Gustavo de Souza
Khaue Rezende Rodrigues

Uma proposta P2P para Publicação e Descoberta de Web Services

Florianópolis, março de 2003.

Gustavo de Souza
Khaue Rezende Rodrigues

Uma proposta P2P para Publicação e Descoberta de Web Services

Trabalho de conclusão de curso
do curso de Bacharelado em Ciências da Computação

Florianópolis, março de 2003.

Resumo

Os Web Services fazem parte de um grupo de aplicações em plena ascensão na Internet. Estes serviços têm se tornado foco de grandes esforços de desenvolvimento no sentido de ampliar sua utilização, como a interligação entre filiais de uma mesma empresa ou entre uma empresa e seus fornecedores. A proposta deste trabalho consiste em realizar a publicação e descoberta de Web Services através de uma rede P2P, representando assim uma alternativa à proposta atual mais aceita para "Registro de Web Services" – a UDDI, que se baseia na idéia de um repositório central. Tentamos, desta forma, propor a descentralização de todo o processo, tornando os documentos de especificação de Web Services disponíveis em qualquer repositório que venha a se tornar disponível publicamente na rede, passando o controle à toda comunidade de desenvolvedores, que fica livre para buscar os melhores caminhos para esta tecnologia, baseando-se unicamente nos fatores relevantes à mesma, tais como eficiência, reusabilidade e otimização dos processos envolvidos. A idéia principal deste trabalho é democratizar o acesso aos Web Services e à sua publicação, simplificando o processo e, por conseguinte, reduzindo custos, uma vez que distribui os gastos de manutenção entre os peers formadores da rede. Assim, qualquer sistema capaz de se conectar à rede P2P de "Registro de Web Services" se torna um repositório de serviços em potencial, capaz de realizar a localização e publicação de documentos WSDL e, conseqüentemente, ter acesso a todas as informações necessárias à operação de vínculo a Web Services.

Abstract

Web Services are a part of an application group in great ascension in the Internet. These services became the focus of great efforts of development in the direction to extend its use, as interconnection between branch offices of one same company or a company and its suppliers. The proposal of this work consists of the publication and discovery of Web Services through a P2P network, thus representing an alternative to the current proposal accepted for "Web Services Registry" – UDDI, which it bases in a central repository. So, we try to purpose the process decentralization, becoming available Web Services specification documents in any repository that comes to become available public in the network, but not involving a central repository under the control of the great corporations, passing the control to developers community, that is free to search the best ways for this technology, solely based on the relevant factors such as efficiency, reusability and optimization in the involved processes. The most important idea of this work is to democratize the access to the Web Services and its publication, simplifying the process and, therefore, reducing costs, once upon that distributes the expenses of maintenance between peers on the network. Thus, becoming any system connected to P2P network a repository of services in potential, able to locate and publish WSDL documents and, consequently, to have access to all necessary information to link the Web Services.

Sumário

1. Introdução

Conforme veremos neste trabalho várias tecnologias discutidas aqui já existiam a muito tempo com outras simbologias, ou apresentadas sobre outras óticas mas sempre há pontos em comum entre as tecnologias utilizadas no passado para desenvolvimento de sistemas distribuídos e as emergentes atualmente, oque fez a diferença entre estas novas tecnologias tem sido a forma como elas se apresentam, buscando a simplicidade, utilização de tecnologias já existentes, e largamente adotadas mundialmente (HTTP, XML, SOAP), sem haver a "reinvenção da roda" mas adotando tecnologias robustas e independentes de plataforma.

Com certeza dentre as tecnologias discutidas neste trabalho, a que iniciou esta revolução no modelo de desenvolvimento de sistemas distribuídos que veio a culminar nos Web Services, foi a XML. A XML surgiu como um padrão para formatação de dados, e se tornou a base da comunicação dos Web Services e de praticamente todo tipo de comunicação moderna seja ela Internet ou não. Seguiram-se a ela a criação de formatos de arquivos e de documentos baseados na XML, assim como de esquemas de formatação e validação de seu conteúdo. Estas características foram absorvidas e incorporadas pelos criadores dos Web Services. São utilizadas para a distribuição de seus serviços e busca de novos serviços.

Este trabalho tem como objetivo apresentar uma proposta de software peer-to-peer para a publicação, a descoberta e a disseminação da tecnologia dos Web Services e a implementação de um sistema que demonstre a viabilidade desta proposta. Nossa principal motivação para este trabalho é oferecer uma alternativa à proposta centralizadora de um repositório central cujas informações sobre os Web Services estariam disponíveis - que é o caso de UDDI.

