

Prüfer: Prof. Dr. Kurt Rothermel
Betreuer: Dipl.-Inf. Alexander Leonhardi
Dr. rer. nat. Joachim Baumann
Begonnen: Juli 1999
Beendet: Januar 2000
CCS-Nummer: C.2.4, D.4.8

Studienarbeit Nr. 1778

Adhoc-Verwendung einer elektronischen Tafel unter Verwendung der Jini-Technologie

Khanh-Loan Nguyen-Salamanis

Universität Stuttgart
Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR)
Breitwiesenstr. 20-22
70565 Stuttgart

Inhaltsverzeichnis

1 Einführung	1
1.1 Motivation	1
1.2 Aufgabenstellung.....	2
1.3 Überblick	3
2 Randbedingungen	4
2.1 Jini	4
2.1.1 Was ist Jini?	4
2.1.2 Jini-Konzept	6
2.1.3 Technischer Hintergrund	18
2.1.4 Anwendungsbeispiele	19
2.2 Smartboard	21
3 Entwurf	21
3.1 Entwurfsentscheidungen.....	21
3.2 Alternativen des Entwurfs	23
3.3 Design: Systemkomponenten	24
3.4 Architektur: Abläufe und Klassenstrukturen.....	27
3.4.1 Abläufe.....	27
3.4.2 Klassenstrukturen.....	30
4 Implementierung	32
4.1 Anmelden des Dienstes	32
4.2 Anfrage des Dienstes	34
4.3 Kommunikation zwischen Server und Klient.....	36
4.3.1 Kommunikation zwischen ET-Server und ET-Proxy	38
4.3.2 Übertragen der Präsentationsdaten vom ET-Proxy an den ET-Server	38
4.3.3 Steuern der PPT-Präsentation vom Laptop aus	38
4.3.4 Übertragen der Notizen von der elektronischen Tafel.....	39
4.4 Benutzungsoberfläche	39
4.4.1 Ausgaben in der Kommandozeile.....	39
4.4.2 Auswahl der elektronischenTafel	40
4.4.3 Das Fenster für den Präsentationsstart.....	42
4.4.4 Das Fenster für die Anzeige der Notizen von der elektronischen Tafel	43

5 Messungen	44
5.1 Messwerte.....	45
5.1.1 Dauer der Dienstanmeldung	45
5.1.2 Dauer der Dienstanfrage	46
5.1.3 Dauer des Transports der PPT-Dateien.....	46
5.2 Ergebnisanalyse.....	47
5.2.1 Analyse der Messungen der Dienstanmeldung.....	47
5.2.2 Analyse der Messungen der Dienstanfrage.....	48
5.2.3 Analyse der Messungen des Transports der PPT-Dateien.....	48
6 Zusammenfassung und Ausblick	49
6.1 Zusammenfassung	49
6.2 Ausblick.....	50
6.2 Anhang	51
A1. Die Befehlzeilen	51
A2. Ausgewählter Pseudocode	52

Abbildungsverzeichnis

Abbildung 2.1: Ablaufschema der Protokolle: Discovery, Join, Lookup	6
Abbildung 2.2: Der Discovery-Vorgang	7
Abbildung 2.3: Das Unicast-Discovery-Protokoll	10
Abbildung 2.4: Das Multicast-Request-Protokoll	11
Abbildung 2.5: Das Multicast-Announcement-Protokoll	12
Abbildung 2.6: Der Join-Vorgang	13
Abbildung 2.7: Der Lookup-Vorgang	15
Abbildung 3.1: Die Systemübersicht	24
Abbildung 3.2: Die Komponenten des ET-Servers	26
Abbildung 3.3: Die Methoden der Schnittstelle ElectronicBoardInterface	26
Abbildung 3.4: Die zeitlichen Abläufe	29
Abbildung 3.5: Die Klassenstrukturen	30
Abbildung 4.1: Die Methoden der Klassen	37
Abbildung 4.2: Die Ausgaben in der Kommandozeile	40
Abbildung 4.3: Die Lokationliste der elektronische Tafeln	41
Abbildung 4.4: Die Anzeige der Lokation der elektronischen Tafel	41
Abbildung 4.5: Der Vorgang des Präsentationsstarts	42
Abbildung 4.6: Das Steuerungsmenü	43
Abbildung 4.7: Das Fenster für die Anzeige der Notizen	43
Abbildung 5.1: Konfiguration für die Messungen	44
Abbildung 5.2: Die Zeitgraphik der Dienstanmeldung	45
Abbildung 5.3: Die Zeitgraphik der Dienstanfrage	46
Abbildung 5.4: Die Zeitgraphik des Dateitransports	47

1 Einführung

1.1 Motivation

Heutzutage gibt es immer mehr kommunikationsfähige Geräte (z.B. Drucker, Scanner usw.), die man miteinander kombinieren und verwenden will. Bisher musste man diese Geräte explizit miteinander verbinden durch Kabelverbindung, Treiber usw. Jini ermöglicht es, die Geräte sich gegenseitig zu finden und miteinander zu kommunizieren. D.h. Jini soll die Zusammenarbeit von beliebigen Geräten (wie Computer, Elektrogeräten, Unterhaltungselektronik) oder Software (wie Applikationen, Hilfsprogramme) ermöglichen und erleichtern, so dass sie sich ohne vorherige Konfiguration oder vorherige Kenntnis voneinander miteinander vernetzen. Edwards beschreibt die Rolle von Jini:

"Jini ermöglicht den Wandel von einer Welt, in der Geräte miteinander verbunden sind (d.h. in der Computer miteinander kommunizieren können), in eine Welt der verteilten Systeme, in der Gruppen vernetzter Geräte zusammenarbeiten" [Edwards 99].

Ein Beispiel für eine Anwendung von Jini wäre: Ein Vortragender will auf einer elektronischen Tafel, die gesteuert von einem Präsentationsserver neben klassischen Präsentationsmöglichkeiten auch weitergehende Interaktionsmöglichkeiten bietet, einen Vortrag oder eine Vorlesung halten. Er hat dazu seine Präsentation auf einen Laptop gespeichert. Die folgende Beispiele zeigen, welche Aktionen unter Verwendung von Jini oder ohne Jini durchgeführt werden müssen.

Szenario ohne Jini:

- Ohne Jini muss man die zu zeigende Präsentationsdaten mühsam von Hand auf den Präsentationscomputer laden. Man kann entweder FTP (File Transfer Protocol) oder Disketten für die Datenübertragung vom Laptop zum Server verwenden. Beide Methoden sind aufwendig. Mittels FTP benötigt man zuerst den Namen des Servers und dann eine Reihe von Befehlen für FTP, um die Präsentationsdaten vom Laptop an den Server zu übertragen. Mittels Disketten benötigt man einen Laptop mit Diskettenlaufwerk und muss dann die Präsentationsdaten erst vom Laptop auf Disketten und dann von Disketten auf den Präsentationsserver kopieren.
- Ohne Jini kann man die Präsentation auf der elektronischen Tafel nicht einfach vom Laptop aus steuern. Für die direkte Steuerung vom Laptop aus benötigt man eine her-

stellerspezifische Steuerungssoftware, d.h. man muss einen Treiber herunterladen und installieren. Sonst ist es auch nicht möglich.

- Ohne Jini kann man die Kommentare und Notizen auf der elektronische Tafel nicht einfach auf den Laptop übertragen, dort anzeigen.

Szenario mit Jini:

- Man kann dann ohne Mühe und mit minimalem Aufwand die Präsentationsdaten auf den Präsentationscomputer (und damit die elektronische Tafel) laden. Jini baut eine Verbindung zwischen Laptop und Präsentationsserver auf, sobald der Vortragende seinen Laptop in dem Raum einschaltet.
- Mit Jini kann man die Präsentation auf der elektronische Tafel vom Laptop aus direkt steuern. Jini sorgt für die Verbindung zwischen dem Laptop und dem Server und damit der elektronischen Tafel.
- Mit Jini kann man die Kommentare und Notizen auf der elektronische Tafel einfach auf den Laptop übertragen, dort anzeigen.

Wegen der Nützlichkeit von Jini soll untersucht werden, wie Jini funktioniert und ob Jini einfach eingesetzt werden kann. Das sind die Ziele der Arbeit, die folgend beschrieben wird.

1.2 Aufgabenstellung

Für Präsentationszwecke werden häufig elektronische Tafeln verwendet, die, von einem Präsentationscomputer gesteuert, Interaktivität mit klassischer Präsentationstechnik verbinden.

Die Nachteile hierbei sind, dass

- die zu zeigende Präsentationsdaten mühsam von Hand auf den Präsentationscomputer geladen werden müssen,
- die Verbindung zwischen den PCs (Personal Computers) konfiguriert werden müssen.

Ziel der Arbeit ist die Schaffung eines Jini-Software-Pakets, das einen Dienst bzw. einen Server auf dem Präsentationscomputer und einen Klienten auf dem Klient-Gerät implementiert. Dieses Klient-Gerät (z.B. mobile Rechner, tragbare Rechner oder PDA (Personal Digital Assistant) usw.) muss java-fähig sein. Dieser Dienst ermöglicht es:

- Informationen von einem Laptop auf den Präsentationscomputer (und damit die elektronische Tafel) bzw. die Darstellung der elektronischen Tafel auf dem Laptop einfach zu übertragen.
- Die Präsentationsdaten auf der elektronischen Tafel vom Laptop aus zu steuern.

Für diesen Dienst soll ein Design erstellt und in Java implementiert werden. Die Effizienz bzw. Eignung der verwendeten Technologien soll durch geeignete Messungen bestimmt werden.

Das Programm soll auf Workstations vom Typ Sun unter Solaris und auf PCs unter Windows 98/NT laufen können. Die Rechner kommunizieren mittels eines Funknetzes oder eines Festnetzes miteinander.

1.3 Überblick

Die Ausarbeitung ist wie folgt strukturiert. In Kapitel 1 wurde die Aufgabenstellung eingeführt. Kapitel 2 beschreibt die Randbedingungen, wobei Jini vorgestellt und beschrieben wird, wie Jini funktioniert. Zusätzlich wird die elektronische Tafel und ihre Funktionalität vorgestellt. Kapitel 3 beschreibt den Entwurf, die Entwurfsentscheidungen und -alternativen. Anschliessend werden das Design, die Klassenstrukturen und die Abläufe erläutert. Kapitel 4 erklärt die Implementierung, wie Anmeldung und Anfrage eines Dienstes implementiert werden und wie der Server und der Klient über Jini miteinander kommunizieren. Dieses Kapitel beschreibt auch die Benutzungsoberfläche. In Kapitel 5 werden die Messungen beschrieben und die Messergebnisse analysiert. Schliesslich wird in Kapitel 6 die Arbeit zusammengefasst.

2 Randbedingungen

2.1 Jini

Jini basiert auf Java und ist eine neue Technologie von Sun Microsystems für die einfache Vernetzung von Geräten und Diensten. Bei Jini handelt es sich um ein dynamisches verteiltes System. Dynamisch, weil permanent neue Dienste hinzugefügt oder entfernt werden können. Aus der Sicht von Jini kann ein **Dienst** sowohl Hardware (Drucker, Scanner, Fax-Gerät, PDA, Digitalkamera, ...) als auch Software (Applikationen und Hilfsprogramme) sein. Jini soll Dienste einfach und unkompliziert miteinander verbinden, kommunizieren und zusammenarbeiten lassen. Damit entsteht ein Gesamtsystem auf einzelnen Diensten.

Um einen Überblick über Jini zu geben, werden in den folgenden Abschnitten dieses Kapitels die Ziele von Jini, das Jini-Konzept, sowie der technische Hintergrund und Jini-Anwendungsbeispiele vorgestellt.

2.1.1 Was ist Jini?

Jini ist ein verteiltes System, das auf der Idee der dynamischen Verbindung und Zusammenarbeit zwischen Klienten und Diensten basiert. Jini bietet den Diensten eine Infrastruktur für das gegenseitige Auffinden und Kommunizieren innerhalb eines Netzwerks. Jini bietet allen Benutzer innerhalb von beliebigen Netzen Einfachheit, Schnelligkeit und Komfort bei der Konfiguration. **Einfachheit und Schnelligkeit**, weil Jini das Plug-and-Play von Diensten innerhalb eines Jini-Netzwerks unterstützt. Dienste werden an einem Jini-Netzwerk angeschlossen und sind gleich verfügbar. **Komfort**, weil Dienste an einem Jini-Netzwerk angeschlossen und benutzt werden können, ohne IP-Adressen zu konfigurieren, Gateways einzurichten, Treiber zu installieren (bzw. zu deinstallieren) usw.

Jini ist in Java realisiert. Deshalb erbt Jini von Java die Eigenschaft der Plattformunabhängigkeit. Jini nutzt neue Funktionen von RMI (Remote Method Invocation), die in Java 2 (bzw. JDK 1.2) enthalten sind, beispielsweise die RMI-Aktivierungsstruktur. Es kann daher nicht mit älteren Java Versionen eingesetzt werden.

In [Bader&Huber2000] steht die Abkürzung Jini für "Java Intelligent Network Infrastructure". Diese Definition wird in dieser Studienarbeit verwendet.

Ziele von Jini

In diesem Abschnitt werden die Ziele von Jini betrachtet.

- **Einfachheit:** Jini nutzt die grundlegenden Konzepte von Java und fügt eine dünne Schicht hinzu, um die Zusammenarbeit von Diensten zu erleichtern. Folgendes ist wichtig: Jini ermöglicht die Verbindung von Diensten und nimmt keinen Einfluss darauf, welche Dienste angeboten werden oder wie sie sich verhalten oder was sie machen. Jini-Dienste müssen nicht unbedingt in Java, sondern können auch in andere Programmiersprachen geschrieben werden. Deswegen ist es erforderlich, dass sich irgendwo innerhalb des Netzwerks etwas Code befindet, der in Java geschrieben wurde und an den Mechanismen teilnehmen kann, die Jini für das Auffinden von Jini-Diensten verwendet.
- **Zuverlässigkeit:** Jini unterstützt die Kommunikation zwischen Diensten und den Klienten dieser Dienste. Dienste können auf einfache Weise angeboten werden, ohne dass irgendeine Art statischer Konfiguration oder Administration erforderlich ist. Über die Änderungen in der Jini-Dienste-Gemeinschaft werden andere Dienste in dieser Gemeinschaft oder Klienten automatisch benachrichtigt. Auf diese Weise unterstützt Jini, ohne Bedarf an zusätzlicher Verwaltung, die spontane Vernetzung von Diensten. Bei Jini ist der Begriff "Gemeinschaft" wie folgend definiert: **Gemeinschaften** sind Gruppen von Jini-Diensten, die sich zu zusammenarbeitenden Einheiten zusammenschliessen. Ausserdem unterstützt Jini redundante Infrastrukturen, um die Wahrscheinlichkeit von Dienstausfällen zu verringern. Jini geht davon aus, dass Netzwerke nicht verlustfrei sind und Software nicht immer fehlerfrei arbeitet. Deshalb bietet Jini ein Konzept für die Selbstheilung des Systems. Das bedeutet, dass Jini-Dienste mit Störungen innerhalb des Netzwerks umgehen können, indem sie sich auf vorhersehbare Weise verhalten und sich nach einiger Zeit wiederherstellen.
- **Skalierbarkeit:** Alle Dienste innerhalb einer Gemeinschaft kennen einander und sind in der Lage, sich gegenseitig zu nutzen. Skalierbarkeit wird von Jini durch Kooperation gewährleistet. Kooperation ist die Fähigkeit von Jini-Gemeinschaften, miteinander verknüpft bzw. zu grösseren Gruppen zusammengeschlossen zu werden. Jini unterstützt den Zugriff auf Dienste anderer Gemeinschaften, indem diese kooperieren können.

2.1.2 Jini-Konzept

Das Jini-Konzept basiert auf fünf Elementen: Discovery, Lookup, Leasing, verteilte Ereignisse und Transaktionen. Für diese Studienarbeit sind Discovery und Lookup wichtig. Daher werden sie im folgenden Abschnitt näher erläutert. Die drei andere Elemente werden nur kurz erklärt.

Das Herz des Jini Systems sind die drei Protokolle Discovery, Join und Lookup. Die Protokolle "Discovery/Join" finden statt, wenn ein Dienst hinzugefügt wird. Bild 2.1 zeigt den Ablauf der Protokolle.

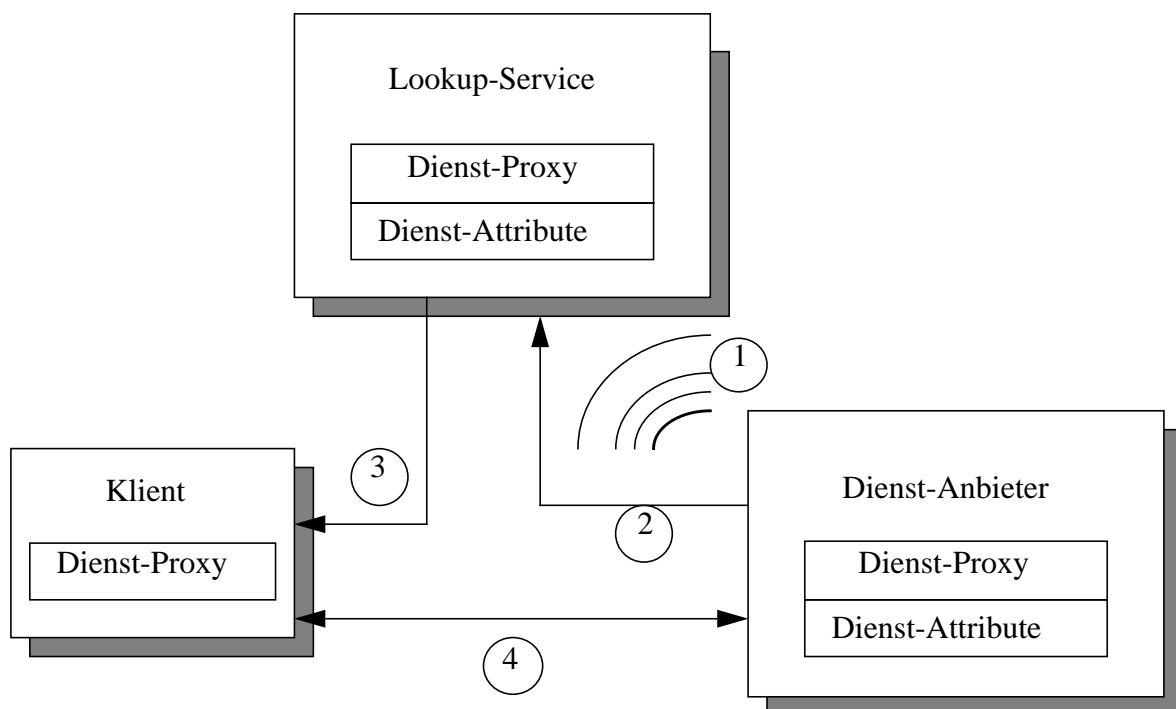


Abbildung 2.1: Ablaufschema der Protokolle: Discovery, Join, Lookup

Der **Lookup-Service** ist eine zentrale Registration von Diensten, die sich bei ihm angemeldet haben und deren Beschreibungen von ihm gespeichert und verwaltet werden. Jeder Dienst enthält einen **Dienst-Proxy**, der Methoden und einige **Dienst-Attribute** umfasst, die den Dienst beschreiben. Der **Klient** (Benutzer oder Applikation) kann diese Methoden aufrufen und ausführen.

- **Discovery** findet statt, wenn ein Dienst nach einem Lookup-Service sucht und sich bei ihm registrieren lassen will (Schritt 1).
- **Join** findet statt, wenn ein Dienst einen Lookup-Service gefunden hat und sich bei ihm anmeldet (Schritt 2).
- **Lookup** findet statt, wenn ein Klient einen Dienst nachfragt (Schritt 3). Danach kann

der Klient direkt mit dem Dienst-Anbieter kommunizieren (Schritt 4).

2.1.2.1 Discovery/Join

Im folgenden Abschnitt werden die Discovery/Join-Protokolle näher erläutert.

Discovery

Discovery wird der für das Auffinden von und das Anbinden an Jini-Gemeinschaften (Jini communities) innerhalb des Netzwerks verwendete Prozess bezeichnet. Aus Discovery resultiert die Fähigkeit des Systems zum spontanen Aufbau von Gemeinschaften.

