MIDDLEWARE-TECHNOLOGIEN 4

Zur Auswahl einer geeigneten Software-Infrastruktur gibt es einige prinzipiell verschiedene Ansätze, von denen in diesem Kapitel eine Reihe von richtungweisenden Middleware-Technologien vorgestellt und im Kapitel 5 verglichen werden. Zunächst richtet sich der Fokus auf die auf dem P2P-Prinzip basierenden Dienstarchitekturen, wie Sun's Jini und Microsoft's UPnP. Darauf folgt die auf dem C/S-Prinzip basierende Middleware. Dabei werden Technologien wie Odysseysoftware's ViaXML und CORBA/JacORB, die Unterstützung für das Zusammenspiel auf Applikationsebene versprechen, betrachtet. Im Folgenden werden diese verschiedenen Middleware-Konzepte erklärt. Der Anfang wird mit Jini gemacht. Des Weiteren werden UPnP, ViaXML und JacORB beschrieben.

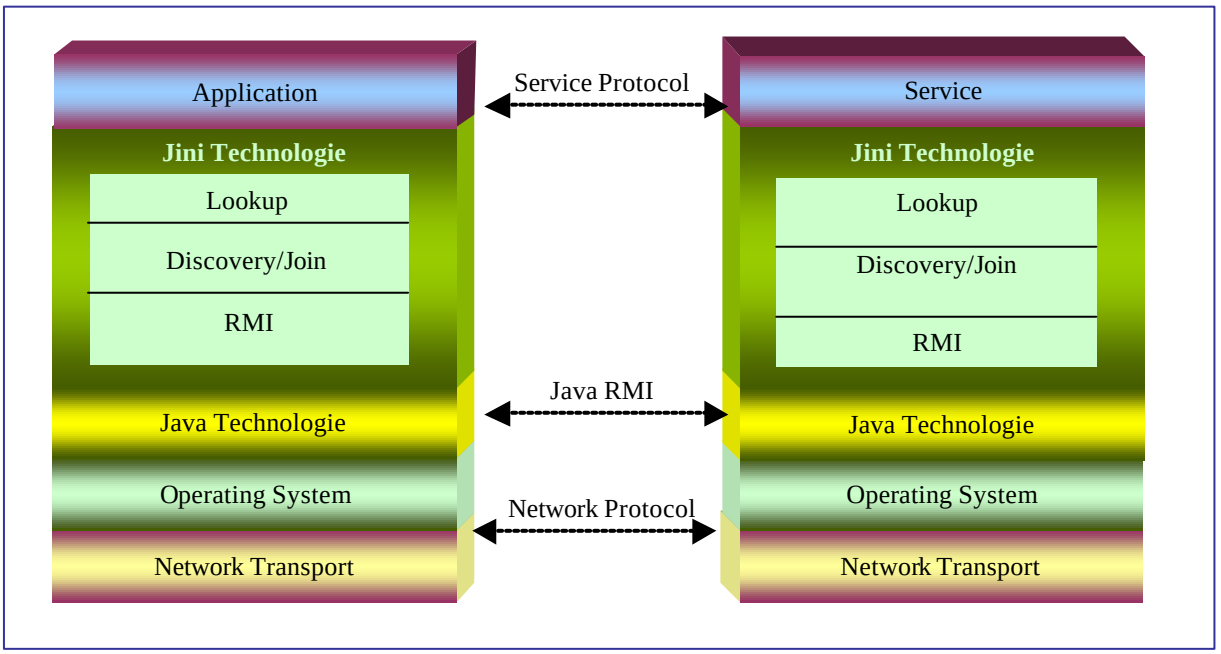
4.1 Java Intelligent Network Interface (Jini)

"Jini network technology is an open architecture that enables developers to create network-centric services -- whether implemented in hardware or software -- that are highly adaptive to change. Jini technology can be used to build adaptive networks that are scalable, evolvable and flexible as typically required in dynamic computing environments" [Sun 2003].

Jini ist ein von der Firma Sun entwickelter Infrastruktur-Dienst für Netzwerkumgebungen, der das Auffinden von Diensten und Ressourcen im Netzwerk erleichtern soll. Jini ist eine Menge von Spezifikationen [Oaks & Wong 2001], mit deren Hilfe Dienste einander im Netz finden können. Diese Menge von Diensten auf einem bestimmten Netzwerk wird die Jini-Gemeinschaft genannt. Jini basiert technisch auf Java und soll damit neben der Vernetzung von Computern auch die Vernetzung von Handys, PDAs, Haushaltsgeräten, Videokameras, Druckern usw. vereinfachen [Sun 2003]. Um dies zu erreichen, stellt Jini über einen so genannten Lookup Service (LUS)[42] nicht nur Informationen über die im Netz verfügbaren Dienst-Ressourcen bereit, sondern sendet auch umgehend einen Proxy-Code, welcher die Nutzung des Dienstes ohne eine zusätzliche Softwareinstallation gewährleistet. Die Benutzung eines Dienstes erfolgt in Jini nur für eine bestimmte Zeit und in einer Art Leasing-Vertrag, der Lease genannt wird. Ein Jini-Dienst muss sich beim Jini-LUS registrieren lassen, damit ihn die Anwendungen finden können [Oaks & Wong 2001]. Wenn ein Dienst seinen Lease nach der Benutzung nicht verlängert, löscht ihn der Lookup Service aus seiner internen Liste der registrierten Dienste und somit auch das Nutzungsrecht, so dass Ressourcen nicht unnötig blockiert werden (vgl. nächsten Abschnitt).

[42] Der *Lookup Service* ist das Herz einer Jini-Gemeinschaft. Er speichert die Registrierung aller anderen in der Jini-Gemeinschaft verfügbaren Dienste. Er ist selber auch nur ein Dienst [Oaks & Wong 2001], ähnlich dem *'naming service'* in anderen Dienstfindungsarchitekturen (z. B. CORBA).

Die *Abbildung 4.1* zeigt den Aufbau der Jini-Architektur und wie die Plattform-unabhängigkeit unter Java realisiert ist. Der Java-Interpreter auf dem dritten Layer bietet den höheren Layers immer das gleiche API. Er übersetzt so alle Systemaufrufe für die entsprechenden Plattformen in die gewünschte plattformabhängige Sprache. Die Pfeile sollen die unterschiedlichen Niveaus der Kommunikation, die angefordert wird, anzeigen.



Quelle: [Sun-Jini 2003]

Abbildung 4.1: Jini Architecture

4.1.1 Das Jini Universum

Das Jini Universum lässt sich in drei unterschiedliche Bereiche aufspalten: Infrastruktur, Programmiermodell und Dienste [Dibit 2003]. Die Infrastruktur ist die Menge von Komponenten, die den Aufbau eines Jini-Systems erlauben. Das Programmiermodell ist eine Menge von Schnittstellen, mit deren Hilfe alle Dienste beschrieben werden. Die Dienste sind die Einheiten in einem Jini-System. Um Jini zu verstehen, müssen daher zuallererst die Konzepte dieser Bereiche näher betrachtet werden.

- **Infrastruktur:** Dieser Bereich, der den Kern von Jini bildet, besteht aus Netzwerkdiensten, die das spontane Zusammenfinden von Diensten ermöglichen. Da

Jini vollständig in Java realisiert ist, ist zu dessen Betrieb eine JVM[43] notwendig. Um die Kommunikation mit verteilten Objekten zu ermöglichen, wird eine RMI benötigt. Beides ist auf den Seiten der Firma Sun im Rahmen des aktuellen JDKs[44] enthalten [Sun 2003]. Folgende Elemente beinhaltet die Infrastruktur (siehe die *Abbildung 4.2*):

- den zentralen Service im Jini „Lookup Service“, der alle Dienste verwaltet und Dienstanbieter und Dienstnutzer miteinander verbindet,
- den Findungsprozess „Discovery“ eines Lookup Service durch einen Dienstanbieter/-nutzer und umgekehrt,
- die Registrierung „Join“ des angebotenen Dienstes beim (zugehörigen) Lookup Service,
- die Anforderung eines Dienstes „Lookup“ durch einen Dienstnutzer.

➢ **Java-Programmiermodell:** Die Infrastruktur ermöglicht und benutzt das Programmiermodell, das folgende Schnittstellen definiert: Leasing, Ereignisse und Transaktionen.

- Das Leasing ist die Zeitspanne, in der ein Dienst beim Lookup Service registriert ist bzw. für die ein Dienst verwendet werden kann. Ein Dienst hat sich in periodischen Abschnitten beim Lookup Service zu melden, sonst wird er gelöscht [Oaks & Wong 2001]. Damit wird ein möglichst störungsfreier Betrieb gewährleistet. Diese Lease-Erneuerung erfolgt mittels Java RMI.
- Ereignisse werden von Jini verwendet, um Diensten das gegenseitige Benachrichtigen über ihren Zustand zu ermöglichen.
- Eine Transaktion ist die Zusammenfassung mehrerer Operationen oder Dienste zu einer Einheit. Die Jini Transaktions-Spezifikation definiert ein Interface, das ein Bestätigungsprotokoll in zwei Phasen implementiert [Oaks & Wong 2001]. Erst werden alle Teilnehmer nach Zustimmung gefragt, bei Erfolg tritt dann die eigentliche Ausführung ein. Eine Transaktion wird von einem Transaktions-Manager erstellt und überwacht. Die Jini-Version (jini-1_2_1_001-src) von Sun hat einen Transaktions-Manager namens ‚mahalo’.

---

[43] JVM (*Java Virtual Machine*) ist die Umgebung, in der die Java-Programme laufen. Die JVM stellt Java-Programmen einen softwarebasierten ‚Computer’ zur Verfügung, mit dem diese interagieren können [Microsoft Press 2000].

[44] Das kostenlose JDK (*Java Development Kit*) ist eine Sammlung von Softwarewerkzeugen, die von Sun als die wichtigsten Entwicklungswerkzeuge für Java und die grundlegenden Java-Objektbibliotheken entwickelt wurden. JDK benötigt man, wenn man Java-Programme selbst übersetzen will. Zum Start von bereits übersetzten Programmen reicht das JRE aus.

- **Dienste:** Ein Server hat ein Interface, ein API, das er der Außenwelt präsentiert. Dieses Interface nennt man Dienst-Interface oder einfacher: Dienst. Dienste können einander und vor allem auch den LUS ohne Kenntnis des Netzes finden. In Jini ist das Interface das Wichtigste, weil dieses sowohl dem Client als auch dem Server bekannt ist [Oaks & Wong 2001]. Die Jini-Version (jini1_2_1_001-src) von Sun wird mit einer LUS-Implementierung namens ‚reggie' geliefert.

❖ **Jini und RMI**

Die Kommunikation zwischen den Diensten in einer Jini-Gemeinschaft wird durch den Mechanismus RMI realisiert, der in der Jini-Technologie implementiert ist. Theoretisch kann auch ein anderes Kommunikationsprotokoll, das RMI unterstützt, eingesetzt werden. RMI bietet eine Reihe von Möglichkeiten, die sich für dynamische Umgebungen gut eignen. Die Dienste von RMI umfassen die verteilten Methodenaufrufe in einfacher Weise, sowie den Transport des heruntergeladenen Implementierungscodes ohne Zusatzaufwand von der Client-Seite und den Start bzw. den Stopp nach Bedarf. Diese Fähigkeiten machen RMI für die Implementierung von Jini-Diensten zur idealen Wahl [Oaks & Wong 2001].
Die RMI-Technologie ist eine Erweiterung der Fähigkeiten, die durch die RPC-Mechanismen bereitgestellt werden (RPC in Java). Dabei ist es nicht nur möglich, Daten zwischen Objekten im Netzwerk auszutauschen, sondern durch die Objekte selbst. Durch RMI wird der Austausch von mobilem Code erreicht, der eine wichtige Basis in der Jini-Technologie bildet.

Das von Java verwendete RMI kann mit einer IIOP-Option[45] RMI-Klassen erstellen, welche IIOP anstelle des RMI-eigenen Protokolls zur Kommunikation mit entfernten Objekten verwenden. RMI implementiert in diesem Modus einen vollständigen CORBA-ORB. Auf diese Weise können Geräte und Programme, die IIOP verstehen, an Jini angekoppelt werden. Der Jini LUS muss seine Interfaces in diesem Fall über das normale RMI Protokoll und über IIOP zur Verfügung stellen. Geräte können denn direkt mit dem Jini LUS kommunizieren, indem sie das IIOP verwenden [Sun-Jini 2003]. Detaillierte Informationen dazu liefert Sun in der Java RMI-IIOP Dokumentation [Sun 2003].

4.1.2 Der Jini-Einsatz und Pocket PC 2002

Jini ist plattformunabhängig, weil sie vollständig in Java realisiert ist. Während bei anderen Sprachen als Endprodukt des Kompilationsvorgangs eine .exe-Datei entsteht, die dann direkt ausgeführt wird, liefert der Java-Compiler als Output ein File in einem

[45] Das *Internet Inter-ORB Protocol* (IIOP) ist von der Object Management Group (OMG) für die Interoperabilität der ORBs entworfen worden. IIOP ist ein Protokoll für den objektbezogenen Zugriff über das Internet. Bei dem IIOP-Protokoll handelt es sich um einen offenen Standard für die Kommunikation zwischen *Object Request Broker* (ORB) über IP-Netze. IIOP wird im CORBA-Framework eingesetzt [Siemens Lexikon 2003].

Zwischenformat, dem sog. Byte-Code, das dann von einer VM (*Virtual Machine*) interpretiert wird. Der Vorteil dieses Systems ist, dass der Byte-Code plattformunabhängig ist, und nur eine VM für die jeweilige Plattform notwendig ist. Allerdings hat dies auch den Nachteil, dass der Speicherbedarf höher ist, da ja auch die VM auf dem Gerät vorhanden sein muss und nicht nur das auszuführende Programm.

