MIDDLEWARE GRUNDLAGEN 3

In vielen Unternehmen besteht die IT-Infrastruktur u. a. aus unterschiedlichen Hardware- und Betriebssystemplattformen. Zur Lösung der Heterogenitäts- und Verteilungsprobleme bei verteilten Anwendungen stehen verteilte Systemservices zur Verfügung, welche eine Standardschnittstelle und ein Standardprotokoll besitzen. Diese Services heißen Middleware Services oder kurz Middleware. Sie ist das zentrale Thema bei der Systemintegration. Ihr Einsatz soll eine Vereinfachung bei der Interoperabilität[28] in verteilten Umgebungen bringen. Je nach IT-Architektur und Problemstellung wird eine unterschiedliche Middleware verlangt. Sie beinhaltet fast immer eine mehr oder weniger hoch entwickelte Kommunikationskomponente, die Clients mit Servern verbindet. Sie kann aber weit darüber hinausgehen und durch unterschiedlichste Dienste und Funktionalitäten weitere Transparenzebenen realisieren.

3.1 Was ist Middleware?

Grob gesagt, ist Middleware alles, was den Software-Entwicklern hilft, vernetzte bzw. verteilte Applikationen zu erstellen. Im Allgemeinen besteht Middleware aus mehreren Komponenten, die auf verschiedenen Schichten des OSI-Modells arbeiten, und bei denen es entsprechende Gegenstücke auf den Client- und Serverrechnern gibt. Mehr Informationen sind [Bernstein 1996] (siehe die beiliegende CD) zu entnehmen.

Der Begriff *‚Middleware'* spielt eine Schlüsselrolle in dieser Arbeit. Leider gibt es keine einheitliche Definition darüber, was Middleware ist. Gleichzeitig aber ist es wichtig, innerhalb dieses Dokuments eine präzise und klare Vorstellung genau davon zu entwickeln, um die untersuchten Technologien richtig einzuordnen. Um dieses Verständnis zu ermöglichen, wird Middleware in Anlehnung an [Coulouris et al. 2002] wie folgt bestimmt:

Definition

„Middleware ist eine Softwareschicht, die eine Programmierabstraktion bereitstellt und die Heterogenität der zu Grunde liegenden Netzwerke, Hardware, Betriebssysteme und Programmiersprachen verbirgt.“

[28] Der Begriff der Interoperabilität besitzt in verschiedenen Kontexten unterschiedliche Bedeutung. Im Kontext dieser Diplomarbeit soll darunter die Fähigkeit unterschiedlicher Hard- und Softwarekomponenten verstanden werden, miteinander kooperieren zu können.

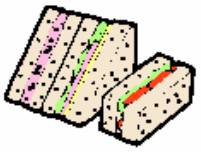
Diese Definition von Middleware ist eine typische, aber bei weitem nicht die einzige. Als die wichtigsten Standards bezüglich Middleware gelten CORBA[29] sowie andere Object Broker-Architekturen z. B. DCOM[30]/ActiveX, Java RMI[31], Jini, etc. [Microsoft GmbH 2002].

3.2 Middleware-Funktion

Unter Middleware-Funktion versteht man eine Menge von allgemeingültigen Diensten, die zur Verringerung der Abhängigkeiten zwischen den Ebenen der Systemplattform (d. h. Hardware und Betriebssystem) und der Anwendung angesiedelt sind. Diese Dienste stellen die Grundlage für die Verteilung von Anwendungen dar. Die Middleware baut auf Standard-Schnittstellen und Protokollen auf und ist selbst über Standard-Schnittstellen nutzbar [Puder & Römer 2001].

Anwendungen können auf vielfältige Weise miteinander kommunizieren. Client-Anwendungen kommunizieren beispielsweise synchron mit Anwendungsservern, um Eingabedaten zu speichern. Middleware stellt für solche Szenarien erweiterte Kommunikationsmodelle zur Verfügung, die die Systemplattform nicht bieten kann. Durch den Grad der Synchronisation zwischen den Anwendungen wird das Kommunikationsmodell vorgegeben. Leicht gekoppelte Kommunikation zwischen Anwendungen kann häufig auch auf Basis der Nachrichtenübermittlung erfolgen.

Zusammenfassend gesagt, ist Middleware eine Softwareschicht, welche zwischen Betriebssystem und Anwendung liegt. Diese Software sorgt für die Überwindung der Verteilung, der verschiedenen Betriebssysteme, der verschiedenen Rechnerplattformen und der verschiedenen Programmiersprachen. Das Resultat ist die in der unteren *Abbildung 3.1* dargestellte Systemarchitektur. Die nachfolgende Metapher soll die Wichtigkeit dieser Softwareschicht betonen.

Was wäre ein Sandwich  ohne was „dazwischen“?

[29] CORBA (*Common Object Request Broker Architecture*) ist ein frei verfügbarer von der OMG (*Object Management Group*) standardisierter objektorientierter Middleware-Entwurf [Puder & Römer 2001].
[30] DCOM (*Distributed Component Object Model*) ist eine Variante des Microsoft-Software-Standards COM „*Component Object Model*”.
[31] Java RMI (*Remote Method Invocation*) ist ein Middleware-Ansatz, der Kommunikation zwischen Servern und Klienten mit Hilfe von Methodenfernaufrufen modelliert.