Wie schon im Kapitel 2 erwähnt ist eine Jini-Gemeinschaft eine Gruppe von Diensten, die - für einander und für Anwendungen, von denen sie genutzt werden - im Netzwerk verfügbar sind. Zwischen Gemeinschaften und Lookup-Services besteht nicht grundsätzlich ein Eins-zu-Eins-Verhältnis. Innerhalb eines Netzwerks kann ein Lookup-Service mehreren Gemeinschaften dienen; eine Gemeinschaft kann einen oder mehreren Lookup-Server umfassen. Lookup-Services werden explizit von Administratoren gestartet. Dies ist der einzige Jini-Dienst, der für das Aufbauen von Jini-Gemeinschaften erforderlich ist.

Der Discovery-Prozess ist so gestaltet, dass ein Dienst beim Start standardmässig und ohne dass er konkret weiss, welche Gemeinschaften vorhanden sind, die Lookup-Services im selben Subnetz findet und sich bei ihnen registrieren lässt (Bild 2.2). Normalerweise gibt es in einem Subnetz einen oder mehrere Lookup-Services, die von Diensten im selben Netzwerk standardmässig gemeinsam genutzt werden.

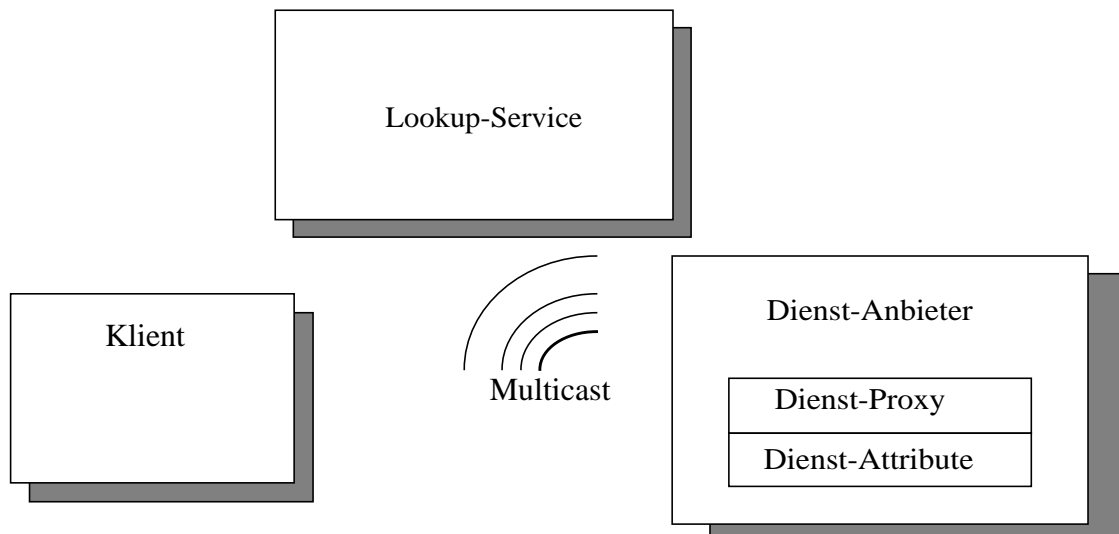


Abbildung 2.2: Der Discovery-Vorgang

Bild 2.2 zeigt den Discovery-Vorgang: der Dienst-Anbieter sucht nach einen Lookup-Service.

Anforderungen an Discovery

Es gibt einige wichtige Anforderungen [Edwards 99] an den Discovery-Mechanismus von Jini, die in die einzelnen folgenden Unter-Protokolle integriert sind:

- Discovery sollte ausreichend flexibel sein, um verschiedene Topologien von Gemeinschaften zu unterstützen.
- Discovery sollte die einfache Wiederherstellung nach Netzwerkpartitionierungen und Rechnerausfällen fördern.
- Discovery sollte ausreichend leichtgewichtig sein, um problemlos auf Systemen mit sehr begrenzter Rechenleistung zu laufen.

Die Discovery-Protokolle

Es gibt nicht nur ein einziges Discovery-Protokoll, sondern drei verschiedene Protokolle für unterschiedliche Situationen.

- Das **Multicast-Request-Protokoll** wird bei der Aktivierung einer Anwendung bzw. eines Dienstes verwendet, um benachbarte aktive Lookup-Services zu ermitteln.
- Das **Multicast-Announcement-Protokoll** wird von Lookup-Services zur Mitteilung des eigenen Vorhandenseins benutzt. Sobald ein neuer Lookup-Service innerhalb einer bestehenden Gemeinschaft gestartet wird, wird jeder interessierte Beteiligte über das Multicast-Announcement-Protokoll darüber informiert.
- Das **Unicast-Discovery-Protokoll** wird verwendet, wenn einer Anwendung bzw. einem Dienst der zu benutzende Lookup-Service bereits bekannt ist. Mit dem Unicast-Discovery-Protokoll wird direkt mit einem Lookup-Service kommuniziert, der nicht unbedingt zum lokalen Netzwerk gehören muss, sofern der Name dieses Lookup-Service bekannt ist. Die Benennung von Lookup-Services erfolgt in URL-Syntax (Uniform Resource Locator), wobei das Protokoll als *jini* spezifiziert wird. (`jini://posaune.informatik.uni-stuttgart.de:4160/` spezifiziert beispielsweise den auf dem Host "posaune.informatik.uni-stuttgart.de" ausgeführten Lookup-Service auf dem Standardport 4160).

Die von Jini verwendete Form von Multicast basiert auf dem Protokoll UDP/IP. UDP (User Datagram Protocol) bietet einen verbindungslosen, nicht zuverlässigen Transport. Das bedeu-

tet, mit UDP ist nicht garantiert, dass ein gesendetes Paket tatsächlich ankommt und die Reihenfolge der Pakete eingehalten wird.

Die Multicast-Protokolle verwenden IP (Internet Protocol)-Multicast. Multicast ist eine Methode, eine Nachricht zu senden, die von einer beliebigen Anzahl von Parteien empfangen werden kann. Bei Multicast wird eine Reihe spezieller IP-Adressen als Multicast-Gruppen benutzt. Interessierte Parteien können an einer Gruppe teilnehmen, indem sie auf Nachrichten prüfen, die an diese Adresse gesendet werden. Jede an diese Adresse gesendete Nachricht wird von allen Parteien empfangen, welche die Adresse abhören. Multicast setzt die Netzwerkressourcen effizient ein, weil eine Nachricht, die an mehrere Empfänger geht, soweit möglich als eine einzige Nachricht übermittelt wird. Wenn sich die Empfänger in zwei unterschiedlichen Netzwerken befinden, müssen die Multicast-Router die Nachricht an die Empfänger im zweiten Netzwerk weiter leiten.

Für die Erklärung der Protokolle werden die folgenden zwei Begriffe definiert: Ein **Diensterbringer** ist ein Dienst, der sich bei einem Lookup-Service anmelden will. Ein **Dienstnehmer** ist ein Klient oder ein anderer Dienst, der nach einem Dienst fragt. D.h ein Dienst kann sowohl ein Diensterbringer als auch ein Dienstnehmer sein, wenn der Dienst sich anmelden und gleichzeitig nach einem anderen Dienst fragen will.

Das Unicast-Discovery-Protokoll

Unicast erfordert explizite Informationen über die Lokation des Lookup-Service. Dazu gehören der Name des Host Rechners, auf dem der Lookup-Service läuft, und der Port, an dem der Lookup-Service auf Anforderungen wartet. Dieses Protokoll umfasst eine Anforderung an den Lookup-Service und eine Antwort an den Diensterbringer/Dienstnehmer. Die Antwort enthält immer den Dienstverwaltungs-Proxy für den Lookup-Service (Bild 2.3). Ein **Dienstverwaltungs-Proxy** ist ein Java-Objekt des Lookup-Service, und wird einem Diensterbringer/Dienstnehmer übergeben, um mit dem Lookup-Service zu kommunizieren.

Sobald der Lookup-Service eine TCP/Unicast-Anfragennachricht erhält, antwortet er durch direkte Verbindungsaufnahme mit dem anfordernden Diensterbringer/Dienstnehmer und durch Senden einer TCP/Unicast-Nachricht. Diese enthält den Dienstverwaltungs-Proxy, den der Diensterbringer/Dienstnehmer verwenden kann, um Kontakt mit dem Lookup-Service aufzunehmen.

Das Unicast-Discovery-Protokoll wird in Situationen verwendet, in denen sich Lookup-Services und Dienste nicht unbedingt in demselben lokalen Netzwerk sind. Der Lookup-Service

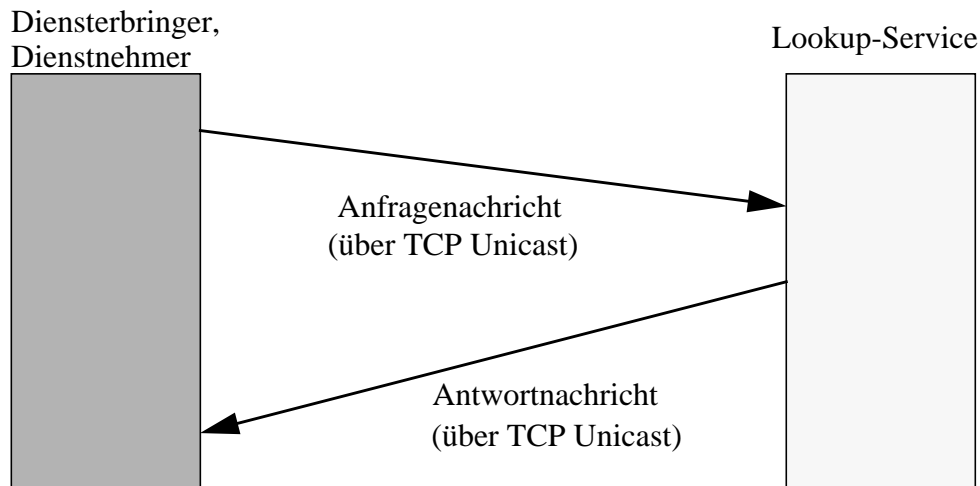


Abbildung 2.3: Das Unicast-Discovery-Protokoll

kann in einem anderen Subnet sein.

Das Unicast-Discovery-Protokoll wird als Teil der beiden Multicast-Protokollen verwendet; dies wird im folgenden Abschnitt erläutert.

Das Multicast-Request-Protokoll

Bei dem Multicast-Request-Protokoll wird eine Anfragennachricht über UDP-Multicast von einem Dienstbringer an allen nahen Lookup-Services, die in Betrieb sind und vom Dienstbringer gefunden wurden, gesendet. Die Anfragennachricht lässt sich in eine einzige UDP-Multicast-Datagramm unterbringen. Das bedeutet, dass die Wahrscheinlichkeit von Paketverlust durch die Einfachheit der Nachricht verringert wird.

Sobald ein Lookup-Service eine Multicast-Anfragennachricht erhält, antwortet er durch direkte Verbindungsaufnahme mit dem anfordernden Dienstbringer und durch Senden einer TCP/Unicast-Nachricht (Bild 2.4). Die Antwortnachricht entspricht der im Unicast-Discovery-Protokoll verwendeten Antwort (Siehe Bild 2.3).

Der Discovery-Prozess ist so gestaltet, dass ein Dienstbringer beim Start die Lookup-Services in seiner "Nähe" findet und sich bei ihnen registrieren lässt, ohne Wissen über das Vorhandensein von Gemeinschaften. Der "Nähe" Begriff des Discovery-Protokolls basiert auf der Netzwerktopologie. Was Nähe für ein konkretes Netzwerk bedeutet, ist abhängig davon, wie es vom Administrator konfiguriert wurde. Bei UDP-Multicast wird die Reichweite der Nachricht durch Angabe eines Wertes für den IP-Parameter TTL (Time-to-Live) festgelegt. Die TTL einer Nachricht legt fest, wie viele Subnetzgrenzen sie überquert. Ist eine TTL 1, erreicht die Nachricht

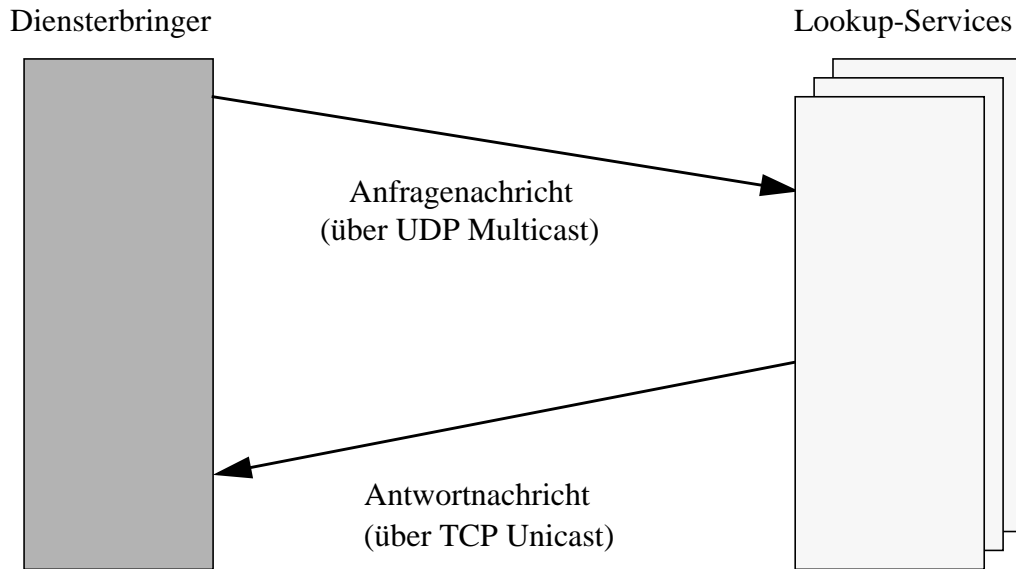


Abbildung 2.4: Das Multicast-Request-Protokoll

Empfänger nur im lokalen Netzwerk; höhere TTLs können dazu führen, dass sie das gesamte Internet durchläuft.

Der Dienstbringer sendet in bestimmten Abständen Multicast-Anfragennachrichten. Diese gehen in eine bekannte Adresse, die Jini-Lookup-Services abhören. Der Abstand zwischen den Anforderungen ist fünf Sekunden. Nach einer gewissen Zeit bricht er die Multicast-Anforderungen ab, weil er annimmt, dass er alle aktiven Lookup-Services gefunden hat. Die Dauer dieses Abschnitts wird von der Jini Spezifikation empfohlen: sieben Nachrichten. Nachdem der Dienstbringer das Senden der Multicast-Anfragennachrichten abgebrochen hat, warten die meisten Dienstbringer auf Multicast-Bekanntmachungsnachrichten, um Lookup-Services zu entdecken, die später gestartet werden oder die Verbindung zum Netzwerk wieder aufbauen. Das Multicast-Announcement-Protokoll wird im folgenden Abschnitt erläutert.

Das Multicast-Announcement-Protokoll

Das Multicast-Announcement-Protokoll funktioniert genau so wie das Multicast-Request-Protokoll, wird jedoch von den Lookup-Services initiiert. Lookup-Services geben ihre Anwesenheit in gewissen Abständen mit Hilfe des Multicast-Announcement-Protokoll bekannt. Wenn ein Dienstbringer/Dienstnehmer eine Bekanntmachung eines Lookup-Service empfängt, von dem er noch nicht gehört hat, kann er Kontakt zu diesem aufnehmen. Er richtet einen TCP-Socket ein und sendet eine Unicast-Anfragennachricht an den Port, an dem der Lookup-Service auf Anforderungen wartet. Nachdem der Lookup-Service die Anfragennachricht vom Dienstbringer/Dienstnehmer empfangen hat, sendet er eine Antwort an den Dienstbringer/Dienst-

nehmer, die den Dienstverwaltungs-Proxy des Lookup-Service enthält (Bild 2.5).

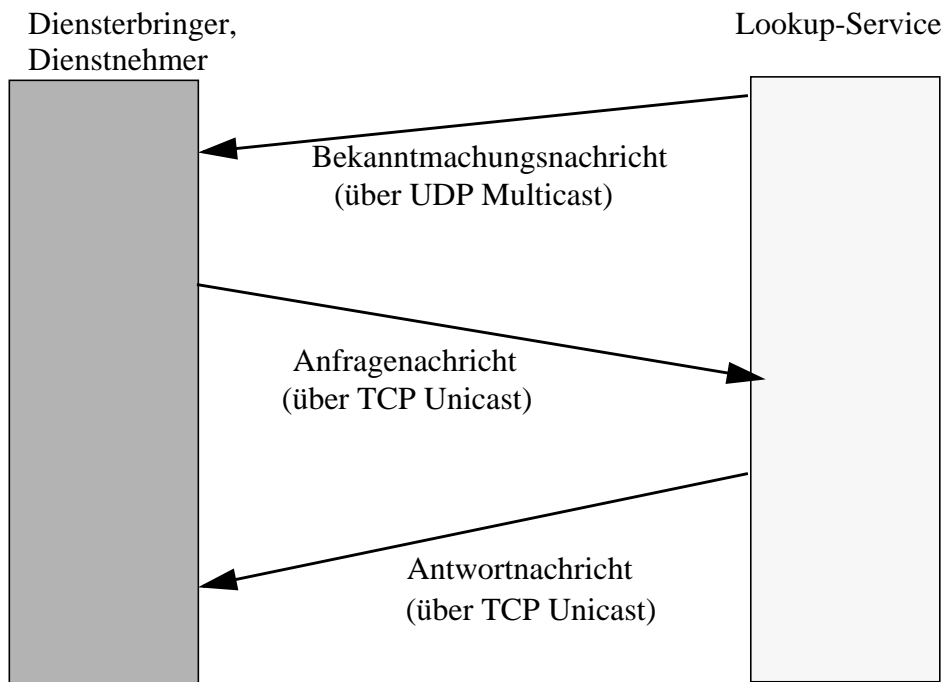


Abbildung 2.5: Das Multicast-Announcement-Protokoll

Wie das Multicast-Request-Protokoll wird das Multicast-Announcement-Protokoll in Situationen verwendet, in denen Lookup-Services und Jini-Dienste in einem lokalen Netzwerk einander finden müssen. Im Unterschied zum Multicast-Request-Protokoll, das nur während der Startphase eines Dienstes eingesetzt wird, wird das Multicast-Announcement-Protokoll während der gesamten Lebensdauer eines Lookup-Service verwendet. Als zeitlicher Abstände zwischen Bekanntmachungsnachrichten werden 120 Sekunden von der Jini-Spezifikation empfohlen.

Wenn Unicast-Discovery als Teil des Multicast-Announcement-Protokoll verwendet wird, dann funktioniert das Protokoll genauso wie bei selbständigem Einsatz (stand alone case). Durch die Bekanntmachungsnachricht können Dienstbringer/Dienstnehmer die Lokation des Lookup-Service erfahren. Dann wird das Unicast-Discovery-Protokoll verwendet (siehe die Anfrage- und Antwortnachricht Pfeile im Bild 2.5 und die Pfeile im Bild 2.3).

Join-Protokoll

Das Join-Protokoll wird verwendet, nachdem ein neuer Dienst mittels des Discovery-Protokolls einen Lookup-Service gefunden hat und sich bei ihm anmelden will. Ein Dienst enthält einen Dienst-Proxy, der Methoden und Attribute umfasst, die den Dienst beschreiben. Dieses Objekt

und die Attribute werden an den Lookup-Service übertragen (siehe Bild 2.6).