Da mesma forma que buscamos por uma alternativa ao repositório central, também foi necessário analisar um novo modelo de arquitetura de rede que fosse mais próprio para a nossa proposta. Devido às características desejáveis das redes peer-to-peer, como a auto-organização, a independência de mecanismo de transporte de rede, a capacidade para replicar espontaneamente informações para usuários e o fomento na construção de serviços e aplicações interoperáveis, nos direcionamos para uma plataforma reconhecida por sua estrutura simples e ao mesmo tempo completa em termos de expansibilidade e versatilidade que é o projeto JXTA. A plataforma do Projeto JXTA provê um ambiente descentralizado que minimiza falhas em pontos isolados e é independente de qualquer serviço centralizado.

Nos próximos capítulos apresentaremos as tecnologias estudadas por nós para o desenvolvimento do nosso sistema, bem como consideraremos suas vantagens e desvantagens. Apresentaremos em seguida os passos do desenvolvimento da nossa proposta, encerrando este trabalho com as nossas conclusões finais e com a indicação de possíveis trabalhos futuros baseados neste tema.

2. Tecnologias estudadas

Neste capítulo apresentaremos as tecnologias utilizadas em nosso projeto, discutindo separadamente cada tecnologia e suas possíveis aplicações nas mais diversas áreas, suas principais características, vantagens, sua importância como padrões de desenvolvimento emergentes e como estas tecnologias tendem a influenciar o futuro do desenvolvimento de sistemas distribuídos em especial.

Iniciamos este capítulo analisando a meta-linguagem XML, apresentado-a como a mais importante de todas as novas tecnologias e responsável pelo principal ponto da interoperação de sistemas heterogêneos, o formato da mensagem (comunicação).

O capítulo seguinte trata os Esquemas, em especial os Esquemas XML e duas de suas principais linguagens de definição a DTD e a XML Schema. Um esquema define os elementos que podem aparecer em um documento e os atributos que podem ser associados a este elemento, é uma descrição da estrutura do banco de dados ou de outras fontes de dados. Ele define dados da estrutura de um documento tais como, estrutura hierárquica, a seqüência dos elementos (ordenação), o número de nós-filhos que um elemento pode ou deve ter além de definir valores default para atributos.

Vemos também os Web Services, seu papel na Internet e no futuro dos sistemas distribuídos, suas estruturas, principais vantagens e as tecnologias das quais os Web Services tiram proveito para apresentar toda sua funcionalidade, a WSDL que têm como e função a padronização da descrição de Web Services e a UDDI, iniciativa de um consórcio de empresas multinacionais para a criação de um padrão de descrição, publicação e localização de Web Services.

Terminamos estudando JXTA, um projeto de software aberto cuja arquitetura utilizada é a P2P. Esta tecnologia serve para o desenvolvimento de sistemas sobre um conjunto de protocolos que embasam o funcionamento dos peers na rede.

2.1. XML

A XML (*Extensible Markup Language*) é uma meta-linguagem de marcação de dados que provê um formato para descrever dados estruturados.

Meta-linguagem é toda linguagem feita para moldar novas linguagens a partir de si própria e a XML é um exemplo disso: a partir de XML foram definidas várias linguagens com XSL, XLink, XPointer, XUL, RDF, entre outras. Todas estas são muito utilizadas atualmente, em funções das mais diversificadas como processamento de dados estruturados, desenvolvimento de interfaces gráficas e definição de formato para a distribuição de notícias pela Internet. Esta diversidade de segmentos que empregam XML como ponto de partida demonstra sua grande capacidade de geração de novas linguagens.

Uma marcação em um documento de dados estruturados é tudo aquilo presente no texto que não lhe acrescenta informação útil. As marcações referiam-se, originalmente, às anotações feitas a mão pelos autores e desenhistas e que seriam adicionadas ao texto escrito posteriormente. Essas anotações eram instruções ao digitador de como o texto deveria ser diagramado e que tipo de letra ele deveria usar. Esse tipo de marcação é conhecido como *marcação de procedimentos*. Um exemplo deste tipo de marcação seria a forma como o HTML apresenta seus elementos. Exemplificando: para que um determinado texto seja apresentado em negrito faz-se necessário colocar o texto entre as *tags* e ; caso desejasse apresentar em itálico, o programador deveria acrescentar as *tags* e , e assim seguindo para cada elemento a ser representado.

O sistema de marcações oposto ao de procedimentos é o de marcações descritivas. Num sistema de marcação descritiva o autor não se preocupa com a aparência do texto, mas sim com as entidades que eles representam. Enquanto num sistema de procedimentos definiríamos o tamanho da letra, o tipo de fonte etc., num sistema descritivo dizemos que aquela porção do texto representa o título do documento, outra porção de texto representa o nome de um capítulo e assim por diante.