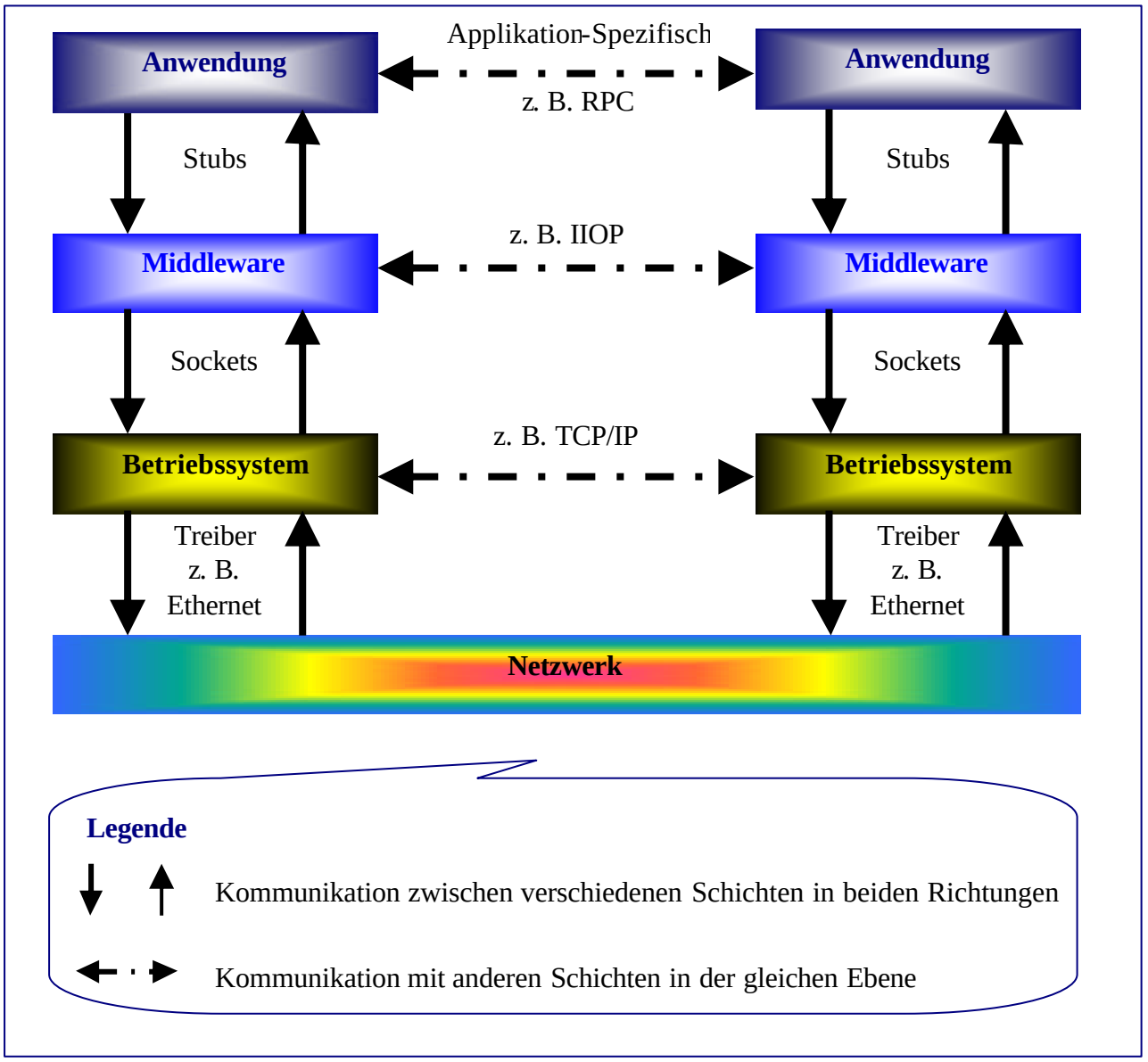


Abbildung 3.1: Middleware

Einige Beispiele für Middleware-Dienste sind:

- GUI-Frameworks zur Abstraktion von konkreten Darstellungsmedien
- Druck-Services

- RPC-Architekturen[32] auf der Basis von CORBA und nachrichtenorientierte Middleware (MOM)[33] wie MQ-Series[34]
- Das Management-System einer verteilten Datenbank
- Remote Login, File Transfer und Authentisierungs-Server.

3.3 Middleware-Mechanismen (Typologie)

Im Grunde gibt es zwei primäre Methoden, zwei Systeme miteinander interagieren zu lassen: per Nachricht oder per Schnittstelle.

- Erfolgt die Interaktion per Nachricht, so sendet ein System, der Sender, eine Nachricht über das Kommunikationssystem an den oder die Empfänger. Die Nachricht enthält Daten und/oder Aktionen, die vom Empfänger verarbeitet werden. Falls der Sender eine Antwort erwartet, so dreht sich die Kommunikationsrichtung, d. h. der Empfänger wird zum Sender und sendet die Antwort an den Empfänger, der zuvor die Anforderung gestellt hat.

- Bei der Interaktion über eine Schnittstelle ruft das anfordernde System direkt eine Funktion oder Prozedur auf einem entfernten System auf, das diese Anforderung bearbeitet und ggf. einen Rückgabewert als Antwort liefert.

Die Middleware-Produkte können anhand der Kommunikationsarten unterschieden werden. Dabei wird man allerdings auch auf Produkte stoßen, bei denen mehrere Wege kombiniert werden. Die Middleware verwendet häufig einen von drei alternativen Mechanismen für die Kommunikation mit der Transportschicht [IBM-Whitepaper 2003]: RPC, ORB (*Object Request Broker*) oder MBQ (*Message Based Queuing*) auch als (*Message based Middleware*) (MOM) bezeichnet. Alle drei Alternativen lösen komplexe, systemspezifische Kommunikations- und Konfigurationsaufgaben. Sie machen die Location der angeforderten Dienstleistung transparent für den anfordernden Klienten. Sie lösen jedoch diese Aufgabe auf eine sehr unterschiedliche Art [Uni-Tuebingen 2003]. Mehr zu Middleware-Produkten ist unter [Middleware 2003] aufgelistet.

[32] Ein Beispiel für RPC (*Remote Procedure Call*), also ein ferngesteuerter Prozedur-Aufruf, wäre ein, über ein Netzwerk, angesprochener Server, der ein Programm startet und das Ergebnis zurückliefert.
[33] MOM (*Message Oriented Middleware*) ist eine Gruppe von Programmen, die Daten und Nachrichten zwischen Anwendungen und Kommunikationsdiensten (z. B. TCP/IP), die unterschiedliche Formate verwenden, übertragen [Microsoft Press 2000] (siehe Abschnitt 3.3.3 Nachrichten orientierte Middleware).
[34] MQ-Series ist ein Middleware-System der Firma IBM, das mit Message Queuing arbeitet, um Programme verteilter Anwendungen miteinander kommunizieren zu lassen. MQ-Series ist sowohl für Großrechner als auch für Unix-, OS/2-, Windows- und Java-Systeme verfügbar und hat 65% Marktanteil [IBM 2003].

