



## Unter vier Augen

# Peer to Peer in Theorie und Praxis

Ludwig Mittermeier und Roy Oberhauser

## Teil 2: Ad-hoc-Web-Services durch P2P-Technologien

Das folgende konkrete Beispiel zeigt die Möglichkeiten auf, die sich aus der Kombination plattformunabhängiger Web-Services mit den Paradigmen des P2P-Computing ergeben können.

### Auffinden und verwenden von Web-Services in P2P-Umgebungen

Web-Services stellen eine betriebssystem-, programmiersprachen- und protokollunabhängige Möglichkeit für entfernte Methodenaufrufe (remote procedure calls – RPCs) zur Verfügung. Dabei wird nur die Schnittstelle im XML-basierten WSDL (Web Services Description Language)-Format zur Verfügung gestellt und die Implementierung bleibt gänzlich verborgen. Doch wie findet ein Web-Service-Client seine Kommunikationspartner? Hierzu könnte der Verzeichnisdienst UDDI (Universal Discovery, Description and Integration) verwendet werden. Ähnlich wie beim Yellow-Pages-System können Web-Services anhand ihrer Eigenschaften gefunden werden.

UDDI ist geeignet für Anwendungsfälle, in denen von einer hohen Verfügbarkeit der Web-Services und von wenigen Konfigurationsänderungen ausgegangen werden kann. Jedoch sollten beim Einsatz von UDDI in einem P2P-Netzwerk folgende Faktoren berücksichtigt werden:

- ▼ Administration von UDDI-Servern: Zeit- und Kostenaufwand für Wartung und Administration, beispielsweise Entfernen nicht-aktueller Einträge, zusätzlich evtl. Datenbankadministration,

- ▼ Administration von Peers: Konfiguration der Adressen von UDDI-Servern,
- ▼ Fehlerquellen: Quality-of-Service-Aspekte von UDDI-Servern und der entsprechenden Peer-Web-Services, Verfügbarkeit, Erreichbarkeit von Peers,
- ▼ Benachrichtigung von Peers über Änderungen in der UDDI-Registry.

Für P2P-Umgebungen ist also UDDI nicht optimal geeignet, um Web-Services ad hoc zu finden oder bekannt zu machen. In der weiter unten beschriebenen **Peer-ChainApplication** wird hierfür ein dezentralisiertes Verfahren verwendet.

Web-Services ermöglichen es Peers, in einer sprach- und plattformunabhängigen Art und Weise, Dienste anzubieten und die Dienste anderer Peers zu nutzen.

P2P-Computing bietet für Web-Services bessere Unterstützung für dynamische Umgebungen und Einsatzgebiete als typische Client-Server-basierte Web-Service-Topologien. Peers können jederzeit im Netz erscheinen und Web-Services anbieten und werden dynamisch von den bereits vorhandenen Peers ohne Administrations- oder Konfigurationsaufwand erkannt. (s. Abb. 1).

### Web-Services mit GLUE

Die in Texas ansässige Firma „The Mind Electric“ bietet mit dem Produkt „GLUE“ [GLUE] ein rein Java-basiertes Web-Services-Toolkit an. Die Personal Edition von GLUE ist kostenlos, die kostenpflichtige Professional Edition bietet zusätzlich EJB- und JMS-Integration sowie einen UDDI-Server.

Ein großer Vorteil von GLUE ist seine Einfachheit in der Nutzung. So kann jedes beliebige Java-Objekt mit wenigen einfachen Java-Anweisungen als Web-Service veröffentlicht werden, ohne dass sich die Entwicklerin mit Stubs, Skeletons oder WSDL-Generatoren herumschlagen müsste. Zudem kann es als Standalone-Lösung eingesetzt werden und benötigt keinen Applikations-Server. Für den Einsatz in Umgebungen mit wenig Ressourcen gibt es auch eine Embedded-Version, welche nur 350KByte groß ist.