A XML utiliza um sistema de marcações descritivas, o que torna possível se fazer declarações mais precisas do conteúdo e resultados mais significativos de busca de informações.

XML é uma meta-linguagem de marcação de dados, ou seja, uma linguagem para se definir linguagens de *markups* ou *tags*. Portanto, XML permite a definição de um número infinito de *tags*. No HTML as *tags* podem ser usadas para definir a formatação de caracteres e parágrafos; já a XML provê um sistema para criar *tags* para dados estruturados, cujo significado é atribuído posteriormente.

Um elemento XML pode ter dados declarados como sendo dados de um paciente, cotações de moedas, endereço, telefone, um título de livro, ou seja, qualquer tipo de elemento de dado. Como as *tags* XML são adotadas por grupos restritos como *intranets* de organizações, e também via Internet, haverá uma correspondente habilidade em manipular e procurar por dados independentemente das aplicações onde os quais são encontrados.

Uma vez que o dado foi encontrado, ele pode ser distribuído pela rede e apresentado em um navegador, ou então esse dado pode ser transferido para outras aplicações para processamento futuro e visualização de subconjuntos de elementos.

Abaixo segue um exemplo do uso de XML usado para valores de retorno e envio de resultados a máquinas interfaceadas em sistemas de automação laboratorial.

Fig. 2.1.1

```
<?xml version="1.0" ?>
<Principal>
<Paciente>
  <Amostra>0140497003</Amostra>
<Exames>
  <Exame>TXM</Exame>
<Resultado>
  <Quantitativo>.05</Quantitativo>
  <Qualitativo>Negativo</Qualitativo>
  <Complemento />
  <Liberado>1</Liberado>
  <Conferido>2</Conferido>
  <Editado>3</Editado>
  </Resultado>
  </Exames>
  <Usuario>administrador</Usuario>
  <Obs1>4</Obs1>
  <Obs2>5</Obs2>
  <Obs3>6</Obs3>
  <Obs4>7</Obs4>
  <Obs5>8</Obs5>
  <Obs6>9</Obs6>
  </Paciente>
 </Principal>
```

Este exemplo apresenta um arquivo XML contendo um resultado de exame médico que, neste caso, foi apresentado de forma integral, mostrando inclusive as *tags* que qualificam os dados. Esta é apenas uma das formas de apresentação dos dados, uma vez que a XML trouxe consigo uma nova geração de aplicações de manipulação e visualização de dados via Internet. Muitas destas aplicações têm suas características aprovadas e formalizadas por entidades de padronização da Internet, como é o caso do W3C, o que aumenta a confiabilidade nas aplicações desenvolvidas com estas tecnologias.

Neste capítulo apresentaremos uma breve descrição da XML, sua definição estrutural, características, formas de representação da informação, benefícios do uso da XML e comparações entre HTML e XML

2.1.1 Definição estrutural de um documento XML

Um documento XML é uma árvore rotulada onde um nó externo consiste de:

- **Dados descritos numa seqüência de texto**: Isto não impede que sejam inseridos dados em outros formatos, como o binário ou hexadecimal. Bastaria apenas que fosse feita a conversão do formato original para uma seqüência de caracteres. A conversão e a leitura desses dados ficam a cargo da aplicação que manipula o documento XML, visto que a função da XML é apenas apresentar os dados de forma organizada e estruturada;

- **Instruções de processamento**: são anotações para os programas que manipularão o documento XML. Apresentam definições como versão do padrão XML usado (atualmente existe apenas a versão 1.0), codificação dos caracteres, nome do documento DTD que atribui significado as *tags*. Tipicamente estão presentes no cabeçalho do documento;

- **Comentários sem semântica**, que acompanham o documento;

Um nó interno é um elemento, o qual é rotulado com:

- **Um nome**, para o qual distinguem-se caracteres maiúsculos e minúsculos;

- **Um conjunto de atributos**, cada qual consistindo de um nome e um valor que qualificam o elemento. Os atributos não são obrigatórios, e seu uso deve ser criterioso.

Normalmente, comentários, declarações de entidades e informações DTD não são explicitamente representadas na árvore.

2.1.2. Uma visão prática das tags

Um documento XML é um texto com tags de marcação (*markup tags*) e outras informações. As *markup tags* denotam a seguinte estrutura:

```
...<bla atrib="valorAtrib" ...>...</bla>...
   |    |                    |    |
   |    |                    |    uma tag finalizadora de elemento
   |    |                    o contexto do elemento
   |    um atributo com nome atrib e valor valorAtrib, com valores limitados por 'ou'
   uma tag inicializadora de elemento com nome bla
```

A notação para elementos vazios é a seguinte: <bla atrib= "valorDoAtrib" ...>...</>

Um documento XML é bem-formado (*well-formed*) quando segue algumas regras básicas. Tais regras são mais simples do que para documentos HTML e permitem que os dados sejam lidos e expostos sem nenhuma descrição externa ou conhecimento do sentido dos dados XML. Um documento que não é bem formado é considerado inválido e deverá apresentar erros.