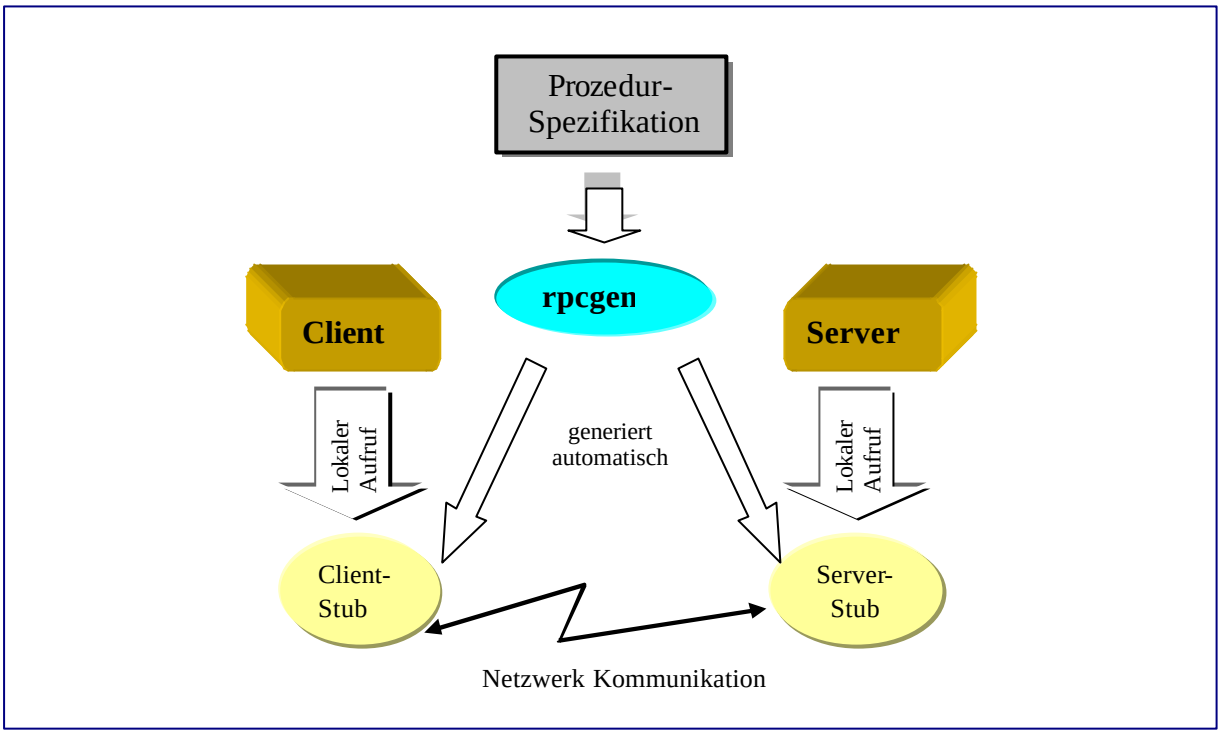
3.3.1 RPC-basierte Middleware-Produkte

RPCs gehören zu den ältesten Formen der Kommunikation zwischen zwei Programmen. Die Programme kommunizieren synchron miteinander und verwenden dabei ein Modell, bei dem ein Programm eine Anforderung sendet und dann auf die Antwort des anderen Programms wartet. Die Prozeduren einer Anwendung können über das Netz aufgerufen werden. Dazu werden diese Prozeduren in einer Interface-Definitionssprache formuliert, aus denen ein Compiler dann einen Code zum Einbinden in den Server und den Client generiert (siehe *Abbildung 3.2*).
RPC verwendet auf Client- und Serverseite so genannte Stubs (Prozedurstümpfe), d. h. kleine Objekte, die für den Client bzw. den Server als lokale Objekte ansprechbar sind und die für die Nachrichtenübertragung zuständig sind. Die Client-Stub-Funktion ist für die Versendung der Argumente zum Server und für die Entgegennahme der Resultate zuständig. Sie enthält den Code zum ‚Verpacken' der Parameter und zum Abschicken der Nachricht über das Netzwerk [Bengel 2002].
Der Server-Teil wird auch Skeleton genannt. In ihm sind bereits die Methoden zum Empfangen der Nachricht über das Netzwerk und das ‚Entpacken' der Parameter realisiert. Er hat zwei Aufgaben zu erledigen (siehe *Abbildung 3.2*):

i. Die Argumente müssen ausgepackt und ggf. konvertiert werden.
ii. Die Resultate müssen berechnet werden, in Nachrichten verpackt (vor dem Verpacken ggf. konvertiert) und zurückgesendet werden. Das Auspacken der Argumente sowie das Konvertieren und Verpacken der Resultate wird dem Server-Stub überlassen.

Der Client-Stub gibt die Ergebnisse nach oben zum Anwendungsprozess ‚der den Eindruck hat' als habe der Client-Stub die Berechnung selbst durchgeführt. Dieses Vorgehen ist in *Abbildung 3.2* grafisch dargestellt.

Ein bekannter Vertreter des RPCs ist die RMI (*Remote Method Invocation*), die ein Teil des JDK (*Java Development Kit*) von Sun ist (siehe hierzu Jini-Abschnitt). RPC ist auch die Grundlage vieler anderer Technologien; u. a. basieren CORBA und Microsoft's DCOM auf den RPC-Prinzipien [Haase 2001]. Die Kommunikation mittels RPC ist am besten für kleine, einfache Anwendungen geeignet, die in erster Linie über Punkt-zu-Punkt-Verbindungen miteinander kommunizieren. Für einen RPC-Überblick lohnt es sich, auf die Seite von Prof. Dr. Klauck zu schauen [Klauck 2003].



Quelle: [Haase 2001] Seite 9

Abbildung 3.2: Generierung und Verwendung von Client- und Server-Stubs

3.3.2 ORB-basierte Middleware-Produkte

ORBs sind sprachenunabhängige, objektorientierte RPCs, die durch die Kapselung einer anwendungsspezifischen Oberfläche den Fernzugriff auf diese Oberfläche ermöglichen. Die Integration eines ORB gestaltet sich ähnlich wie beim RPC, nur dass man statt einzelner Methoden nun komplette Objekte deklariert. Auch hier werden wieder Stubs und Skeletons von einem Compiler generiert.

Der ORB ist die Kernkomponente von OMA (*Object Management Architecture*) und fungiert als Telefonzentrale in einem verteilten System (siehe *Abbildung 3.3*). Er stellt Dienste für die Initialisierung und Auffindung von Zielobjekten und für die Übermittlung von Methodenaufrufen (requests) sowie deren Resultaten zwischen Clients und Zielobjekten zur Verfügung. Er vermittelt zwischen den verteilten Objekten und sorgt dafür, dass die Methodenaufrufe zum passenden Zielobjekt geleitet werden, dieses aktiviert wird (falls erforderlich) und die Ergebnisse zum Aufrufer gelangen [Bengel 2002]. Beispiele für ORB sind: JacORB[35] [JacORB] und MICO [MICO 2003].

[35] JacORB ist eine nichtkommerzielle Implementation eines CORBA ORBs, die in Java programmiert ist und nur mit dieser Sprache zusammenarbeitet.