Verfolgt man diesen Gedanken weiter, so kommt man zu dem Schluss, dass sich das Problem der Java-Entwicklung für die Pocket PC 2002 Plattform darauf reduzieren lässt, eine JVM für diese Plattform zu finden, um Jini benutzen zu können. Weil die Systemressourcen von Pocket PC 2002 im Gegensatz zu einem Desktop-PC eher knapp bemessen sind, ist die Verwendung der normalen JVM nicht möglich. Aus diesem Grund wurde von Sun Microsystems eine reduzierte Variante der JVM (*Kilo Virtual Machine* KVM) für Umgebungen entwickelt, in denen die Speicherressourcen knapp sind (128 bis 512 KB). Die dazugehörigen Lösungen heißen Java 2 Platform Micro Edition (J2ME)[46] [Muller 2001] und PersonalJava, die speziell für Geräte wie PDAs mit wenig Speicher und begrenzten Ressourcen optimiert sind.

- **PersonalJava:** Im Java-Bereich gibt es den PersonalJava Standard, der als eine abgespeckte Java-Version für den Einsatz auf mobilen Geräten spezifiziert ist. Die aktuelle, dazugehörige VM von Sun sowie mehr Informationen zur PersonalJava Application Environment sind auf der Adresse [Sun 2003] erhältlich (siehe hierzu Softwarequelle). Auf dem iPAQ 3870, der für diese Arbeit als Endgerät benutzt wurde, ist bereits die Java Runtime Environment von Jeode „Insignia Solution JeodeRuntime“ installiert. Die Jeode PDA Edition VM ist eine Implementation der Personal Java Specification 1.2 von Sun.

4.1.3 Voraussetzungen für Jini

Für den Einsatz von Jini sind folgende Voraussetzungen zu erfüllen [Oaks & Wong]:

- Zuerst ist ein TCP/IP Netzwerk erforderlich, von dem die JVM abstrahiert. Jini soll mit allen Betriebssystemen in jeglicher Art von Netzwerken zusammenarbeiten können.

- Zweitens muss im ganzen Netzwerk, das Multicasting unterstützen muss, Java laufen. Die Idee dabei ist, dass auf jedem Gerät eine JVM installiert ist und die Dienste entweder direkt auf einer JVM oder auf einer JVM als Proxy ausgeführt werden müssen.

- Drittens müssen sich die Geräte, auf denen JVM läuft, im Netzwerk befinden.

[46] J2ME (*Java 2 Micro Edition*) ist ein Standard Java Plattform für mobile Geräte, bestehend aus einer JVM, sowie aus Programmierschnittstellen, die auf mobile Geräte abgestimmt sind.

- Wenn die Teilnehmer an einer Jini-Gemeinschaft einander nutzen sollen, müssen sie die Interfaces der anderen kennen, aber nicht deren Implementierungen. Als Kommunikationsprotokoll benötigt Jini im Moment RMI. Jini-Protokolle bauen auf TCP/IP auf. Es wird auch einen HTTP-Server benötigt.

- JDK bzw. JRE ist für die Jini-Implementierung erforderlich. Die RMI-Abhängigkeit wichtiger Dienste ist ein wichtiger Grund für dieses Erfordernis (Sun RMI).

- Als letztes ist es notwendig, mindestens einen LUS im Netz installiert zu haben.

Weitere Informationen zu Jini findet man bei Sun [Sun 2003] und in dem empfehlenswerten Buch [Oaks & Wong 2001].

❖ Plattformen, auf denen das JSK geprüft worden ist

Die *Tabelle 4.1* zeigt, welche Plattformen mit JSK (*Jini Technology Starter Kit*) von Sun in Betrieb genommen und getestet worden sind [Sun 2003].

- Java 2 SDK, Standard Edition, v1.3.1_01 und v1.3.1_02 mit JSK v1.2 und v1.2.1 für Windows 2000, Windows NT 4.0 (SP6) und RedHat 7.1.
- Java 2 SDK, Standard Edition, v1.4.0 und v1.4.1 mit JSK v1.2.1_001 für Linux (INTEL x86) on RedHat 7.2, Windows XP Professional, Windows 2000 Professional und Windows NT 4.0 (SP6).

SDK	**Starter Kit v1.2**	**Starter Kit v1.2.1**	**Starter Kit v1.2.1_001** (released 8/2002)
v1.3.1_01	RedHat 7.1 TCK[47] Windows 2000 Windows NT 4.0 (SP6) TCK	-	-
v1.3.1_02	-	Windows 2000 (2nd Edition) Windows NT 4.0 (SP6) RedHat 7.1 TCK	-

⇨ **Weiter**

Tabelle 4.1: Plattformen und Jini Technology Starter Kit

[47] *Technology Core Platform Compatibility Kit.*

SDK	Starter Kit v1.2	Starter Kit v1.2.1	Starter Kit v1.2.1_001 (released 8/2002)
v1.4.0	-	-	RedHat 7.2 TCK Windows XP Professional Windows 2000 Professional Windows NT 4.0 (SP6)
v1.4.1	-	-	RedHat 7.2 Windows XP Professional Windows 2000 Professional Windows NT 4.0 (SP6) TCK

Quelle: [Sun 2003] http://developer.java.sun.com/developer/products/jini/tested.html

Tabelle 4.1: Plattformen und Jini Technology Starter Kit

❖ Jini API

Das Jini-1.2.1_001 von Sun umfasst drei Teile [Sun 2003]:

- Jini Technology Core Platform (JCP) beinhaltet die Kerninfrastruktur von Jini mit Spezifikationen, Interfaces und Klassen für Discovery, Join, Lookup, Distributed Events, Leasing und Distributed Transactions

- Jini Technology Extended Platform (JXP) enthält erweiterte Infrastruktur-Service-Interfaces, wie zB. discovery utilities und die Spezifikation der JavaSpaces[48]

- Jini Software Kit (JSK) enthält Implementationen einiger Spezifikationen des JCP und JXP, z. B. die Implementation von JavaSpaces

❖ Jini-Installation

Dieser Abschnitt zeigt, wie eine Arbeitsumgebung eingerichtet werden muss, damit die Jini-Services, die die Jini-Technologie beinhaltet, funktionieren.
Zwingend erforderlich für eine Nutzung von Jini-Diensten ist eine installierte JVM, da Jini auf RMI und einigen Java 2-Klassen basiert. Auf den Webseiten der Firma Sun ist derzeit die kostenlose Version J2sdk-1_4_0_03 als komplettes Entwicklungskit oder als lauffähige Runtime-Umgebung per Download erhältlich. Folgender Link führt zu den

[48] Gemäß [Oaks & Wong 2001] ist der JavaSpaces-Dienst sehr vielseitig. Er eignet sich ideal zum Entwickeln verteilter Anwendungen, da er dafür geschaffen ist, Daten zwischen den einzelnen Teilen einer solchen Anwendung fließen zu lassen. Zwar ist das grundlegende API des JavaSpaces-Dienstes einfach, aber der Dienst lässt sich für viele komplexe Tasks einsetzen.

entsprechenden Versionen (siehe hierzu Softwarequelle): Java JDK und Java JRE[49] für Windows 95/98/ME, NT 4, 2000, XP [Sun 2003a].

Nach der erfolgreichen Installation des JDK müssen die Jini-spezifischen Dienste installiert werden. Für eine funktionstüchtige Jini-Umgebung ist das JSK notwendig. Es umfasst die Bibliotheken der Jini-Klassen, die Jini-Spezifikationen, die Services, die zur Infrastruktur der Jini-Technologie gehören sowie die javadoc-Dokumentation der Jini-Klassen. Die derzeit aktuelle Version des JSK v1.2.1_001 ist auf der Homepage von Sun erhältlich unter [Sun 2003a] (siehe hierzu Softwarequelle). Allerdings muss man Mitglied der Java Developer Connection (JDC) sein (bzw. werden), um sich Jini herunterladen zu können.

- **Jini-Dienste starten**: Um die Sun-Implementierung des Jini-Frameworks zu starten, sind viele Schritte notwendig. Die Jini-Dienste erfordern in ihrer gegenwärtigen Implementierung viel Arbeitsspeicher und es kann problematisch werden, sie alle auf einem einzigen Rechner auszuführen. Dies gilt besonders für Windows-Plattformen, wobei mindestens 64 MB Arbeitsspeicher benötigt werden, wenn mehrere Dienste auf demselben Rechner ausgeführt werden [Oaks & Wong 2001].

1. HTTP-Server für die Jini-Dienste starten

Für die Kerndienste von Jini müssen die Clients in der Lage sein, Klassen herunterzuladen, die sich in den im lib-Verzeichnis installierten Jar-Dateien befinden. Daher wird der HTTP-Dienst benötigt, um die Dienste aus diesem Verzeichnis herunterzuladen.
Mit dem folgenden Befehl wird der HTTP-Server gestartet:

```
c:\java> –jar c:\files\jini1_2_1_001\lib\tools.jar –port 8080 –dir c:\java\jini1_1\lib\ -verbose
```

2. RMI-Aktivierungsdämon starten

reggie, mahalo, outrigger etc. benötigen den RMI-Aktivierungsdämon. Das ist der Hauptgrund, weshalb für die Jini-Implementierung von Sun Java 2 erforderlich ist. Um rmid zu starten, muss folgender Befehl, der die Sicherheits-Richtliniendatei definiert, ausgeführt werden:

```
c:\java> start rmid -J-Djava.security.policy= c:\files\jini1_2_1_001\policy\policy.all
```

[49] JRE (*Java Runtime Environment*) ist ein reduziertes JDK, mit dem nur bereits kompilierte Java-Programme gestartet werden können.

3. Lookup-Dienst „reggie" starten

Der erste Parameter auf der Kommandozeile definiert die Sicherheits-Richtliniendatei, unter der dieser Befehl registriert wird. Durch den Parameter http://localhost:8080/reggie-dl.jar wird der HTTP-Server referenziert. Der letzte Parameter c:\files\jinitemp\reggie_log ist das Verzeichnis, das reggie zum Speichern seiner Log-Dateien verwendet, die die Persistenz des Lookup-Servers verwalten.

```
c:\java> java    –Djava.security.policy= c:\files\jini1_2_1_001\policy\policy.all
                 –jar c:\files\jini1_2_1_001\lib\reggie.jar
                 http://localhost:8080/reggie-dl.jar
                 c:\files\jinitemp\reggie_log
```

4. JavaSpaces-Dienst „outrigger" starten

Im Gegensatz zu den anderen bisher erwähnten Diensten ist dieser Dienst nicht Teil des JSK. Um ihn zu starten, muss erst das JSTK „JavaSpaces Technologie Kit" geladen und installiert werden. Der JavaSpaces-Startbefehl ist ähnlich wie bei reggie.

```
c:\java> java    -Djava.security.policy= c:\files\jini1_2_1_001\policy\policy.all
                 -jar c:\files\jini1_2_1_001\lib\outrigger.jar
                 http://localhost:8080/outrigger-dl.jar
                 c:\files\jini1_2_1_001\policy\policy.all
                 c:\files\jinitemp\js_log
```

5. Sources kompilieren und Programm ausführen

Nach dem Starten aller notwendigen Dienste kann das Programm kompiliert und ausgeführt werden.

4.1.4 Protokolle, auf denen Jini basiert

Die Protokolle Discovery und Join kommen zum Einsatz, sobald ein neues Gerät angehängt werden will. Damit Services in den Jini Gemeinschaften sichtbar und damit auch nutzbar sind, müssen sie sich bei den entsprechenden LUSs anmelden. Hierzu dient das Join Protokoll. Vor der Anmeldung muss der Lookup Service kontaktiert werden. Dazu dient das Discoveryprotokoll, das aus zwei Einzelprotokollen besteht: Unicast- und Multicast-Discoveryprotokoll. Mit Hilfe des Unicast-Discoveryprotokolls können bereits bekannte

Lookup Services direkt kontaktiert werden. Im Gegensatz dazu ist bei dem Multicast-Discoveryprotokoll kein derartiges Wissen notwendig. Hier werden Multicast-Discoveryrequests versendet, welche von dem Lookup Service empfangen und in entsprechender Weise beantwortet werden[50]. Dafür gibt es im JDK einen Manager, den Lookup-Discovery-Manager, der das automatisiert,. Diese Klasse beherrscht sowohl Unicast wie auch Multicast Discovery [Oaks & Wong 2001].

Die *Abbildung 4.2* verdeutlicht die Zusammenarbeit der oben beschriebenen Protokolle. Sie zeigt auch die drei Hauptkomponenten, die normalerweise an einem funktionierenden Jini-System beteiligt sind, nämlich den Service Provider, den Client und den Lookup Service. Das Lookup und Discovery Protokoll bestehen aus drei Protokollteilen: Discovery (Client und Server), Join (Server) und Lookup (Client), wobei Join und Lookup als Dienst implementiert sind. Der Lookup Service ist die zentrale Schnittstelle zwischen Dienstanbietern und Dienstnutzern. Ein Client muss zuerst den LUS mittels eines Service Providers im Netzwerk ausfindig machen (Discovery 1/Join 2), um einen Dienst anmelden zu können. Hat er den LUS gefunden, bekommt er von diesem den LUS Proxy, über welchen die Kommunikation mit dem eigentlichen LUS stattfindet. Nun steht er als Dienst zur Verfügung. Will ein Dienstnutzer diesen Dienst in Anspruch nehmen, so stellt er eine Anfrage (Lookup) mit den charakteristischen Attributen des Dienstes und erhält das Proxy-Objekt zurück. Über dieses Proxy-Objekt kann er nun den Dienst nutzen, als wenn er lokal verfügbar wäre. Der Proxy selbst kommuniziert mit dem Gerät und der Software per RMI [Oaks & Wong 2001].