Nos documentos XML bem estruturados, as tags possuem sempre uma parte inicial e outra final. Elas poderão ter dois formatos:

```
<minha_tag> exemplo </minha_tag>  ou
<minha_tag atrib= "exemplo" />
```

As tags de elemento têm que ser apropriadamente posicionadas, ou seja, não deve haver sobreposição de seus elementos. Um exemplo de sobreposição é o seguinte:

```
<titulo>XML <subtitulo> Guia de Consulta Rápida </titulo>
<autor> Zé Pedro </autor> </subtitulo>
```

Claramente percebe-se que a tag `<subtitulo>` está sobrepondo a tag `<titulo>`. Corrigindo o erro, tem-se:

```
      <titulo>XML <subtitulo> Guia de Consulta Rápida </subtitulo>
</titulo>
      <autor> Zé Pedro </autor>
```

Caracteres especiais podem ser digitados usando referências de caracteres Unicode. Por exemplo, o "." corresponderia ao caracter `#38;` , e assim por diante.

A seguir serão apresentadas mais características de um documento XML bem-formado:

```
<!-- comentário -->
```

Todo comentário é ignorado pelos processadores do documento, servindo ao único propósito de armazenar informações ao programador quando este abrir o documento.

```
<?Target data...?>
```

Uma instrução para um processador. "Target" identifica o processador para o qual ela foi direcionada e "data" é a string contendo a instrução a ser passada ao processador.

```
<!ENTITY name value>
```

Declara uma entidade com um nome e um valor; expandida usando a referência `ENTITY`: "&`name`" (entidades externas e referências de entidades de parâmetros são ignoradas aqui).

```
<!ELEMENT...>, <!ATTLIST...>,...
```

Informações de definição de tipos no documento XML. As melhores alternativas encontradas atualmente são a DTD (*Document Type Definition*) e o *XML Schema*.

2.1.3. Características de um Documento XML

2.1.3.1. Representação estruturada dos dados

A XML tem como sua mais marcante característica a representação dos dados em árvore, que se deve ao fato de derivar da linguagem chamada SGML, que significa *Standard Generalized Markup Language* ou, Linguagem Padrão de Marcações Genéricas.

A SGML é um padrão internacional (ISO 8879) publicado em 1986 e descreve o uso de marcações descritivas mescladas a um documento. A SGML também fornece um método padronizado para nomear as estruturas de um texto, definindo modelos hierárquicos para cada tipo de documento produzido. A linguagem força que cada um dos elementos descritos, como "capítulo", "título" e "parágrafo", a se ajustarem na estrutura lógica e previsível de seu documento.

Há diferentes estruturas de documentos para cada diferente tipo de informação criada: boletins informativos, manuais técnicos, catálogos, especificações de projeto, relatórios, cartas e memorandos.

A SGML permite a criação de documentos independentes do tipo de máquina e dos programas usados já que, como a SGML é um padrão internacional, ela é portável. Pode-se trocar informações entre usuários em diferentes sistemas e plataformas sem nenhuma alteração necessária.

Um exemplo da importância dessa independência é a fotografia: pode-se comprar um filme no padrão ISO 100, por exemplo, sem se preocupar com o fabricante, ajustar-se a velocidade da máquina para 100 (automático em muitas máquinas) e sair tirando fotos. A SGML representa (ou tenta representar) para a indústria da documentação o que os padrões ISO e ASA representam para a indústria fotográfica: uma universalização dos formatos, tornando-os 100% compatíveis e acessíveis a partir de qualquer plataforma ou sistema usado.

A XML, como subconjunto da SGML que é, foi otimizada para distribuição através da Internet, e é definida pela W3C (*World Wide Web Consortium)*, assegurando que os dados estruturados serão uniformes e independentes de aplicações e fornecedores.

XML provê um padrão para codificar o conteúdo e as semânticas de uma grande variedade de aplicações, das mais diferentes complexidades, dentre elas:

- Um simples documento, com título, subtítulo etc.
- Um registro estruturado tal como uma ordem de compra de produtos.
- Um objeto com métodos e dados como objetos Java ou controles ActiveX.
- Um registro de dados como o resultado de uma consulta a bancos de dados.
- Apresentação gráfica, como interface de aplicações de usuário, através de hipertexto ou gerando interfaces gráficas próprias.