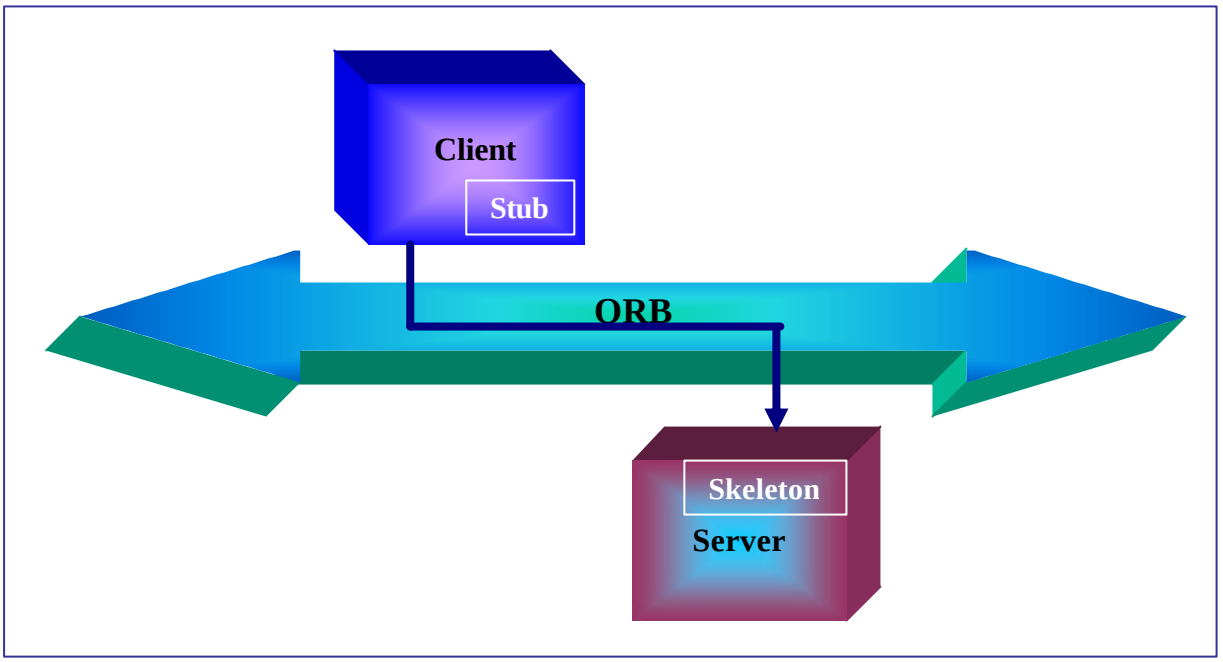


Abbildung 3.3: ORB-Struktur

ORBs bieten den Vorteil, dass entfernte Clients und Server wesentlich einfacher programmiert werden können, da die Komplexität der verteilten Programmierung verborgen bleibt. Sie vereinfachen die Zusammenarbeit zwischen Systemen, die auf verteilten Objekten basieren, weil die Anwendungen aus Objekten verschiedener Anbieter erstellt werden können, die über den ORB miteinander kommunizieren [Uni-Tuebingen 2003].

3.3.3 Nachrichtenorientierte Middleware-Produkte

"Message Oriented Middleware (MOM) is one of three categories of connectivity middleware that provide program-to-program communications by message passing. MOM, because it generally supports multiple protocols, comprises an infrastructure, that will support reliable and scalable high-performance distributed application networks. Most Message Oriented Middleware is implemented with queued message store-and-forward capability, that is Message Queuing Middleware (MQM)" [Middleware 2003].

Der asynchrone Austausch von Nachrichten wird in verteilten Softwaresystemen immer wichtiger. MOM (*Message Oriented Middleware*)[36] ermöglicht einen solchen asynchronen Austausch von Nachrichten und erlaubt somit eine lose Kopplung der Softwarekomponenten.

[36] MOM basiert auf dem asynchronen (store-and-forward) Modell, welches die Zustellung wichtiger Nachrichten selbst dann garantiert, wenn ein Gerät zeitweise keine Netzwerkverbindung hat [Raasch 2002].

MOM kommt immer dann zum Einsatz, wenn Informationen in einem heterogenen Netzwerk ausgetauscht werden sollen, bei denen die verschiedenen Teilnehmer unterschiedliche Protokolle verwenden. MOM ist eine Middleware, die schnellstmöglich die Kommunikation zwischen Client- und Server-Anwendungen über Nachrichten abwickelt [Raasch 2002]. Sie unterstützt die Kommunikation zwischen Objekten und Anwendungen, indem Nachrichten gespeichert und zu einem späteren Zeitpunkt asynchron an die Zielanwendungen gesendet werden. Gerade diese Nachrichten geben dieser Middleware den Namenszusatz *‚Message-Oriented'*. Eine solche Nachricht besteht wenigstens aus der ID des Absenders, der ID des Empfängers und natürlich dem eigentlichen Request oder Response als Inhalt. Zur Übermittlung der Nachrichten werden primär zwei Modelle eingesetzt: Das Message Queuing (Nachrichtenwarteschlange) und das Message Passing (Nachrichtenaustausch). Beide Modelle gewährleisten eine sichere Übertragung einzelner Nachrichten [IBM-Whitepaper 2003].

- Message Queuing ist eine Form der asynchronen verteilten Programm-Programm-Kommunikation. Programme einer Anwendungslösung kommunizieren miteinander, in dem sie Messages über Queues austauschen, statt sich direkt aufzurufen, wie es bei RPC-Lösungen der Fall ist. Die Programme können bei einigen Lösungen auch auf verschiedenen Rechnern im Netzwerk verteilt sein [IBM 2003]. Beim Message Queuing wird die Punkt-zu-punkt-Kommunikation über eine Nachrichtenwarteschlange unterstützt.

- Das Message Passing basiert auf dem Publish/Subscribe-Modell[37], bei dem eine Nachricht automatisch an alle Interessenten gesendet wird, die die Nachricht abonniert haben. Message-Passing-Systeme erlauben die direkte Übertragung von Nachrichten vom Sender zum Empfänger, ohne die Verwendung expliziter Warteschlangen. Die Kommunikation erfolgt verbindungsorientiert.

Die Nachrichtenorientierung gibt MOM die Fähigkeit, neben der synchronen auch die asynchrone Kommunikation zwischen Client und Server verwalten zu können[38]. Dieser Umstand ist entscheidend für den Einsatz von MOM. Für die synchrone Kommunikation alleine würde auch eine weniger komplexe Middleware ausreichen. Wird MOM doch einmal für synchrone Kommunikation benutzt, so kommt die Möglichkeit des asynchronen Datenaustausches wieder im Falle der Nichterreichbarkeit eines Servers zum Tragen [Raasch 2002]. Ein Beispiel für MOM ist die IBM WebSphere MQ Produkt Familie.

Anmerkung: Der Einsatz von RPCs ist vor allem bei der Entwicklung von neuen Anwendungen vorteilhaft, wenn keine oder nur wenige nicht-RPC Anwendungen integriert werden müssen. Message Based Queuing ist besser geeignet, wenn es um die Integration

[37] Das Publish-/Subscribe-Modell ermöglicht, dass es mehrere Nachrichtenquellen in der Rolle eines Senders (Publisher) und ggf. mehrere Empfänger (Subscriber) für diese Nachrichten gibt.

[38] Bei synchroner Nachrichtenübermittlung wird der Request einfach direkt an den Server weitergeleitet, ebenso der Response zum Client. Im Gegensatz dazu wird bei asynchroner Server-/Client-Kommunikation der Request als Nachricht gespeichert, bis sich der Server meldet. Dieses Konzept bietet den Vorteil, dass der Server weder direkt mit dem Client verbunden noch überhaupt verfügbar sein muss, damit ein Request auch abgesetzt werden kann und trotzdem erfolgreich bearbeitet wird [Raasch 2002].

unterschiedlicher heterogener Anwendungen geht. Die Middleware verbirgt die Komplexität beim Einsatz von RPC oder MBQ vor uns als Anwendungsprogrammierern [Uni-Tuebingen 2003].