Darüber hinaus offeriert GLUE vollkommen transparent eine nicht unerhebliche Optimierung: Falls sich Web-Service-Server und Client innerhalb derselben JVM befinden, sorgt GLUE für einen optimierten Ablauf, der je nach Recherausstattung laut GLUE-Dokumentation etwa 100 000 bis 300 000 Messages pro Sekunde bietet.

Die Standard Edition von GLUE liegt zum Zeitpunkt der Erstellung dieses Artikels in der Version 2.1 vor. Die Installation gestaltet sich denkbar einfach: Zip-Datei downloaden, entpacken, fertig.

Sind die \*.jar-Dateien, welche in der eben entpackten Verzeichnisstruktur unter „electric/lib“ zu finden sind, der Lieblings-IDE bekannt gemacht, so kann das Entwickeln von Web-Services mit GLUE auch schon losgehen.

Im Folgenden soll nun als erstes einfaches Beispiel ein Objekt der Klasse **Calculator.java** (s. Listing 1) als Web-Service veröffentlicht werden und dessen Methoden sollen von einem Web-Service-Client aufgerufen werden.

Wie in **Server.java** in Listing 2 zu sehen ist, sind zum Veröffentlichen eines Java-Objektes als Web-Service mit GLUE lediglich zwei Anweisungen notwendig, nämlich:

```
HTTP.startup( "http://localhost:8004/glue" );
Registry.publish( "urn:Calculator", new Calculator() );
```

**Server.java** startet einen Web-Server auf Port 8004 und wartet auf ankommende Nachrichten unter der URL /glue. Mit **Registry.publish()** wird das **Calculator-Ob-**

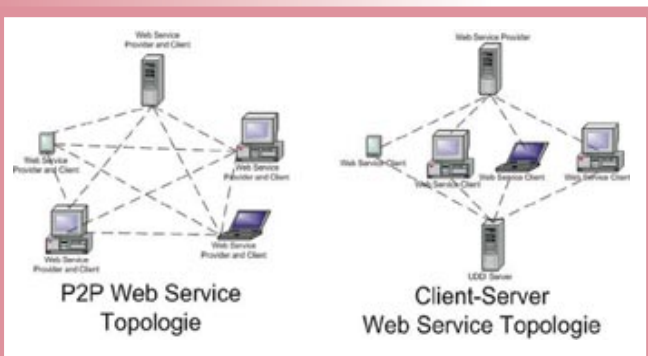


Abb. 1: P2P- und Client-Server-Web-Service-Topologien

jekt unter dem Pfad „Calculate“ veröffentlicht. Das Programm beendet seine Ausführung nicht nach der letzten Anweisung, da im Hintergrund Threads des gestarteten Web-Servers laufen. Die Ausgabe des Servers sieht folgendermaßen aus:

```
GLUE 2.1 (c) 2001 The Mind Electric
tartup HTTP server on http://127.0.0.1:8004/glue
```

Unter der URL <http://localhost:8004/glue/urn:Calculator.wsd> kann die automatisch generierte und veröffentlichte Beschreibung des Web-Service im WSDL-Format begutachtet werden.

Möchte man nun beispielsweise nicht alle Methoden der Klasse `Calculator` als Web-Service veröffentlichen, so kann man der Methode `Registry.publish` anstelle eines Objektes der Klasse `Calculator` ein Interface übergeben, welches von `Calculator` implementiert wird und nur diejenigen Methoden anbietet, welche als Web-Service veröffentlicht werden sollen.

```
public class Calculator {
    public float doubler(float value) {
        return 2*value;
    }
}
```

Listing 1: Calculator.java

```
public class Server {
    public static void main( String[] args )
    throws Exception {
        // start a web server on port 8004,
        // accept messages via /glue
        HTTP.startup( "http://localhost:8004/glue" );
        Registry.publish( "urn:Calculator", new Calculator() );
    }
}
```