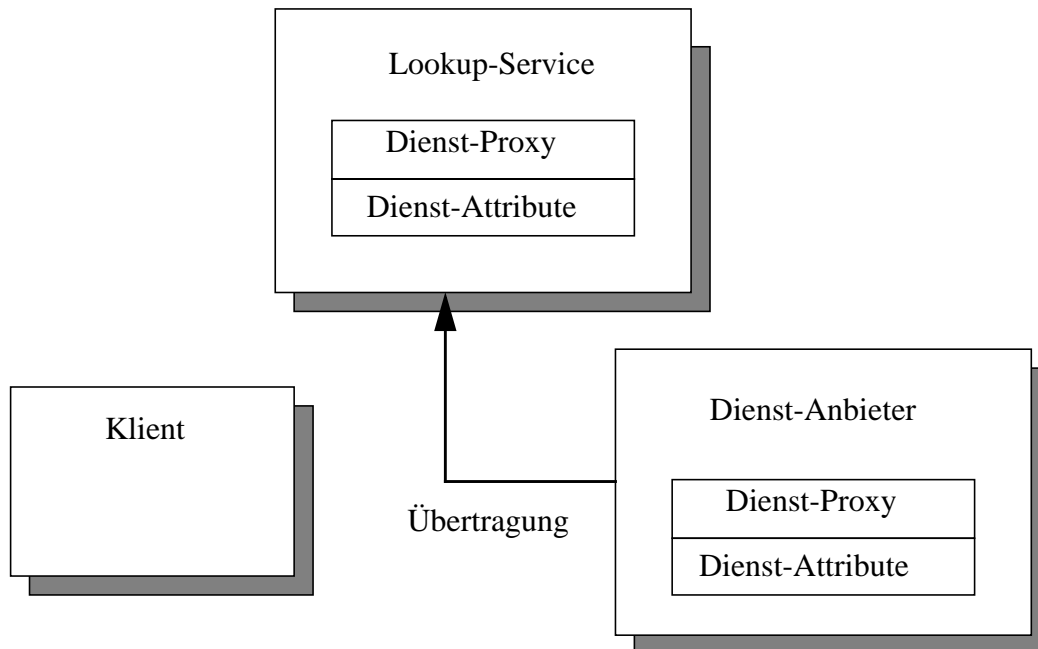


Abbildung 2.6: Der Join-Vorgang

Das Join-Protokoll besteht aus einer Reihe von Konventionen. Diese definieren das Zusammenspiel zwischen Diensten und Discovery sowie Lookup. Das Join-Protokoll legt fest, wie Dienste Discovery einsetzen, was geschieht, wenn sie die Verbindung zu einem Lookup-Service verlieren, welchen Lookup-Services sie beizutreten versuchen usw. Die besonderen Anforderungen des Join-Protokolls sind komplex. Jini bietet einige Klassen zum Automatisieren dieses Prozesses.

Mehrere Informationen sind erforderlich, um einen Dienst zu charakterisieren:

- eine Service-ID
- eine Menge von Attribute, die den Dienst beschreiben. Z.B. sind die Attribute eines Druckers die Lokation und der Name des Druckers, ob es ein farbiger oder schwarzweiss Drucker ist oder dessen Auflösung.
- eine Menge von Dienst-Gruppen, an welchen der Dienst teilnehmen will
- eine Menge an Lookup-Services, bei denen sich der Dienst registriert hat

Die ersten beiden Informationen werden auf seiten des Lookup-Service gespeichert und die letzten beiden auf seiten des Dienstes. Der Dienst muss eine eindeutige Service-ID für alle Registrierungen verwenden. Diese Service-ID wird vom ersten Lookup-Service zugewiesen, mit

dem er Verbindung aufnimmt. Der Dienst muss sich diese Service-ID merken und sie zukünftig bei allen Lookup-Services verwenden, bei denen er sich registriert.

2.1.2.2 Lookup

Während der Discovery-Prozess für das Auffinden von Lookup-Services zuständig ist, bezieht sich Lookup auf die eigentliche Verwendung der Lookup-Services. Lookup-Services können als ein vernetzter Speichermechanismus betrachtet werden. Jeder Lookup-Service speichert eine Gruppe von Dienstelementen, **Service-Items**, die dem Lookup-Service bekannte Dienste beschreiben. Wenn ein neuer Dienst gestartet wird, erstellt er ein Service-Item, das sowohl den Dienst-Proxy des Dienstes als auch die ihn beschreibenden Attribute enthält. Ein **Dienst-Proxy** ist ein Java-Objekt eines Dienstes, das der Klient herunterladen und verwenden kann. Der Dienst registriert dieses Service-Item bei dem gefundenen Lookup-Service. Wenn ein neuer Klient gestartet wird und einen Dienst verwenden will, nimmt er Kontakt mit einem gefundenen Lookup-Service auf und fragt nach dem gewünschten Dienst (Bild 2.7).

Lookup-Services sind selbst Jini-Dienste, d.h. jeder Lookup-Service hat einen eigenen Dienstverwaltungs-Proxy. Benutzer von Lookup-Services laden und verwenden diese Proxies wie bei "normalen" Diensten. Der einzige Unterschied zwischen dem Proxy eines Lookup-Service, dem **Dienstverwaltungs-Proxy**, und dem Proxy eines normalen Dienstes, dem **Dienst-Proxy**, ist, dass Lookup-Proxies durch den Discovery-Prozess und Jini-Dienst-Proxies durch den Lookup-Services übertragen werden.

Bild 2.7 zeigt den Lookup-Vorgang: der Klient fragt den Lookup-Service nach einem bestimmten Dienst. Eine Kopie des Dienst-Proxy wird vom Lookup-Service an den Klient übertragen. Jetzt kann der Klient mit dem Dienst-Anbieter über den Dienst-Proxy kommunizieren.

Ein Lookup-Service ist nicht dasselbe wie ein Name Server. Im Gegensatz zu traditionellen Name Servern, die Strings als Bezeichnungen gespeicherten Objekten zuordnen, ist die Funktionsweise des Lookup-Service komplizierter als die eines einfachen Name Server. Die Lookup-Funktionen sind für die Hierarchie der Java Datentypen ausgelegt. Daher kann eine Lookup-Suche auf dem Typ eines Objektes basieren und auch die Vererbung von Objekten berücksichtigen.

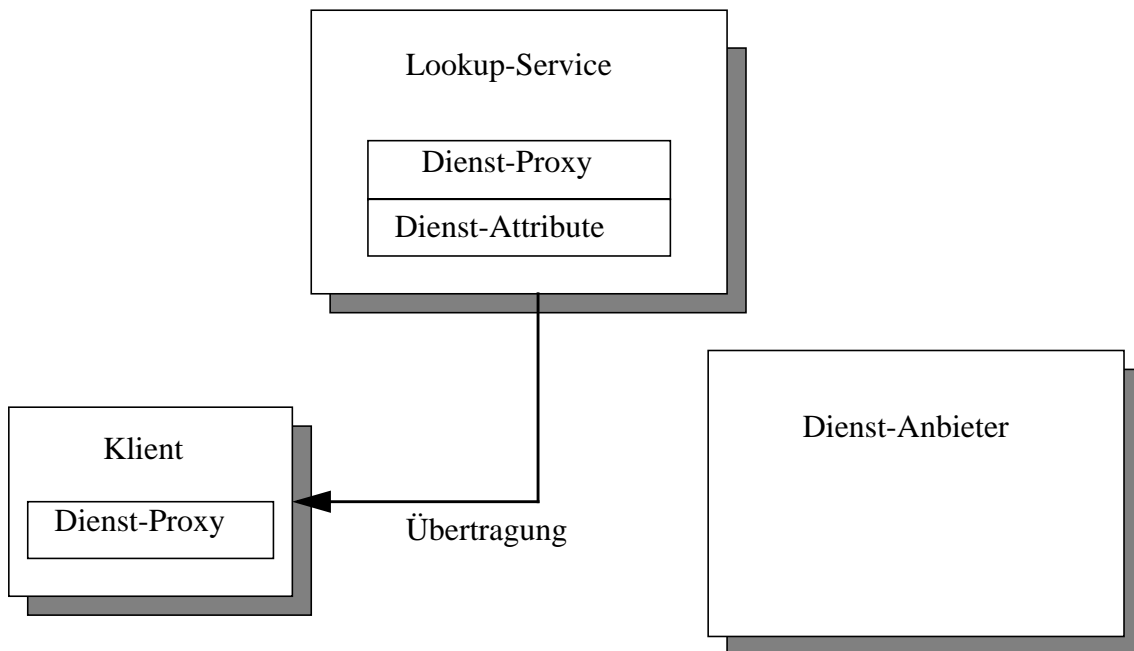


Abbildung 2.7: Der Lookup-Vorgang

Sowohl Klienten als auch Dienste verwenden Lookup-Services. Dienste verwenden Lookup zum Speichern ihrer Service-Proxies und ihrer Attribute. Klienten verwenden Lookup zur Suche nach Diensten und um Benachrichtigungen zu erhalten, wenn neue Dienste verfügbar werden.

Wie Dienste Lookup verwenden

Ein Dienst erstellt ein Service-Item-Objekt, das dessen Dienst-Proxy und alle Attribute aufnimmt, die der Dienst diesem hinzufügen möchte. Dann sucht er mit Hilfe des Discovery nach Lookup-Services und meldet sich mittels des Join-Protokolls bei ihnen an.

Jeder Dienst hat eine eindeutige 128 bit lange Service-ID, die vom ersten gefundenen Lookup-Service zugewiesen wird. Der Dienst muss sich diese ID merken und bei allen anderen Lookup-Services verwenden, bei denen er sich registriert. Durch die eindeutige Service-ID kann der Dienst von anderen Diensten unterschieden werden.

Wenn sich ein Dienst bei einem Lookup-Service registriert, gibt der Lookup-Service ihm ein Lease zurück. Ein **Lease** ist eine Garantie von einem Lookup-Service für die Verfügbarkeit der Ressourcen eines Dienstes für einen bestimmten Zeitraum. Solange der Dienst aktiv ist, sollte das Lease verlängert werden. Wenn der Dienst nicht mehr verfügbar ist, sollte er seine Leases stornieren, um sicherzustellen, dass der Lookup-Service ihn so schnell wie möglich aus der Re-

gistrierung löscht und alle reservierten Ressourcen freigeben kann.

Wie Klienten Lookup verwenden

Mit Hilfe von Lookup-Services können Klienten nach Diensten suchen, die sich bei einem Lookup-Service registriert haben. Es gibt mehrere Verfahren, die Klienten für die Suche einsetzen können. Klienten können einen bestimmten Dienst finden, indem sie seine Service-ID (falls diese bekannt ist) oder eine Dienstbeschreibung (Template) von Dienst-Attributen verwenden.

Eine Dienstbeschreibung, die den gesuchten Dienst beschreibt, wird dem Lookup-Service übergeben. Der Lookup-Service gibt dann die gewünschten Dienste zurück, die mit der Dienstbeschreibung übereinstimmen. Falls keine Dienste mit der Dienstbeschreibung übereinstimmen, wird dem Klient der Wert Null zurückgegeben.

Es gibt zwei Methoden von Lookup für die Suche mit Dienstbeschreibungen. Bei der ersten Methode wird dem Klient nur ein Dienst zurückgegeben, obwohl es mehrere Dienste gibt, die mit der angegebenen Dienstbeschreibung übereinstimmen. Der Lookup-Service wählt einen aus und gibt dessen Dienst-Proxy dem Klient zurück. Diese Methode erfordert, dass die Dienstbeschreibung genau und vollständig beschrieben ist.

Bei der zweiten Methode übergibt der Klient dem Lookup-Service nicht nur die Dienstbeschreibung, sondern zusätzlich noch einen Wert, der die Anzahl der übereinstimmenden Dienste angibt, die zurückgegeben werden sollen. Diese Methode liefert eine Liste der übereinstimmenden Dienste mit der Service-ID und den Attributen.

Je nach Anwendungen können beide Methoden kombiniert verwendet werden. D.h. wenn man die Attribute des gesuchten Dienstes nicht genau weiss, verwendet man zuerst die zweite Methode. Man bekommt eine Liste der Dienste mit der Beschreibung ihrer Attribute. Dann kann man diese Informationen für die genaue Dienstbeschreibung verwenden und benutzt die erste Methode mit dieser Dienstbeschreibung für die Suche nach dem Dienst.

2.1.2.3 Leasing, verteilte Ereignisse und Transaktionen

Im folgenden Abschnitt werden die drei weiteren Elemente von Jini-Konzept kurz vorgestellt, die im Rahmen der Studienarbeit nicht so von Bedeutung sind.

Leasing

Leasing basiert auf dem Prinzip, dass der Zugriff auf eine Ressource nur für einen bestimmten Zeitraum erlaubt wird. Leases werden für Lookup-Services genutzt. Wenn sich ein Dienst bei

einem Lookup-Service registriert, gibt der Lookup-Service ihm ein Lease zurück. Solange der Dienst aktiv ist, sollte das Lease verlängert werden.

Leases können vom Lease-Anbieter (Lookup-Service) abgelehnt und vom Lease-Abnehmer (Klienten) verlängert werden. Ohne Verlängerung laufen Leases zu einem bestimmten Zeitpunkt ab. Ausserdem können Leases vorzeitig beendet und zwischen dem Lease-Anbieter und dem Lease-Abnehmer ausgehandelt werden. Sobald ein Dienst beabsichtigt oder unbeabsichtigt nicht mehr verfügbar ist, ohne sich abzumelden, laufen seine Leases nach einiger Zeit ab und der Dienst wird aus der Registrierung gelöscht und die verwendete Ressourcen für die Registrierung können freigegeben werden. Alle Klienten, die diesen Jini-Dienst benutzen, werden über dessen Ausfall informiert.

Verteilte Ereignisse

"Ein Ereignis ist ein Objekt, das Informationen über eine externe Zustandsänderung enthält, an der eine Software-Komponente interessiert sein könnte" [Edwards99]. Im Gegensatz zu lokalen Ereignissen finden verteilte Ereignisse auf verschiedenen Rechner statt. Es gibt mehrere Situationen, in denen verteilte Ereignisse benötigt werden. Beispielsweise möchten eine Anwendung oder ein Jini-Dienst einen anderen Jini-Dienst verwenden. Sie benötigt dabei Meldungen über externe Zustandsänderungen. Zum Beispiel will eine Digitalkamera einen Drucker verwenden. Es gibt den Fall, dass der Drucker schon mit dem Netzwerk verbunden und verfügbar ist, bevor die Kamera da ist. In diesem normalen Fall werden die Discovery-Mechanismen (siehe Kapitel 2.1.2) verwendet. Aber es gibt auch den Fall, in dem der Drucker verfügbar wird, nachdem die Kamera mit dem Netzwerk verbunden wurde. In diesem Fall benötigt die Kamera Meldungen, wenn Drucker an der Gemeinschaft teilnehmen und verfügbar sind. Der Lookup-Service sendet Ereignisse an Interessenten, wenn sich Dienste bei ihm anmelden, abmelden oder ändern.

Transaktionen

Wie in Datenbanken sind Transaktionen in Jini zusammenhängende Operationen, die entweder erfolgreich und vollständig oder gar nicht ausgeführt werden. In beiden Fällen nimmt das System einen bekannten Zustand an - war die Transaktion erfolgreich, wird das Programm fortgesetzt.

Transaktionen ermöglichen Datenmanipulationen mit ACID-Eigenschaften (Atomicity, Consistency, Isolation, Durability). Um die ACID-Eigenschaften einzuhalten, wird ein Two-Phase Commit Protokoll verwendet (siehe [Edwards 99], Seite 124-128).

2.1.3 Technischer Hintergrund

Nachdem das Jini-Konzept vorgestellt wurde, wird im folgenden Abschnitt erklärt, wie Jini eingerichtet und gestartet wird.

Klient-Geräte müssen java-fähig sein, d.h. auf diesen Geräten kann die Java-Virtual-Maschine (JVM) laufen. Die Java-2(JDK 1.2) wird benötigt, weil Jini einige Funktionen erfordert, die in Java-2 enthalten sind zum Beispiel RMI-Aktivierungsstruktur. Für das Einrichten und Starten von Jini sind die folgenden Schritten notwendig:

1. Installieren von Java-2
2. Installieren von Jini
3. Die Umgebungsvariablen *path* für die ausführbaren Java-Dateien und *classpath* für Jini einrichten.
4. Jini-Laufzeitdienste starten: die drei Jini-Laufzeitdienste sind der HTTP-Server, RMI und der Jini-Lookup-Service.

Den HTTP-Server konfigurieren und starten:

Der HTTP-Server überträgt den Code für den Dienst-Proxy (Klassendateien), den Klienten verwenden können, um mit dem Dienst zu interagieren. Die Anforderungen an diesen Server sind minimal, im Grunde braucht er lediglich das Abrufen von Daten zu unterstützen. Deshalb sollte jeder HTTP-Server dafür ausreichend sein. Im Paket *tools.jar* der Jini-Software befindet sich ein einfacher HTTP-Server. Um den HTTP-Server zu starten, wird die folgende Befehlszeile eingegeben (für die Erklärung siehe Anhang A1):

```
java -jar <tools-jarfile> [-port <port-number>] [-dir <document-root-dir>] [-trees] [-verbose]
```

Der HTTP-Server kann auf demselben Rechner laufen wie der Lookup-Service.

Den RMI-Aktivierungs-Daemon konfigurieren und starten:

Auf jedem Host, auf dem sich aktivierbare Objekte befinden, muss eine Instanz des Aktivierungs-Daemon ausgeführt werden. Dies betrifft den Jini-Lookup-Service, den Transaktionsmanager und JavaSpaces. Der RMI-Prozess ermöglicht es Objekten, die nur selten verwendet werden, einzuschlafen, um bei Bedarf automatisch geweckt zu werden. Der RMI-Aktivierungs-Daemon reguliert das Wechseln zwischen den aktiven und inaktiven Zuständen der Objekte und wird von den anderen Jini-Laufzeitdiensten genutzt.

Der RMI-Aktivierungs-Daemon muss auf demselben Rechner laufen wie der Lookup-Service und zwar *vor* dem Lookup-Service, weil die Sun-Implementierung des Lookup-Service Aktivierung verwendet. Um den RMI-Aktivierungs-Daemon zu starten, wird einfach die Befehlszeile *rmid* eingegeben, falls das Verzeichnis *jdk1.2/bin/rmid* in der Umgebungsvariable *path* angegeben ist.

Den Lookup-Service konfigurieren und starten:

Der Lookup-Service stellt die eigentliche Schlüsselkomponenten von Jini dar. Er verwaltet die gegenwärtig aktiven Jini-Dienste, die innerhalb eines Subnetzes verfügbar sind. Die Sun-Implementierung des Lookup-Service heisst "**Reggie**" und befindet sich in dem Archiv *reggie.jar* der Jini-Software. Das Format für das Ausführen des Lookup-Service sieht folgendermassen aus (für die Erklärung siehe Anhang A1):

```
java -Djava.security.policy=<security_policy> -jar <lookup-server-jarfile> <lookup-client-codebase> <lookup-policy-file> <output-log-dir> <lookup-service-group>
```

Der Lookup-Service sollte auf einem Rechner im selben Subnetz gestartet werden, auf dem der Dienst läuft, weil die Multicast-Discovery-Protokolle standardmässig so konfiguriert sind, dass Dienste Lookup-Services, die im selben Subnetz sind, suchen. Ausserdem muss der Lookup-Service *nach* dem RMI gestartet werden, weil er den RMI für die Aktivierung benutzt.

2.1.4 Anwendungsbeispiele

Im folgenden Abschnitt werden Jini-Anwendungsbeispiele für verschiedene Szenarien vorgestellt: ein Flughafeninformationssystem, eine universelle Haussteuerung und ein Stadtinformationssystem vorgestellt:

Flughafeninformationssystem

Beispielsweise kommt jemand an einen Flughafen an, der mit einem Funknetz ausgestattet ist. In der Hand hat er einen Palm PDA (Personal Digital Assistant). Mit Hilfe von Jini kann sein PDA das lokale Informationssystem spontan finden und sich bei ihm anmelden. Der Benutzer muss selbst dazu nichts machen. Er wird über seine Abflug-Gate sofort benachrichtigt und sein Name wird in die Passagierliste aufgenommen. Er muss dazu nicht an einem Schalter stehen und warten. Seine Ankunft startet auch einen anderen Netzwerk-Dienst, das Flugticketverwaltungssystem (the airline's frequent-flyer administration system). Er hat genug Kilometer gesammelt, um einen Bonus zu bekommen; die Belohnung ist ein Sitz in der ersten Klasse. Das

Netzwerk ändert seine Sitznummer und er kann in die Flughafenhalle gehen und dort auf die Abflugszeit warten. Das Informationssystem wird ihn benachrichtigen, wenn es Zeit für das Einsteigen ist. Inzwischen kann er seine E-mail lesen. Mit Hilfe von Jini kann er auch einen lokalen Drucker benutzen und die E-Mail ausdrucken lassen.