3.4 Mobile Middleware

Bei der Entwicklung von mobilen Informationssystemen spielt mobile Middleware mehr und mehr eine zentrale Rolle. Mobile Middleware ermöglicht dem Endgerät, eine Verbindung mit dem Server selbst dann aufrecht zu erhalten, wenn die Netzabdeckung zeitweise nicht gewährleistet ist. Mobile Middleware kann zudem eine Verschlüsselung der Daten bewerkstelligen, Transaktionen steuern oder mobilen Endgeräten einen Zugriff auf einen JNDI[39] Namendienst im Rechenzentrum ermöglichen [Softwired 2002].
Die Übertragungsraten schwanken oft stark und mobile Geräte verlieren oft die Netzwerkabdeckung (d. h. die "Verbindung" zum Rechenzentrum). Daher muss speziell das Kommunikationsumfeld von mobilen Anwendungen berücksichtigt werden. Zudem gibt es in der mobilen Welt etliche Standards zur mobilen Datenübertragung. Das bedeutet, dass Middleware eine Auswahl an Übertragungsprotokollen (Bluetooth, IrDA, GPRS, UMTS, etc.) hat und Dienstqualitäten (zuverlässig, verschlüsselt) unterstützen muss [Softwired 2002].

3.5 Client/Server Middleware

Das Client/Server Modell beschreibt das wohl geläufigste Paradigma zur Modellierung und Realisierung von verteilten Anwendungen. Bei diesem C/S-Prinzip wird das gesamte System in die Funktionseinheiten des Dienstgebers (Servers) zerlegt. Der Server stellt einen Prozess oder eine Prozessgruppe dar. Die Dienstnehmer (Clients) sind andere Prozesse, die die Dienste des Servers in Anspruch nehmen. Die Kommunikationsobjekte zwischen Client und Server sind im allgemeinen Kanäle und zwar ein Eingangskanal, an dem der Server die Aufträge entgegennimmt, und ein Ausgangskanal, an dem der Client das Ergebnis entgegennimmt. Eine allgemeine Diensterbringung hat beispielsweise die Form in der *Abbildung 3.4*, wobei die Server wiederum Clients anderer Server sein können. Detaillierte Informationen dazu liefert [Bengel 2002].

[39] JNDI (*Java Naming and Directory Interface*) ist ein API, das von Sun den Entwicklern zur Verfügung gestellt wird und die Möglichkeit bietet, Naming- und Directory Service-Funktionalitäten in Java-Anwendungen zu nutzen. Mehr Information befinden sich auf der Sun Homepage [Sun 2003].

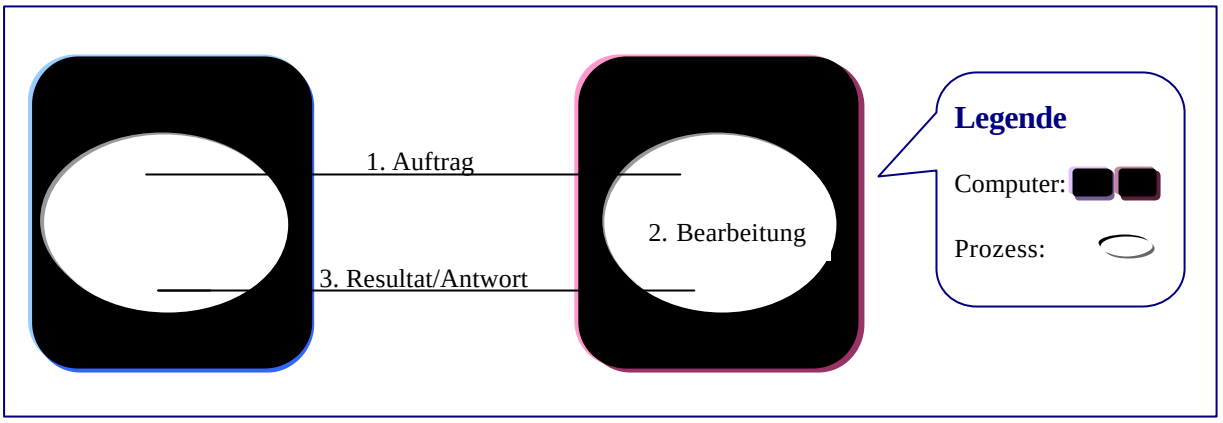
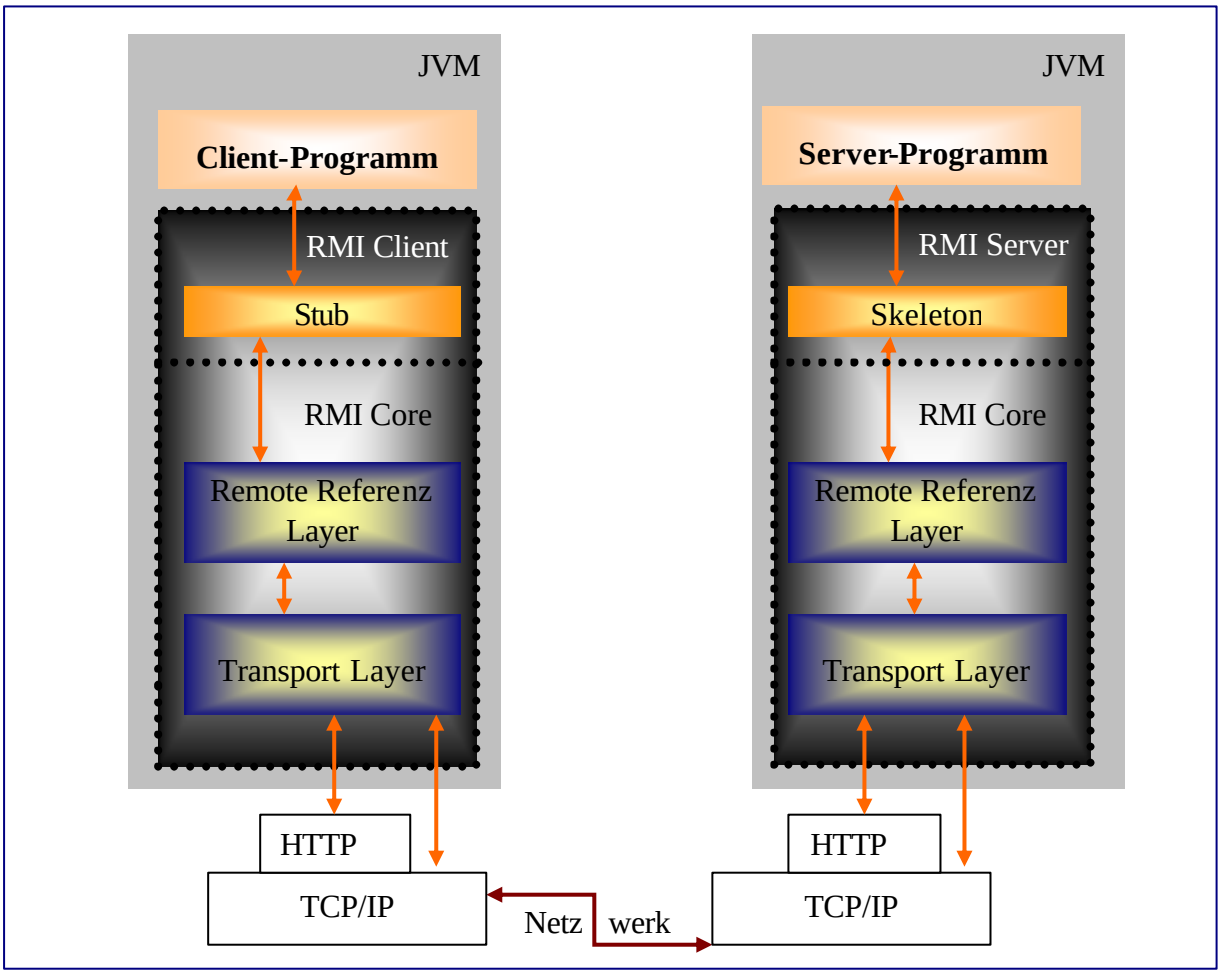