Zusammenfassend sind folgende Schritte in einer Jini-Gemeinschaft notwendig (siehe *Abbildung 4.2*):

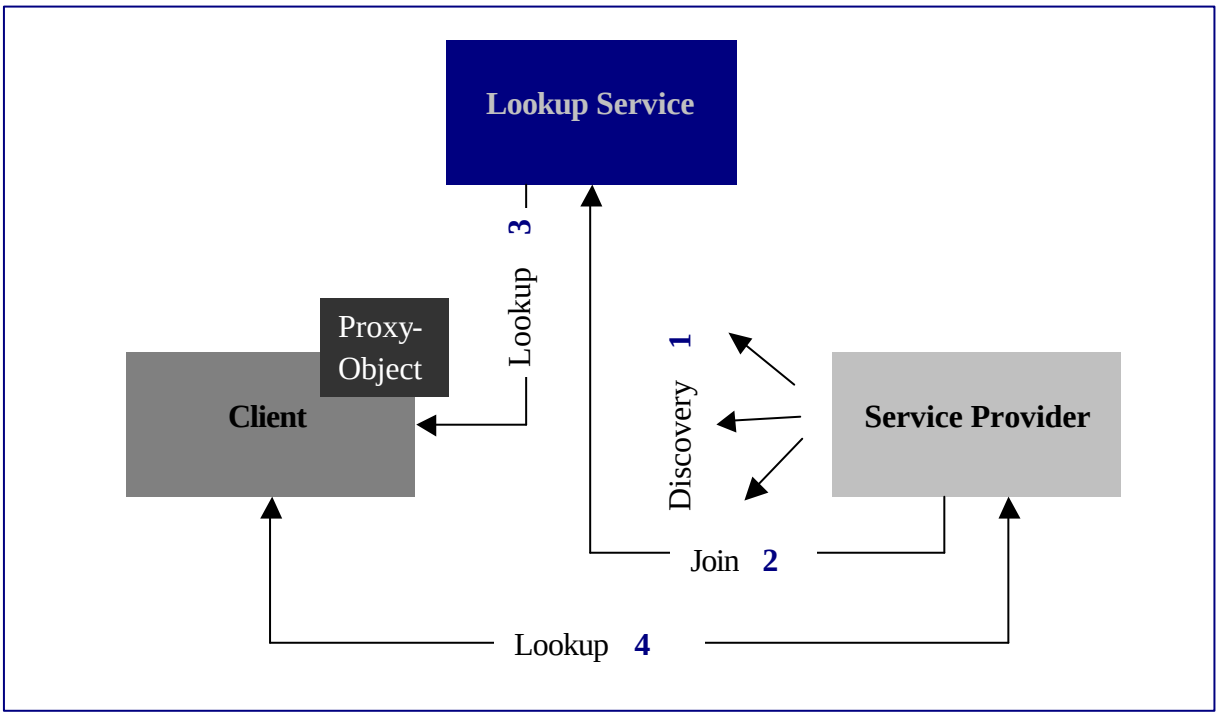
1 Discovery: Um Kenntnis der lokalen Gegebenheiten zu erhalten, müssen die Dienstnutzer zuerst die Instanz eines Dienstvermittlers ausfindig machen. Diesen Prozess bezeichnet man als Discovery. Ein Service Provider sucht also einen Lookup Service.

2 Join: Der Service Provider registriert ein Service Objekt und seine Service Attribute beim Lookup Service.

3 Lookup: Ein Client sucht einen Service über seinen Java Typ und ggf. andere Service Attribute. Eine Kopie (Proxy-Objekt) wird zum Client geschickt, womit er mit dem gesuchten Service kommunizieren kann.

4 Lookup: (Enderbringung) Mit Hilfe des Service Objekts kann der Client direkt mit dem Service interagieren.

[50] Anders ausgedrückt: Wenn ein Dienst weiß, wo sich der LUS befindet, kann er ihn über Unicast mit einem Lookup-Locator ansprechen. Anderen falls muss er ihn über Multicast mit Hilfe von Lookup-Discovery suchen.



Quelle: [Sun-Jini 2003]

Abbildung 4.2: Jini-Service: Lookup, Discovery, Joining

4.1.5 Eigenschaften und Bewertungen

Aus [Sun-Whitepaper] und [Oaks & Wong 2001] sind folgende Punkte herausgearbeitet worden, die für den Einsatz von Jini sprechen:

- Jini vereinfacht die Interaktion mit Netzwerken,
- Für den Dienstanbieter vereinfacht Jini die Netzadministration. Ein angeschlossenes Gerät meldet sich selbst automatisch an, identifiziert sich und gibt seine spezifischen Eigenschaften bekannt. Der Dienstanbieter wird nun aufgrund der gemeldeten Eigenschaften das Gerät in den Informationsaustausch einbeziehen,
- Jini gestattet es, dass ein Gerät sein eigenes Leistungsangebot dem Netz mitteilt und zur Verfügung stellt,
- Für den Programmierer soll Jini die Aufgabe vereinfachen, Software für verteilte Anwendungen zu schreiben,
- Jini soll das Netz intelligent machen, da Dienste und Clients spontan auftauchen und wieder verschwinden können,

- Jini basiert auf Java, das plattformunabhängig und objektorientiert ist,
- Die Adresse des Service Anbieters und des Lookup Services sowie die Netzwerkverbindung zum Service müssen nicht bekannt sein (Netzwerktransparenz),
- Jini ermöglicht spontane Netzwerke. Ressourcen im Netzwerk können einfach aufgefunden und genutzt werden,
- Ressourcen können gleichermaßen von Software und direkt von Benutzern genutzt werden.

Aus [Uni-Tuebingen 2003] sind folgende Punkte herausgearbeitet worden, die gegen den Einsatz von Jini sprechen:

- Jini ist von Java abhängig,
- Jini stellt hohe Anforderung an Speicher und CPU und verbraucht viele Ressourcen (JDK, Proxy-Ausführung), da der Java Code auf einer Stack Machine ausgeführt wird, welche um einiges langsamer ist als der native Code[51],
- Durch die Multicasts und den mobilen Code (der Proxy eines Service), der immer wieder aus dem Netz geladen werden muss, entsteht eine hohe Netzlast,
- Im Byte Code Verifier der VM[52] ist ein Fehler,
- Man muss JDC-Mitglied werden, um Jini (auch nur als Anwender) herunterladen zu können,
- Installation und Konfiguration der Basisdienste ist alles andere als leicht verständlich und wirkt leicht chaotisch,
- Ein weiterer Schwachpunkt ist, dass der Jini-Einsatz offensichtlich Probleme hat, vom experimentellen Status zu einer reifen Technologie fortzuschreiten, denn seit dem ersten Release ist es nun auch schon ein paar Jahre her, ohne dass der Markt Jini-fähige Endgeräte zu bieten hat. Es gibt nur wenige Beispiele für Jini Anwendungen und es gibt praktisch keine Verwendung von Jini,
- Man ist auf einen HTTP-Server angewiesen.

[51] Mit Native Code (Maschinensprache) ist eine Datei gemeint, die kompilierten Source Code enthält, der einem prozessorspezifischen Bytecode entspricht.

[52] Der Fehler liegt im Byte Code Verifier, der VM, der eine spezielle ungültige Code-Sequenz nicht korrekt überprüft, wodurch aber auch alle möglicherweise folgenden Sicherheits-Checks in der VM ausgehebelt werden. Ein Angreifer kann auf einer Web-Seite nun ein Java-Applet einbauen, das diese Code-Sequenz enthält; besucht ein Surfer diese Seite oder wird sie ihm per E-Mail ‚untergejubelt', kann der Angreifer Code mit den Rechten des lokal angemeldeten Users auf dem System ausführen lassen [Computer-security 2003].

❖ Sicherheit in Jini

Da die Jini-Technologie auf dem Konzept der RMI-Technologie basiert, kann auch das darin eingebaute Sicherheitskonzept übernommen werden. In der gerade aktuellen Jini-Implementierung werden die Sicherheitskonzepte des Auftraggebers (*Principal*[53]) und der Zugangskontrollliste (*Access Control List*) verwendet (*Policy* Dateien). Wird nun ein Dienst aufgerufen, so überprüft dieser anhand seiner Zugangskontrollliste, ob der Benutzer die benötigten Rechte hat, den Dienst zu nutzen. Dies bedeutet, dass ein Dienst nur dann nutzbar ist, wenn der Nutzer des Dienstes die entsprechende Berechtigung dazu hat. Die Zugangserlaubnis ist wie erwähnt in der Zugangskontrollliste spezifiziert, die mit dem Aufrufer des Dienstes assoziiert ist [Sun-Jini 2003].
Java besitzt ein starkes Sicherheitsmodell, das darauf beruht, dass es bekannt ist, woher der Code kommt und welche spezifischen Rechte benötigt werden. Dieses Modell ist nicht die beste Wahl für Jini, bei dem der Code von unbekannten Orten stammt, aber dennoch spezifische Rechte benötigt. Einstweilen ist es am einfachsten, Jini-Dienste und -Clients in einer Umgebung zu betreiben, die ihnen das Recht gibt, jede beliebige Operation auszuführen. Standardmäßig besitzen Anwendungen keinen Sicherheitsmanager. Allerdings müssen RMI-Clients einen Sicherheitsmanager installiert haben, um Code von RMI-Servern herunterladen zu können [Oaks & Wong 2001]. Authentifizierung und Verschlüsselung werden nicht angeboten. Ein weiterer ungelöster Sicherheitsaspekt betrifft die übertragenen Proxy-Objekte. Sobald diese auf dem anfordernden Gerät heruntergeladen sind, werden sie automatisch ausgeführt.

Zusammengefasst verfügt Jini über Sicherheitsdienste, die überwiegend Funktionen zur Realisierung der Zugangskontrolle beinhalten. Andere Sicherheitsfunktionen wie Anonymität, Integrität und Vertraulichkeit müssen vom Nutzer selbst implementiert werden.

4.2 Universal Plug and Play (UPnP)

"Universal Plug and Play (UPnP) is an architecture for pervasive peer-to-peer network connectivity of PCs and intelligent devices or appliances, particularly within the home. UPnP builds on Internet standards and technologies, such as TCP/IP, HTTP, and XML, to enable these devices to automatically connect with one another and work together to make networking - particularly home networking - possible for more people" [UPnP-Microsoft 2003].

Eine große Alternative zu Jini ist die Microsoft-Technik UPnP (Universal Plug and Play), die laut Microsoft im Gegensatz zur Sun-Technologie den PC zum Mittelpunkt des Heimnetzwerks macht. Bei der Jini-Technik kann prinzipiell jedes angeschlossene Gerät die Steuerung des Netzwerkes übernehmen. Die Idee von UPnP ist es, verschiedene unabhängige

[53] Der *Principal* ist in der Regel ein Benutzer, dem gewisse Zugriffsrechte erteilt worden sind.

Einheiten mit dem PC zu verbinden, und sein Ziel ist die Erweiterung der Plug and Play Idee für den Fall, dass Geräte über TCP/IP miteinander verbunden sind. Wie bei Jini sollen auch bei UPnP verschiedenartige Geräte miteinander kommunizieren können. UPnP ist eine verteilte, geöffnete, vernetzte Architektur, welche TCP/IP und das Web zur Steuerung und zur Datenübertragung zwischen vernetzten Geräten verwendet [Microsoft 2003]. Dieser Standard setzt komplett auf XML zur Ressourcenbeschreibung und den Internet Protokollen HTTP über UDP[54] auf, das seinerseits nicht zwingend auf IP aufsetzen muss. Und somit kann auch direkt Infrarot, Bluetooth oder Ähnliches betrieben werden. Die Dienstvermittlung ist dabei mit oder ohne zentrale Komponente (Proxy) möglich. Im Gegensatz zu Jini verzichtet man auf den mobilen Code. Das Netz kann ohne PC auskommen. Aktiviert wird es durch eine Peer device to device Verbindung. Um Geräte im Netz zu finden, wird ein Multicast verwendet. Die Services warten, bis eine Anfrage kommt, sehen, ob sie den Service liefern und wenn ja, geben sie eine Antwort [TU-Darmstadt 2003].

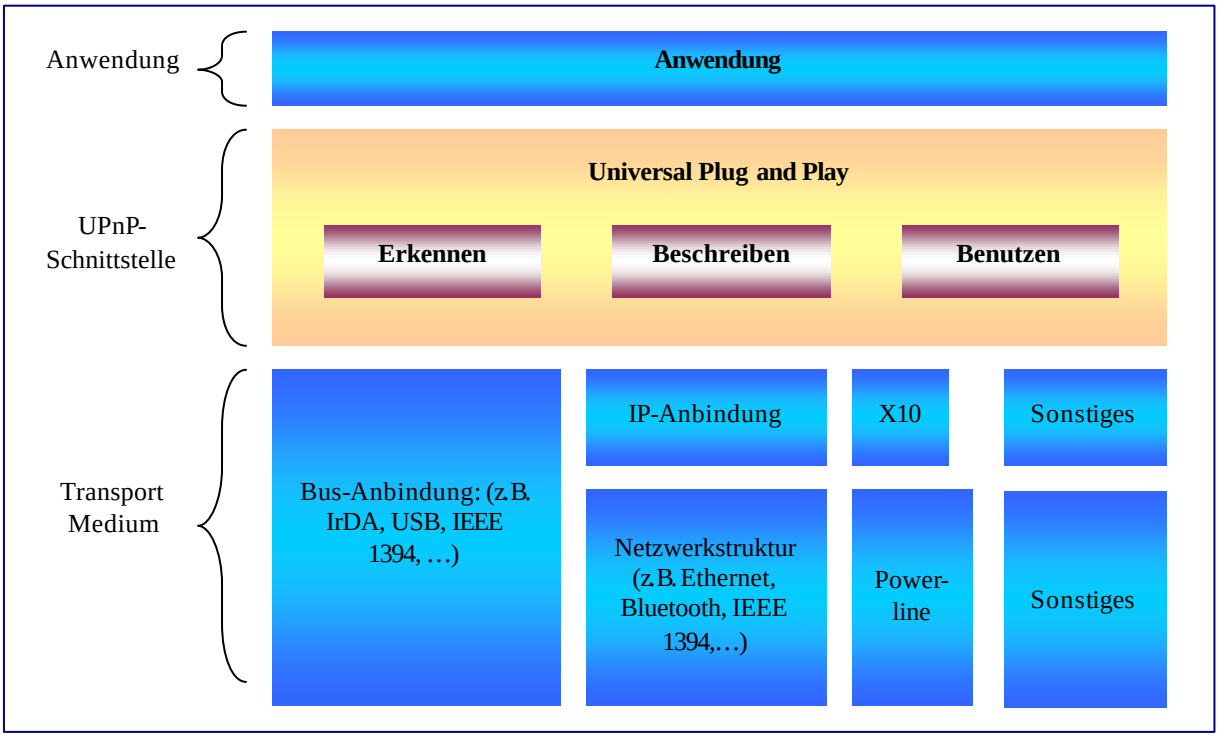
UPnP kann in verschiedenen Bereichen eingesetzt werden: in Heimnetzwerken, kleinen Netzwerken oder in Netzwerken kleinerer Unternehmen und Geschäftsgebäude. UPnP ist nicht auf ein bestimmtes Betriebssystem, eine bestimmte Programmiersprache oder ein bestimmtes physisches Medium festgelegt.
UPnP unterstützt Netzwerkverbindungen ohne Konfigurationsbedarf und automatische Ermittlung. Auf diese Weise kann sich ein Gerät dynamisch einem Netzwerk anschließen und es kann eine IP-Adresse erhalten, seinen Namen ankündigen, seine Fähigkeiten auf Anfrage angeben und Informationen zum Vorhandensein und zu den Fähigkeiten anderer Geräte erhalten. DHCP- und DNS-Server sind optional und werden verwendet, wenn sie im Netzwerk verfügbar sind. Darüber hinaus kann ein Gerät ein Netzwerk problemlos und automatisch verlassen, ohne einen nicht gewünschten Status zu hinterlassen [UPnP-Microsoft 2003].

