High Performance Serial Bus (High Speed Supplement)*. 2001.

[IEEE1394B-2002] IEEE: *1394B-2002 IEEE Standard for a Higher Performance Serial Bus-Amendment 2*. 2002.

[IEEE802.11] IEEE: *IEEE Standard for Information Technology - Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Network - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - IEEE Std. 802.11*. 1999.

[IEEE802.3] IEEE: *IEEE Standard for Information Technology - Telecommunications and Information Exchange between Systems - Local and Metropolitan Area networks - Specific requirements, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications - IEEE Std. 802.3*. 2002.

[IETFRFC2617] FRANKS, J.; Hallam-Baker, P; HOSTETLER, J; et al.: *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*. 1999.

[IETFRFC2246] ALLEN, C. ; DIERKS, T.: *RFC2246: The TLS Protocol Version 1.0*. 1999.

[IETFRFC3261] ROSENBERG, J.; SCHULZRINNE, H.;CAMARILLO, G.; et al.: *Request for Comments 3261:Session Initiation Protocol. Internet Engineering Task Force*. 2002.

[ISOIEC13213] ISO/IEC : *ISO/IEC 13213: 1994 [ANSI/IEEE Std 1212, 1994 Edition], Information technology - Microprocessor systems - Control and Status Registers (CSR) Architecture for microcomputer buses*. 1994.

[HAVi2000] HOME AUDIO/VIDEO INTEROPERABILITY, INC.: *The HAVi Specification - Specification of the Home Audio/Video Interoperability (HAVi) Architecture*. 2000.

[HAVi2001] HOME AUDIO/VIDEO INTEROPERABILITY, INC.: *The HAVi Specification - Specification of the Home Audio/Video Interoperability (HAVi) Architecture Version 1.1*. 2001.

[ITUH.323] INTERNATIONAL TELECOMMUNICATION UNION: *Packet-based multimedia communications systems - Recommendation H.323 (11/00)*. 2000.

[Romkey1988] ROMKEY, J.: *RFC 1055 - A nonstandard for transmission of IP datagrams over serial lines: SLIP*. l988.

[SIEMENSDECT] SIEMENS: *Siemens DECT Engine MD32, Technical Description – Programmers Reference Manual*. 1999.

[SUNCLDC] SUN MICROSYSTEMS, Inc.: *Connected, Limited Device Configuration Specification Version 1.0 Java 2 Platform Micro Edition*. 2000.

[SUNRMI] SUN MICROSYSTEMS, Inc.: *Java™ Remote Method Invocation Specification Revision 1.7, Java™ 2 SDK, Standard Edition, v1.4 Beta 2*. 2001.

[UML2001] OBJECT MANAGEMENT GROUP, Inc. (OMG): *OMG Unified Modeling Language Specification Version 1.4*. 2001.

[Williams2001] WILLIAMS, S.; COOKLOV, T. V.; KRISTENSEN, P. H.; et al.: *Bluetooth Specification Version 1.1*. 2001.

Anhang

Abkürzungsverzeichnis

ACL	Aynchronous Connection-Less Link
ADSL	Asymmetric Digital Subscriber Line
AHRG	Architecture for HomeGate
API	Application Programming Interface
ARP	Address Resolution Protocol
ASI	Asynchron-Serielles Interface
ASIC	Application-specific Integrated Circuit
AWT	Abstract Window Toolkit (Java)
BAU	Bus Access Unit (EIB)
BAV	Basic Audio/Video device
BCU	Bus Coupling Unit
CAN	Controller Area Network
CCD	Charge Coupled Device
CGI	Common Gateway Interface
CLDC	Connected Limited Device Configuration
CMM	Comunication Media Manager (HAVi)
CORBA	Common Object Request Broker Architecture
CRC	Cyclic Redundancy Checksum
CSCW	Computer Supported Cooperative Work
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access / Collision Detection
CTS	Command and Transaction Set
DCM	Device Control Module (HAVi)
DDI	Data Driven Interaction
DECT	Digital Enhanced Cordless Telecommunication
DES	Data Encryption Standard
DGC	Distributed Garbage Collection
DLC	Data Link Control
DMA	Direct Memory Access
DPRS	DECT Packet Radio Service
DSL	Digital Subscriber Line
DSP	Digital Signal Processor
DTD	Document Type Declaration
DVB	Digital Video Broadcasting
EEPROM	Electrically Erasable Programmable Read-Only Memory
EHS	European Home System
EIB	Europäischer Installationsbus
EIS	EIB Interworking Standard