Listing 2: Server.java

Mit wenigen einfachen Java-Anweisungen kann also in einem eigenen Java-Programm mit GLUE ein Web-Service angeboten werden. Wer jedoch keinen eigenen Java-Code schreiben will und eine Java-Klasse hat, von der ein Objekt als Web-Service veröffentlicht werden soll, der kann auch ein mit GLUE mitgeliefertes Kommandozeilenwerkzeug verwenden. Nähere Informationen hierzu finden sich unter [GLUE].

```
public class Client {
    public static void main( String[] args )
    throws Throwable {
        String url =
            "http://localhost:8004/glue/urn:Calculator.wsd";
        Float value = (Float)Registry.invoke(url, "doubler",
            new Float[]{ new Float(2.95) });
        System.out.println(value);
    }
}
```

Listing 3: Client.java

Als nächstes wird nun gezeigt, wie mit GLUE der Aufruf eines Web-Service erfolgt. GLUE ermöglicht den Aufruf beliebiger Web-Services beinahe genauso einfach wie den Aufruf von Methoden lokaler Java-Objekte. Die Generierung von Proxies geschieht dabei automatisch und transparent zur Laufzeit.

`Client.java` (s. Listing 3) zeigt, wie der zuvor veröffentlichte Calculate-Web-Service genutzt werden kann:

```
Float value = (Float)Registry.invoke(url, "doubler",
    new Float[]{ new Float(2.95) });
```

Die Klasse `Calculator` zeigt, wie ein `float` als Aufrufparameter verwendet wird. Die Menge der Parameter beschränkt sich jedoch nicht auf diese einfachen Datentypen, sondern es kann jedes beliebige Java-Objekt als Parameter übergeben werden. GLUE serialisiert dabei solche Objekte vollkommen transparent in einen XML-Complex-Typ. Dabei ist es nicht notwendig, aber möglich, Mapping-Informationen zur Verfügung zu stellen. Allerdings kann man hierbei leicht auf Interoperabilitätsprobleme mit anderen Web-Service-Implementierungen stoßen, da das Mapping von komplexen Datentypen nicht standardisiert ist.

## JXTA und GLUE: Die PeerChainApplication als Beispiel

Das Zusammenspiel von P2P und Web-Services wird an einem konkreten Beispiel, nämlich der auf JXTA und GLUE basierenden `PeerChainApplication`, vorgestellt.

Folgende Schritte zeigen den möglichen Ablauf einer Applikation, die JXTA und GLUE kombiniert:

1. Veröffentlichung eines Java-Objektes als Web-Service mit GLUE, dabei wird eine WSDL-URL generiert.
2. Bekanntgabe des Web-Services an andere Peers: Start von JXTA und Generierung eines `PeerAdvertisements`, welches als Beschreibung die in Schritt (1) erhaltene WSDL-URL enthält. Start eines separaten Threads, der fortlaufend in einem bestimmten Zeit-Intervall das generierte `PeerAdvertisement` veröffentlicht.
3. Registrieren eines Listeners auf ankommende Discovery-Ereignisse. Dieser Listener extrahiert aus einem ankommenden Discovery-Event ein darin enthaltenes `PeerAdvertisements` und überprüft, ob dieses eine URL zu einer WSDL enthält. Falls dies der Fall ist, kann ein Web-Service-Aufruf erfolgen.

Die `PeerChainApplication` (s. Listing 4) ist ein einfaches Beispiel dafür, wie Peers zusammenarbeiten und gegenseitige Web-Service-Aufrufe nutzen können. Der Ablauf der Applikation ist folgendermaßen: Jeder Peer lernt im Laufe der Zeit, welche Nachbarpeers er hat und stößt von Zeit zu Zeit ein so genanntes „PeerChaining“ an.