Laut Microsoft sollen die Netzwerkprodukte funktionieren, die die UPnP-Technologie enthalten, sobald sie physisch an das Netzwerk angeschlossen werden. UPnP soll mit fast jeder verkabelten oder kabellosen Netzwerkmedientechnologie verwendet werden können.

Die untere *Abbildung 4.3* veranschaulicht einen exemplarischen Aufbau des UPnP-Schichtenmodells als Vermittler zwischen Netzwerk/Hardware und Applikation. Wie Jini bietet auch UPnP standardisierte Schnittstellen zum Auffinden, Erkennen und Benutzen vernetzter UPnP-Geräte. Ein UPnP-Element registriert sich bei einem UPnP-Host und erhält von diesem eine eigene IP-Adresse im Netzwerk. Ein Service wird bei UPnP mit XML beschrieben und diese Beschreibung wird dann an den anfragenden Control Point gesendet. Per XML meldet das Gerät seine Eigenschaften per HTTP im Netzwerk an. Die UPnP-Technologie setzt auf bereits eingeführte Standards und Protokolle zur Kommunikation

[54] Das UDP (*User Datagram Protocol*), das direkt auf dem darunter liegenden IP-Protokoll aufbaut, ist ein Transportprotokoll, das die vierte Schicht des OSI-Referenzmodells realisiert und den verbindungslosen Datenaustausch zwischen Rechnern unterstützt. Das UDP wurde definiert, um auch Anwendungsprozessen die direkte Möglichkeit zu geben, Datagramme zu versenden und damit die Anforderungen transaktionsorientierten Verkehrs zu erfüllen [Siemens Lexikon 2003].

zwischen einzelnen Elementen, wie der Protokollstack in der *Abbildung 4.7* darlegt (Abschnitt 4.2.4 Protokolle, auf denen UPnP basiert). Insbesondere ist kein Übertragungsstandard vorgeschrieben. D. h., man kann ein beliebiges, auch hybrides, Netz zur Verbindung der beteiligten Geräte verwenden, sofern TCP / IP unterstützt wird. Etablierte und neue Netzwerktechniken wie zum Beispiel Ethernet, IEEE 1394 (Firewire), HomePNA (Home Phone Networking Association) und Powerline sollen ebenfalls direkt unterstützt werden [UPnP-Microsoft 2003] (siehe auch die beiliegende CD, Datei: Technet-Microsoft-UPnP).



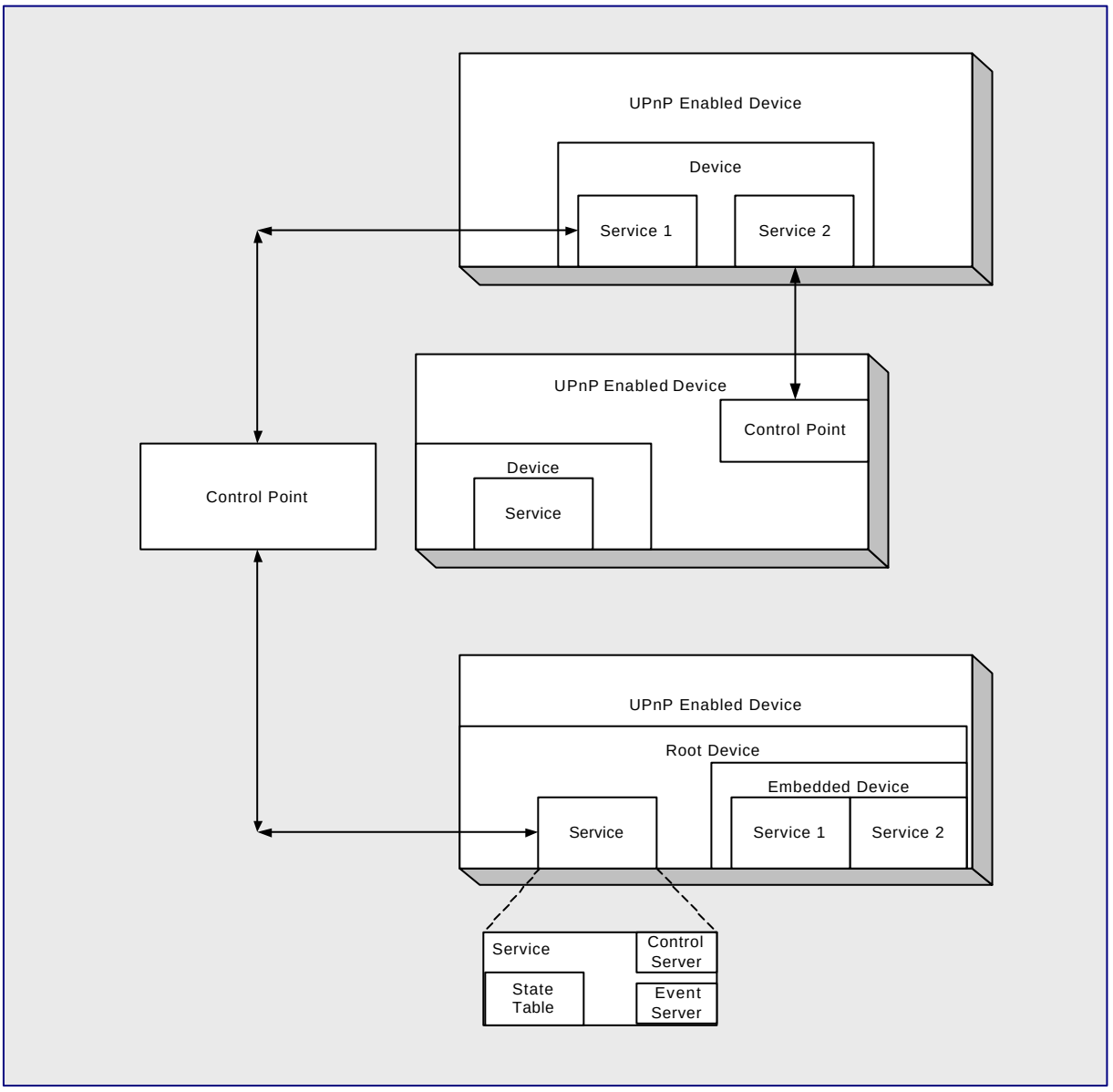
Quelle: [UPnP-Microsoft 2003]

Abbildung 4.3: Exemplarischer Aufbau des UPnP-Schichtenmodells

4.2.1 Komponenten eines UPnP Netzwerks

Ein Gerät ist ein Container von Diensten und von eingebetteten Geräten. Zum Beispiel besteht ein Videorecorder aus einem Kassettentransport-Mechanismus, einem Tuner und einer Uhr. Und eine Fernseher/Videorecorder Kombination bietet nicht nur Dienste, sondern auch ein eingebettetes Gerät an. Es werden sich Kategorien von UPnP Geräten bilden, verschiedene Geräte wird man mit verschiedenen Diensten und eingebetteten Geräten assoziieren.

Zu einem UPnP Netzwerk gehören Geräte, Dienste und Kontrollpunkte. Ein möglicher Aufbau ist in der *Abbildung 4.4* zu sehen.



Quelle: [UPnP-Microsoft 2003] (siehe die beiliegende CD, Datei: UPNP_Understanding UPNP-1.doc)

Abbildung 4.4: UPnP-Netzwerk

Dienste, die ein Videorecorder bietet, werden zum Beispiel anders sein als diejenigen, die ein Drucker bietet. Ein Dienst ist die kleinste Kontrolleinheit in einem UPnP Netzwerk, z. B. der Uhrdienst auf einem Videorecorder. Jeder Dienst hat Aktionen und Zustandsvariablen. Dieser

Uhrdienst hätte möglicherweise eine Zustandsvariable, ‚aktuelle_zeit', die uns den Zustand der Uhr anzeigt, und zwei Aktionen, ‚zeit_einstellen' und ‚alarmieren', die uns erlauben, den Dienst zu kontrollieren. Die Geräte sowie die Dienst-Beschreibung sind Teile von XML-Beschreibungen, die vom UPnP Forum[55] standardisiert werden. Die Gerätebeschreibungs-Dokumente enthalten einen Zeiger (eine URL) zu den Dienst-Beschreibungen. Geräte können mehrere Dienste beinhalten. Dienste in UPnP Geräten bestehen aus einer Zustandstabelle, einem Kontrollserver und einem Ereignisserver (siehe *Abbildung 4.4*). Die Zustandtabelle zeigt den Zustand des Dienstes durch Zustandsvariablen an. Der Kontrollserver erhält Aktionsanfragen (z. B. ‚zeit_einstellen'), führt sie aus, ändert die Zustandstabelle und gibt Antworten zurück. Der Ereignisserver zeigt die Ereignisse jedes Mal an, wenn sich der Zustand des Dienstes ändert. Ein Kontrollpunkt ist fähig, andere Geräte zu erkennen und zu kontrollieren und Informationen über die Geräte und deren Dienste abzufragen. D. h., der Kontrollpunkt holt sich nach dem Erkennen eines Geräts seine Geräte- und Dienst-Beschreibungen und kann dann Aktionen auf dem Gerät aufrufen oder über die Ereignisse auf dem Gerät informiert werden.

❖ Dienstvermittlung beim UPnP

Das Zusammenspiel der UPnP-Komponenten ist in der *Abbildung 4.5* dargestellt. Wenn ein Device einem Netzwerk beitritt, erlaubt das UPnP Simple Service Discovery Protokoll (SSDP) dem Device, seine Services dem Control Point im Netzwerk anzubieten. Gleicherweise erlaubt das Discovery Protokoll von UPnP einem Control Point, wenn er das Netz betritt, nach für ihn interessanten Services zu suchen. Weiterhin kann der Control Point die Standard Multicast Adresse abhören, um zu erfahren, ob für ihn neue interessante Services im Netzwerk zur Verfügung stehen. Im Gegensatz dazu muss ein Device die Standard Multicast Adresse abhören, um auf alle Anfragen antworten zu können, bei denen ein Service nachgefragt wird, den es beinhaltet. Ein Service muss aber nicht auf eine Anfrage eines Control Points antworten.

Ein Service wird nur für eine bestimmte Zeit angeboten. Für diesen Zeitrahmen schreibt der Control Point die Adresse, unter der er den Service erreichen kann, in seinen Cache. Läuft die Zeit eines Devices ab, muss sie durch ein Advertisement[56] oder einen beantworteten Suchvorgang[57] verlängert werden. Es sollte sich in diesem Zusammenhang auch jedes Device abmelden, bevor es das Netzwerk verlässt, um vielleicht noch

[55] Das Forum für UPnP ist ein im Juni 1999 gegründetes offenes Industriekonsortium, das beim Definieren der UPnP-Standards beteiligt ist, wodurch die Zusammenarbeit intelligenter Geräte in Heimnetzwerken und längerfristig und innerhalb von Unternehmen vereinfacht werden soll. Dies erreicht das Forum durch das Definieren und Veröffentlichen von Steuerungsprotokollen für UPnP-Geräte und Dienste [UPnP-Microsoft 2003].

[56] Als Advertisement wird der Vorgang bezeichnet, wenn ein Device das Netzwerk betritt und seine Services via Multicast anbietet. Dabei benutzt es das HTTPMU-Protokoll.

[57] Der Suchzyklus wird von einem Control Point gestartet, der gerade in das Netzwerk eingetreten ist oder einen Service sucht.

bestehende Speicherungen seiner Angaben löschen zu können, da die Services des Devices nicht mehr benutzt werden können.