ETSI	European Telecommunications Standards Institute
FAV	Full Audio/Video device
FCM	Functional Component Module
FCP	Functional Control Protocol
FDMA	Frequency Division Multiple Access
FEC	Forward Error Correction Code (Bluetooth)
FP	Fixed Part (DECT)
FREL	Frame Relay
FTP	File Transfer Protocol
GOF	Glass Optical Fiber
GUID	Global Unique Identifier (IEEE 1394)
HAVi	Home Audio/Video interoperability
HKL	Heizung/Klima/Lüftung
HTTP	Hypertext Transfer Protocol
HVAC	Heating/Ventilating/Air-Conditioning
IAONA	Industrial Automation Open Networking Association
IAV	Immediate Audio/Video device
IDA	Interface for Distributed Automation
IDEA	International Data Encryption Algorithm
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IPSEC	IP Security Protocol
IrDA	Infrared Data Associaton
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
ISO/OSI	International Organization for Standardization/Open System Interconnection
IWF	Interworking Functions
IWU	Interworking Unit
J2ME	Java 2 Platform Micro Edition
Jar	Java Archive
JES	Java Embedded Server
Jini	Java Intelligent Network Infrastructure
JNI	Java Native Interface
JVM	Java Virtual Machine
KNX	Konnex
KSG	Key Stream Generator
LAN	Local Area Network

LAV	Legacy Audio/Video device
MAC	Medium Access Control
MACL	Medium Access Control Layer (DECT)
MHP	Multimedia Home Platform
MIDP	Mobile Information Device Profile
MPEG	Moving Picture Experts Group
NWK	Network
OBEX	Object Exchange Protocol
ODVA	Open DeviceNet Vendors Association
OHCI	Open Host Controller Interface
OMG	Object Management Group
OSGi	Open Services Gateway Initiative
PCMCIA	Personal Computer Memory Card International Association
PDA	Personal Digital Assistant
PEI	Periphal External Interface
PHY	Physical (DECT)
PIN	Personal Identification Number
POF	Polymer Optical Fiber
POP	Post Office Protocol
PP	Portable Part (DECT)
PROFIBUS	Professional Fieldbus
RAM	Random Access Only
RFC	Request for Commands
ROM	Read-Only Memory
RMI	Remote Method Invocation (Java)
SCL	Synchronous Connection-Oriented Link
SDD	Self Describing Device
SDSL	Symmetric Digital Subscriber Line
SEID	Software-Element ID(HAVi)
SIP	Session Initiation Protocol
SLIP	Serial Line Internet Protocol
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
TAM	Transport Adaptation Module
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TIME	Telekommunikation, Information, Medien und Entertainment
TLS	Transport Layer Security
TPUART	Twisted Pair Universal Asynchronous Receiver Transmitter (EIB)
TRUP	Transparent Unprotected Service
UDP	User Datagram Protocol

UI	User Interface
UML	Unified Modelling Language
UPnP	Universal Plug and Play
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTF	Unicode Transformation Format.
W3C	World Wide Web Consortium
WAN	Wide Area Network
WDM	Windows Driver Model
WEP	Wired Equivalent Privacy
WLAN	Wireless LAN
XDDML	Extensible Device Description Markup Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

Tabellenverzeichnis

Tabelle 3.1: Übersicht über einige Bussysteme 15
Tabelle 4.1: Dienste und die zugehörigen Datentransferraten 24
Tabelle 4.2: Zusätzliche Stereotypen zur Beschreibung der Dienste 36
Tabelle 5.1: Kategorisierung der Methoden innerhalb der Dienste 59
Tabelle 5.2: Element des RMI Wire Protocols 65
Tabelle 5.3: Elemente der RMI Rückgabedaten 66
Tabelle 5.4: Datenaufkommen auf Ethernet Ebene (Zeiten bezogen auf 100 MBit/s) 70
Tabelle 5.5: Vergleich der Anforderungen und der Umsetzung des HAVi-Stacks 81
Tabelle 5.6: Nachrichtenformat für einen ausgehenden Operationsaufruf 88
Tabelle 5.7: Datenaufkommen der Protokollebenen 93