Dies bedeutet, er sucht sich per Zufall einen seiner bekannten Peers aus und ruft dessen Web-Service-Methode `chainPeers` auf (s. Listing 5 und 6). Parameter dieser Methode ist eine Liste derjenigen Peers, die schon an dem aktuellen PeerChaining-Vorgang teilgenommen haben. Derjenige Peer, dessen `chainPeers`-Web-Service-Methode aufgerufen worden ist, fügt sich selbst an diese Liste an und sucht nun seinerseits in seiner Liste der bekannten Peers, ob sich darin ein Peer befindet, der noch nicht an diesem PeerChaining-Vorgang teilgenommen hat, wählt einen dieser Peers per Zufall aus und ruft in einem neuen Thread dessen `chainPeers`-Methode als Web-



```

public class PeerChainApplication {
    private Set foundPeers =
        Collections.synchronizedSet(new HashSet());
    ...
    public void discoveredPeer(String wsdlURL) {
        if (foundPeers.add(wsdlURL))
            System.out.println("discovered new peer=" + wsdlURL);
    }
    private void startPeerChaining() throws RegistryException
    {
        PeerChainService peerChain =
            (PeerChainService)Registry.bind(myWSDLURL,
                PeerChainService.class);
        peerChain.chainPeers(new ArrayList());
        // start chain with empty list
    }
    private DiscoveryService startJxta()
    throws PeerGroupException {
        return PeerGroupFactory.newNetPeerGroup()
            .getDiscoveryService();
    }
    private PeerAdvertisement
    createAdvertisement(String wsdlURL) {
        PeerAdvertisement adv =
            (PeerAdvertisement)AdvertisementFactory
                .newAdvertisement(
                    PeerAdvertisement.getAdvertisementType());
        adv.setName("Java Spektrum P2P Example");
        adv.setDescription(wsdlURL);
        // description is web service URL
        return adv;
    }
    public static void main(String args[]) throws Exception {
        int port = 8004; // default port
        ...
        HTTP.startup("http://localhost:" + port + "/P2P");
        // start GLUE HTTP server
        PeerChainApplication app = new PeerChainApplication();
        // publish peer chain web service on GLUE HTTP server
        app.myWSDLURL =
            PeerChainServiceImpl.publish(app, port) + ".wsdl";
        // publish JXTA advertisement that contains
        // peer chain web service url
        DiscoveryService discoSvc = app.startJxta();
        discoSvc.addDiscoveryListener(new Listener(app));
        PeerAdvertisement adv =
            app.createAdvertisement(app.getWSDLURL());
        new Publisher(adv, discoSvc).start();
        while (true) { // start peer chaining periodically
            Thread.sleep(app.randomInt(MAX_SLEEP) * 1000);
            if (!app.foundPeers.isEmpty()) app.startPeerChaining();
        }
    }
}

```

Listing 4: PeerChainApplication.java

```

public interface PeerChainService {
    void chainPeers(List peers);
    // find other peers not yet on the list
    void chainCompleted(List peers);
    // follow peer list backwards
}

```

Listing 5: PeerChainService.java

Service auf. Dieser Aufruf erfolgt in einem neuen Thread, damit der aufrufende Peer nicht warten muss, bis die Kette der `chainPeers`-Aufrufe fertig ist.

Findet ein Peer in seiner Liste der bekannten Peers keinen, der noch nicht an diesem PeerChaining-Vorgang teilgenommen hat, so ist er der letzte Peer in der Kette und er ruft in einem neuen Thread die Methode `chainCompleted` desjenigen Peers auf, der sich in der Liste vor ihm befindet. Diese Aufrufkette geht nun ihrerseits weiter, bis sie an dem ersten Peer in der PeerChain angelangt ist. Dieser Peer erkennt dies daran, dass er sich an Position 0 in der PeerChain befindet und gibt die komplette PeerChain aus. (s. Abb. 2)

Dabei laufen in der Regel mehrere PeerChaining-Vorgänge gleichzeitig ab, weil jeder Peer unabhängig von den anderen nach einer zufälligen Wartezeit einen Peer-Chaining-Vorgang startet.

Dieses Szenario zeigt sehr deutlich die Gleichberechtigung aller Peers, was typisch für eine P2P-Applikation ist: Jeder Peer bietet Web-Services an und ruft im gleichen Maße die Web-Services anderer Peers auf. Es können jederzeit Peers zum Netzwerk hinzukommen oder offline gehen. Über neu hinzugekommene Peers werden die vorhandenen Peers über den Discovery-Mechanismus informiert und falls versucht wird, einen Web-Service eines Peers aufzurufen, der nicht mehr online ist, so entfernt der aufrufende Peer diesen aus seiner Menge der bekannten Peers.