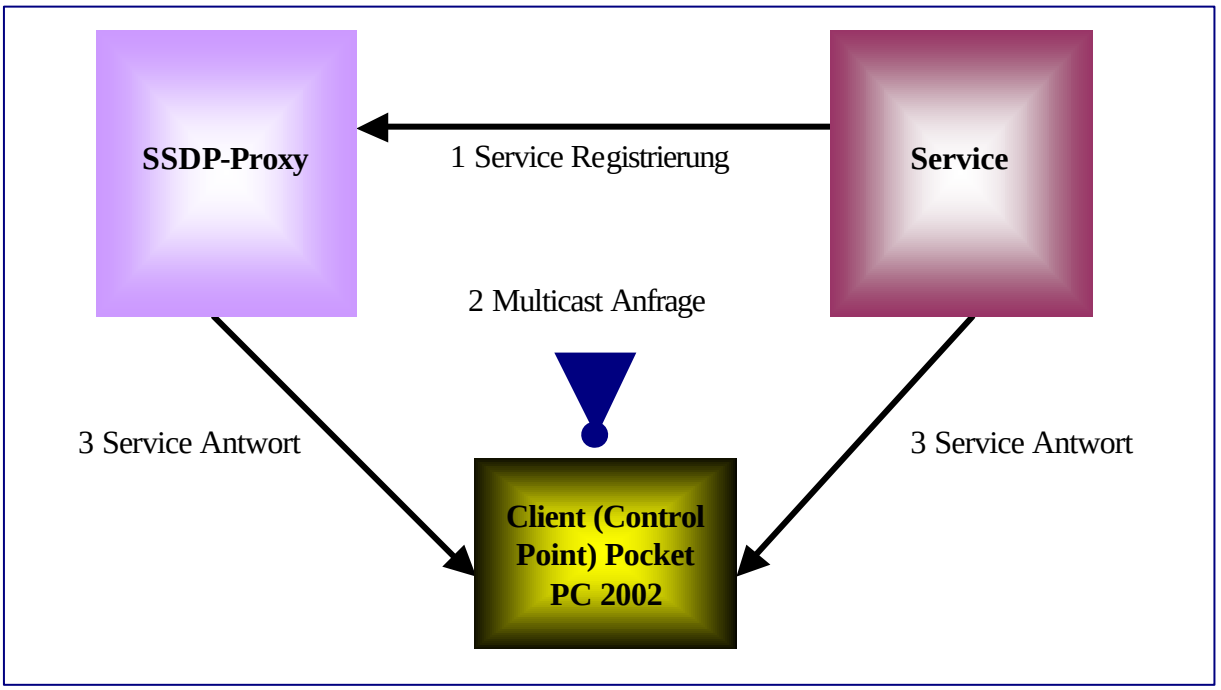


Abbildung 4.5: UPnP Dienstvermittlung mit SSDP

Nachdem ein Control Point ein passendes Device gefunden hat, nimmt er Verbindung mit dem Host auf, der ihm beim Discoveryvorgang übermittelt wurde. Von dort aus kann der Control Point weitere Informationen über den benötigten Service herunterladen. Z. B. sind in den heruntergeladenen Daten URLs enthalten, über die man den Service steuern kann, usw. [Uni-Stuttgart 2003].

❖ Die Schritte zu einem UPnP Netzwerk

Die nötigen Interaktionen bei der UPnP-Vernetzung können gemäß [UPnP 2003] und [UPnP-Altotec 2003] in fünf Schritte unterteilt werden. Dies wird in der nachfolgenden *Abbildung 4.6* verdeutlicht.

0 **Addressing** (IP mittels DHCP oder AutoIP)**:** Es beginnt mit der Zuteilung einer Adresse, sobald ein Gerät oder ein Control Point an ein physikalisches Netzwerk (LAN) angeschlossen wird.

1 **Discovery** (via SSDP & HTTP Multicast)**:** Der nächste Schritt heißt Discovery, bei dem sich Control Points und Geräte gegenseitig finden. Die Geräte werben in

regelmäßigen Abständen durch Aussenden von SSDP-Nachrichten für ihre Dienste. Diese zeitbegrenzten Werbenachrichten kann jeder Control Point empfangen und bei Interesse mit dem Gerät Kontakt aufnehmen. Umgekehrt kann ein Control Point gezielt nach einem speziellen Gerät oder Dienst im Netzwerk suchen, und bei Erfolg einen Kontakt herstellen. SSDP unterstützt auch ein explizites Abmelden eines Gerätes.

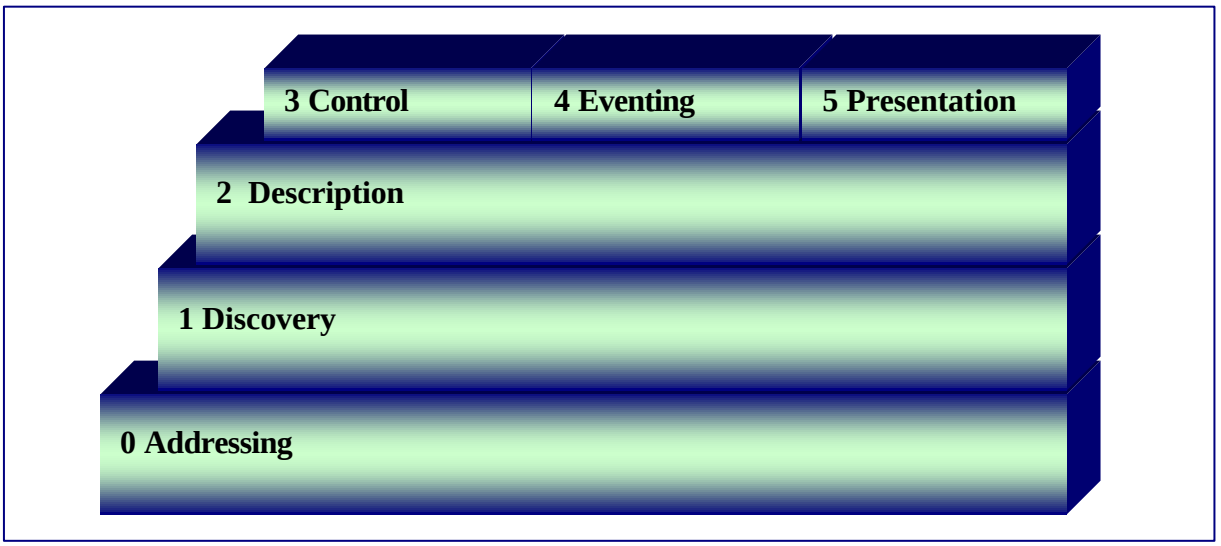


Abbildung 4.6: Die Schritte zu einem UPnP Netzwerk

2 **Description** (via XML-Messages)**:** Sobald der Kontakt zwischen Gerät und Control Point hergestellt ist, lernen sich die beiden mit Hilfe der Gerätebeschreibung näher kennen. Die UPnP Beschreibung (in XML) eines Dienstes enthält Informationen über das Gerät, den Hersteller, die Seriennummer und das Dienstangebot.

3 **Control** (via SOAP[58]: XML- Messages): Zur Steuerung eines Gerätes benutzt UPnP das SOAP-Protokoll (Simple Object Access Protocol). Dieses Protokoll benutzt HTTP und mit XML formatierte Nachrichten, um beim Gerät Aktionen auszulösen und die Werte von Variablen abzufragen.

4 **Eventing** (via GENA[59]: XML-Messages): UPnP sieht einen Mechanismus vor, der einem Gerät erlaubt, an einen Control Point Nachrichten über die Änderungen von

[58] Mit dem Ziel der programmiersprachen- und plattformunabhängigen Kommunikation zwischen verteilten Anwendungskomponenten entstand das auf XML basierende Simple Object Access Protocol (SOAP). SOAP definiert ein XML-Format für Nachrichten, die mittels eines darunter liegenden Transportprotokolls übertragen werden.
[59] *General Event Notification Architecture*

Variablen zu senden. Bevor dies geschehen kann, muss ein Control Point diese Nachrichten beim Gerät abonnieren. Dieses Abonnement ist zeitbegrenzt und muss vor Ablauf vom Control Point erneuert werden. Zusätzlich kann dieses Abonnement vom Control Point jederzeit gekündigt werden. UPnP verwendet das GENA-Protokoll (General Event Notification Architecture) für diesen Mechanismus.

5 **Presentation** (via HTTP): Unter Presentation verbirgt sich nichts anderes als eine Webseite des Gerätes. Diese Seite kann mit jedem Web-Browser dargestellt werden. Das UPnP Forum schreibt keine speziellen Eigenschaften für diese Seite vor. Es ist dem Gerätehersteller überlassen, eine ansprechende, eventuell mehrsprachige Webseite für das Gerät zu hinterlegen.

Details zur UPnP-Architektur können auf der Webseite des UPnP Forums [UPnP 2003] abgerufen werden.

4.2.2 Einsatz von UPnP

Wenn man sich einen neuen Drucker kauft und zurück nach Hause kommt, möchte man ihn sicher ans Heimnetzwerk anschließen. Man hätte es gerne so, dass er für alle PCs und für alle anderen Geräte des Netzwerks direkt verfügbar ist. Die heute dafür benötigten Schritte wären, den Drucker an einem PC des Netzwerks anzuschließen, den Treiber zu laden, den Drucker im Netzwerk zur Verfügung zu stellen, und jeden einzelnen PC des Netzwerks mit dem neuen verfügbaren Drucker zu verbinden. Wäre es nicht besser, schneller und effektiver, wenn man den Drucker einfach an irgendeinen verfügbaren Netzwerk-Port anschließen könnte, und er würde dann automatisch allen Geräten und Benutzern des Netzwerks zur Verfügung stehen?
Es gibt natürlich eine Fülle an weiteren Szenarien, wobei sie alle erst gestartet werden, wenn die jeweilige zu ihnen passende Anwendung auf einem PC gestartet wird und läuft [UPnP-Microsoft 2003] (siehe auch die beiliegende CD Datei: UPNP_Understanding UPNP-1.doc).

❖ **Voraussetzung für UPnP**

UPnP schreibt die Verwendung eines DHCP (*Dynamic Host Configuration Protocol*) Servers, DNS-Server (domain name service) oder AutoIP[60] vor. Darüber hinaus kann ein Gerät ein Netzwerk problemlos und automatisch verlassen, ohne einen nicht gewünschten Status zu hinterlassen [UPnP-Microsoft 2003]. Weiterhin benötigt UPnP XML und HTTP.

[60] AutoIP benutzt reservierte Adressen im 169.254/16 Bereich. Damit entfällt die lästige manuelle Zuteilung einer IP-Adresse [UPnP-Altotec 2003].

❖ **UPnP, Pocket PC 2002 und Windows**

Von Intel wurde ein spezielles Entwicklungskit SDK für Pocket PC Geräte entwickelt, die UPnP verwenden. Mit UPnP und dem Intel UPnP SDK für Pocket PC kann ein Pocket PC-Gerät (z. B. PDA) alle UPnP fähige Geräte im Netz steuern und in einem ad-hoc personal Networks an der Umgebung der drahtlosen Geräte teilnehmen (UPnP-SDK [Intel 2003]). Als Entwicklungsumgebung stehen momentan zwei SDK zur Verfügung: Microsoft UPnP Device Kit sowie Intel UPnP SDK für Linux. UPnP ist verfügbar unter J2SE (Java 2 Platform Standard Edition), J2EE (Java 2 Platform Enterprise Edition) und J2ME (Java 2 Platform Micro Edition).

Windows ME verfügt über UPnP-Funktionalität (Support [UPnP-Microsoft 2003]). In Windows XP wird die UPnP-Funktionalität durch folgende Dateien bereitgestellt,

- Upnpcont.exe
- Upnphost.dll - for hosting UPnP devices
- Upnp.dll - the main UPnP dynamic-link library (DLL)
- Upnpui.dll - used by Windows XP to create the interface

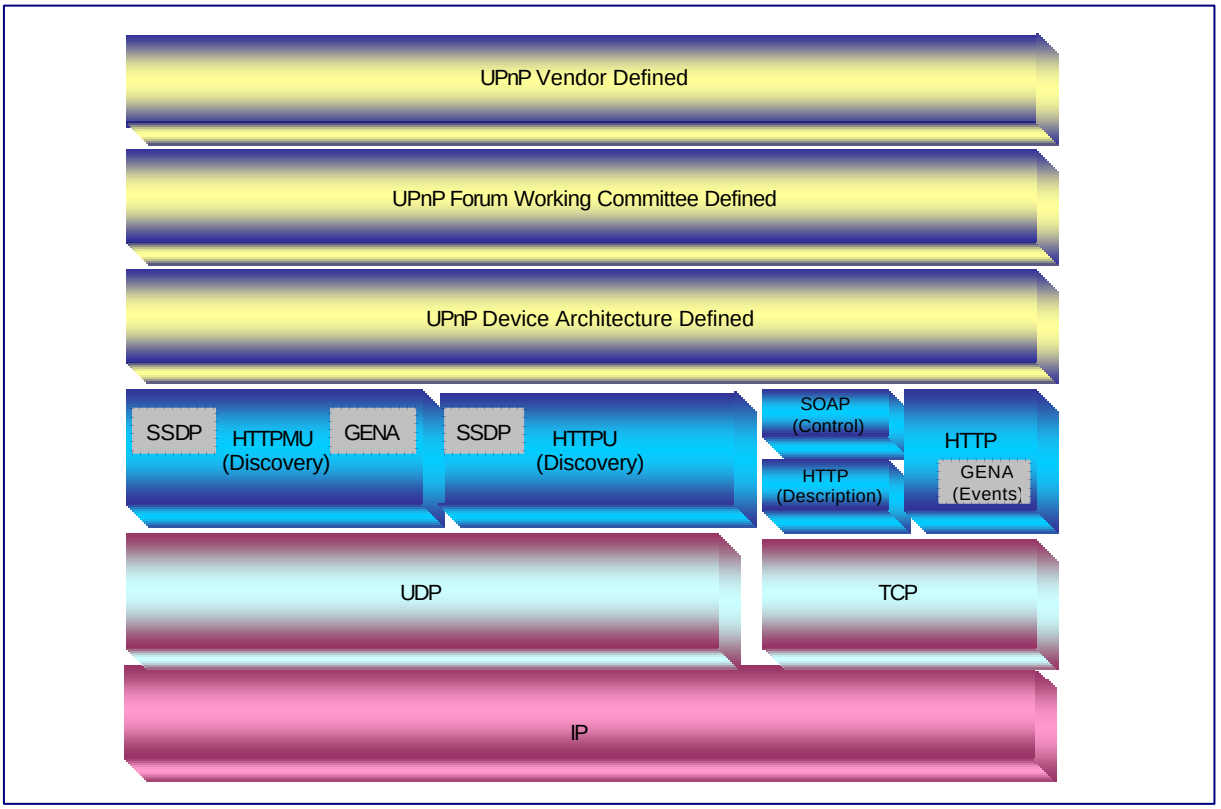
UPnP-Dateien sind im Pfad /Windows/System32/ enthalten. Sobald ein UPnP-Gerät an das Netzwerk angeschlossen wird, erscheint ein Icon in der Statusleiste. In der Netzwerkumgebung werden alle integrierten Unterstützungen der verfügbaren UPnP-Geräte aufgelistet.