Abbildungsverzeichnis

Abbildung 1.1: Bussysteme innerhalb eines Gebäudes 2
Abbildung 3.1: Paketformate in der Bustechnik 16
Abbildung 3.2: Strukturansicht des EIB 18
Abbildung 3.3: Konfiguration für den Frame Relay Service (verbindungslose und verbindungsorientierte Dienste) 20
Abbildung 4.1: Gateway mit unterschiedlichen Bussystemen 21
Abbildung 4.2: Logische Verknüpfung der Bussysteme 22
Abbildung 4.3: Hierarchie des mehrstufigen Systems 26
Abbildung 4.4: Definitionsiumfang von Middleware 28
Abbildung 4.5: Konzept zur Entkopplung von Systemen und Diensten 31
Abbildung 4.6: Austausch von Nachrichten 32
Abbildung 4.7: Dienstangebot und Buseigenschaften 34
Abbildung 4.8: Basisdienst 37
Abbildung 4.9: Videoservice 39
Abbildung 4.10: Dienst zur Abfrage von Audiodaten 40
Abbildung 4.11: Dienst zur Steuerung und Wartung der HKL-Technik 40
Abbildung 4.12: Service zur Verwaltung von dezentralen Informationen 41
Abbildung 4.13: Dienstkommunikation bei bekannten Schnittstellen 43
Abbildung 4.14: Komponenten der Dienstumgebung 44
Abbildung 4.15: Dienstverknüpfungsprozess im Sequenzdiagramm 45
Abbildung 4.16: Agent zur Vereinfachung der Dienstnutzung 47
Abbildung 4.17: Agent außerhalb des Systems 47
Abbildung 5.1: Gesamtsystem im Überblick 49
Abbildung 5.2 Stapelbausweise des Kameramoduls 51
Abbildung 5.3 Kamera Mikrosystemstapel 51
Abbildung 5.4: Verwendung der Set-top Box 52
Abbildung 5.5: Palm Vx 53
Abbildung 5.6: Kommunikationsstruktur von Jini 54
Abbildung 5.7: Eine mögliche Jini-Konfiguration 55
Abbildung 5.8: Jini-Protokoll für Client/Server/LookupServer 56
Abbildung 5.9: Dienstimplementierung bei Jini 56
Abbildung 5.10: Der Basisdienst in Jini 57
Abbildung 5.11: DTD für alle Dienste im System 58
Abbildung 5.12: Dateidienst 59
Abbildung 5.13: Ausschnitt aus der Schnittstellenbeschreibungsdatei 60
Abbildung 5.14: Aufzeichnungsdienst, vereinfacht 61
Abbildung 5.15: E-Mail-Dienst 61
Abbildung 5.16: Aufbau des Senders 63
Abbildung 5.17: Anwesenheitsdienst 63
Abbildung 5.18: Anzeige der verfügbaren Dienste innerhalb des Systems 64

Abbildung 5.19: Auszug RMI Client/Server-Kommunikation 68
Abbildung 5.20: Nachrichtensequenz bei einer RMI Anfrage 69
Abbildung 5.21: HAVi-Architektur 73
Abbildung 5.22: Einbettung von HAVi in das Systemkonzept 74
Abbildung 5.23: HAVi-Stack unter Linux 76
Abbildung 5.24: Benutzerinteraktion des BAV/EIB-Gateways 79
Abbildung 5.25: Nachrichtenablauf zwischen DDI Controller und DDI Target 80
Abbildung 5.26: Paketeinteilung des Java HAVi-Stacks 82
Abbildung 5.27: Umsetzung des Datenmodells für den EIB 84
Abbildung 5.28: Systemübergang zwischen zwei EIB Installation durch IEEE 1394 85
Abbildung 5.29: Einordnung des EIB-Datenmodells in HAVi 86
Abbildung 5.30: Verbindung der Dienstebenen Jini und HAVi 87
Abbildung 5.31: Protokollaufzeichnung des DDI-Protokolls 89
Abbildung 5.32: Verbindung zwischen den DECT-Modems 91
Abbildung 5.33: Kommunikation zwischen PDA und Proxy 92
Abbildung 5.34: Darstellung der Steuerung auf dem Palm V 93
Abbildung 5.35: Struktur des Kamera-Proxys 96
Abbildung 5.36: Nachrichtenabfolge zwischen OBEX-Client und -Server 97
Abbildung 5.37: EIB Geräte als Jini Dienste 99
Abbildung 5.38: Darstellungen einiger EIB-Dienste 100
Abbildung 5.39: Umsetzung des Basisdienstes zur graphischen Anzeige 100
Abbildung 5.40: *Reflection* in Java 102
Abbildung 5.41: Verknüpfung von Dienstanbieter und Dienstnehmer durch Service-Agenten 103
Abbildung 5.42: Konfigurationen von Client/Server Verbindungen 106
Abbildung 5.43: HAVi DDI Elemente des FAV 108
Abbildung 5.44: Steuerung von EIB Geräten über DDI Buttons 108
Abbildung 5.45: Abbildung der Anzeige auf einem Web Pad 109
Abbildung 6.1: Aufgaben bei der Konfiguration und Instandhaltung des Systems 115
Abbildung 6.2: Ausschnitt aus einer importierten EIB Konfigurationsdatei 118
Abbildung 6.3: Ansicht des Jini Editors zur Festlegung der Eigenschaften des EIB 119
Abbildung 6.4: Konfigurationssoftware im Modus HAVi/Fernzugriff/lokal 121
Abbildung 6.5: Konfiguration des PDA – hier der Palm Vx 122