- Entidades e tipos de esquema padrões para a formatação de documentos.

- Links entre informações, pessoas e comunidades na *Web*.

2.1.3.2. Pequeno uso de recursos

Uma característica importante é que, uma vez tendo sido recebido o dado pelo cliente, tal dado pode ser manipulado, editado e visualizado sem a necessidade de reacionar o servidor. Dessa forma, os servidores têm menor sobrecarga, reduzindo a necessidade de computação e reduzindo também a requisição de banda passante para as comunicações entre cliente e servidor.

2.1.3.3. Separação entre dados e apresentação

A mais importante característica da XML se resume em separar a interface com o usuário (apresentação) dos dados estruturados. O HTML especifica como o documento deve ser apresentado na tela por um navegador. Já a XML define o conteúdo do documento. Por exemplo, em HTML são utilizadas tags para definir tamanho e cor de fonte, assim como formatação de parágrafo. No XML você utiliza as tags para descrever os dados, como exemplo tags de assunto, título, autor, conteúdo, referências, datas, etc...

O XML ainda conta com recursos tais como folhas de estilo definidas com *Extensible Style Language* (XSL) e *Cascading Style Sheets* (CSS) para a apresentação de dados em um navegador. O XML separa os dados de apresentação e processo, o que permite visualizar e processar o dado como quiser, utilizando diferentes folhas de estilo e aplicações. Pode-se criar aplicativos para ter acesso aos dados de um documento XML utilizando linguagens de programação convencionais, pois o fato de XML ser padronizada, aberta, e escrita em texto puro, torna relativamente fácil criar bibliotecas de desenvolvimento para aproveitar os recursos de XML, permitem a criação de múltiplas formas de visualização dos dados estruturados.

2.1.4. Principais vantagens da linguagem XML

O XML tem por objetivo trazer flexibilidade e poder às aplicações Web. Dentre os benefícios para desenvolvedores e usuários temos:

- Buscas mais eficientes;

- Desenvolvimento de aplicações Web mais flexíveis;

- Distribuição dos dados via rede de forma mais comprimida e escalável;

- Padrões abertos.

2.1.4.1. Buscas mais eficientes

Os dados em XML podem ser unicamente "etiquetados", o que permite que, por exemplo, uma busca por livros seja feita em função do nome do autor. Atualmente, uma busca pelo nome do autor poderia levar a qualquer site que tivesse referência a tal nome, não importando se fosse o autor do livro ou simplesmente um livro sobre o autor. Sem o XML é necessário para a aplicação de procura saber como é construído cada banco de dados que armazena os dados de interesse, o que é impossível. O XML permitiria definir livros por autor, título, assunto, etc., o que facilitaria enormemente a busca.

2.1.4.2. Desenvolvimento de aplicações flexíveis para a Web

O desenvolvimento de aplicações Web em três camadas, ou *three-tier*, é altamente factível com o XML. Os dados XML podem ser distribuídos para as aplicações, objetos ou servidores intermediários para processamento paralelo, criando e simplificando protocolos entre outras coisas. Esses mesmos dados também podem ser distribuídos para o desktop (PC e similares) para ser visualizado em um navegador, facilitando a disseminação de conteúdo informativo. A XML é considerada de grande importância na Internet e em grandes intranets porque provê a capacidade de interoperação dos computadores por ter um padrão flexível, aberto e independente de dispositivo.

2.1.4.3. Integração de dados de fontes diferentes

Nos dias de hoje é praticamente impossível a busca de informação em múltiplos bancos de dados, que na sua grande maioria são incompatíveis. O XML permite que tais dados possam ser facilmente combinados. Essa combinação seria feita via software em um servidor intermediário, estando os bancos de dados na extremidade da rede. Os dados poderiam ser distribuídos para outros servidores ou clientes para que fizessem o processamento, a agregação e a distribuição da informação. A simples migração de informações de um banco de dados para outro causa muitas dores de cabeça ao desenvolvedor. Documentos XML poderiam servir de formato intermediário para o armazenamento de informações.

2.1.4.4. Computação e manipulação local

Os dados XML recebidos por um cliente são analisados e podem ser editados e manipulados de acordo com o interesse do usuário. Ao contrário de somente visualizar os dados, os usuários podem manipulá-los de várias formas. Os recursos disponíveis do Document Object Model (DOM), ou em outros analisadores de documentos XML (parser's), permitem que os dados sejam manipulados via scripts ou outra linguagem de programação.

2.1.4.5. Múltiplas formas de visualizar os dados

Os dados recebidos por um usuário podem ser visualizados de diferentes formas, uma vez que o XML define somente os dados e não o visual. A interpretação visual poderia ser

dada de várias maneiras diferentes, de acordo com as aplicações. Os recursos de CSS e XSL permitem essas formas particulares de visualização. A separação da interface visual dos dados propriamente ditos permite a criação de aplicações mais poderosas, simples e flexíveis.