4.2.3 UPnP-Besonderheiten

UPnP ist universal. Die Gründe dafür sind [UPnP 2003]:

- Medium- und Geräteunabhängigkeit: Die UPnP-Technologie kann auf jedem Medium laufen, wie beispielsweise Telefonlinie, Stromlinie, Ethernet, RF und 1394 (Firewire).
- Plattformunabhängigkeit: um UPnP-Produkte zu erstellen, ist jedes Betriebssystem und jede Programmiersprache verwendbar.
- Internetbasierte Technologien: Die UPnP-Technologie basiert zu einem großen Teil auf bekannten Internet-Protokollen wie IP, TCP, UDP, HTTP, und XML.
- Benutzeroberfläche-Kontrolle: Der Browser kann verwendet werden, um die Benutzeroberflächen der Geräte zu kontrollieren.
- Erweiterbar: Jedes UPnP-Produkt kann value-added services zu seinen Basisdiensten und seiner Basisarchitektur hinzubekommen.

4.2.4 Protokolle, auf denen UPnP basiert



Quelle: [UPnP-Microsoft 2003] (siehe auch die beiliegende CD, Datei: UPNP_Understanding UPNP-1.doc)

Abbildung 4.7: UPnP Protokoll Stack

Die obere *Abbildung 4.7* stellt den UPnP Protokoll Stack dar. Diese Protokolle basieren auf dem vorhandenen Protokoll TCP/IP. UPnP vendors, UPnP Forum Working Committees und die UPnP Device Architecture Document definieren die höchsten Schicht-Protokolle der UPnP-Implementierung. Das Service Discovery wird über das Simple Service Discovery Protocol (SSDP) abgewickelt. Bei SSDP kann die Dienstsuche durch Angabe bestimmter Suchkriterien verfeinert werden. Bspw. werden nur Services zurückgeliefert, die gewünschte Attribute aufweisen. SSDP verwendet das HTTP-Pendant auf UDP-Seite (User Datagram Protocol) HTTPU bzw. HTTPMU (Unicast-bzw. Multicast-Variante over UDP). Der Dienstnutzer startet bei einer SSDP Service Anfrage ähnlich wie z. B. bei DHCP einen UDP-Broadcast bei einer vordefinierten Multicast-Adresse. Jeder Knoten, der den geforderten Service unterstützt, sendet per UDP-Unicast eine entsprechende Erfolgsmeldung an die, dem anfragenden UDP-Paket beiliegende, MAC-Adresse zurück.

Die Kommunikation von Control Points und Controlled Devices (Controlling und Eventing) erfolgt hingegen durch den, auf der aus dem Internet bekannten Kombination HTTP / TCP / IP aufbauenden, SOAP-Standard [UPnP-Microsoft 2003].
Zusammengefasst findet die Dienstvermittlung mittels SSDP statt. Hier sendet der Client eine Anfrage. (die im Wesentlichen den Service-Typ (Bezeichner für Dienst) enthält) an eine feste Multicast-Adresse. Die Server, die diesen Dienst anbieten, antworten mit: Unique Service Name, Time-to-live und Description-URL, wobei der SSDP-Proxy als zentrale Komponente dient. Mittels der Description-URL wird eine XML-basierte Dienstbeschreibung beim Description-Server angefordert. Bei der Dienststeuerung wird das Service Control Protocol (SCP) zur Fernsteuerung des Dienstes auf Basis einer Service Control Protocol Declaration (SCPD) und bei dem Notifikationsmechanismus eine Generic Event Architecture für Distributed Events benutzt.

4.2.5 Eigenschaften und Bewertungen

Aus Microsoft Technet [UPnP-Microsoft 2003] werden im Nachfolgenden die wesentlichen Vorteile des UPnP gefasst:

- Man hat kaum Konfigurationsaufwand, d. h. UPnP unterstützt Netzwerkverbindungen ohne Konfigurationsbedarf und automatische Ermittlung,
- Es ermöglicht eine direkte Kommunikation der Geräte untereinander (P2P-Networking),
- Es ermöglicht Datenkommunikation zwischen zwei beliebigen Geräten, wobei jedes Gerät auch ein Kontrollpunkt für diese Kommunikation sein kann,
- Die UPnP-Technologie ist unabhängig vom Betriebssystem, von der Programmiersprache und vom Übertragungsmittel,
- Ein Gerät kann das Netzwerk reibungslos und automatisch verlassen, ohne irgendwelche ungewollte Zustandsinformationen zu hinterlassen,
- UPnP ist in Windows Betriebsystemen integriert,
- UPnP setzt auf etablierte Internettechniken auf,
- Bei UPnP ist die Implementierung komplexerer Anwendungen einfach.

Problembereiche von UPnP:

- UPnP ist in der Spezifikation noch nicht so fortgeschritten wie Jini,
- Es gibt nur wenige Beispiele für UPnP Anwendungen,
- UPnP ist als Open-Source verfügbar aber nur für Mitglieder,

- Wenn der Remote-Zugang zusammen mit der UPnP des Betriebssystems aktiv ist, kann ohne jede Bestätigung mit Hilfe von UPnP der Verbindungsname und das Kennwort zurückgeholt werden. Es existieren bereits Exploits, die diese Sicherheitslücke ausnutzen (Stand: 09.04.2003) [Computer-security 2003]. Diese Schwachstelle ist mittlerweile mit einem Sicherheitspatch gelöst,

❖ **Sicherheitsfunktionen in UPnP**

Aufgrund des ursprünglichen Design-Ziels, eine nahtlose und einfache dynamische Integration und Vernetzung von Geräten und Diensten zu realisieren, bietet UPnP kein explizites Sicherheitskonzept an. Es wird davon ausgegangen, dass Sicherheitsmaßnahmen wie Subnetze (Gateways, Firewalls oder Proxies), MAC-Schicht-Verschlüsselung (durch Netzwerkadapter) sowie physikalische Zugriffskontrolle implementiert werden, um den UPnP Netzverkehr zu schützen [BSI 2003].

Anmerkung: Microsoft stellt durch die Technologien der Internetverbindungsfreigabe (ICS „Internet Connection Sharing“) und des Internetverbindungsfirewalls (ICF „Internet Connection Firewall“) umfassende Unterstützung von UPnP unter Windows XP bereit. Weitere Informationen zu ICS und ICF sind auf der Microsoft-Seite [Microsoft 2003] unter ‚What's New in Security for Windows XP’ zu finden.

4.3 ViaXML

“ViaXML is a mobile and wireless application infrastructure that enables web services (which provide access to data, business logic, knowledge, and application components) to be exposed and called over the Internet and corporate Intranet using open Internet standards – XML, HTTP and HTTPS. With ViaXML, developers can rapidly build and deploy a new class of node to node, fully interactive, platform independent applications, for Pocket PCs, devices powered by Windows CE, Palm OS, Windows 95/98/ME and Windows NT/2000 – over wireless LAN or WAN, direct Internet, or dial-up connection” [Odysseysoftware 2003].

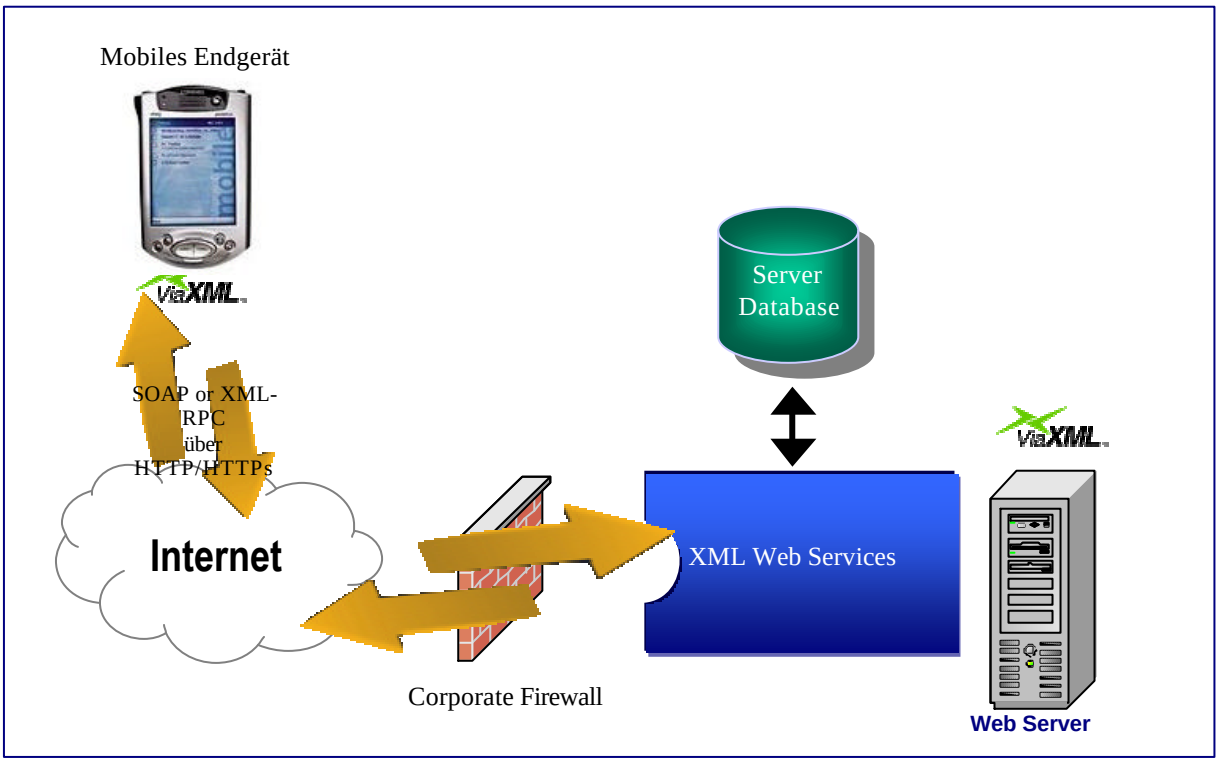
ViaXML von Odyssey Software [61] ist eine bewegliche und drahtlose Anwendungs-Infrastruktur, die Netzdienstleistungen über das Internet ermöglicht und die Fähigkeiten des mobile application developer auf einem höheren Niveau bringt. ViaXML basiert auf XML-Standards [62] (siehe den XML Abschnitt), was die Beschränkungen der neuen beweglichen Anwendungsmöglichkeiten beseitigt und sie plattformunabhängig macht.

[61] Odyssey-Software wurde 1996 gegründet und ist ein Microsoft-Partner.

[62] Das XML (*Extensible Markup Language*) ist eine reduzierte Variante von SGML (*Standard Generalized Markup Language*), das es Webentwicklern und Designern ermöglicht, individuelle Tags zu erzeugen, mit denen sich eine größere Flexibilität bei der Organisation und der Darstellung von Informationen erreichen lässt als mit dem älteren HTML-System (siehe Kapitel 3 XML-Abschnitt) [Microsoft Press 2000].

❖ ViaXML Kommunikationsmodell

Wie aus der *Abbildung 4.8* ersichtlich, findet die Dienstleistung bzw. die Kommunikation durch ViaXML zwischen einem mobilen Endgerät und dem Rechner, der über einen Web Server verfügen muss, über das Internet statt. Dieses verwendet XML-RPC mit HTTP/HTTPs[63] als Unterstützung für den Datentransport. Die Bezeichnung XML Web Services bezieht sich auf eine Reihe zusammenhängender Standards, die die Kommunikation von Programmen untereinander ermöglichen.
ViaXML kann auch für P2P-Lösungen eingesetzt werden, damit z. B. ein automatischer Informationsabgleich zwischen mobilen Endgeräten möglich ist (mobile device to mobile device, desktop to desktop, server to server).



Quelle: [Odysseysoftware 2003]

Abbildung 4.8: ViaXML Scenario

[63] HTTPs (*Hyper Text Transport Protocol Secure*) ist ein Software-Protokoll zur Kommunikation zwischen Web-Browsern und Web-Servern. HTTPs bietet, wie der Name sagt, sicherheitsrelevante Aspekte wie Informationsverschlüsselung, Identitätsnachweis und elektronische Unterschrift [Siemens Lexikon 2003].

4.3.1 ViaXML-Preis

Die ViaXML ist kostenpflichtig. Als Auswertungstest dient die Shareware-ViaXML, die auf der Odysseysoftware-Internetseite für 30 Tage zur Verfügung steht. Sie kann unter der Adresse [Odysseysoftware 2003a] bezogen werden. Allerdings muss man ein Formular ausfüllen, um sich ViaXML herunterladen zu können und für die Installation bekommt man per e-Mail den License Key. Weitere Informationen sind auf der Homepage von Odyssey-Software zu finden. Die *Tabelle 4.2* gibt einen Überblick über die ViaXML-Preise im Verhältnis zu Lizenzen.

Lizenz	Preis
5-User	$1.995,00
25-User	$6.725,00
50-User	$11.450,00
100-User	$17.900,00
200-User	$29.800,00
500-User	$65.400,00
1000-User	$99.000,00
2500-User	$222.500,00
5000-User	$345.000,00

Quelle: [Odysseysoftware 2003] Stand 03.01.03 um 13:00

Tabelle 4.2: ViaXML-Preise im Überblick

4.3.2 ViaXML System-Anforderungen

Da ViaXML auf freien XML-Standard basiert und Standard-HTTP/HTTPs Internet-Protokolle verwendet, ist sie plattformunabhängig (UNIX, Win32). ViaXML unterstützt alle Geräte, auf denen Windows CE Version 2.11 und höher (HPC Pro, Palm-Sized PC und Pocket PC) oder Palm OS Version 3.0 und höher installiert ist. Windows CE Pocket PC Emulators und cradle connections werden auch unterstützt [Odysseysoftware 2003]. Für eine funktionstüchtige ViaXML-Umgebung ist Folgendes notwendig:

- entweder Microsoft eVT 3.0 (*eMBedded Visual Tools*) oder höher (ViaXML ist in eVT mitintegriert). Diese beinhalten außer eVB (*eMBedded Visual Basic*) und eVC++ (*eMBedded Visual C++*) SDKs [Microsoft-Technet 2003], für die verschiedenen Windows CE Gerätetypen. Einigermaßen erstaunlich für ein Microsoft-Produkt ist die Tatsache, dass die eVT frei erhältlich ist. Man kann sie sich unter [Microsoft 2003a] downloaden (Das File hat 210485 KB),

- oder Microsoft Visual Basic 6.0 oder höher,
- oder Microsoft Visual C++ 6.0 oder höher,
- ActiveSync 3.1 oder höher,
- IIS oder der kostenfreie Apache-Server [Jakarta 2003] (siehe Anhang E Softwarequellen).