Abbildung 3.4: Interaktion im Client/Server-Modell

Die Möglichkeit zur Implementierung von Client und Server auf unterschiedlichen Systemen bringt einige Konsequenzen mit sich. Weitere Komponenten werden erforderlich, die die gegenseitige Verständigung der zusammenwirkenden Anwendungen bzw. Prozesse ermöglichen. Diese Aufgabe wird von der Middleware erledigt. Sie schafft eine einheitliche Schnittstelle zwischen den Clients und Servern in verteilten Systemen. RPC und ORB (vgl. Abschnitt 3.3 Middleware-Mechanismen) gehören u. a. zu diesen C/S-Technologien.

3.5.1 Kommunikation über RMI

Die Technik RMI (Methodenfernaufruf) ist die logische Übertragung des Prinzips des Prozedurfernaufrufs in der objektorientierten Programmierung. Im Gegensatz zum RPC, wo Prozeduren von einem Server für einen entfernten Aufruf zur Verfügung gestellt werden, werden beim Methodenfernaufruf Objektmethoden bereitgestellt [Haase 2001]. D. h., die RMI Technologie stellt eine Anpassung der allgemeinen RPC-Grundidee an verteilte Objekte dar. Deren allgemeine Architektur zeigt die folgende *Abbildung 3.5*. Mittels RMI kann ein Client transparent serverseitige, in einem Java-Interface definierte Methoden aufrufen. Dabei besteht die Möglichkeit eine Objektaktivierung zu nutzen. Das bedeutet, dass erst zum Zeitpunkt der Dienstnutzung das jeweilige Objekt aktiviert wird. Dadurch können Ressourcen gespart werden. Vertiefende Informationen zu RMI liefert [Bengel 2002] und das RMI-Tutorial auf der Sun-Homepage [Sun 2003].



Quelle: [Uni-Magdeburg 2002]

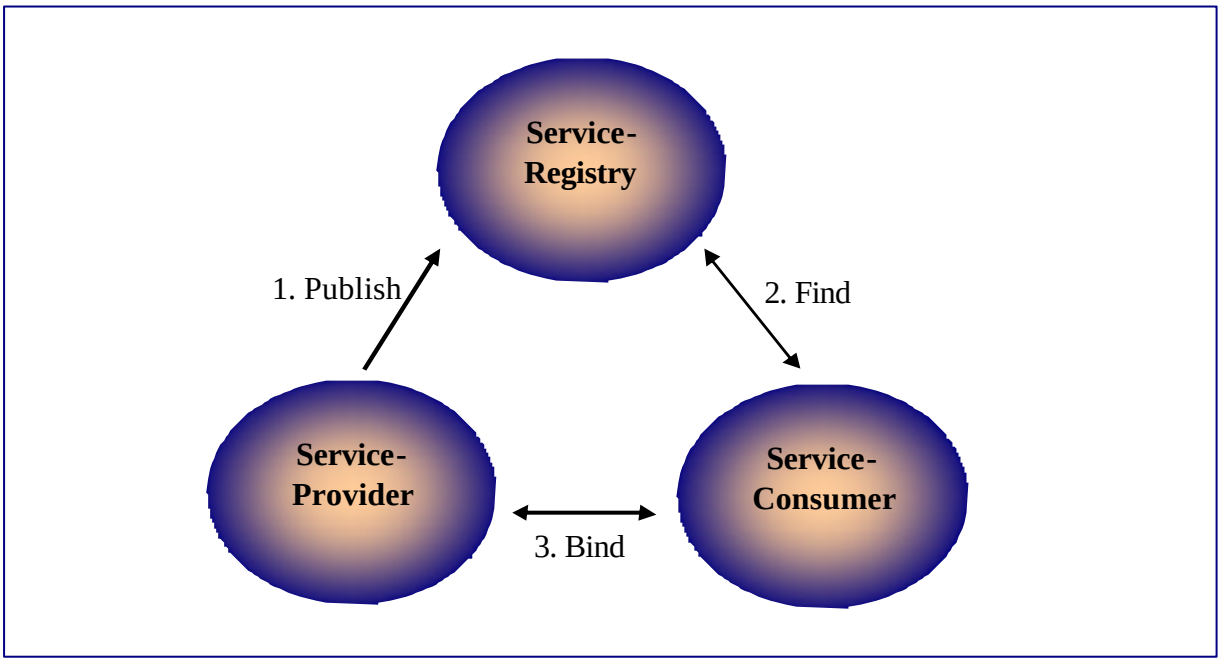
Abbildung 3.5: RMI-Funktionsweise

3.5.2 Web Service

Im Grunde folgen Web Services einem sehr einfachen Konzept: Funktionen auf Systemen werden durch das Senden von Anforderungen und Daten zwischen Computern unter Verwendung von XML und HTTP ausgeführt.
Ein Web Service bietet Schnittstellen an, um auf Funktionen einer Anwendung über Datenkommunikationsnetzwerke zuzugreifen. Dadurch ist diese Anwendung von einem System, das sie nutzen will, gekapselt und kann plattformunabhängig realisiert werden (vgl. SOAP-Abschnitt Kapitel 4).

Das Ergebnis einer solchen Strategie ist in *Abbildung 3.6* angedeutet. Aus Architektursicht werden Web Services i. d. R. als Dreigespann aus Provider, Consumer und Registry

dargestellt. Der Service-Provider stellt in die Service-Registry eine Beschreiburg seiner Dienste (Publish). Der Service-Consumer (Mensch, Programm) sucht in der Service-Registry nach einem Service (Find). Der Service-Consumer nutzt einen angebotenen Service des Service-Providers (Bind). Diese Sicht beschreibt jedoch nur sehr grob die Beziehungen zwischen den Beteiligten [Uni-Leipzig 2003]. Mehr Informationen sind [Web Services 2003] zu entnehmen.



Quelle: [Uni-Leipzig 2003]

Abbildung 3.6: Web Services-Architektur

Da die meisten Web Services und einige Infrastrukturen auf dem Standard XML bzw. XML-RPC und SOAP basieren, ist es von Vorteil, zunächst die Begriffe XML und XML-RPC kurz zu beschreiben.