Universelle Haussteuerung

Beispiele für Jini-Dienste im Heimbereich können Lampen, Kaffe-Maschinen, Geschirrspülmaschinen und Wasser-Erhitzer, die Jini-Software verwenden. Jini bietet hier eine höhere Bequemlichkeit. Geräte, die sich selbst durch entfernte Diagnose warten, können ihre Verhalten verbessern oder regulieren. Während man mit warmen Wasser duscht, reguliert der Wasser-Erhitzer die Temperatur und den Wasserdruck, damit jemand anderes gleichzeitig die Geschirrspülmaschine einschalten kann.

Stadtinformationssystem

Mobile Netzwerke sind weit verbreitet und ein ideales Anwendungsgebiet für die Jini-Technologie. Mit Hilfe der Jini-Technologie können Provider Zusatz-Dienste (value-added services) anbieten und dem Benutzer z.B. Zugriff auf ein elektronische Bezahlungssystem geben. Beispielsweise fährt ein Benutzer mit dem Taxi ohne Bargeld. Wie kann er den Taxifahrer bezahlen? Er hat ein Jini-fähiges Handy und sein Provider benutzt Jini, um seinen Kunden netzwerk-fähige Dienste anzubieten. Mit seinem Handy kann der Benutzer auf das elektronische Bezahlungssystem der Taxi-Gesellschaft zugreifen, um die Fahrt zu bezahlen. Die Taxi-Gesellschaft nimmt die Transaktion entgegen und sendet eine Quittung an den Drucker im Taxi. Der Fahrgast nimmt die Quittung und steigt aus.

2.2 Smartboard

Das Smartboard ist eine elektronische, interaktive Tafel. Wie bei einer Beamer-Präsentation zeigt das Smartboard den Inhalt des Präsentationsbildschirms. Ausserdem kann man mit dem Smartboard:

- Notizen, die man auf die Tafel geschrieben hat, in einer Datei speichern.
- mit Finger auf die Tafel drücken, um Windows-Anwendungen zu steuern.

Es bietet noch mehr Funktionen wie Ausdrucken, Senden der E-Mails usw.

3 Entwurf

Nachdem die Grundlagen von Jini und Smartboard erläutert wurden, werden in den folgenden Abschnitten die Entwurfsentscheidungen, die Alternativen des Entwurfs, das Design und die Abläufe sowie die Klassensstrukturen beschrieben.

3.1 Entwurfsentscheidungen

Zuerst werden einige wichtige Begriffe für die Aufgabe hier erklärt. In dieser Studienarbeit ist das Smartboard, die elektronische Tafel, der Dienst. Die Abkürzung für elektronischer Tafel Dienst ist **ET-Dienst**. Das Java-Objekt des ET-Dienstes ist der **ET-Proxy**. Als **ET-Server** wird die Software auf dem Präsentationscomputer für den ET-Dienst bezeichnet. Jeder ET-Server ist zuständig für einen ET-Dienst im Raum, wo der Präsentationscomputer bzw. der ET-Server ist. Der ET-Dienst wird durch den ET-Proxy und den ET-Server implementiert. Als **ET-Klient** wird im folgenden die Software auf dem Laptop bezeichnet.

Für die Arbeit gibt es vier wichtige Entwurfsentscheidungen:

Serverseitige Implementierung der Funktionen des ET-Proxy

Auf der Klient-Seite gibt es nur eine einfache Komponente für die Suche nach einem Lookup-Service und für die Anfrage nach dem ET-Dienst. Wenn der Klient den ET-Proxy bekommen hat, muss er nichts weiter machen. Der ET-Proxy kümmert sich im weiteren um die Benutzungsoberfläche, die Steuerung der Präsentation vom Laptop aus und die Kommunikation mit dem Server. Die Vorteile sind dabei die Einfachheit und Flexibilität der Erweiterung und Wiederverwendbarkeit der Software. Jedes Mal, wenn man eine andere elektronische Tafel mit neuen Funktionen hat oder wenn man einige Funktionen der elektronischen Tafel erweitern will, braucht man nur die Software auf der Server-Seite zu ändern oder anzupassen. Die Software auf der Klient-Seite wird nicht geändert.

ET-Proxy ohne RMI

Wie in Kapitel 2 erwähnt, ermöglicht das Jini-Konzept des herunterladbaren Dienst-Proxies, Dienste ohne explizite Installation von Treibern oder sonstiger Software zu verwenden. Ein Klient kann den Dienst-Proxy herunterladen und ihn benutzen, nachdem er den Lookup-Service

gefunden und nach dem Dienst gefragt hat. Wie der Dienst-Proxy mit dem Dienst interagiert, ist abhängig von der Implementierung, die dem Klient verborgen bleibt. In dieser Arbeit verwendet der ET-Proxy kein RMI. D.h. der heruntergeladene ET-Proxy verwendet ein eigenes Kommunikationsprotokoll für die Verständigung mit dem Server. Die Benutzungsoberfläche und der Ablauf der Präsentation werden vom Benutzer gesteuert. Deshalb ist es sinnvoll, dass die Funktionen des ET-Proxy beim Klient ausgeführt werden. Damit werden die Methoden des ET-Proxy lokal, nicht von einem anderen Rechner aufgerufen. In diesem Fall fungiert der Proxy als herunterladbarer Treiber für elektronische Tafel. Ausserdem ist RMI für die Aufgabe der Übertragung von Massendaten nicht geeignet. Das lokale Aufrufen der Methode ohne RMI geht schneller als das Aufrufen der Methoden mit RMI. Dadurch läuft die Interaktion mit dem Benutzer flüssiger.

Verwendung des Multicast-Discovery

Diese Aufgabe legt nicht fest, auf welchem Rechner der Lookup-Service läuft, d.h. der Klient muss zuerst nach den Lookup-Services im lokalen Netzwerk suchen, da er diese benötigt, um nach der elektronischen Tafel zu suchen. Dafür werden die beiden Multicast-Protokolle verwendet. In diesem Fall braucht man nicht, die Namen der Host-Rechner anzugeben, auf denen die Lookup-Services laufen.

Verwendung einer Dienstbeschreibung für die Dienst-Suche

Wie im Kapitel 2 erwähnt wurde, kann ein Klient entweder eine Service-ID oder eine Dienstbeschreibung für die Suche nach einem Dienst verwenden. In dieser Aufgabe soll jemand einen beliebigen ET-Dienst für seine Präsentation benutzen können. Er kann also die Service-ID des ET-Dienstes nicht wissen. Auf diesem Grund wird eine Dienstbeschreibung für die Suche verwendet.

3.2 Alternativen des Entwurfs

Im obigen Abschnitt wurden die Entwurfsentscheidungen beschrieben, zu denen in diesem Abschnitt die Alternativen vorgestellt werden sollen: Klientseitige Implementierung der Funktionen und ET-Proxy mit RMI.

Klientseitige Implementierung der Funktionen des ET-Proxy

Ausser der Funktion zur Suchen nach Lookup-Services und zur Anfragen nach Diensten könnte die Software auf der Klient-Seite zusätzliche Funktionen enthalten. Beispielsweise könnten die Benutzungsoberfläche und die Steuerung der Präsentationsdaten hier implementiert werden. Für die Aufgabe ist es keine gute Lösung, weil jedes Mal, wenn man eine andere elektronische Tafel mit neuen Funktionen hat oder wenn man einige Funktionen der elektronischen Tafel erweitern will, muss man die Software auf der Klient-Seite ändern oder anpassen. Das ist mühsam und kostet viel Aufwand, weil es nicht nur einen Klient gibt, d.h. die Software auf allen Klient-Rechner muss neu installiert werden; Im schlimmsten Fall muss man die Software auf dem Klient-Rechner jedes Mal neu installieren, wenn man in einem anderen Raum geht, weil die Räume verschiedene elektronische Tafeln haben. Daher ist die klientseitige Implementierung der Funktionen des ET-Proxy ungeeignet.

ET-Proxy mit RMI

Eine Alternative des Aufrufen der Funktionen des ET-Proxy **ohne RMI** ist das Aufrufen der Funktionen des ET-Proxy **mit RMI**. Mit RMI werden die Methoden des ET-Proxy an dem ET-Server ausgeführt und die Ergebnisse werden an den Klient bzw. an den ET-Proxy gesendet. Anschliessend ist RMI für die Übertragung von Massendaten ungeeignet. Das entfernte Aufrufen der Methoden geht langsamer als das lokale Aufrufen. Dadurch läuft die Interaktion mit dem Benutzer nicht flüssig und daher wird diese Lösung nicht verwendet. Ausserdem ist das wichtige Jini-Konzept die Möglichkeit des Herunterladen des Dienst-Proxy und deshalb wird eine Lösung ohne RMI gegenüber einer Lösung mit RMI bevorzugt.

3.3 Design: Systemkomponenten

Nach dem Überblick über die Entwurfsentscheidungen wird in diesem Kapitel das Design mit den Systemkomponenten vorgestellt; das Zusammenspiel der Komponenten und die Rolle jeder Komponente werden hier näher betrachtet.

Systemkomponenten

Das System (Bild 3.1) besteht aus mehreren Komponenten: dem ET-Klient, dem ET-Server, dem Smartboard, einem HTTP-Server und dem Jini-Lookup-Service.

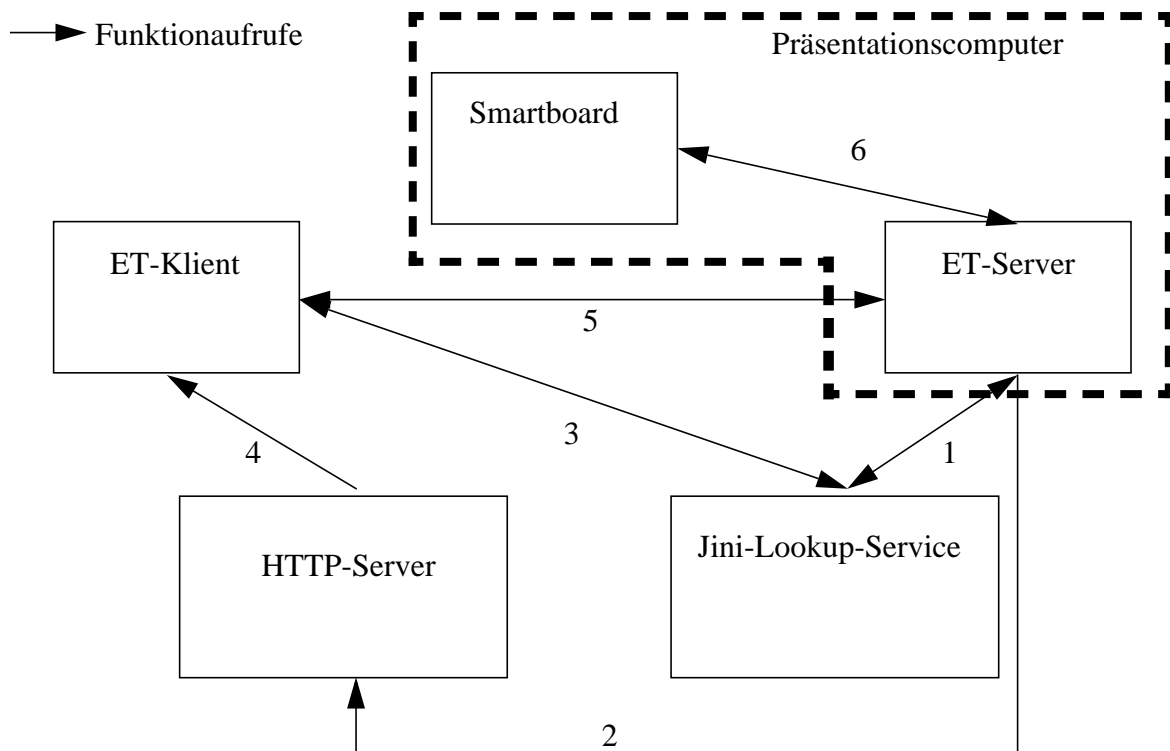


Abbildung 3.1: Die Systemübersicht

Der **Smartboard-Treiber** ist eine herstellereigenspezifische Software. Eine elektronische Tafel, die von diesem Treiber gesteuert wird, zeigt die Präsentationsdaten an und ermöglicht es, Notizen, die auf die Tafel geschrieben werden, an den ET-Server zu übertragen. Der **HTTP-Server** überträgt die Klassendateien an den ET-Klienten, die für die Kommunikation mit dem ET-Server benötigt werden. Der **Lookup-Service** ist ein Teil der Jini-Software, bei dem sich der ET-Server anmeldet und bei dem der ET-Klient nach dem ET-Dienst fragt. Die **ET-Klient**- und **ET-Server**-Komponenten sind in dieser Studienarbeit entstanden. Diese Komponenten werden

nach Erklärung der Systemübersicht im Bild 8 näher erläutert.

1. Nach der Suche nach den vorhandenen Lookup-Services meldet der ET-Server bei ihnen den ET-Dienst an.
2. Die Klassendateien des ET-Proxy werden dem ET-Klienten über den HTTP-Server zur Verfügung gestellt
3. Nach der Suche nach einem Lookup-Service fragt der ET-Klient ihn nach dem ET-Dienst.
4. Der HTTP-Server überträgt den Code des ET-Proxy an den ET-Klienten.
5. Der ET-Klient und der ET-Server kommunizieren über den ET-Proxy miteinander.
6. Der ET-Server und das Smartboard kommunizieren über die Smartboard-API miteinander.

ET-Klient

Der ET-Klient in dieser Arbeit dient eigentlich nur dem Laden des ET-Proxy, der das Objekt des ET-Dienstes repräsentiert. Der ET-Klient hat keine Benutzungsoberfläche oder weitere wichtige Funktionen neben einfachen Funktionen, die der Dienstanfrage und dem Start des ET-Proxy dienen. Deshalb besitzt der ET-Klient nur eine einfache Struktur und wird hier nicht beschrieben.

ET-Server

Der ET-Server ist ein wesentlicher Bestandteil dieser Studienarbeit. Seine Rolle ist die eines Diensteanbieters und er meldet dazu den ET-Proxy beim Lookup-Service an. Ausserdem ist er der Vermittler zwischen dem ET-Klienten bzw. dem ET-Proxy und der elektronischen Tafel:

- Der ET-Server empfängt die Präsentationsdatei sowie die Steuersignale vom ET-Proxy bzw. vom Laptop und diese Datei wird auf der elektronischen Tafel angezeigt bzw. aktualisiert.
- Der ET-Server sendet Notizen, die auf die elektronische Tafel geschrieben wird, an den ET-Proxy und diese Notizen werden auf dem Laptop-Bildschirm angezeigt.

Die Komponenten des ET-Servers werden im Bild 3.2 gezeigt.

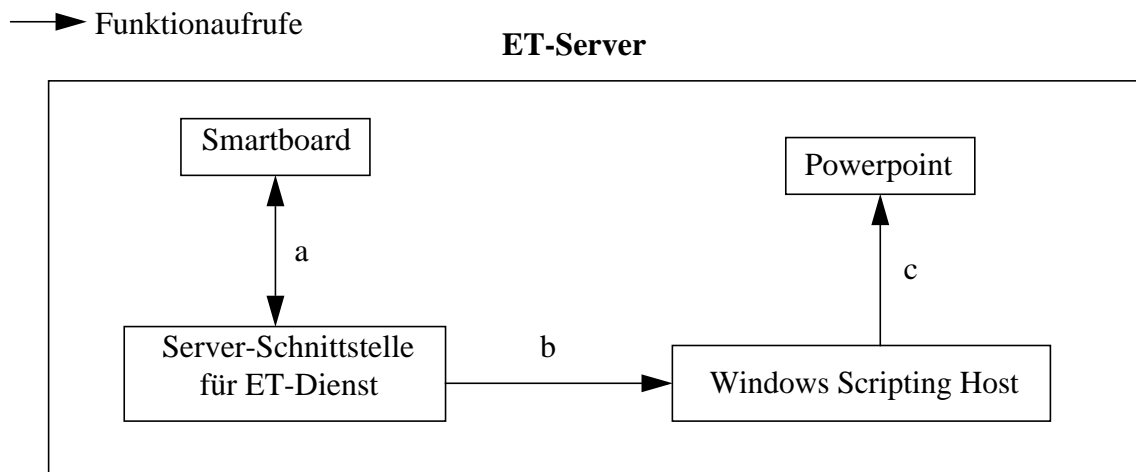


Abbildung 3.2: Die Komponenten des ET-Servers

Die Server-Schnittstelle für den ET-Dienst nutzt die Smartboard-API zur Übertragung der Notizen von der elektronischen Tafel an den ET-Klient (a). Ausserdem verwendet sie Windows Scripting Host (WSH) zur Steuerung der Powerpoint (PPT)-Dateien vom Laptop aus (b und c).

Kommunikation zwischen dem ET-Klienten bzw. dem ET-Proxy und dem ET-Server

Die Schnittstelle *ElectronicBoardInterface* (Bild 3.3) ist dem ET-Klient bekannt und wird von dem ET-Dienst implementiert. Diese Schnittstelle bietet die Methoden für Dienstanfragen (*ready()*, *askService()*) und für Dienststart (*startService()*) sowie für das Senden der Benutzerinformationen (*sendInfo()*). Diese Benutzerinformation z.B. der Name des Benutzers wird für die Ablehnungsmeldung verwendet, falls es eine Kollision bei der Benutzung der elektronischen Tafel gibt (siehe Kapitel 4.4).

ElectronicBoardInterface

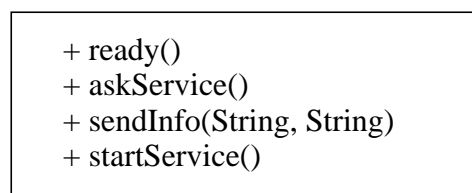


Abbildung 3.3: Die Methoden der Schnittstelle ElectronicBoardInterface

Nach dem Herunterladen des ET-Proxy kann der ET-Klient den ET-Proxy starten und dadurch kann der Benutzer über diesen ET-Proxy mit dem ET-Server interagieren. Die Präsentationsdaten sowie die Steuersignale werden vom ET-Proxy an den ET-Server gesendet.

In der umgekehrten Richtung werden Notizen, die auf die elektronische Tafel geschrieben werden, vom ET-Server an den ET-Proxy übertragen.

3.4 Architektur: Abläufe und Klassenstrukturen

Nach dem Überblick über die Systemkomponenten werden in diesem Kapitel die zeitlichen Abläufe sowie die Klassenstruktur des Programms im Detail erläutert.

3.4.1 Abläufe

Das Bild 3.4 stellt den Überblick über den Vorgang ab dem Programmstart bis zum Beenden der Präsentation vor. Zur Vereinfachung wird dabei angenommen, dass es nur einen Lookup-Service, einen ET-Klient, einen ET-Dienst bzw. einen ET-Server und ein Smartboard gibt. Die zeitliche Abläufe lassen sich in drei Phasen einteilen:

Phase 1: Dienstanmeldung und Dienstanfrage

- 1.1 Der ET-Server meldet sich beim Lookup-Service an (Schritt 1).
- 1.2 Der ET-Proxy wird an den Lookup-Service übertragen (Schritt 2).
- 1.3 Der ET-Klient fragt nach dem ET-Dienst (Schritt 3).
- 1.4 Der ET-Proxy wird vom ET-Klienten heruntergeladen (Schritt 4).

Phase 2: Ausführen des ET-Dienstes

- 2.1 Der ET-Klient startet den ET-Proxy (Schritt 5).
- 2.2 Der ET-Proxy öffnet zwei Fenster für die Steuerung der Präsentation (Schritt 6).
- 2.3 Der Benutzer gibt die Präsentationsdaten an, die an den ET-Server gesendet und auf dem Smartboard angezeigt wird (vom Schritt 7 bis zum Schritt 9).
- 2.4 Der ET-Proxy öffnet ein Steuerungsmenü auf dem Bildschirm des Klienten (Schritt 10).
- 2.5 Der Benutzer steuert die Präsentationsdaten auf dem Smartboard durch Steuersignale, die

an den ET-Server weiter gesendet werden. Die Präsentationsdaten auf dem Smartboard werden aktualisiert (vom Schritt 11 bis zum Schritt 13).