Zur Installation der eVT ist mindestens ein PC mit 150MHz Pentium-Prozessor notwendig. Als Betriebssystem ist entweder Windows 2000 Professional, Windows NT Workstation 4.0 (SP5) oder Windows 98 SE notwendig.

❖ **Installation der ViaXML-Software**

ViaXML™ (Mobile XML Web Services) ist mit einem Installationsprogramm ausgestattet. Bei der ViaXML-Installation muss der PDA mit dem PC verbunden sein (über Kabel, über Infrarot oder über Funk. Siehe Abschnitt 2.3.4 Eigenschaften der Vernetzung). Zur Installation der Software auf dem PDA muss Microsoft ActiveSync 3.1 oder eine höhere Version auf dem PC installiert sein. Wenn Microsoft ActiveSync nicht bereits installiert ist, bricht das Installationsprogramm den Installationsvorgang ab. Zum Ausführen des Installationsprogramms werden die Administratorrechte benötigt.

Nach der erfolgreichen Ausführung des Programms "ViaXML Device Install" wird eine viaxml.dll-datei auf dem Pfad \Windows\System32/ und als Hilfsmittel die Deinstallation-Datei "Odyssey Software ViaXML" (Mobile Device/ Windows) abgelegt, die innerhalb der Deinstallationsanwendung ausgeführt wird.
Es muss ein virtuelles Verzeichnis unter IIS namens ViaXML, das auf den ASP-Ordner "\Program Files\Odyssey Software\ViaXML\ASP" verweist, erstellt werden.

4.3.3 Protokolle, auf denen ViaXML basiert

- Übertragungsprotokoll: HTTP/HTTPs.
- Kommunikationsprotokoll: Bei der Dienstleistung verwendet ViaXML entweder XML-RPC oder SOAP (siehe hierzu XML-RPC und SOAP Abschnitte Kapitel 3).
- Web Services: Durch Web Services werden Informationen, die auf einem Webserver liegen (z. B. IIS), durch andere Programme genutzt, wobei der Aufruf der Methoden mittels SOAP oder XML-RPC erfolgt. Dabei dient ein Directory- oder Verzeichnisdienst der Verwaltung von Informationen.

4.3.4 Eigenschaften und Bewertung

Für den Einsatz von ViaXML sprechen:

- ViaXML-Technologie unterstützt HTTP,
- ViaXML verwendet sichere verschlüsselte Datenübertragung (SSL[64]),
- Integration mit Microsoft IIS (*Internet Information Server*),
- Standardanzeigenformate der XML-Unterstützung,
- ViaXML 1.30 unterstützt Pocket PC, Windows CE, Win32, and Palm OS Clients,
- Der große Vorteil ist die relativ simple Art und Weise der Funktionalitäten, da diese auf Basis von HTTP ablaufen. So ist es beispielsweise nicht notwendig, eine Firewall extra dafür zu konfigurieren.

Gegen den Einsatz von ViaXML sprechen:

- Es gibt kaum Know-How auf dem Markt,
- ViaXML ist nicht kostenfrei,
- Man ist auf einen HTTP-Server wie bei Jini angewiesen.

❖ **ViaXML-Sicherheit**

Bei der Verwendung von verteilten Anwendungen spielt häufig auch die Sicherheit eine große Rolle. Da SOAP und XML-RPC als Transportprotokoll HTTP und als Datenprotokoll XML verwenden, werden alle Daten unverschlüsselt übertragen. Zur Verschlüsselung der Kommunikation benutzt ViaXML das Standardprotokoll des Internets: HTTP- und HTTPs-Internet Protocols mit SSL (*Secure Socket Layer*) (siehe *Abbildung 4.9*), wobei keine zusätzlichen Ports in der Firewall notwendig sind, weil der über die Firewall gelieferte Inhalt ein Standard XML-Inhalt und mit der Firewall kompatibel ist. ViaXML verwendet die Authentisierung, die Autorisierung und die Verschlüsselungsservices von IIS, und mit SSL werden empfindliche Benutzerdaten geschützt [Odysseysoftware 2003].

[64] SSL (*Secure Sockets Layer*) ist ein von der Firma Netscape entwickeltes Verschlüsselungsverfahren bzw. Standard zur Verschlüsselung bei der Datenübertragung im Internet. SSL kann in Verbindung mit den Anwendungsprogrammen SMTP, Telnet, FTP und HTTP eingesetzt werden und setzt auf TCP/IP auf [Siemens Lexikon 2003].

4.3.5 Fallstudie: ViaXML

Nachdem nun die grundlegenden Mechanismen der ViaXML-Technologie erarbeitet wurden, wird im folgenden Abschnitt anhand einer Fallstudie auf die praktische Arbeit mit der Technologie eingegangen. Aufgrund des Umfangs der Quelltexte wird hier der Code nicht vollständig abgedruckt (siehe hierzu Anhang G) und die einzelnen Programmzeilen werden auch nicht diskutiert. In diesem Abschnitt werden nur einige Besonder- und Eigenheiten erwähnt, die während des Ablaufs aufgetreten sind. Dadurch wird erkennbar, auf welche Probleme man bei der Verwendung der ViaXML-Technologie stoßen kann oder auch nicht.

Im Folgenden ist in einem ViaXML-Beispielszenario eine Anwendung dargestellt, die drei Zahlen addiert und das Ergebnis zurück liefert. Das Szenario musste nicht so kompliziert bzw. im Detail ausmodelliert sein, sondern vielmehr die folgenden Grundprinzipien aufgeführt werden: Wie funktioniert die Zusammenarbeit zwischen PC und PDA mittels der ViaXML-Middleware? Wie wird ein Web Service mittels ViaXML und Microsoft IIS erstellt? Wie funktioniert der Aufruf dieser Services von eVB-Applikation (Client), die auf Pocket PC 2002 installiert sind? Das folgende Beispiel, das die ViaXML-Software beinhaltet, soll das Ganze demonstrieren.

Man kann sich das Beispielszenario so vorstellen: Ein Server bietet im Internet eine Add-Methode an. Das Script, welches die Formulareingaben entgegennimmt, ruft hierbei eine Funktion auf dem entfernten Server auf und übergibt die drei Zahlen. Als Antwort des Funktionsaufrufes erhält das Script dann das Resultat. In diesem Falle baut das damit aufgerufene Script eine HTTP-Verbindung zu einem vorher festgelegten Server auf und sendet die Zahlen in den Variablen X, Y, Z dorthin. Dieser berechnet das Resultat und sendet das Ergebnis zurück.

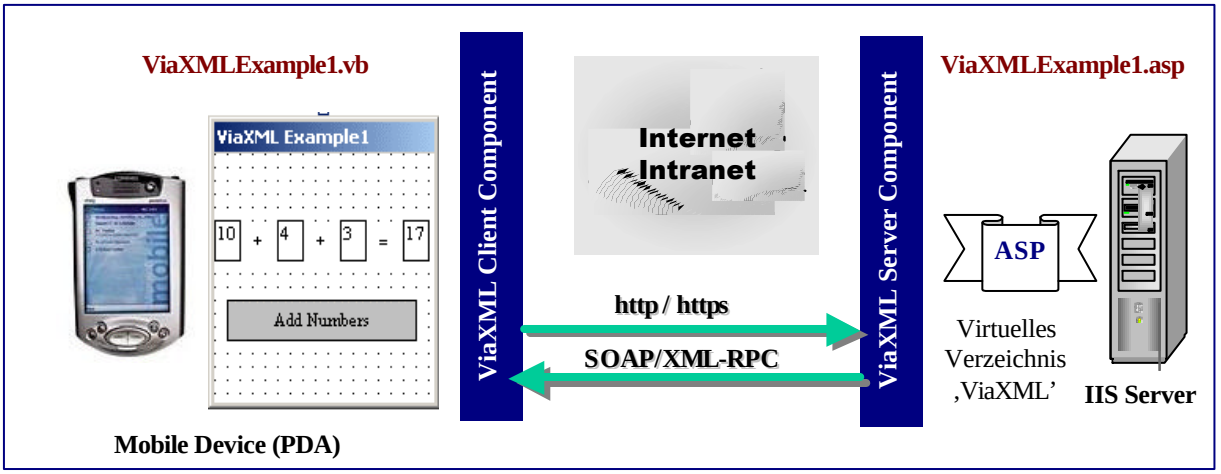


Abbildung 4.9: Applikation Scenario

Die *Abbildung 4.9* zeigt eine grafische Darstellung der Datenkommunikation zwischen PDA und PC durch ViaXML. Die Client-Seite beinhaltet: viaxml.dll, die Applikation ‚ViaXMLExample.vb', den License Key und die ServerURL. Auf der Server-Seite ist ein ViaXML-Virtuelles Verzeichnis, dem der Pfad zu ViaXML-ASP Pages bekannt ist, und ‚ViaXMLExample1.asp' aufruft.

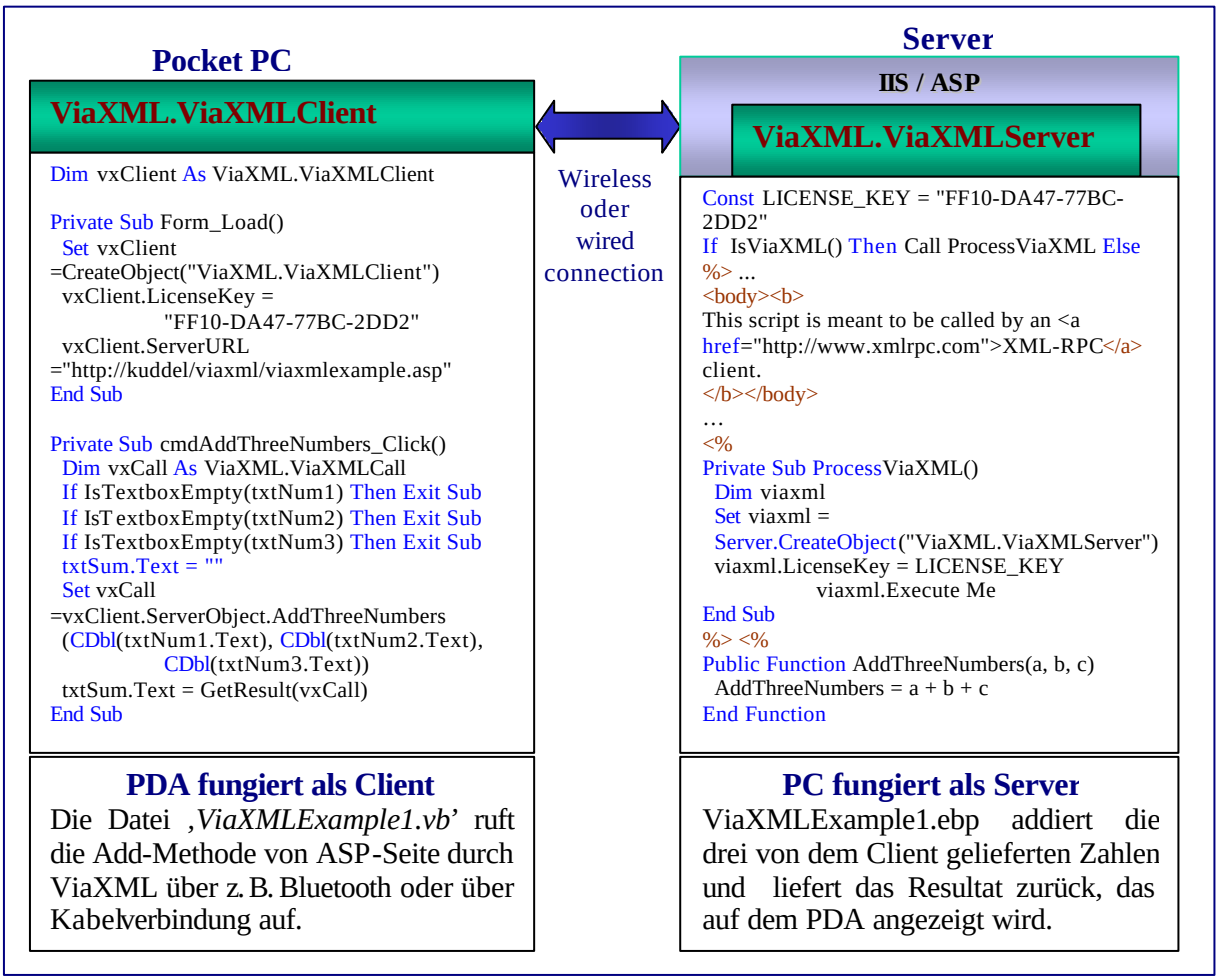


Abbildung 4.10: ViaXML Sample Code - Das typische Client-Server Scenario

Das Beispiel besteht aus folgenden Dateien:

- In dem ASP-Skript[65] ‚*ViaXMLExample.asp*', das ViaXML zum Definieren der Web Services benutzt und unter IIS abgelegt ist, ist die Methode ‚*AddThreeNumber(x, y, z)*' definiert, die nur drei Zahlen addiert und das Resultat zurück liefert.
- ‚*ViaXMLExample1.ebp*' ist die eVB-Projektdatei.
- ‚*frmMain.ebf*' ist die eVB-Form, die auf dem PDA angezeigt wird.

Die *Abbildung 4.10* zeigt die Kommunikation zwischen dem ViaXMLClient und dem ViaXMLServer. Es wird nicht auf die einzelne Codezeile eingegangen, da der aufgeführte Code nur einen groben Überblick über die verwendeten Methoden auf beiden Seiten geben soll (der Programmcode ist im Anhang G zu finden).

Anmerkung: Für den nicht vernetzten Fall hat jeder PC den Namen ‚local host', der sich in die IP-Adresse 127.0.0.1 auflöst.