2.1.4.6. Atualizações granulares dos documentos

Os dados podem ser atualizados de forma granular, evitando que uma pequena modificação no conjunto de dados implique na busca do documento inteiro novamente. Dessa forma, somente os elementos modificados seriam enviados pelo servidor para o cliente. Atualmente, uma modificação em um ítem de dados acarreta na necessidade de atualização da página inteira. O XML também permite que novos dados sejam adicionados aos já existentes, sem a necessidade de reconstrução da página.

2.1.4.7. Fácil distribuição na Internet

Assim como HTML, a XML, por ser um formato baseado em texto aberto, pode ser distribuída via HTTP sem necessidade de modificações nas redes existentes, atravessando facilmente sistemas como Firewalls.

2.1.4.8. Compressão

A compressão de documentos XML é fácil devido à natureza repetitiva das tags usadas para definir a estrutura dos dados. A necessidade de compressão é dependente da aplicação e da quantidade de dados a serem movidos entre clientes e servidores. Os padrões de compressão do HTTP 1.1 podem ser usados para o XML.

2.2. Esquemas XML

Nos últimos anos, a XML vem se tornando um formato de dados cada vez mais utilizado e popular, isso deve-se principalmente ao fato de sua sintaxe ser facilmente extensível e baseada em texto (fácil compreensão ao seres humanos). Infelizmente, isto também significa que XML é uma linguagem muito geral, e para que possa ser utilizada em aplicações mais sérias (troca de documentos eletrônicos em geral), é necessário definir uma especificação (restrições, estrutura, tipos de dados) para representa-la, é neste ponto que percebemos a importância dos esquemas XML .

Um esquema define os elementos que podem aparecer em um documento e os atributos que podem ser associados a este elemento, é uma descrição da estrutura do banco de dados ou de outras fontes de dados. Um esquema também define dados da estrutura de um documento tais como, estrutura hierárquica, a seqüência dos elementos (ordenação), o número de nós-filhos que um elemento pode ou deve ter, além de também definir valores default para atributos. Dentre as principais linguagens de definição de esquemas XML estão a DTD (Document Type Definition) e a XML Schema.

2.2.1. Exemplo de esquemas XML (XML Schema e DTD)

Para exemplificar o uso de esquemas XML e a sintaxe dos dois principais modelos de definição de esquemas XML (XML Schema e DTD), considere o documento XML abaixo (fig 2.2.1) com apenas três elementos: "colaboradores", "funcionario" e "descricao".

Fig. 2.2.1

```
<?xml version=”1.0” ?>
<colaboradores qtd=”3”>
    <funcionario>Sergio</ funcionario >
    <funcionario>Gustavo</ funcionario >
    <funcionario>Marcelo</ funcionario >
    <funcionario>Mariana</ funcionario >
    <descricao>Funcionarios TCM</descricao>
</colaboradores>
```

Neste exemplo, o elemento principal é “<colaboradores>”. Neste exemplo, "<colaboradores>" não contém texto, mas contém um ou mais elementos filhos chamados "<funcionarios>", e um elemento filho chamado "descricao". Os elementos "pessoa" e "descricao" contêm somente texto e não contém nenhum outro elemento.

Uma maneira com a qual poderíamos descrever este esquema é utilizar uma DTD (Document Type Definition), que nada mais é que uma gramática utilizada para descrever formalmente um esquema em particular.

Uma DTD para o documento exemplo seria:

Fig. 2.2.2

```
<!DOCTYPE colaboradores [
<!ELEMENT colaboradores (funcionario+, descricao)>
<!ELEMENT funcionario (#PCDATA)>
<!ELEMENT descricao (#PCDATA)>
<!ATTLIST colaboradores qtd CDATA> ]>
```

Um segundo e mais interessante método para fazer a descrição deste esquema seria utilizar XML Schema. Um esquema do tipo XML Schema é uma alternativa ao DTD com inúmeras vantagens, e por isso vem a ser apresentado como seu sucessor. Sua proposta é definir as regras de formação de documentos XML, descrevendo desta forma a estrutura e as regras de validação de um documento XML. A linguagem XML Schema também é chamada de XML Schema Definition (XSD).

XML Schema foi proposto inicialmente pela Microsoft, mas tornou-se rapidamente uma recomendação oficial da W3C (W3C Recommendation) em maio de 2001.