2.6 Die Notizen auf dem Smartboard werden auf dem Server-Bildschirm angezeigt und an den ET-Proxy gesendet und damit auf dem Klient-Bildschirm angezeigt (vom Schritt 14 bis zum Schritt 16).

2.7 Der Benutzer beendet die Präsentation. Das entsprechende Steuersignal wird an den ET-Server gesendet. Die Präsentationsdaten werden nicht mehr auf dem Smartboard angezeigt (vom Schritt 17 bis zum Schritt 19).

Phase 3: Abmelden und Beenden des ET-Klienten

3.1 Der Benutzer beendet das Klient-Programm (Schritt 20).

3.2 Das Steuersignal wird an den ET-Server übertragen und damit die Verbindung abgebaut. Der ET-Server wartet auf neue Anfragen eines ET-Klienten (Schritt 21).

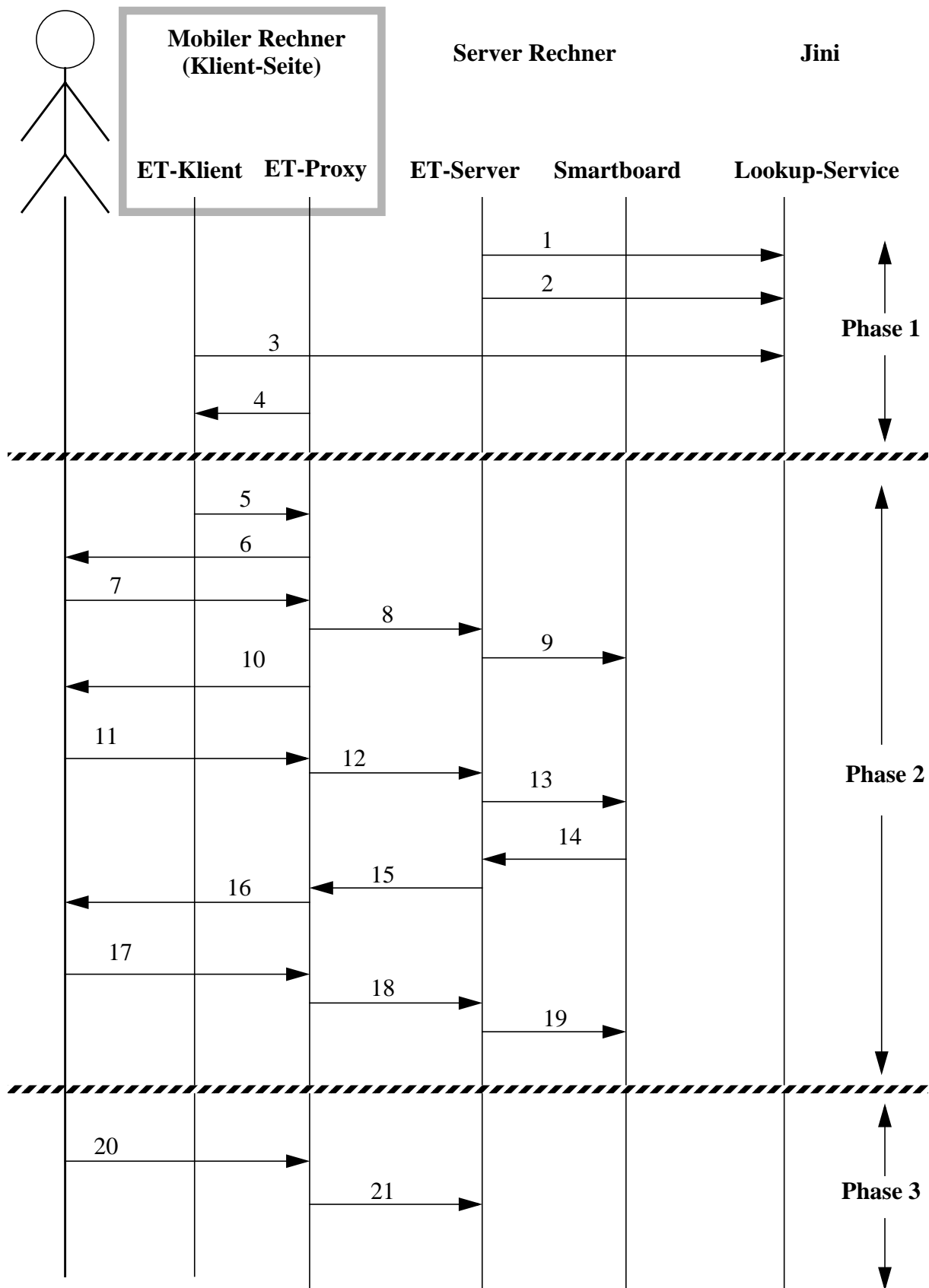


Abbildung 3.4: Die zeitlichen Abläufe

3.4.2 Klassenstrukturen

In diesem Abschnitt werden die Aufgaben und die Methoden der Klassen (Bild 3.5) des Systems beschrieben.

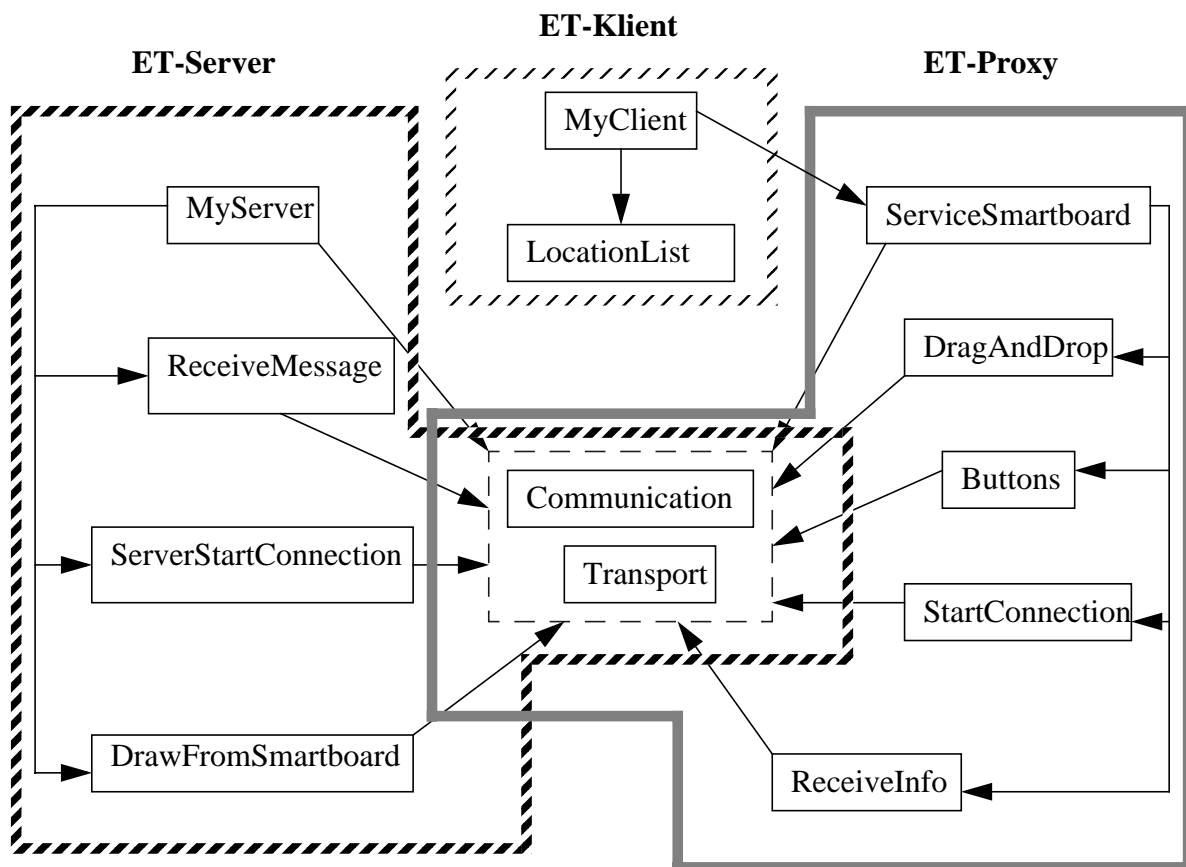
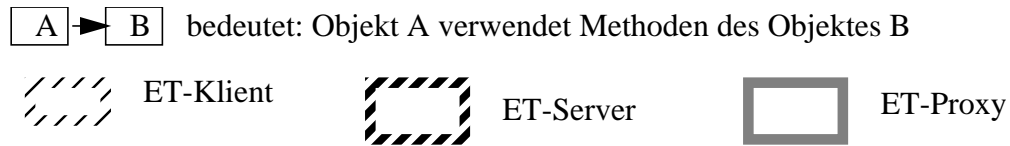


Abbildung 3.5: Die Klassenstrukturen

Die Aufgaben der Klassen:

Der ET-Klient enthält die Klassen: MyClient und LocationList. Die Klasse **MyClient** bietet die Methoden für Dienstanfrage und Dienststart. Die Aufgabe der Klasse **LocationList** ist die Interaktion mit dem Benutzer. Der Benutzer wählt dort eine elektronische Tafel aus der Liste der verfügbaren Tafel aus.

Der ET-Server enthält die Klassen **MyServer**, **ReceiveMessage**, **ServerStartConnection**, **DrawFromSmartboard**, **Communication** und **Transport**. Die Aufgaben der Klasse **MyServer** sind die Suche nach dem Lookup-Service, die Dienstanmeldung sowie die Kommunikation mit dem ET-Proxy über Netzwerk. Ausserdem überträgt sie die vom ET-Proxy gesendete Präsentationsdaten an die elektronische Tafel zur Anzeige. Die Klasse **ServerStartConnection** sorgt für die Netzwerk-Verbindung mit dem ET-Proxy. Die Klasse **ReceiveMessage** empfängt die Nachrichten vom ET-Proxy und überprüft diese Nachrichten, ob sie Benutzer-Informationen, Steuersignale oder die Präsentationsdaten enthalten. Die Klasse **DrawFromSmartboard** nimmt die Notizen von der Tafel auf, zeigt sie auf dem Server-Bildschirm an und sendet sie an den ET-Proxy.

Die Klasse **Communication** ist für das Senden und das Empfangen der Nachrichten zwischen zwei Parteien zuständig. Die Klasse **Transport** wird als Container beim Transport von verschiedenen Objekttypen verwendet.

Der ET-Proxy besteht aus den Klassen: **ServiceSmartboard**, **DragAndDrop**, **Buttons**, **StartConnection**, **ReceiveInfo**, **Communication** und **Transport**. Die Klasse **ServiceSmartboard** implementiert die Methoden der Schnittstelle **ElectronicBoardInterface**. Sie sorgt dafür, dass der Benutzer die Präsentationen starten kann und die Notizen von der Tafel, die vom ET-Server an den ET-Proxy gesendet und auf dem Laptop angezeigt werden. Die Klasse **StartConnection** startet die Verbindung mit dem ET-Server. Die Klasse **ReceiveInfo** empfängt Nachrichten vom ET-Server und prüft den Typ der Nachricht. Die Klasse **DragAndDrop** nimmt die Präsentationsdaten durch Drag&Drop auf und sendet diese an den ET-Server. Die Klasse **Buttons** erzeugt ein Steuerungsmenü, mit deren der Benutzer die Präsentation auf der Tafel vom Laptop aus steuern kann.

Der Überprüfung der Existenz des Klienten dienen zusätzlich bei dem ET-Server die Klassen **ServerStartControlClient**, **ControlClient** und **ReceiveClientReply** sowie bei dem ET-Proxy die Klassen **StartControlConnection** und **ReceiveClientRequest**. Zur Vereinfachung werden diese Klassen im Bild 3.5 nicht gezeigt. Der ET-Server muss regelmässig wissen, ob der ET-Klient noch da ist. Ist der ET-Klient da, findet die Kommunikation weiter statt. Ist der ET-Klient ohne offizielle Abmeldung abgestützt, wird die Verbindung abgebaut und werden alle Zustände zurückgesetzt. Der ET-Server kann daher auf die Anfrage neuen ET-Klienten warten.

4 Implementierung

Dieses Kapitel erläutert die Implementierung der ET-Dienst-Anmeldung, der ET-Dienst-Anfrage sowie der Kommunikation zwischen dem ET-Klient und dem ET-Server. Dabei wird die Benutzungsoberfläche näher erklärt.

4.1 Anmelden des Dienstes

Die Anmeldung findet in der Klasse `MyServer` statt. Für die Anmeldung des ET-Dienstes ist zuerst die Suche nach den vorhandenen Lookup-Services erforderlich. Beim Anlegen der Klasse `LookupDiscovery` werden die Namen für die Gruppe von Lookup-Services übergeben, nach denen gesucht werden soll. Die Übergabe von `null` bedeutet, dass nach allen Gruppen gesucht werden sollen. Die Übergabe eines leeren Arrays (mit Länge 0) signalisiert, dass keine Gruppen gesucht werden sollen. Für die beide letzteren Möglichkeiten stellt die Klasse `LookupDiscovery` die Konstanten `ALL_GROUPS` und `NO_GROUPS` bereit. Beim Erstellen eine Instanz von `LookupDiscovery` beginnt der Discovery-Prozess, wobei alle Lookup-Services im Subnetz mit Hilfe von Multicast-Anfragen gesucht werden. Nach einer definierten Zeit hört der Discovery-Prozess auf (siehe Kapitel 2.1.2.1) Anfragenachrichten zu senden und wartet dann auf Bekanntmachungsnachrichten (wie in Kapitel 2 beschrieben).

```
public class MyServer implements ServiceIDListener, DiscoveryListener
{
    ..
    // Start der Suche nach Lookup Services
    LookupDiscovery discovery;
    discovery = new LookupDiscovery (LookupDiscovery.ALL_GROUPS);
    discovery.addDiscoveryListener (this);
    ...
    // Die Methoden der Schnittstelle DiscoveryListener
    public void discarded(DiscoveryEvent devn){...}
    public void discovered(DiscoveryEvent devn){...}
    // Methode der Schnittstelle ServiceIDListener
    public void serviceIDNotify (ServiceID sidIn)
    {
        System.out.println ("server: received ServiceID = " + sidIn);
    }...
}
```

```
}
```

Die Methode `addDiscoveryListener(DiscoveryListener l)` wird verwendet, um Listener in die Liste aufzunehmen. Ein Listener ist der Diensterbringer, der von Lookup-Services hören will. Sobald ein neuer Lookup-Service ermittelt wird, ruft `LookupDiscovery` alle registrierten Listener auf. Das *MyServer-Objekt* muss dazu die Schnittstelle *DiscoveryListener* implementieren. Diese Schnittstelle wird zur Benachrichtigung über die Existenz von Lookup-Services verwendet und hat zwei Methoden. Die Methode `discovered()` wird immer dann aufgerufen, wenn ein neuer Lookup-Service mittels des Discovery-Protokolls gefunden wurde. Die Methode `discarded()` wird aufgerufen, wenn ein bereits gefundener Lookup-Service nicht mehr relevant ist. Das kann z.B. geschehen, wenn Lookup-Services aus der Liste der Gruppen entfernt werden, an denen man interessiert ist.

Nachdem der Lookup-Service gefunden wurde, meldet sich der ET-Dienst bei ihm mit Hilfe des Join-Protokolls (beschrieben in Kapitel 2) an (siehe den folgenden Code). Es ist nötig, einen Sicherheit-Manager einzurichten, um alle entfernten Klassen über eine Codebase zu laden. Wenn keiner Sicherheit-Manager eingerichtet wird, werden nur Klassen geladen, die im CLASSPATH der Anwendung vorkommen. Das ET-Dienst Smartboard hat mehrere Attribute z.B. Name, Lokation usw. Die Klasse *JoinManager* bietet alle Funktionen, die zur Implementierung des Join-Protokolls erforderlich sind. Diese Klasse stellt drei Konstruktoren bereit, die zu verschiedenen Zeiten des Lebenszyklus eines Dienstes verwendet werden. Der erste Konstruktor wird benutzt, wenn die Service-ID des Dienstes noch nicht bekannt ist. Der zweite Konstruktor wird benutzt, wenn die Service-ID des Dienstes noch nicht bekannt ist und der Dienst an einer Dienstgruppe teilnehmen sowie die Verbindung mit einer bestimmten Menge der Lookup-Services herstellen will. Der dritten Konstruktor wird benutzt, falls die Service-ID hier bekannt ist. Für diese Studienarbeit wird der erste Konstruktor wie im folgenden benutzt:

```
JoinManager(Objekt proxy, Entry[] atts, ServiceIDListener listener, LeaseRenewalManager mgr)
```

Der erste Parameter ist das Dienst-Objekt, hier das *ServiceSmartboard-Objekt*. Der zweite Parameter ist die Dienstbeschreibung. Der dritte Parameter ist das Objekt, das über die Service-ID benachrichtigt wird. Der letzte Parameter ist das Objekt zur Verwaltung und Verlängerung von Leases (siehe Kapitel 2).

Hier ist der Ausschnitt des Codes:

```
public static void main (String[] args)
{
```

```
MyServer myServer;
Entry[] attributes;
JoinManager joinmanager;
try
{
    System.setSecurityManager (new RMISecurityManager ());
    attributes = new Entry[2];
    attributes[0] = new Name("ServiceSmartboard");
    attributes[1] = new Location ("0", "183", "20");
    myServer = new MyServer ();
    joinmanager = new JoinManager(myServer.serviceSB, attributes, myServer,
                                new LeaseRenewalManager ());
    ...
}
}
```

4.2 Anfrage des Dienstes

Die Dienstanfrage findet in der Klasse `MyClient` statt. Zunächst wird der Lookup-Service gesucht. Der Suche-Vorgang bzw. die Schnittstelle `DiscoveryListener` wurde in Kapitel 4.1 beschrieben und wird deshalb hier nicht weiter erklärt. Anschliessend findet die Dienstanfrage mit einer Dienstbeschreibung bestehend aus Dienst-Attributen statt und wird hier näher erläutert. Die Dienstanfrage ist der Vorgang der Anfrage vom ET-Klient nach dem ET-Dienst und dient dem Herunterladen des ET-Proxy.

Die Klasse `DiscoveryEvent` der Jini-API bietet die Methode `getRegistrars()`. Diese Methode gibt ein Array mit Instanzen von `ServiceRegistrar` zurück. Jede Instanz ist ein Dienstverwaltungs-Proxy, der verwendet werden kann, um mit einem bestimmten Lookup-Service zu kommunizieren. Die Schnittstelle `ServiceRegistrar` bietet die Methode `Lookup()`, die ein Objekt `ServiceMatches` zurückgibt. Die Klasse `ServiceMatches` wiederum bietet zwei Eigenschaften: `totalMatches` zur Bestimmung der Anzahl der Dienste eines Lookup-Service und `items` zum Zugriff auf Attribute, Service-ID usw. eines Dienstes.