❖ XML zusammengefasst

Die Extensible Markup Language (XML) ist ein Anwendungsprofil bzw. eine eingeschränkte Form von SGML[40], deren Ziel es ist, die Akzeptanz und Nutzbarkeit der SGML-Idee zu verbessern und seine Anwendbarkeit in der Form auf das Web zu übertragen, wie sie mit HTML möglich ist. XML wurde für eine einfache Verwendung und Interoperabilität sowohl

[40] SGML (*Standard Generalized Markup Language*) wurde von der ISO (*International Standardization Organization*) als Standard zur Beschreibung von Dokumenten festgelegt.

mit SGML als auch mit HTML entwickelt [XML2Day 2003]. Durch XML können Daten strukturiert und mit Bezeichnerelementen versehen werden. XML-Daten oder -Dokumente liegen in einer textuellen Form vor, welches die Verarbeitung auch durch den Menschen ermöglicht. XML wird vor allem für den Austausch von Daten und Dokumenten zwischen Plattformen und Applikationen eingesetzt [Weyer 2002]. In Anlehnung an [XML2Day 2003] und an [Weyer 2002] wird XML in den folgenden Punkten zusammengefasst:

1. XML ist keine Programmiersprache, sondern ein Satz von Regeln für die Erstellung von Textformaten zur Datenstrukturierung. XML erleichtert es einem Computer, Daten zu generieren oder zu lesen und sorgt dafür, dass eine bestimmte Datenstruktur eindeutig bleibt. XML ist erweiterbar und plattformunabhängig.

2. Wie HTML verwendet XML Tags und Attribute. Während HTML festlegt, was jedes Tag und Attribut bedeutet und auch wie der Text dazwischen in einem Browser aussehen sollte, benutzt XML die Tags nur zur Abgrenzung von Daten und überlässt die Interpretation der Daten allein der Anwendung, die sie verarbeitet.

3. Genau wie bei HTML bestehen XML-Dateien aus reinem Text, der zwar von Menschen nicht gelesen werden sollte, aber gelesen werden kann, wenn es notwendig ist. Anders als bei HTML, sind die Regeln bei XML strikt. Ein fehlendes Tag macht eine XML-Datei unbenutzbar.

4. Weil XML ein Textformat ist und Tags verwendet, um die Daten abzugrenzen, sind XML-Dateien fast immer größer als vergleichbare binäre Formate.

5. Die Entwicklung von XML begann 1996. In Wirklichkeit ist die Technologie nicht sehr neu. Vor XML gab es SGML, das in den frühen 80er Jahren entwickelt wurde. Die Entwickler von XML nahmen aufgrund der Erfahrung mit HTML einfach die besten Teile von SGML und produzierten etwas, was nicht weniger mächtig als SGML, aber einfacher in der Anwendung ist.

6. XML ist modular und damit erlaubt es, ein neues Dokumentenformat zu definieren, indem man andere Formate kombiniert oder wiederbenutzt. Wenn jedoch zwei Formate völlig unabhängig voneinander entwickelt worden sind, können sie Elemente und Attribute enthalten, die in beiden Formaten mit dem gleichen Namen vorkommen. Um bei der Kombination von Formaten Namenskollisionen zu vermeiden, stellt XML den Namensraummechanismus zur Verfügung.

7. XML ist lizenzfrei und gut unterstützt. Man hat Zugang zu einer großen und wachsenden Ansammlung von Werkzeugen und zu vielen versierten Fachleuten.

❖ XML-RPC

Im Rahmen der XML-RPC-Spezifikation dient XML zur Codierung der Übergabeparameter an die entfernten Prozeduren und RPC als Kommunikationsmechanismus, daher der Name XML-RPC (Siehe XML und RPC Abschnitte). XML-RPC ist sprach-, plattform- und herstellerunabhängig, er basiert auf dem C/S-Prinzip, ermöglicht einem Client eine auf dem Server befindliche Funktion aufzurufen und verbindet Software verschiedener Betriebssysteme und verschiedener Umgebungen über das Internet [XML-RPC 2003]. Der Transport bei XML-RPC erfolgt durch HTTP, also ein www-Client zu einem www-Server (*Abbildung 3.7*), wobei das Transport-Schema „request-response-Schema“ zur Anwendung kommt.

XML-RPC stellt eine Architektur für netzwerkbasierte Methodenaufrufe auf der Basis von XML zur Verfügung. Eine typische XML-RPC-Kommunikation hat den in *Abbildung 3.7* dargestellten Aufbau. Die Informationen werden in das XML-Datenformat umgewandelt und per HTTP zwischen den Rechnern übertragen. Wie das XML für XML-RPC aussieht, ist in der XML-RPC Spezifikation festgelegt.

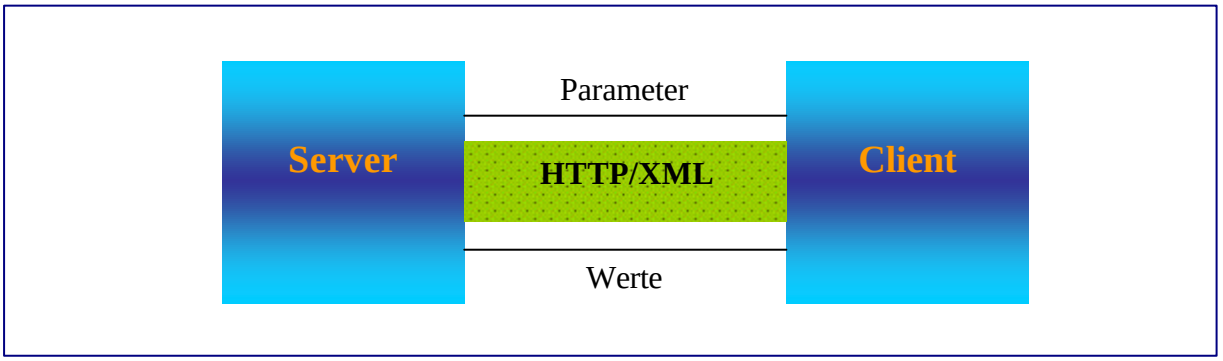


Abbildung 3.7: XML-RPC-Schema

❖ Simple Object Access Protocol (SOAP)

“SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework.” [Soap 2003].

Mit dem Ziel der programmiersprachen- und plattformunabhängigen Kommunikation zwischen verteilten Anwendungskomponenten entstand der von W3C[41] entwickelte Standard namens Simple Object Access Protocol (SOAP). SOAP ist ein selbstbeschreibendes, einfaches, XML-basiertes Nachrichtenformat, das zusammen mit jedem Betriebssystem, jeder Programmiersprache und jeder Plattform arbeitet. Dabei werden die Nachrichten mittels eines darunter liegenden Transportprotokolls übertragen [SOAP 2003].

SOAP findet in zweierlei Kommunikationsmodellen Verwendung. Die erste Variante ist die nachrichtenorientierte Kommunikation, die die Übertragung von beliebigen XML-Dokumenten ermöglicht. Darauf baut die zweite Variante auf, mit der sich ein entfernter Methodenaufruf (RPC) realisieren lässt. Hierfür wird festgelegt, dass eine XML-Nachricht bestimmten Formats die Anfrage darstellt, worauf eine Antwortnachricht folgt, die das Ergebnis oder eine Fehlermeldung enthält.