Um esquema em XML Schema para o documento exemplo seria:

Fig. 2.2.3

```
<?xml version="1.0"?>
<schema xmlns=http://www.w3c.org/1999/XMLSchema
xmlns:grp=http://meunamespace.com/Colaboradores
targetNamespace="http://meunamespace.com/colaboradores">
    <complexType name="tColaboradores" content="elementOnly">
        <element name="funcionario" type="string" minOccurs="1" maxOccurs="*"/>
        <element name="decricao" type="string" minOccurs="1" maxOccurs="1"/>
        <attribute name="qtd" type="integer"/>
    </complexType>
<element name="colaboradores" type="tColaboradores">
</schema>
```

Um XML Schema sempre inicia com "<schema>" e termina com "</schema>". Todas as declarações de elementos, atributos e tipos devem ser colocadas entre estas duas tags. Isto acontece porque um documento XML Schema deve ser um documento XML bem formado, e para isso deve possuir apenas um elemento raiz, que neste caso é "<schema>".

A parte mais importante de qualquer esquema é a definição de quais elementos e atributos são permitidos em uma determinada classe de documentos e como eles se relacionam uns com os outros. XML Schema permite que se faça isso através da utilização de tipos. Os tipos podem ser simples (simpleType) ou complexos (complexType). Um simpleType é um dos tipos básicos do XML Schema, tais como string, data, float, double, timeDurations, etc. A declaração de um complexType define a estrutura de um elemento, ou seja, define os sub-elementos, atributos, cardinalidades dos sub-elementos, obrigatoriedade dos atributos, etc. No exemplo do esquema para a turma, temos um exemplo de declaração de um complexType "tColaboradores", também existe uma outra declaração "<element name="colaboradores" type=tColaboradores">" depois da declaração do complexType "tColaboradores". Esta declaração liga o nome "colaboradores" ao complexType "tColaboradores". Isso significa que em uma instância de um documento XML que segue este esquema deveremos ter um elemento "colaboradores" com sub-elementos "funcionario" e "descricao", como mostrado no primeiro exemplo.

A cardinalidade é indicada pelos atributos "MinOccurs" e "MaxOccurs", ou seja, o sub-elemento "funcionario" deve aparecer pelo menos uma vez, e não existe um limite máximo. Já o sub-elemento "descricao" deve aparecer no mínimo 1 e no máximo 1 vez, fazendo com que ele seja obrigatório. Pode-se especificar o conteúdo de um elemento através do atributo "content". Um elemento pode conter:

- Somente texto, neste caso seu "content" deve ser declarado como "textOnly";

- Somente sub-elementos, neste caso seu “content” deve ser declarado como “elementOnly”;

- Texto e sub-elementos, neste caso seu “content” deve ser declarado como “mixed”;

- Nada, neste caso seu “content” deve ser declarado como “empty”.

O complexType “colaboradores” também possui um atributo “qtd”, que foi declarado no esquema através da tag <attribute>. Um atributo pode ser declarado como um tipo simpleType apenas, complexTypes não são permitidos para atributos, além disso, pode-se especificar se um atributo é obrigatório ou opcional. Isto é feito através do atributo “use”. Se o atributo é obrigatório, então declara-se use=“required”. Um atributo também pode ser opcional (use=“optional”) ou fixo (use=“fixed”), no caso de ser fixo, deve-se dizer o valor default do atributo utilizando “value”, por exemplo:

<attribute name="qtd" type="integer" use=”required”/>

2.2.2. Propostas alternativas para esquemas XML

Além da DTD e da XML Schema, há diversas outras propostas de esquemas XML surgindo, cada uma adaptada de certa forma a um tipo de negócio e melhorando em alguns pontos as representações existentes atualmente, seja dando-lhes maior abrangência ou adicionando novos atributos. Dentre elas podemos citar:

- **RELAX NG**, desenvolvido na OASIS (Organization for Advancement in Structured Information Standards) é resultado da junção de 2 propostas anteriores:

- **TREX (Tree Regular Expressions)**, proposta por James Clark - líder técnico das recomendações XML 1.0 e editor do XSLT e Xpath.

- **RELAX**, proposta por Murata Makoto.

- **XML-Data Reduced (XDR),** foi uma iniciativa BizTalk promovida pela Microsoft, é uma modificação da proposta XML-Data submetida ao W3C em 1998.

2.2.3. Vantagens da XML Schema

XML Schema se apresenta como uma proposta para substituir a DTD, e para tanto introduz uma série de conceitos que a DTD não possui, possibilitando a representação de uma gama maior de estruturas que podem ser representadas e permite uma maior flexibilidade. Mas é claro que novas estruturas podem ser adicionadas à proposta da W3C antes que XML Schema se torne padrão, e quem sabe estas estruturas adicionadas tornem XML Schema ainda mais flexível e adaptável às mais diversas aplicações.