## Hinweise zum Start der PeerChainApplication

JXTA legt eine Konfigurationsdatei im aktuellen Verzeichnis an, in der unter anderem Ports festgelegt werden. Sollen nun mehrere JXTA-Peers auf einem Rechner laufen, so müssen diese verschiedene Ports verwenden, was in unterschiedlichen Konfigurationsdateien beschrieben sein muss. Aus diesem Grund müssen, falls auf einem Rechner mehrere Instanzen der `PeerChainApplication` laufen sollen, diese aus verschiedenen Verzeichnissen heraus gestartet werden.

## Fazit

Die beschriebene Möglichkeit, JXTA mit Web-Services zu verbinden, stellt eine sehr komfortable Möglichkeit dar, schon heute plattform- und betriebssystemunabhängige RPCs zwischen Peers zu realisieren. Während das Web-Service-Toolkit GLUE keinerlei Schwierigkeiten bereitet, ist die Java-Implementierung der plattformunabhängigen P2P-Umgebung JXTA jedoch noch nicht so ausgereift, wie man sich das wünschen würde. Außerdem verwendet JXTA in IP-Subnetzen als Discovery-Mechanismus IP-Multicasting. Dieses Verfahren skaliert allerdings möglicherweise schlecht mit der Größe des Subnetzes.

Unter [JXTAPr] finden sich mehrere Projekte, die sich mit RPCs und Web-Services in Verbindung mit JXTA befassen. Diese Projekte befinden sich allerdings noch in einem sehr frühen Stadium.

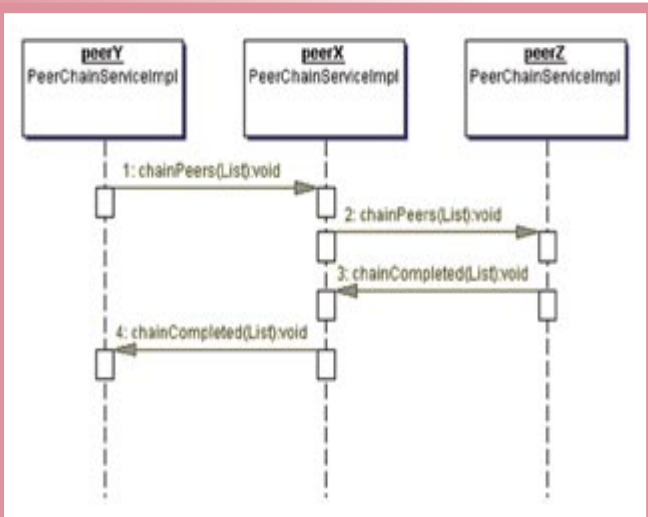


Abb. 2: Sequenz-Diagramm der PeerChainApplication

## ▼ Titelthema

Die Kombination aus P2P-Technologien und Web-Services könnte insbesondere im Mobile-Bereich neue Anwendungsmöglichkeiten eröffnen. Während bisher mobile Endgeräte lediglich als Clients Verwendung fanden, könnten P2P-Technologien in Verbindung mit Web-Services in diesem Bereich einen Paradigmenwechsel herbeiführen. Mobile Endgeräte können eine bessere Integration in bestehende IT-Infrastrukturen finden und ihre Ressourcen und Dienste zur Verfügung stellen. Die Besitzer eines mobilen Endgerätes können von neuen Einsatzmöglichkeiten und neuen Arten von Applikationen profitieren.

## Literatur und Links

[GLUE] das Web-Service-Toolkit GLUE,

<http://www.themindelectric.com>

[JXTA] JXTA,

<http://www.jxta.org>

[HJXTA] Hello JXTA!,

<http://www.onjava.com/lpt/a/onjava/2001/04/25/jxta.html>

[GJXTA] Get connected with Jxta,

[http://www.javaworld.com/javaworld/jw-06-2001/j1-01-jxta\\_p.html](http://www.javaworld.com/javaworld/jw-06-2001/j1-01-jxta_p.html)

[JXTAP2P] The Jxta solution to P2P,

[http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta\\_p.html](http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta_p.html)