Für SOAP ist kein bestimmtes, darunter liegendes Transportprotokoll vorgeschrieben, sondern es kann auf verschiedenen Protokollen aufsetzen. Üblicherweise wird das HTTP verwendet, aber es können auch andere Protokolle zum Einsatz kommen [W3C 2003].

SOAP ist für Web Services die zentrale Middleware Technologie. Sie definiert einen Nachrichtenaustausch zwischen 2 Partnern im Internet:

- Ein Dienstanbieter (Server), auch Web Service Provider genannt, stellt eine beliebige Online-Dienstleistung via SOAP zur Verfügung. Er akzeptiert Nachrichten über das Standard Internet Protokoll HTTP, die in XML formatiert sind.
- Ein Dienstnehmer (Client), der den Service des Anbieters durch eine entsprechend formulierte SOAP Nachricht nutzen kann.

3.6 Peer to Peer Middleware

Peer to Peer (P2P) ist eine Alternative zum C/S Paradigma, wobei die Interaktion gleichberechtigter Partner ohne permanente Präsenz im Netz und ohne feste IP-Adresse geschieht, d. h. in einem P2P-Netzwerk ist jeder angeschlossene Computer Server und Client zugleich. Der User bestimmt selbst, welche Ressourcen (Software, Dateien) er dem P2P-Netz zur Verfügung stellt, d. h. P2P ermöglicht den direkten Austausch von Daten zwischen Computern, ohne Umweg über einen Server (*Abbildung 3.8*).

Im Lexikon von Siemens ist folgende allgemeine Erklärung für P2P zu finden: „*P2P-Netze gehören zu den PC-Netzen und bieten allen Rechnern im Netz eine gleichberechtigte Zusammenarbeit. Das bedeutet, dass jedes System im Netz anderen Systemen Funktionen und Dienstleistungen anbieten und andererseits von anderen Systemen angebotene Funktionen und Dienstleitungen nutzen kann. P2P-Networking ist für kleinere Arbeitsgruppen zwischen 5*

[41] *World Wide Web Consortium* (W3C)

und 20 Arbeitsplätzen praktikabel. In einer solchen Umgebung kann jeder Mitarbeiter seinen Partnern den Zugang zu Betriebsmitteln wie Plattenspeicher oder Drucker erlauben" [Siemens Lexikon 2002]. Weitere Informationen zu P2P sind bei der P2P Working Group [P2P 2003] zu finden.

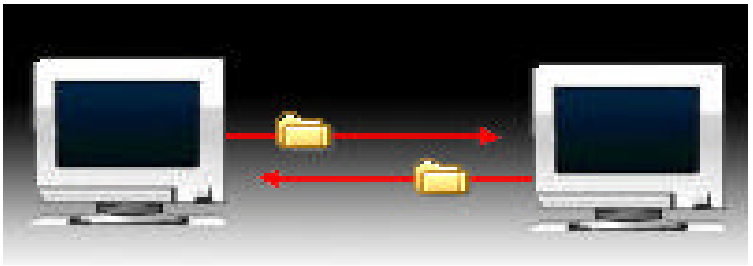


Abbildung 3.8: P2P-Netzwerk

3.7 Genereller Ablauf der Dienstvermittlung

Die heute verfügbaren Architekturen zur Dienstbeschreibung und Dienstvermittlung können trotz Unterscheidungen nach deren vorhandenen Ähnlichkeiten generalisiert werden. Dies wird in *Abbildung 3.9* verdeutlicht. Es gibt im Wesentlichen drei unterschiedliche Rollen: den Dienstanbieter, den Nutzer eines Dienstes sowie den Dienstvermittler. Dieser bildet die Anlaufstelle zum Anfragen, Anbieten und Suchen von Diensten. Die Kommunikation zwischen diesen Rollen basiert auf TCP/IP.

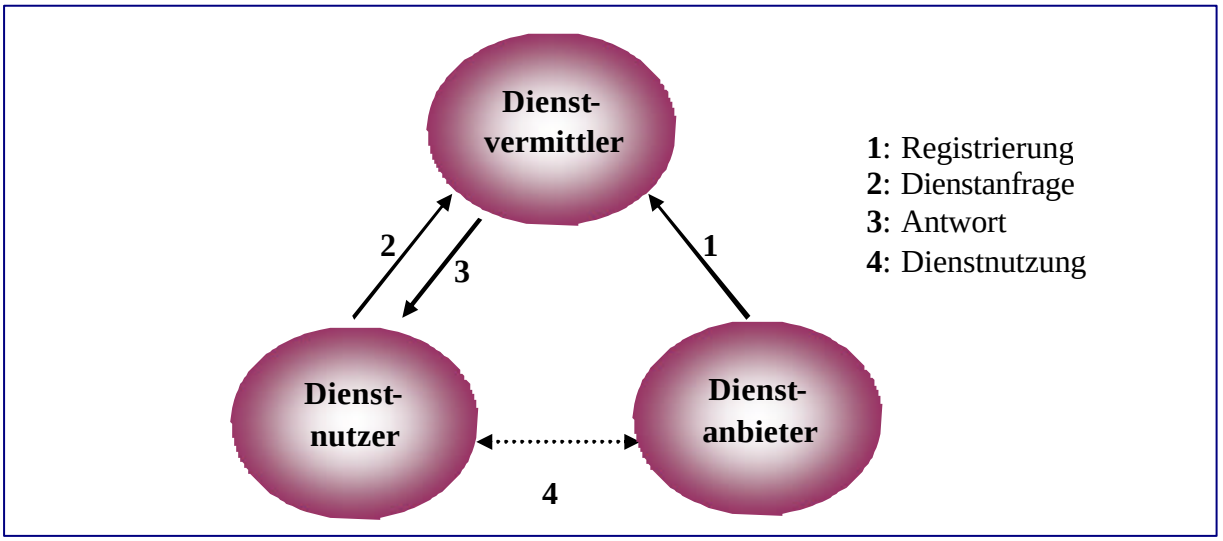


Abbildung 3.9: Genereller Ablauf der Dienstvermittlung

Bei denjenigen Architekturen (*Abbildung 3.9*), die einen Dienstvermittler zur Verfügung stellen, können die Dienstanbieter zunächst einmal ihre Dienste registrieren (1). Später kann ein anderes Gerät, der Dienstnutzer, der nach einem bestimmten Dienst sucht, den Dienstvermittler befragen (2). Der Dienstvermittler kann, falls ein geeigneter Dienst existiert, diesen ermitteln und das dem suchenden Dienstnutzer mitteilen (3). Dieser kann nun direkt den Dienstanbieter kontaktieren (4). In diesem Kontext ist Jini als Beispiel für die Dienstvermittlung in lokalen Netzen zu nennen (siehe Kapitel 4 Abschnitt 4.1 Java Intelligent Network Interface (Jini)).