Nachdem der ET-Klient den Lookup-Service gefunden hat, fragt er ihn nach einem ET-Dienst. Der ET-Klient verwendet eine Dienstbeschreibung mit Attributen, die den ET-Dienst beschreiben. Hier wird der Name des Dienstes angegeben (siehe den *kursiven* Code). Damit bekommt der ET-Klient eine Liste aller verfügbaren elektronischen Tafeln mit ihren vollständigen Attributen (siehe den **fettgedruckten** Code). Der Benutzer kann dann die gewünschte elektronische

Tafel mit ihren vollständigen Attributen von der Liste auswählen. Der ET-Klient verwendet diese Attribute für die Dienstanfrage noch mal aber mit einer vollständigen Dienstbeschreibung beispielsweise zusätzlich mit der Lokation der elektronischen Tafel (siehe den *kursiven* Code).

```

class MyClient implements DiscoveryListener
{
    ServiceRegistrar[] registrars;
    // Start der Suche nach Lookup-Services
    discovery = new LookupDiscovery (LookupDiscovery.ALL_GROUPS);
    discovery.addDiscoveryListener (this);
    ...
    // Die Methoden der Schnittstelle DiscoveryListener
    public void discarded (DiscoveryEvent evt) {...}
    public void discovered (DiscoveryEvent evt)
    {
        ...
        int i, j, k; // für Schleife for
        ServiceID id; // für den Lookup-Service
        ServiceID serviceID; // für den ET-Dienst
        Entry[] attributes, attrib; // zum ersten Mal und zweiten Mal der Suche
        ServiceTemplate template, template1; // Dienstbeschreibung
        Object object;
        ServiceMatches matches;
        String room, floor, building;
        boolean ready, available, allRight;
        ElectronicBoardInterface electronicBoardInterface;
        // Dienstanfrage bei jedem gefundenen Lookup-Service
        registrars = evt.getRegistrars();
        for (i = 0; i < registrars.length; i++)
        {
            id = registrars[i].getServiceID (); // ServiceID des Lookup-Service
            attributes = new Entry[1];
            attributes[0] = new Name ("ServiceSmartboard"); // Attribute nicht genau beschrieben
            template = new ServiceTemplate (null, null, attributes);
            try
            {
                // jeden gefundenen ET-Dienst mit seinen Attributen angezeigt
                matches = registrars[i].lookup(template, 10);
                for (k = 0; k < matches.totalMatches; k++)
                {
                    serviceID = matches.items[k].serviceID;
                    for (j = 0; j < matches.items[k].attributeSets.length; j++)
                    { // eine Liste der ET-Dienst mit vollständigen Attributen
                        object = matches.items[k].attributeSets[j];
                        System.out.println ("client: " + k + ":" + j + ": attrib Set: " + object);
                    }
                }
            }
        }
    }
}

```

```
    }  
  } ...  
  // die Dienstbeschreibung mit vollständigen Attributen  
  attrib = new Entry[2];  
  attrib[0] = new Name("ServiceSmartboard");  
  attrib[1] = new Location(floor, room, building);  
  template1 = new ServiceTemplate (null, null, attrib);  
  try  
  {  
    electronicBoardInterface =  
      (ElectronicBoardInterface)registrars[i].lookup (template1);  
    if(electronicBoardInterface == null)  
    {  
      System.exit(0);  
    }  
    // richtiger ET-Dienst gefunden, seine Methoden werden aufgerufen  
    if (electronicBoardInterface instanceof ElectronicBoardInterface)  
    {  
      electronicBoardInterface.sendInfo(userName, address);  
      allRight = electronicBoardInterface.allRight();  
      ...  
    }...  
  }
```

4.3 Kommunikation zwischen Server und Klient

In diesem Kapitel wird zunächst die Übersicht der Methoden der Klassen des ET-Servers und ET-Dienstes vorgestellt (Bild 4.1). Danach wird die Implementierung der Kommunikation zwischen dem ET-Server und dem ET-Klienten bzw. dem ET-Proxy beschrieben.

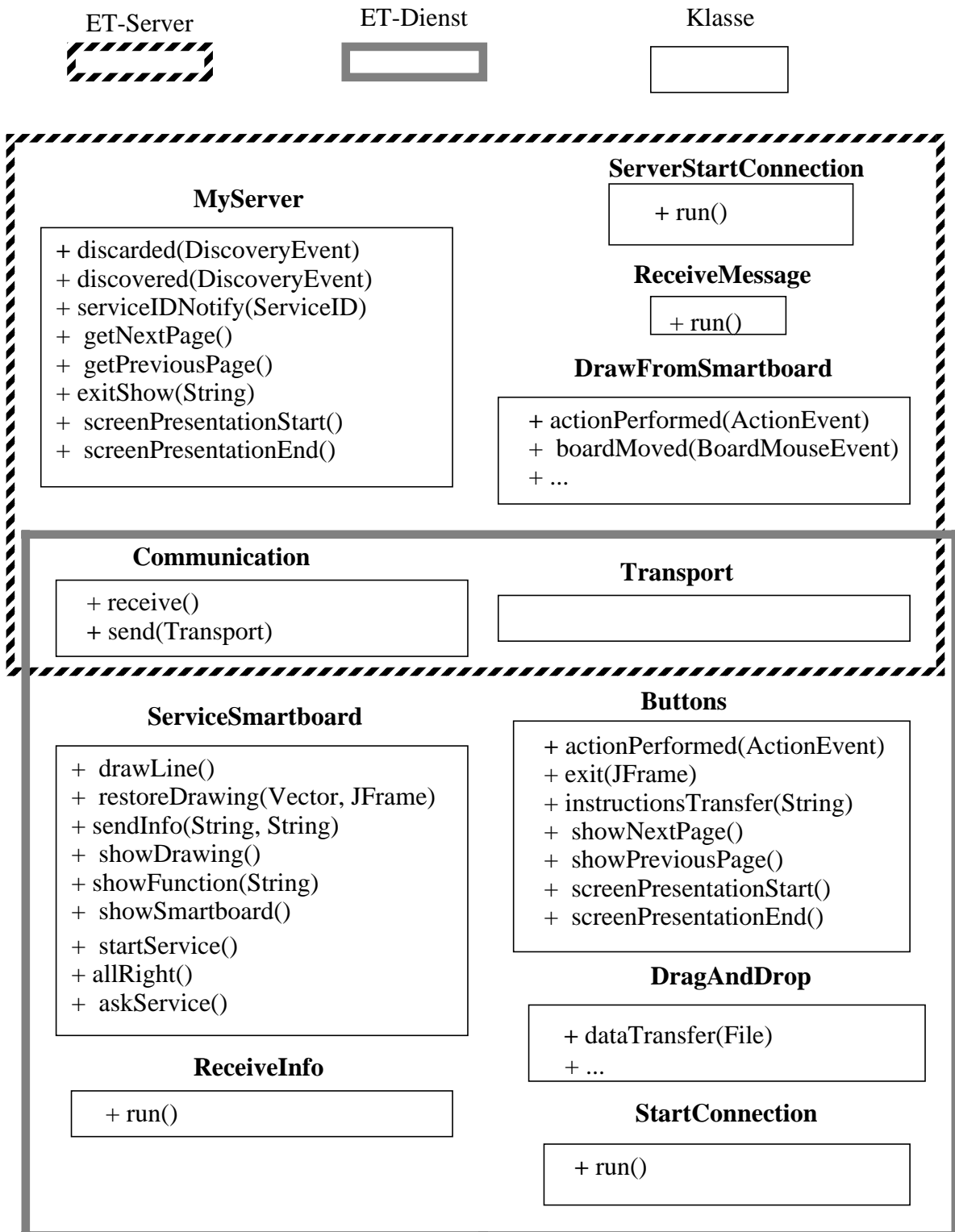


Abbildung 4.1: Die Methoden der Klassen

4.3.1 Kommunikation zwischen ET-Server und ET-Proxy

Der ET-Server und der ET-Proxy kommunizieren über das Netzwerk miteinander. Auf der Server-Seite wird die Methode *run()* der Klasse **ServerStartConnection** aufgerufen. Der ET-Server wartet daraufhin auf Anfragen des ET-Proxy. Auf der ET-Proxy-Seite wird die Methode der Klasse **StartConnection** *run()* aufgerufen und die Verbindung zwischen ihnen wird aufgebaut. Über diese Verbindung werden Nachrichten, die Präsentationsdaten und die Notizen von der Tafel durch die Methode *send()* und *receive()* der Klasse **Communication** gesendet und empfangen.

4.3.2 Übertragen der Präsentationsdaten vom ET-Proxy an den ET-Server

Wenn der ET-Klient den ET-Proxy startet, bekommt er mittels Methode *showSmartboard()* der Klasse **ServiceSmartboard** ein Fenster mit dem Titel "Smartboard" auf dem Klient-Bildschirm zum Starten der Präsentation. Die angezeigten Präsentationsdaten werden mit Drag&Drop in das Fenster gezogen und durch die Methode *dataTransfer()* der Klasse **DragAndDrop** (siehe Bild 4.1) an den ET-Server übertragen. Nachdem die Präsentationsdaten gesendet wurden, erscheint ein Steuerungsmenü mit dem Titel "PPT Functions" (PPT: Powerpoint) auf dem Klient-Bildschirm über die Methode *showFunction()* der Klasse **ServiceSmartboard**.

4.3.3 Steuern der PPT-Präsentation vom Laptop aus

Die Klasse **Buttons** bietet verschiedene Methoden z.B. *showNextPage()*, *showPreviousPage()* usw. (siehe Bild 4.1) für die Steuerung der PPT-Präsentation. Jeder Menüpunkt auf dem Steuerungsmenü entspricht einer dieser Methode. Wenn die Präsentationsdaten eine Powerpoint (PPT)-Datei ist, sind alle Menüpunkte des Steuerungsmenüs aktiv. Wenn die Präsentationsdaten eine andere Datei (wie Datei mit Endung bmp, gif oder jpg) ist, ist nur der Menüpunkt "Exit" aktiv, weil andere Menüpunkte nicht unterstützt werden.

Wenn ein Knopf gedrückt wird, wird der entsprechende Befehl an den ET-Server gesendet. Dort wird der Befehl mittels Windows Script Host (WSH) ausgeführt und auf dem Server-Bildschirm und damit auf der elektronische Tafel angezeigt. Dazu werden die Methoden der Klasse **MyServer** (siehe Bild 4.1) wie *getNextPage()*, *getPreviousPage()* usw. aufgerufen.

Mit Hilfe des Windows Script Host wird die PPT-Präsentation vom Server gesteuert. Der WSH wird hier kurz erklärt, die Details des WSH siehe [Born2000]. Der WSH ermöglicht den Zugriff auf die Schnittstelle von Anwendungsprogrammen wie Powerpoint. Der WSH erlaubt es, in Da-

teilen gespeicherte Skriptprogramme zu laden und auszuführen, die in VBScript bzw. JScript geschrieben wurden. Um ein neues Objekt anzulegen bzw. auf ein Objekt zuzugreifen, bietet der WSH zwei Methoden: *CreateObject* und *GetObject*. Die beiden Methoden werden für die Steuerung der PPT-Präsentation verwendet (siehe den Code in Anhang A2).

4.3.4 Übertragen der Notizen von der elektronischen Tafel

Die Notizen, die von Benutzer auf der elektronischen Tafel gemacht werden, werden mittels der Smartboard-API und der Methoden der Klasse **DrawFromSmartboard** an den ET-Server übertragen. Die Klasse **DrawFromSmartboard** ist eine Erweiterung der Klasse *TestFrame* der Smartboard-API. Wenn der ET-Server die Verbindung mit dem ET-Proxy aufgebaut hat, erzeugt er ein Objekt der Klasse **DrawFromSmartboard**. Dieses Objekt öffnet ein Fenster mit dem Titel "Drawing from Smartboard" auf dem Server-Bildschirm. Was auf der Tafel gezeichnet wird, wird genauso auf diesem Fenster angezeigt und diese Notizen werden an den ET-Proxy bzw. den ET-Klient weiter gesendet und auf dem Fenster auf dem Klient-Bildschirm angezeigt. Dies geschieht durch die Methoden *restoreDrawing()* und *drawLine()* der Klasse **ServiceSmartboard**.

4.4 Benutzungsoberfläche

Dieses Kapitel beschreibt die Funktionalität sowie die Art und Weise der Interaktion zwischen dem Benutzer und dem ET-Dienst.

4.4.1 Ausgaben in der Kommandozeile

Zuerst startet der Benutzer Jini-Software und sieht auf dem DOS-Fenster die Ausgaben des Suchvorgangs: wie viele Lookup-Services gefunden wurden, wie viele ET-Dienste sich bei jedem Lookup-Service registriert haben und die Attributen sowie die ServiceIDs jedes ET-Dienstes (Bild 4.2).

```
client: Begin search for a lookup service...

client: Found = 1 lookup service[s]
client: ServiceRegistrar = com.sun.jini.reggie.RegistrarProxy@fa60f421
client: ServiceID = 454f3af5-5a8d-47f4-abb7-feca4e1577ea
client: ServiceMatches = net.jini.core.lookup.ServiceMatches@2bb0589
client: Number of matches = 2

client: 0: svc item:net.jini.core.lookup.ServiceItem@790b0589
client: 0: svc ID: bc50e1fb-8de4-4d18-84a1-c42a5cb911d7
client: 0: svc object: ServiceSmartboard@7b6f0589
client: Length : 2
client: 0:0: attrib Set: net.jini.lookup.entry.Name(name=ServiceSmartboard)
client: 0:1: attrib Set: net.jini.lookup.entry.Location(floor=0,room=183,building=20)

client: 1: svc item:net.jini.core.lookup.ServiceItem@6e870589
client: 1: svc ID: 8b526a35-30f8-4afd-b13b-b15a5fb11e27
client: 1: svc object: ServiceSmartboard@614b0589
client: Length : 2
client: 1:0: attrib Set: net.jini.lookup.entry.Name(name=ServiceSmartboard)
client: 1:1: attrib Set: net.jini.lookup.entry.Location(floor=2,room=121,building=20)
```

Abbildung 4.2: Die Ausgaben in der Kommandozeile

4.4.2 Auswahl der elektronischen Tafel

Gleichzeitig erscheint ein Fenster mit dem Titel "Electronic Board List", das die Lokationliste der angebotenen elektronischen Tafeln in verschiedenen Räume anzeigt. Die Lokation ist die Information über Gebäude, Stockwerk und Raum, in dem sich die elektronische Tafel befindet. Diese Informationen sind Attribute der ET-Dienste (siehe Bild 4.2: die fettgedruckten Attribute). Der Benutzer wählt die elektronische Tafel des Raumes aus, in dem er steht und in dem er seine Präsentation halten will. Nach der Auswahl drückt er den Knopf "OK" (siehe Bild 4.3) und die Lokationinformation wird für die weitere Dienstsuche aufgenommen. Ausserdem gibt der Benutzer in dem Textfeld seinen Namen an. Dieser Name wird für die Ablehnungsmeldung verwendet, falls andere Benutzer gleichzeitig diese elektronische Tafel benutzen wollen. Diese Auswahl könnte automatisch geschehen, wenn der Laptop des Benutzers mit einem Positionssystem ausgestattet ist (wie Aktive Badge System).

Die Nummer 20.0.183 im Bild 4.3 bedeutet, dass sich die elektronische Tafel im Gebäude 20, im Erdgeschoss und im Raum 183 (analog für 20.2.121) befindet.

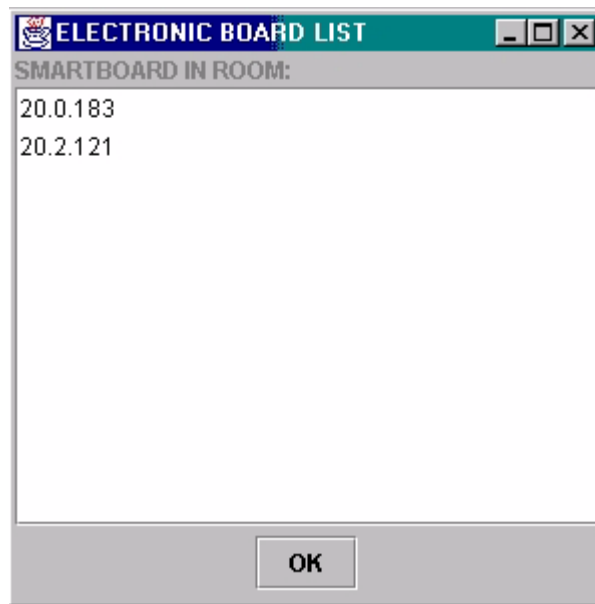


Abbildung 4.3: Die Lokationliste der elektronische Tafeln

Wenn der Benutzer eine elektronische Tafel in einem falschen Raum auswählen würde, gibt es zwei Fälle:

- Wenn die elektronische Tafel in Benutzung ist, bekommt er eine Meldung mit der Ablehnung seiner Anfrage. Danach kann er die Jini-Software erneut starten und die richtige Lokation aus der Liste auswählen.
- Wenn die elektronische Tafel nicht in Benutzung ist und wenn der Benutzer sie falsch ausgewählt hat, läuft trotzdem der ET-Dienst. Er sieht jedoch seine Präsentationsdaten nicht, weil die Präsentationsdaten auf der elektronischen Tafel des anderen Raumes angezeigt werden. In diesem Fall kann dem Benutzer keine Meldung angezeigt werden, weil der ET-Server die geographischen Lokation des Benutzers nicht feststellen kann. Der ET-Server kann eine Unterstützung geben. Zu Beginn wird nochmals ein Fenster mit dem Titel "Announcement" mit der Lokation des gewählten ET-Dienstes angezeigt (siehe Bild 4.4). Diese Anzeige dient dem Benutzer als Hinweis auf die Lokation der elektronischen Tafel.

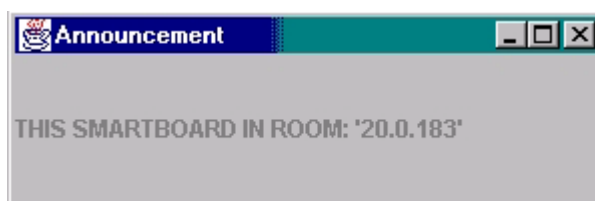


Abbildung 4.4: Die Anzeige der Lokation der elektronischen Tafel

4.4.3 Das Fenster für den Präsentationsstart

Nachdem die Lokation einer elektronischen Tafel ausgewählt und für die Suche des ET-Dienstes benutzt wurde, wird der ET-Proxy von dem ET-Klient heruntergeladen. Der ET-Proxy öffnet ein Fenster mit dem Titel "Smartboard" auf dem Klient-Bildschirm für den Präsentationsstart. Der Benutzer kann seine Präsentation starten, indem er die Präsentationsdatei mittels Drag&Drop in das Fenster "Smartboard" zieht (siehe Bild 4.5). Präsentationsdateien können nicht gleichzeitig geöffnet werden, sondern nur nacheinander, nach dem Beenden der vorherigen Präsentationsdatei. Macht der Benutzer das Fenster "Smartboard" zu, wird das Programm beendet.

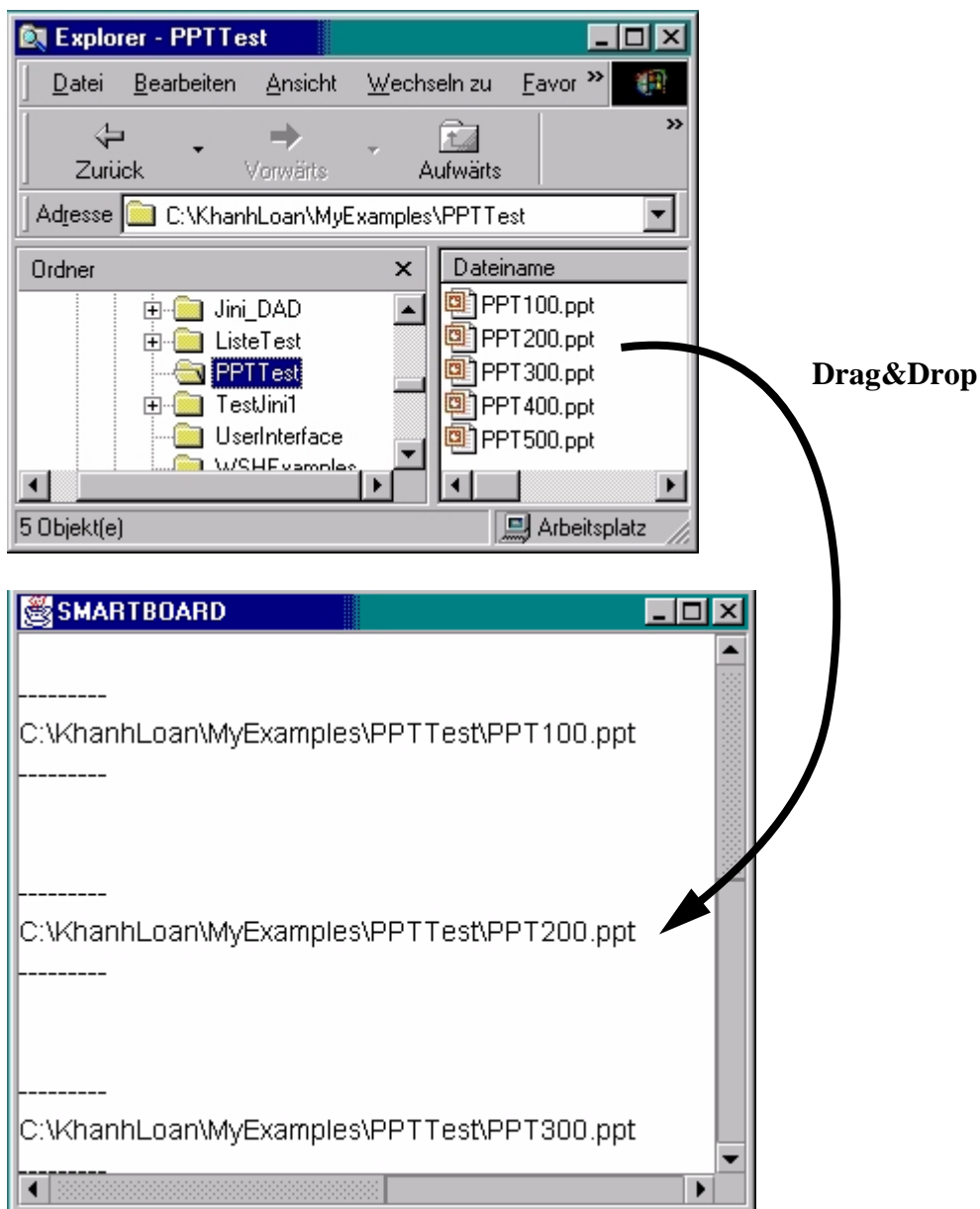


Abbildung 4.5: Der Vorgang des Präsentationsstarts

Danach erscheint ein Steuerungsmenü auf dem Klient-Bildschirm (siehe Bild 4.6). Wenn die Präsentationsdaten eine PPT-Datei ist, sind alle Menüpunkte aktiv. Der Benutzer kann mit diesen Menüpunkten die nächste bzw. vorherige Seite anzeigen oder die Bildschirmpräsentation starten und beenden. Wenn die Präsentationsdatei keine PPT-Datei ist (wie z.B. mit Endung bmp, jpg oder gif), ist nur der Menüpunkt "EXIT" aktiv für das Beenden der Darstellung der Präsentationsdaten. Das Programm für die Anzeige der Dateien mit Endung bmp, jpg und gif wird zu Beginn in eine Konfigurationsdatei jenach dem Rechner eingestellt.