4.4 CORBA / JacORB

CORBA[66] ist ein, von OMG standardisiertes, Framework zur Entwicklung von verteilten objektorientierten Anwendungen. Ein Object Request Broker (ORB) dient als Vermittler zwischen einem Client und einem Server. Die Plattform- und Programmiersprachen-Unabhängigkeit wird durch die Definition der Objekt-Schnittstellen in einer Interface Definition Language (IDL) gewährleistet. Aus dieser IDL-Definition generiert ein IDL-Compiler programmiersprachenspezifische Schnittstellen und Vertreter-Klassen, die für die Anwendungsentwicklung und zur Realisierung einer CORBA-Objektreferenz verwendet werden. Weitere Details über CORBA sind den entsprechenden Internetseiten der OMG unter [OMG 2003] zu entnehmen.

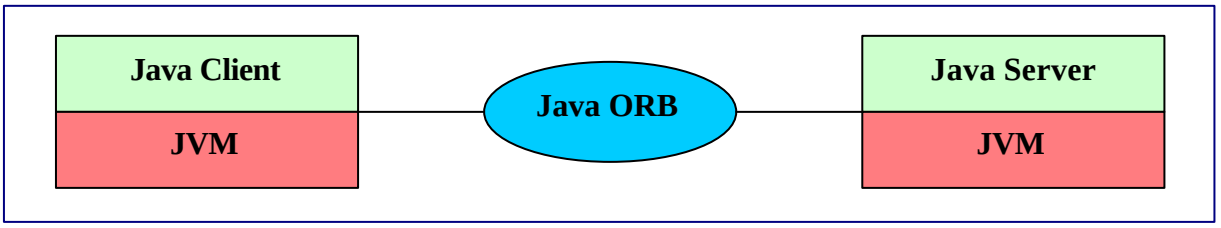
CORBA ist besonders für den Einsatz in heterogenen Umgebungen gut geeignet [OMG 2003]. Weiterhin hat man eine große Produkt-Auswahl durch die Existenz einer Vielzahl von ORB-Implementierungen. Für die HAW sind im Moment die Open-Source-Produkte von Interesse. Aus der Vielzahl der frei verfügbaren CORBA-Implementierungen sollen in diesem Zusammenhang der auf C++ basierende MICO[67] und der auf Java basierende JacORB erwähnt werden. Da MICO in meiner Studienarbeit untersucht wurde (siehe [Heinrich 2003]), wird JacORB im nachfolgenden Abschnitt erklärt.

[65] *Active Server Page*
[66] Um die Wiederholungen meiner Studienarbeitsabschnitte in dieser Diplomarbeit zu vermeiden, wird hier nicht auf CORBA-Spezifikationen eingegangen. Interessierte werden gebeten, meine Studienarbeit [Heinrich 2003] zu lesen. Für mehr Information zu diesem Thema ist [Coulouris et al. 2002] empfehlenswert.
[67] MICO steht für (*Mico Is CORBA*), ist CORBA 2.1-konform und wurde speziell für Lehrzwecke entwickelt.

4.4.1 Kombination von CORBA und Java

Aufgrund der Nähe der Objektmodelle von CORBA und Java ließ sich laut Gerald Brose und Boris Bokowski von der Freien Universität Berlin (FUB) [JacORB 2003] bei der Erweiterung des Java-Objektmodells um CORBA-Konzepte ein hoher Grad an Verteilungstransparenz erreichen. Die Abbildung des CORBA-Objektmodells in das Java-Objektmodell soll verhältnismäßig einfach sein. Entsprechend sollen die Entwicklung von CORBA-Anwendungen mit JacORB (Java Object Request Broker) kein großes konzeptionelles Umdenken gegenüber der Entwicklung verteilter Anwendungen in einer homogenen Java-Umgebung erfordern, denn fast alle Java-Interfaces lassen sich in CORBA-IDL ausdrücken. Das bedeutet, dass die Entwicklung von CORBA-Applikationen mit JacORB, der unter [JacORB 2003a] verfügbar ist, sich nur wenig von der herkömmlichen Entwicklung in Java unterscheidet [JacORB 2003].

Eine Kombination von CORBA und Java bietet sich an, da sich die Stärken der beiden Technologien ideal ergänzen und die jeweiligen Schwächen beheben. Wie in der *Abbildung 4.12* ersichtlich ist, können Clients und Server durch Java ORB kommunizieren, die komplett in Java implementiert ist und von der Portabilität dieser Programmiersprache profitiert. Auf Server-Seite kann für eine geforderte Aufgabe die am besten geeignete Plattform und Programmiersprache ausgewählt werden. Clients wiederum lassen sich durch die Portabilität von Java nahezu überall einsetzen.



Quelle: [Uni-Ulm 2003]

Abbildung 4.11: Die Verbindung von CORBA und Java

In einer CORBA und Java Welt kommt IIOP als Protokoll zwischen ORBs zur Anwendung und Java-IDL als Schnittstelle im JDK.

4.4.2 JacORB Architektur

Das Ziel der Entwicklung von JacORB war es, eine Plattform für die verteilungstransparente Entwicklung verteilter Anwendungen in Java und deren Integration in CORBA bereitzustellen. JacORB besteht aus einem IDL-Compiler, einem Vertretergenerator, einem Laufzeitsystem und einem als CORBA ObjectService implementierten Namensdienst. Intern

verwendet JacORB IIOP (Internet Inter-ORB Protocol) und ist daher mit anderen CORBA-Implementierungen interoperabel[68]. Die ORB-Funktionalität wird vollständig durch die Stubs und Skeletons realisiert und bereitgestellt. Es ist keine spezielle Konfiguration von Netzwerkknoten nötig, auf denen JacORB-Anwendungen oder -Server laufen sollen.

❖ **Voraussetzung für JacORB**

Die einzige Voraussetzung ist der Zugriff auf einen HTTP-Server, der zum Auffinden des JacORB-Namensdienstes benötigt wird, denn die vom Generator erzeugten Stubklassen verwenden generell den Namensdienst, um den Dienstanbieter zu lokalisieren, den sie bei den Clients vertreten. Ein explizites Adressieren von Objekten geschieht nur über Namen, nicht über Netzwerkadressen [JacORB 2003]. Ein JVM ist auch bei dem JacORB-Einsatz erforderlich.

4.4.3 JacORB Eigenschaften

"JacORB is a partial implementation of the OMG CORBA standard. It was developed at Freie Universität Berlin as a freely distributable ORB with the aim of making CORBA technology easily accessible and providing an implementation of an ORB which could be used as a kind of textbook example for teaching basic ORB technology and principles" [JacORB 2003].

JacORB ist eine auf Java basierende Open-Source ORB-Implementierung eines CORBA ORBs, die von der Software Engineering und Systems Software Group des Fachbereichs Informatik an der FUB entwickelt wird. JacORB ist ein compliant CORBA V2.3 mit den folgenden Eigenschaften:

- Support for IIOP 1.2,
- Java Bindings,
- Open Source (LGPL[69]),
- 100% reines Java, JDK 1,1 und höher, alle Plattformen mit JVM,
- High-performance, Fully Multithreaded ORB,

[68] Für die Interoperabilität zwischen ORBs ist das *General Inter-ORB Protocol* (GIOP) zuständig, das Transfersyntax und Nachrichtenformate CDR (*Common Data Representation*) spezifiziert. Für ein beliebiges verbindungs-orientiertes Transportsystem wird auf ein konkretes Transportprotokoll abgebildet. IIOP ist GIOP auf TCP/IP [JacORB 2003].
[69] *Library General Public License.*

- POA (Portable Object Adapter)[70],
- JacORB unterstützt die folgenden CORBA Services:
  - Naming,
  - Trading,
  - Transaction,
  - CORBA Event-Service (COS Event),
  - Collection,
  - Concurrency[71],
- Implementation und Interface Repositories,
- Portable Interceptors,
- GUI tools für POA, Interface Repository, Implementation Repository, Naming Service, Domain Management,
- HTTP Tunneling und IIOP Proxy,
- IIOP über SSL.

Die *Tabelle 4.3* stellt die Core-Funktionen und Services-Funktionen, die von JacORB unterstützt bzw. nicht vorhanden sind, dar. Die Unterstützung für Asynchronous Messaging Interface, Security Service sowie Notification Service sind geplant.

❖ JacORB und Pocket PC 2002

Konkrete Versuche auf mobilen Plattformen haben die JacORB-Hersteller Laut Gerald Brose -JacORB Mitarbeiter- nicht selbst gemacht. Aber ein Studienkollege von mir, Piotr Wendt, hat die JacORB Version 1.3.30 auf dem Pocket PC 2002 installiert und ohne Schwierigkeiten ein C/S-Beispiel zum Laufen gebracht. Zusammengefasst: JacORB läuft auf Pocket PC 2002 ohne Probleme. Zu mehr Informationen wird hier auf Piotr Wendts Diplomarbeit unter der Betreuung von Prof. Dr. Kai von Luck verwiesen.

[70] Ein Objekt Adapter ist ein Mechanismus, der objektspezifisches Verhalten vom ORB-Kern entkoppelt und eine Anfrage mittels einer Objektreferenz mit dem korrekten Objekt für diese Referenz verbindet. Ein POA ist ein spezieller Objekt Adapter, welcher in der CORBA-Spezifikation definiert ist und die Verwaltung von Objekten mit sehr unterschiedlichen Eigenschaften ermöglicht [Puder & Römer 2001].

[71] Der Concurrency Control Service enthält Mechanismen, um den gleichzeitigen und konkurrierenden Zugriff mehrerer Objekte auf ein Objekt zu kontrollieren und die nebenläufigen Ausführungen zu managen. Es werden Verfahren unterstützt, um Deadlocks zu verhindern, zu entdecken und zu beheben.

4.4.4 JacORB Produkt Profile

Core	
Dynamic Invocation Interface (DII)	Yes
Dynamic Skeleton Interface (DSI)	Yes
Portable Object Adapter (POA)	Yes
Basic Object Adapter (BOA)	No
Portable Interceptor	Yes
Current interface	Yes
Interoperable Naming Service	Yes
Interface repository	Yes
Value Type Semantics	Yes
Wide characters	Yes
CORBA Component Model	No
Minimum CORBA	No
Multi-Threading	Yes
Real-Time CORBA	No
Asynchronous Messaging Interface	Planned
Quality of Service (QoS) Management	No
Fault tolerance	No

Services	
Collection Service	Yes
Persistent State Service	No
Concurrency Service	Yes
Property Service	No
Event Service	Yes
Query Service	No
Externalization Service	No
Relationship Service	No
Naming Service	Yes
Security Service	Planned
Licensing Service	No
Time Service	No
Life Cycle Service	No
Trading Service	Yes
Notification Service	Planned
Transaction Service	Yes

Quelle: [Puder 2003] Last updated: Freitag, den 18 Okt. 2002

Tabelle 4.3: JacORB Matrix

4.5 Einordnung der Middleware in das OSI-Modell

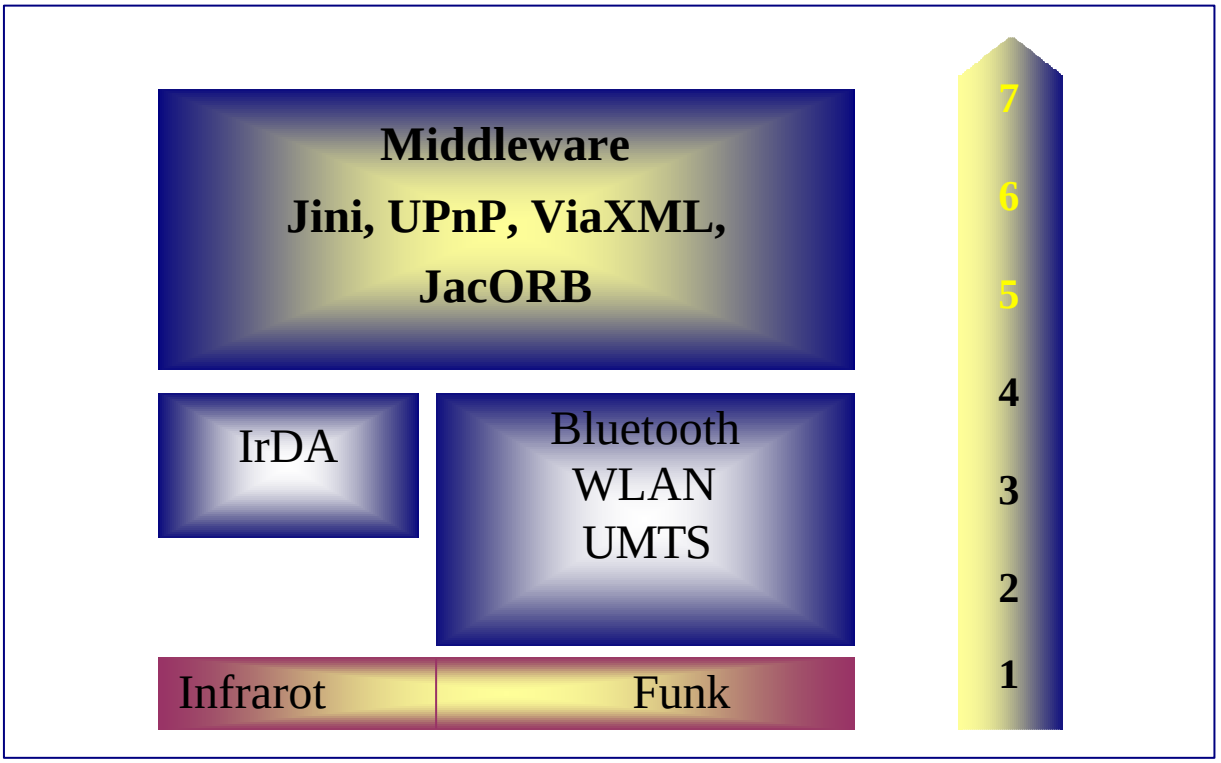


Abbildung 4.12: Einordnung der Middleware in das OSI-Referenzmodell

Da Jini-, UPnP-, ViaXML- und JacORB-Technologien die Transportart verbergen und manche von ihnen das TCP/IP Netzwerk voraussetzen wie Jini und Java 2 Standard Edition, sind sie oberhalb des OSI Transport Layers 4 anzusiedeln (*Abbildung 4.11*).