[JXTAShell] Master the Jxta shell, Part 1,

[http://www.javaworld.com/javaworld/jw-01-2002/jw-0111-jxtashell\\_p.html](http://www.javaworld.com/javaworld/jw-01-2002/jw-0111-jxtashell_p.html)

[JXTAPr] JXTA-Sub-Projekte,

<http://www.jxta.org/servlets/DomainProjects>

[Ora01] A. Oram, Hrsg., Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology, O'Reilly, Sebastopol, 2001

[P2P] [www.openp2p.com](http://www.openp2p.com)

[Qua02] T. Quas, M. Völter, P2P und Komponenten: Eigenschaften, Unterschiede, Ergänzungsmöglichkeiten, in: ObjektSpektrum, 1/2002



**Ludwig Mittermeier** arbeitet als Engineer bei der Zentralabteilung „Corporate Technology“ der Siemens AG. Er konzentriert sich auf Software-Architekturen im Java-Umfeld. E-Mail: [ludwig.mittermeier@mchp.siemens.de](mailto:ludwig.mittermeier@mchp.siemens.de).

**Roy Oberhauser** ist als Senior Engineer bei der Zentralabteilung „Corporate Technology“ der Siemens AG tätig. Seine Schwerpunkte sind Java-basierte Software-Architekturen und Web-Services. E-Mail: [roy.oberhauser@mchp.siemens.de](mailto:roy.oberhauser@mchp.siemens.de).

## ▼ Weiterführende Informationsquellen

<http://www.openp2p.com>

<http://www.jxta.org>

```
public class PeerChainServiceImpl
implements PeerChainService {
    private PeerChainApplication app; // reference to application

    public static String publish(PeerChainApplication app,
                                int port)
    throws RegistryException {
        PeerChainServiceImpl svc = new PeerChainServiceImpl(app);
        Registry.publish(app, PEER_CHAIN_SVC_URN, svc);
        // publish web service
        return Registry.getPath(svc); // return web service URL
    }

    public void chainPeers(List peers) {
        new Thread(new ChainPeers(peers)).start();
    }

    public void chainCompleted(List peers) {
        new Thread(new ChainCompleted(peers)).start();
    }

    class ChainPeers implements Runnable {
        ...
        public void run() {
            // find an unchained peer, else callback chainCompleted
            peers.add(app.getWSDLUrl());
            // add ourselves to the chain list
            Set known = app.getKnownPeers();
            List unchained = new ArrayList(known);
            // create copy of known peers first
            unchained.removeAll(peers);
            // get set of known that are unchained
            while (true) {
                // attempt to reach one other peer
                // until no other or success
                if (unchained.isEmpty()) { // if no peers left to call
                    chainCompleted(peers);
                    // start chain completed callback
                    break; // finished
                } else { // pick a random peer url
                    String url = (String)unchained.get(
                        app.randomInt(unchained.size()));
                    PeerChainService svc =
                        (PeerChainService)Registry.bind(url,
                            PeerChainService.class); // bind to web service
                    svc.chainPeers(peers);
                    // invoke chainPeers web service on peer
                    break; // finished
                }
            }
        }
    }

    class ChainCompleted implements Runnable {
        ...
        public void run() {
            // inform peer up the chain that the chain is complete
            int mypos = peers.indexOf(app.getWSDLUrl());
            // find this peer's position
            for (int i = mypos; i >= 0; i--) {
                // attempt call up the chain
                if (i > 0) { // this is not the last peer in the chain
                    String url = (String)peers.get(i - 1);
                    // get peer up the chain
                    PeerChainService svc =
                        (PeerChainService)Registry.bind(url,
                            PeerChainService.class); // bind to web service
                    svc.chainCompleted(peers);
                    // invoke chainCompleted on peer
                    break; // finished
                } else if (i == 0) {
                    // we are the last peer to take the list
                    System.out.println(
                        "in : peer chain completed at position " + mypos);
                }
            }
        }
    }
}
```

Listing 6: PeerChainServiceImpl.java

<http://www.javaspektrum.de>