Abbildung 4.6: Das Steuerungsmenü

4.4.4 Das Fenster für die Anzeige der Notizen von der elektronischen Tafel

Auf dem Klient-Bildschirm erscheint ein Fenster mit dem Titel "Drawing From Smartboard" für die Anzeige der Notizen von der elektronischen Tafel. Im Bild 4.7 zeigt ein Beispiel. Die Notizen werden vom ET-Server an den ET-Klient übertragen und auf dem Klient-Bildschirm angezeigt.

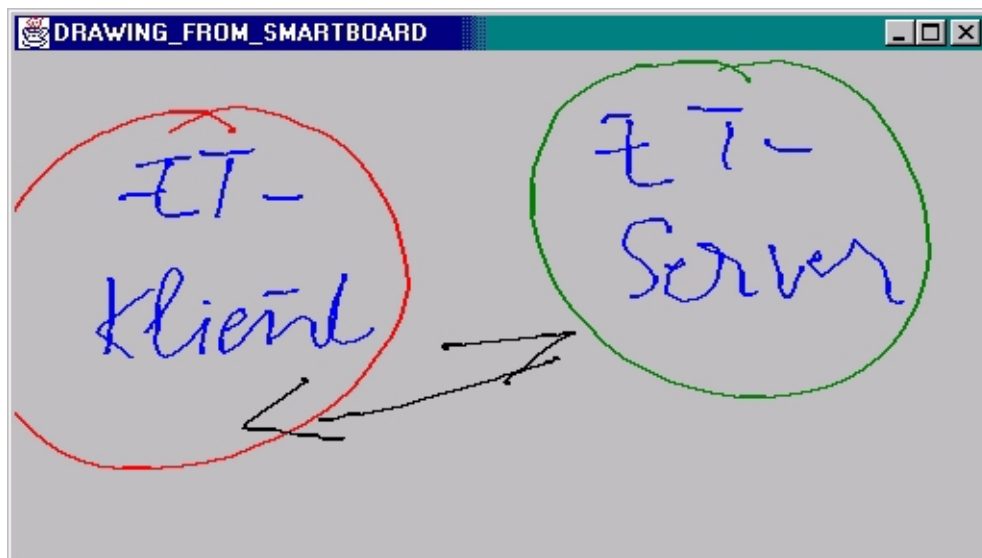


Abbildung 4.7: Das Fenster für die Anzeige der Notizen

5 Messungen

Der Schwerpunkt dieser Studienarbeit liegt auf Jini. Deshalb sollte die Benutzbarkeit und die Brauchbarkeit von Jini festgestellt werden, indem für diese Arbeit Messungen durchgeführt und die Messergebnisse analysiert werden.

Für die Messungen wurde eine Konfiguration (Bild 5.1) gewählt, bei der sich der Lookup-Service in einem anderem Subnetz befindet als der Server. Die Subnetze sind über einen multicastfähigen-Router verbunden. Bei dieser Konfiguration handelt es sich um einen Fall, der in der Praxis denkbar ist, wenn nicht für jedes Subnetz ein eigener Lookup-Service installiert werden soll. Dieser Fall ist wesentlich kritischer als eine Konfiguration, bei der sich alle Komponenten in einem Subnetz befinden, da die Kommunikation durch den Router zusätzlich verzögert wird. Bei Vergleichsmessungen in einem Subnetz ergab sich als einziger Unterschied eine etwas kürzere Dauer der Dienstanmeldung.

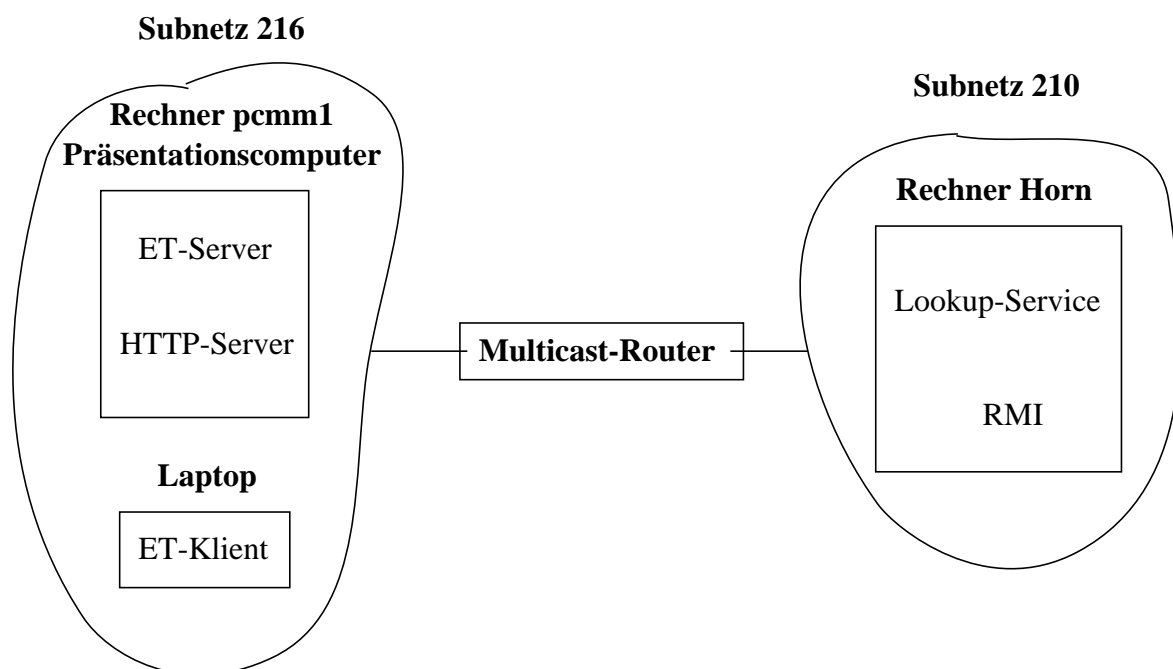


Abbildung 5.1: Konfiguration für die Messungen

Wegen der Abhängigkeit von der Latenzzeit des Netzwerks und der Rechner sind die Messwerte unregelmäßig. Daher werden im weiteren nur Mittelwerte angegeben.

5.1 Messwerte

5.1.1 Dauer der Dienstanmeldung

Auf der Server-Seite wird die Dauer der Anmeldung des ET-Dienstes gemessen. Diese Zeit besteht aus der Dauer der Suche nach Lookup-Services (Discovery-Vorgang) und der Dauer der Anmeldung des ET-Dienstes (Join-Vorgang). Der Zeitraum der Messungen wurde ab dem Start des Discovery-Vorgang bis zum Zeitpunkt des Eintreffens der Service-ID des ET-Dienstes durchgeführt. Die Werte der Messungen werden im Bild 5.2 gezeigt. Die Werte, die extrem oder unüblich sind, werden im Bild 5.2 nicht gezeigt. Der Mittelwert ist aus 10 Messungen berechnet.

Die Messungen der Dienstanmeldung vom ET-Server wurden für drei Fälle mit einem, zwei und drei Lookup-Services durchgeführt. Der ET-Server ist für die elektronische Tafel im Raum zuständig. Das Netzwerk hat 10Mbit Bandbreite.

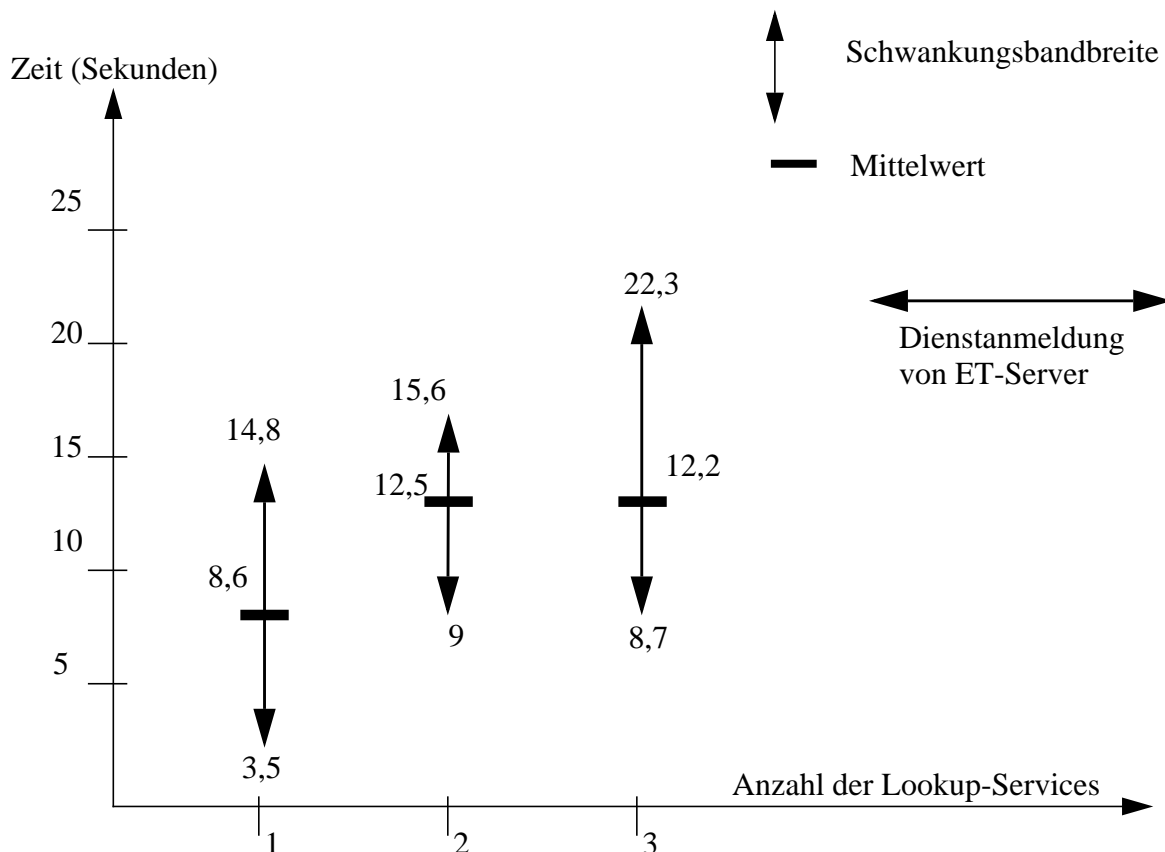


Abbildung 5.2: Die Zeitgraphik der Dienstanmeldung

5.1.2 Dauer der Dienstanfrage

Auf der Klient-Seite wird die Dauer der Anfrage des ET-Dienstes gemessen. Diese Zeit besteht aus der Dauer der Suche nach Lookup-Services und der Dauer der Anfrage nach dem ET-Dienst. Die Dauer des Suchvorgangs nach einem Lookup-Service ist kleiner als 1ms (oder 0,001s) und wird deshalb hier nicht beachtet. Praktisch die gesamte Zeit wird für die Übertragung der Klassendateien vom HTTP-Server benötigt. Die Klassendateien oder der Code des ET-Proxy sind ungefähr 40 KB. Die Werte der Messungen der Dienstanfrage vom ET-Klient werden im Bild 5.3 gezeigt. Die Werte, die extrem oder unüblich sind, werden im Bild 5.3 nicht gezeigt. Die Messungen wurden für drei Fälle mit einem, zwei und drei Lookup-Services durchgeführt.

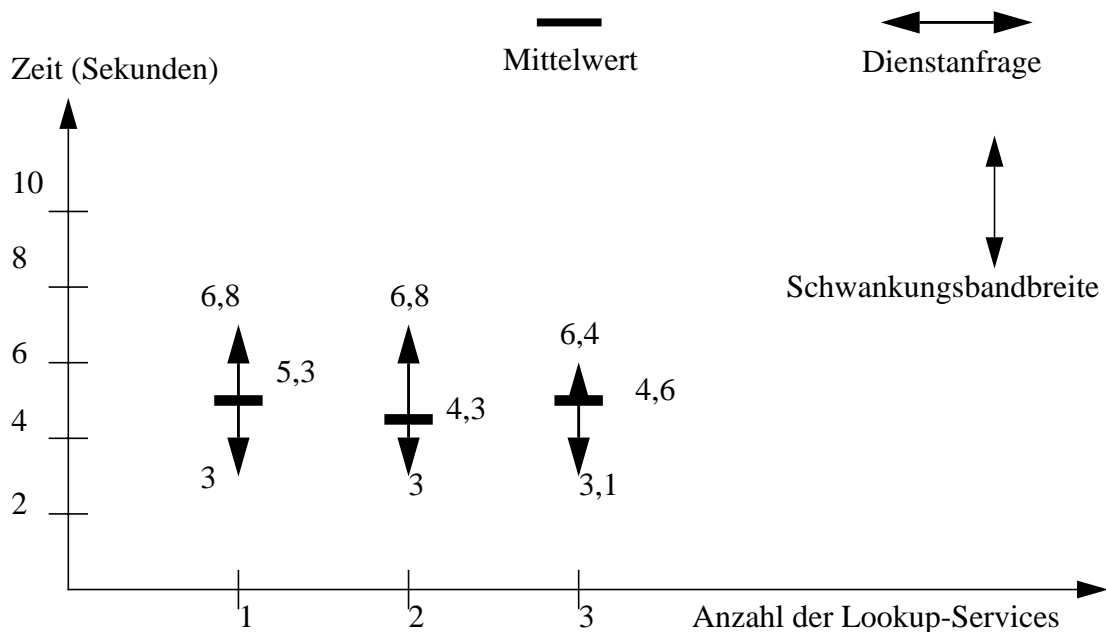


Abbildung 5.3: Die Zeitgraphik der Dienstanfrage

5.1.3 Dauer des Transports der PPT-Dateien

Der Zeitraum, der für den Transport der Powerpoint (PPT)-Dateien vom ET-Klient bis zum ET-Server benötigt wird, wurde gemessen. Es wurden Messungen mit PPT-Dateien verschiedener Größe von 100 KB bis 500 KB durchgeführt (Bild 5.4). Auf der Klient-Seite wird dazu der Zeitpunkt t_1 des Sendens einer Datei an den ET-Server genommen. Nach dem Empfang dieser Datei sendet der ET-Server sie sofort an dem ET-Klient zurück und der Zeitpunkt t_2 des Empfangs der Datei auf der Klient-Seite wird genommen. Hieraus wurde dann der Wert für eine

Richtung berechnet.

Die Gesamtzeit vom Start des Programms bis zur Anzeige der Präsentationsdatei (mit der Grösse bis zu 500KB) auf der elektronischen Tafel liegt zwischen 5s und 8s.

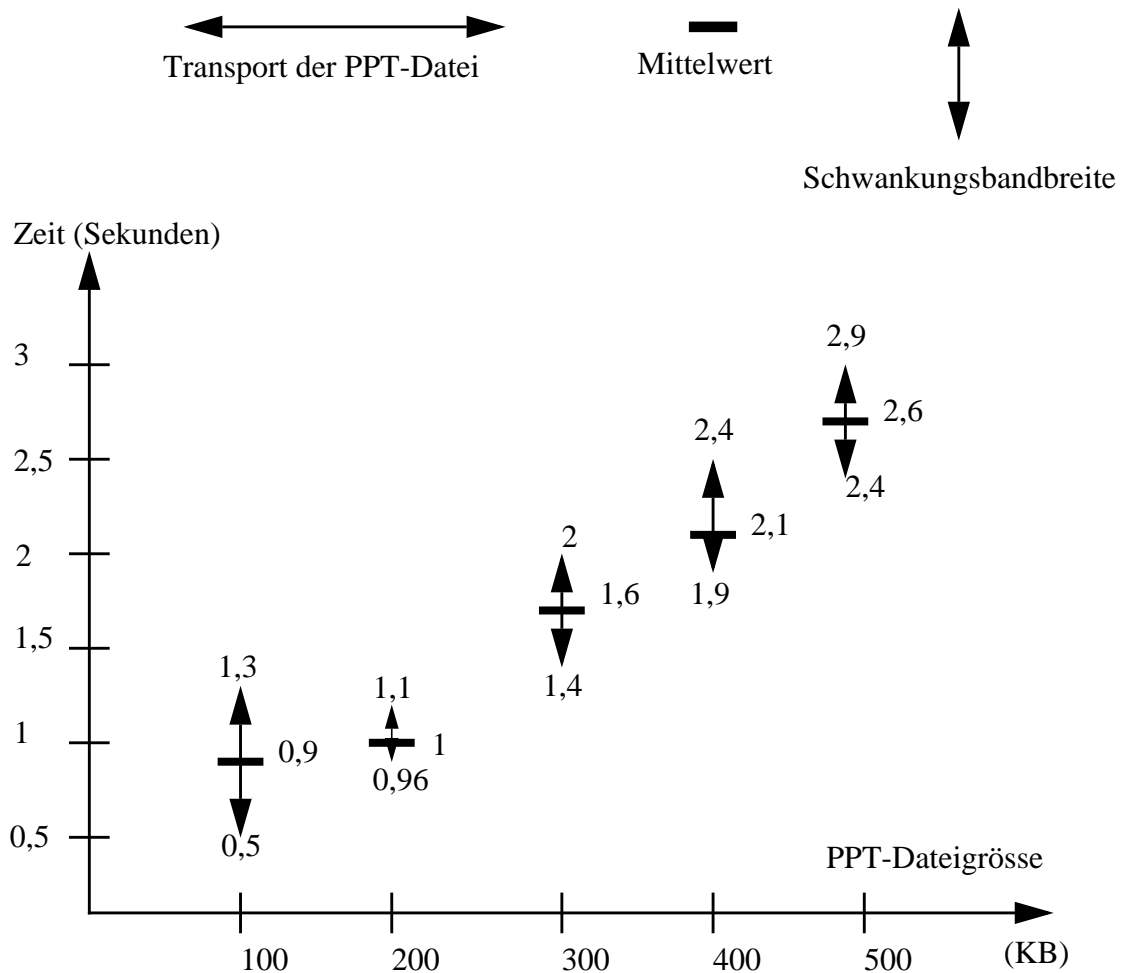


Abbildung 5.4: Die Zeitgraphik des Dateitransports

5.2 Ergebnisanalyse

5.2.1 Analyse der Messungen der Dienstanmeldung

Die Dauer der Dienstanmeldung ist von der Anzahl der Lookup-Services abhängig. Je mehr Lookup-Services aktiv vorhanden sind, desto grösser ist die Zeit, die für die Dienstanmeldung benötigt wird. Dieser Anstieg ergibt sich, weil sich der ET-Dienst nicht gleichzeitig bei allen gefundenen Lookup-Services anmelden kann. Die Anmeldung bei dem ersten gefundenen

Lookup-Service muss sequentielle erfolgen, da der ET-Dienst bei der ersten Anmeldung eine eindeutige Service-ID bekommt und diese Service-ID bei den Anmeldungen bei den anderen Lookup-Services verwenden muss. Deshalb kann der erste Anmeldungsvorgang nicht gleichzeitig bei allen aktiven Lookup-Services stattfinden, sonst würde der ET-Dienst mehrere Service-IDs bekommen. Die Service-ID des ET-Dienstes wäre dann nicht mehr eindeutig. Alle weitere Anmeldungen des ET-Dienstes können dann parallel erfolgen.

Die Dauer der Dienstanmeldung ist für die meisten Anwendungen unerheblich, weil ein Dienst im normalen Fall ständig läuft. Auch für Anwendungen, bei denen sich der Dienst öfter an- und abmeldet, ist die Anmeldedauer von einigen Sekunden akzeptabel.

5.2.2 Analyse der Messungen der Dienstanfrage

Die Dauer der Dienstanfrage ist unabhängig von der Anzahl der Lookup-Services. Nach dem Empfang der ersten positiven Antwort (bzw. des ET-Proxy) von einem Lookup-Service ignoriert der ET-Klient weitere Antworten der anderen Lookup-Services. Der grösste Teil der Zeit wird für das Herunterladen des ET-Proxy benötigt. Dies muss bei der Realisierung von Diensten mit aufwendigen Proxy-Objekten beachtet werden. Für den ET-Dienst ergibt sich jedoch eine ausreichend kurze Zeit für die Dienstanfrage.

5.2.3 Analyse der Messungen des Transports der PPT-Dateien

Je grösser die PPT-Dateien sind, desto mehr Zeit wird für den Transport der PPT-Dateien benötigt. Dies ist abhängig von der Bandbreite des Netzwerkes und beträgt auch für grösseren Dateien nur wenige Sekunden. Die Automatisierung der Übertragung durch den ET-Dienst bringt in jedem Fall einen Vorteil gegenüber einer Übertragung mit FTP oder Diskette, da FTP auch nur die Bandbreite des Netzwerkes zur Verfügung hat und bei Disketten der Zeitaufwand in jedem Fall grösser ist.

Die Gesamtzeit vom Start des Programms bis zur Anzeige der Präsentationsdatei (mit der Grösse bis zu 500KB) auf der elektronischen Tafel liegt zwischen 5s und 8s. Diese Zeit ist abhängig von

- der Grösse der Klassendateien des ET-Proxy,
- der Grösse der PPT-Dateien und
- der Latenzzeit des Netzwerkes.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Diese Arbeit hat sich mit dem Entwurf und der Implementierung eines Dienstes für elektronische Tafeln mit mobilen Klienten befasst. Es wurde dazu Jini eingesetzt und daher eine serverseitige Implementierung der Funktionen gewählt. Der Klient enthält so nur eine einfache Funktionalität (wie Dienstanfrage und Dienststart). Diese Lösung bringt den Vorteil einer sehr generischen Klient Implementierung, die auch mit sehr unterschiedlichen elektronischen Tafeln zusammenarbeitet. Durch die lokale Ausführung auf dem Klient-Rechner wird auch eine flüssigere Interaktion mit dem Benutzer erreicht. Bei dem dadurch komplexeren ET-Proxy ergibt sich jedoch das Problem einer längeren Ladezeit von zusätzlich benötigter Klassen vom HTTP-Server.

Die Startzeit von ca. 8s vom Start des Programms bis zur Anzeige der Präsentationsdaten auf der elektronischen Tafel, ist eine erhebliche Verbesserung gegenüber der bisherigen Methoden der Übertragung der Präsentationsdaten mit FTP oder Diskette. Der Arbeitsaufwand für den Benutzer wird durch die Automatisierung der Übertragung sehr verringert. Ausserdem kann man mit Hilfe des ET-Dienstes die Präsentation auf der elektronischen Tafel vom eingenen Laptop aus steuern. Dafür muss die Software der elektronischen Tafel und der sonstigen für die Präsentation genutzten Software eine von Java nutzbare Schnittstelle besitzen. Der Windows Scripting Host wurde für die Steuerung der Präsentation von PPT-Dateien verwendet. Für die Verbindung mit der elektronischen Tafel konnte eine Java-Schnittstelle genutzt werden.

Jini hat sich als einfach zu erlernen und einfach einzusetzen erwiesen. Der Implementierungsaufwand wird durch mitgelieferte Bibliotheken zusätzlich verringert. Als Problem hat sich der Mangel einer genauen geographischen Lokation des Klienten erwiesen. Deshalb kann es zu Problemen kommen wie sie in Kapitel 4.4.2 beschrieben sind. Dieses Problem wird sich nicht einfach beseitigen lassen, da heutige Netze keine entsprechenden Funktionen unterstützen. Für die Weiterentwicklung von Jini sollte aber eine entsprechende Erweiterung vorgesehen werden. Trotzdem ist Jini gut geeignet für Anwendungen, die einfach bedienbar sein sollen oder bei denen Verbindungen dynamisch wechseln. Zu beachten sind die Bedingungen:

- Jini funktioniert nur mit Java-fähigen Geräten oder Systemen (von denen noch wenig vorhanden sind).

- Jini funktioniert nur mit IP-Netzwerken (erforderlich für die Discovery-Protokolle).
- Es muss eine Standardschnittstelle für den Dienst vorhanden sein.

Der letzte Punkt ist eigentlich keine notwendige Bedingung, da Dienstschnittstellen beliebig definiert werden können. Um jedoch die Zusammenarbeit verschiedener Klient-Implementierungen mit unterschiedlichen Dienst-Implementierungen zu gewährleisten, müssen entsprechende Standards definiert und eingehalten werden.

6.2 Ausblick

Diese Arbeit könnte noch in Richtung der Kooperation mit anderen Benutzern weiter ausgebaut werden. Ein Beispiel wäre: Ein Vortrag findet in einem Raum mit einer elektronischen Tafel statt. Einige Zuhörer befinden sich in anderen Räumen wie bei einer Tele-Konferenz. Der Vortragende wird dabei zum Moderator. Der ET-Dienst ermöglicht es, dass der Moderator bestimmen kann, welcher Zuhörer Anmerkungen auf der elektronischen Tafel machen oder den Inhalt der Präsentation des Moderators ergänzen darf. Auf der elektronischen Tafel wird dazu nicht nur die Präsentation des Moderators, sondern noch ein Fenster für die Diskussion mit den Zuhörern angezeigt. Für Zuhörer, die nicht im Raum anwesend sind, und für Eingaben durch Zuhörer, wird der Inhalt der elektronischen Tafel auf den Laptops der Zuhörer angezeigt.

In dieser Arbeit wurde mit Jini die Steuerung einer elektronischen Tafel realisiert. Jini kann ebenso die Realisierung und Benutzung anderer Anwendungen vereinfachen, z.B. die Steuerung von Geräten wie Drucker, Scanner, dem Alarmsystem eines Hauses, Lichtschalter, Schliessvorrichtungen von Türen usw. oder die Zusammenarbeit zwischen Stereoanlagen, CD- und DVD-Player, Video-Recorder und Fernsehern.

Anhang

A1. Die Befehlszeilen

In diesem Anhang werden die Parameter der Befehlszeilen für den Start von Jini-Laufzeitdiensten erklärt.

Start den HTTP-Server:

Um den HTTP-Server zu starten, wird die folgende Befehlszeile eingegeben:

```
java -jar <tools-jarfile> [-port <port-number>] [-dir <document-root-dir>] [-trees] [-verbose]
```

Beispiele für Windows und Solaris:

Unter Windows: `java -jar c:\jini1_0\lib\tools.jar -port 8080 -dir c:\jini1_0\lib -verbose`

Unter Solaris: `java -jar /files/jini1_0/lib/tools.jar -port 8080 -dir /files/jini1_0/lib -verbose`

Im Paket *tools.jar* der Jini-Software befindet sich ein HTTP-Server. Der HTTP-Server arbeitet standardmässig mit dem Port 8080. Mit Hilfe der Option `-port <port-number>` kann man eine andere Portnummer auswählen. Mittels der Option `-dir <document-root-dir>` kann man das Stammverzeichnis bestimmen, das vom Server verwendet wird. Die Option `-verbose` zeigt jede an den HTTP-Server gestellte Anfrage und deren Herkunft.

Start den Lookup-Service:

Um den Lookup-Service zu starten, wird die folgende Befehlszeile eingegeben:

```
java -Djava.security.policy=<security_policy> -jar <lookup-server-jarfile> <lookup-client-codebase> <lookup-policy-file> <output-log-dir> <lookup-service-group>
```

Die Option `-Djava.security.policy` ermöglicht es, eine Zeiger auf die Sicherheitsrichtliniendatei anzugeben. Die Java Virtual Maschine (JVM) verwendet diese Datei, um das Programm auszuführen. Die Option `-jar` weist darauf hin, dass die `main()`-Routine des Programms sich in der angegebenen JAR-Dateien befindet. An dieser Stelle wird die JAR-Datei des Lookup-Service (*reggie.jar*) übergeben, welche sich im Verzeichnis *lib* der Jini-Software befindet. Der nächste Teil der Befehlszeile ist die Codebase, die angibt, von wo aus Code zu den Klienten heruntergeladen wird. An dieser Stelle wird eine URL angegeben, die auf die Datei *reggie-dl.jar* weist, die den für das Herunterladen zu Anfragen verwendeten Code enthält. Die Option `<lookup-policy-file>` legt die Sicherheit für künftige Aktivierungen des Dienstes durch den Ak-

tivierungs-Daemon fest. Die nächste Option `<output-log-dir>` bestimmt das Verzeichnis, in dem Reggie seine Protokolle speichert. Die letzte Option bestimmt eine Liste von Gruppen, denen der Lookup-Server Dienste anbieten soll.

Beispiele für Windows und Solaris:

Unter Windows: `java -Djava.security.policy=c:\jini1_0\example\lookup\policy
-jar c:\jini1_0\lib\reggie.jar
http://hostname/reggie-dl.jar
c:\jini1_0\example\lookup\policy
c:\temp\reggie_log
public`

Unter Solaris: `java -Djava.security.policy=/files/jini1_0/example/lookup/policy
-jar /files/jini1_0/lib/reggie.jar
http://hostname/reggie-dl.jar
/files/jini1_0/example/lookup/policy
/tmp/reggie_log
public`

Hostname ist der Name des Rechners, auf dem der HTTP-Server gestartet wird.

A2. Ausgewählter Pseudocode

Der Code für die Dienstanfrage:

```
import net.jini.core.entry.*;
import net.jini.core.lookup.*;
import net.jini.lookup.entry.*;
import net.jini.discovery.*;
import java.rmi.*;
import java.net.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/*****

MyClient -- Uses Discovery to find the Jini Lookup Service
It finds a desired service and gives necessary user information (name)
and chooses smartboard's location (building, floor, room) and starts it.
*****/

class MyClient implements DiscoveryListener
```

```
{
  /**
   * Address of client
   */
  String address;
  /**
   * For discovery
   */
  LookupDiscovery discovery;
  /**
   * For registrar which is service-proxy and talks to lookup service
   */
  ServiceRegistrar[] registrars;
  /**
   * accept one service-proxy of the lookup service, ignore another reply
   */
  boolean oneTime;

  /**
   * Constructor
   * Discover the desired service
   */
  MyClient ()
  {
    try
    {
      oneTime = true;
      // get address of the client
      address = InetAddress.getLocalHost().getHostAddress();
      // start discovery process
      discovery = new LookupDiscovery (LookupDiscovery.ALL_GROUPS);
      discovery.addDiscoveryListener (this);
      System.out.println ("\nclient: Begin search for a lookup service...");
    }
    catch (Exception e)
    {
      System.out.println ("MyClient(): " + e);
      e.printStackTrace();
    }
  }
  /**
   * The method discovered of DiscoveryListener interface
   */
  public void discovered (DiscoveryEvent evt)
```

```
{
    int i, j, k, position;
    int maxService = 10;
    ServiceID id, serviceID;
    Entry[] attributes, attrib;
    ServiceTemplate template, template1;
    String[] array = new String[maxService];
    Object object;
    Location loc;
    ServiceMatches matches;
    String room, floor, building, info, userName;
    boolean ready, available, availableReady;
    ElectronicBoardInterface electronicBoardInterface;
    if(oneTime == true)
    {
        oneTime = false;
        registrars = evt.getRegistrars();
        System.out.println ("client: Found = " + registrars.length + " lookup service[s]");
        /* Each cycle of this loop attempts to find MyServerInterface
        in one of the registrars that was passed to this method.
        */
        for (i = 0; i < registrars.length; i++)
        {
            id = registrars[i].getServiceID ();
            System.out.println ("client: ServiceRegistrar = " + registrars[i]);
            System.out.println ("client: ServiceID = " + id);
            attributes = new Entry[1];
            attributes[0] = new Name ("ServiceSmartboard");
            template = new ServiceTemplate (null, null, attributes);
            position = 0;
            try
            {
                matches = registrars[i].lookup(template, 10);
                System.out.println ("client: ServiceMatches = " + matches);
                System.out.println ("client: Number of matches = " + matches.totalMatches);
                for (k = 0; k < matches.totalMatches; k++)
                {
                    System.out.println ("client: " + k + ": svc item:" + matches.items[k]);
                    System.out.println ("client: " + k + ": svc ID: " + matches.items[k].serviceID);
                    serviceID = matches.items[k].serviceID;
                    System.out.println ("client: " + k + ": svc object: " + matches.items[k].service);
                    for (j = 0; j < matches.items[k].attributeSets.length; j++)
                    {
                        System.out.println ("client: Length : " + matches.items[k].attributeSets.length);
```

```
object = matches.items[k].attributeSets[j];
System.out.println ("client: " + k + ":" + j + ": attrib Set: " + object);
if(object instanceof Location)
{
loc = (Location)matches.items[k].attributeSets[j];
info = new String("");
info = info + loc.building + "." + loc.floor + "." + loc.room;
array[position] = info;
position++;
}
}
}
// smartboard's location list
LocationList locList = new LocationList(array);
locList.setSize(300, 300);
locList.addWindowListener(new WindowAdapter()
{public void windowClosing(WindowEvent e)
{
System.out.println("\nIf you want, you can start client again.\n");
System.exit(0);
}});
locList.show();
ready = locList.finished;
synchronized(locList)
{
while(!locList.finished)
{
locList.wait();
ready = locList.finished;
}
}
userName = locList.userName;
building = locList.building;
floor = locList.floor;
room = locList.room;
//desired service with determined template
attrib = new Entry[2];
attrib[0] = new Name("ServiceSmartboard");
attrib[1] = new Location(floor, room, building);
template1 = new ServiceTemplate (null, null, attrib);
try
{
electronicBoardInterface =
(ElectronicBoardInterface)registrars[i].lookup (template1);
```

```
System.out.println ("client: Service object = " + electronicBoardInterface);
if(electronicBoardInterface == null)
{
System.out.println ("PLEASE START CLIENT AGAIN!");
System.exit(0);
}
// If the correct object was returned, call one of its methods
if (electronicBoardInterface instanceof ElectronicBoardInterface)
{
System.out.println ("Client: ---> Calling sendInfo(<---");
electronicBoardInterface.sendInfo(userName, address);
availableReady = electronicBoardInterface.ready();
while(availableReady == false)
{
//ignore
availableReady = electronicBoardInterface.ready();
}
if(availableReady == true)
{
System.out.println ("Client: ---> Calling askService(<---");
available = electronicBoardInterface.askService();
if(available == false)
{
System.exit(0);
}
else
{
System.out.println ("Client: ---> Calling startService(<---");
electronicBoardInterface.startService();
}
}
}
}
catch (Exception e)
{
System.out.println ("Client: exception by lookup(template)! : " + e);
}
}
catch (Exception e)
{
System.out.println ("Client: exception by matches: " + e);
System.exit(0);
}
}
```

```
    }
}
/**
 * The methode discarded of DiscoveryListener interface
 */
public void discarded (DiscoveryEvent evt)
{
    int i;
    registrars = evt.getRegistrars ();
    System.out.println ("client: discarded() = " + registrars.length + " registrar[s]");
    for (i = 0; i < registrars.length; i++)
    {
        System.out.println ("client: registrar = " + registrars[i]);
    }
}
public static void main (String[] args)
{
    MyClient myClient;
    try
    {
        /* Set the RMI security manager and instantiate this class.
        The search for lookup servers begins in the class's
        constructor.
        */
        System.setSecurityManager (new RMISecurityManager ());
        myClient = new MyClient ();
        Thread.currentThread().join ();
        myClient.discovery.terminate ();
    }
    catch (Exception e)
    {
        System.out.println ("MyClient.main(): " + e);
    }
}
}
```

Der Code für die Dienstanmeldung ist kurz und wurde in Kapitel 4.1 gezeigt.

Skripte für die Steuerung der PPT-Dateien

```
' *****  
' File: StartPPT.vbs  
' First create a PPT object and then start it with a PPT file  
' which is read by starting the script with command - line parameter  
' *****  
  
Dim objPPT, objArgs  
fileName = ""  
Set objArgs = WScript.Arguments  
If objArgs.Count = 1 Then  
    fileName = fileName + objArgs.Item(0)  
Else  
    WScript.Echo "Error by giving command - line parameter"  
End If  
Set objPPT = WScript.CreateObject("PowerPoint.Application")  
objPPT.Visible = TRUE  
objPPT.Presentations.Open(fileName)
```

```
' *****  
' File: ScreenPresentationStart.vbs  
' First get the PPT object and then show screen presentation  
' *****  
  
Dim objSSW , obj  
Set obj = WScript.GetObject("", "PowerPoint.Application")  
Set objSSW = obj.Presentations(1).SlideShowSettings.Run
```

```
' *****  
' File: NextPage.vbs  
' First get the PPT object and then show next page  
' *****  
  
Dim objSSW , obj  
Set obj = WScript.GetObject("", "PowerPoint.Application")  
Set objSSW = obj.Presentations(1).SlideShowSettings.Run  
objSSW.View.Next
```

```
' *****  
' File: PreviousPage.vbs  
' First get the PPT object and then show previous page  
' *****  
  
Dim objSSW , obj  
Set obj = WScript.GetObject ("", "PowerPoint.Application")  
Set objSSW = obj.Presentations(1).SlideShowSettings.Run  
objSSW.View.Previous  
  
' *****  
' File: ScreenPresentationEnd.vbs  
' First get the PPT object and then exit screen presentation  
' *****  
  
Dim objSSW , obj  
Set obj = WScript.GetObject ("", "PowerPoint.Application")  
Set objSSW = obj.Presentations(1).SlideShowSettings.Run  
objSSW.View.Exit  
  
' *****  
' File: ExitPPT.vbs  
' First get the PPT object and then exit it  
' *****  
  
Dim obj  
Set obj = WScript.GetObject ("", "PowerPoint.Application")  
obj.Quit
```

Literaturverzeichnis

[Edwards 99]:

W. Keith Edwards

Core Jini

Prentice Hall 1999

[Arnold et al. 99]:

Ken Arnold, Bryan O' Sullivan, Robert W. Scheifler, Jim Waldo, Ann Wollrath

The Jini Specification

Addision-Wesley 1999

[Bader&Huber 2000]:

Herbert Bader, Walter Huber

Jini-Die intelligente Netzwerkarchitektur in Theorie und Praxis

Addision-Wesley 2000

[Born 2000]:

Günter Born

Inside Windows Script Host

Microsoft Press 2000

[Jini]:

<http://www.sun.com/jini/>

[Smartboard]:

<http://www.smarttech.com/smartboard/>

[Script]:

<http://msdn.microsoft.com/scripting/>