Dentre as principais vantagens da XML Schema podemos destacar:

- Utiliza a sintaxe da XML e neste ponto, está uma de suas principais vantagens, tornando-as mais inteligíveis que DTD's;

- Possibilita o processamento de documentos identificando as limitações de estrutura e conteúdo e validando o mesmo;

- Permite a utilização de tipos de dados diversos (string, data, inteiro, float, etc.) (Anexo 1), o que não é possível na DTD;

- XML Schema possui um mecanismo de derivação de tipos, ou seja, permite a criação de novos tipos a partir de outros já existentes, e pode ser feito de duas maneiras: por restrição ou por extensão. Tipos simples só podem ser derivados por restrição, e isto é feito aplicando a um tipo básico "facetas" definidas na XML Schema, ou através do uso de uma linguagem de expressões regulares que também foi definida pela XML Schema;

- Provê integração com namespaces XML;

- Provê herança.

2.3. Web Services

Historicamente, os desenvolvedores construíam aplicativos integrando serviços em um sistema local, este modelo garantia acesso a uma ampla gama de recursos de desenvolvimento e o controle preciso de como o aplicativo se comportaria. Este modelo, porém, já está superado, hoje os desenvolvedores constroem sistemas complexos que unificam o uso dos mais diversos aplicativos (n-tier), através de intranets ou Internet.

"Na prática, as máquinas do amanhã não serão puras máquinas de rede que adquirem suas funções on-line nem puros PCs, recheados de softwares de fábrica. Eles usarão uma mistura de recursos locais e remotos por meio de softwares nômades, fluentes, porque isso atenderá melhor às necessidades das pessoas" [DERT, 2002].

Com o aumento da complexidade dos sistemas devido à própria evolução dos recursos computacionais surgiram novos paradigmas como concorrência, e desafios característicos da interoperação entre sistemas heterogêneos tais como:

- Grande diversidade de componentes (EJB, CORBA, DCOM);

- Diversidade de linguagens: Java, C/C++, C#, ASP;

- Firewalls;

- Ausência de padrões de desenvolvimento amplamente adotados.

O desenvolvimento baseado no modelo n-tier resulta em menor tempo de acesso ao mercado, maior produtividade no desenvolvimento e em última análise, melhor qualidade de software já que um mesmo componente ao ser atualizado influencia todos os sistemas que o utilizem, garantindo com isso atualizações a um menor custo e mais freqüentes, como por exemplo, um componente de acesso a disco.

A necessidade de superar os desafios citados, a combinação dos processos coesos e altamente produtivos do modelo n-tier com a flexibilidade e os conceitos orientados para a mensagem da Web, levaram ao surgimento deste estilo de computação chamado Web Service. Um Web Service é um aplicativo que expõe suas propriedades programaticamente dentro da Internet ou de uma intranet usando protocolos padrão, como HTTP (Hypertext Transfer Protocol) e XML. É um sistema publicado, localizado e chamado através da Internet de forma síncrona ou assíncrona, que encapsula funções e objetos remotos via um protocolo padrão conhecido.

Apontado por muitos como o próximo estágio evolutivo da Internet (no que diz respeito ao desenvolvimento de sistemas) e consequentemente dos sistemas distribuídos, Web Services são oque há, de mais recente em desenvolvimento de sistemas distribuídos. Grandes empresas como Microsoft, IBM e Sun já possuem implementações das principais tecnologias utilizadas em Web Services como SOAP (Simple Object Access - Protocolo padrão de aplicações, baseado em XML, para o vínculo a Web Services), XML e das outras camadas constituintes dos Web Services e estão investindo fortunas nestas novas tecnologias.

O conceito fundamental é simples, os Web Services nos permitem compor as RPC's (Remote Procedure Calls) de um objeto pela Internet ou por uma rede, seguindo as especificações (WSDL - Web Service Description Language) que encontramos em repositórios (UDDI - Universal Description, Discovery, and Integration) por exemplo. Os Web Services não são a primeira tecnologia a nos permitir isto, mas são diferentes das tecnologias existentes pelo fato de usarem padrões comuns e já existentes, como HTTP, XML, SOAP. Em um Web Service ocultamos os detalhes relativos à implementação do cliente, ao qual precisa conhecer apenas a URL do serviço, os métodos disponíveis neste e os tipos de dados usados para as chamadas do método, mas não precisa saber se o serviço foi construído em Java e se está sendo executado em uma plataforma Linux, ou se é um Web Service escrito em ASP.NET, que é executado em uma plataforma Windows. Notamos neste ponto então uma das grandes vantagens dos Web Services, a Independência de plataforma.

2.3.1. Definição estrutural

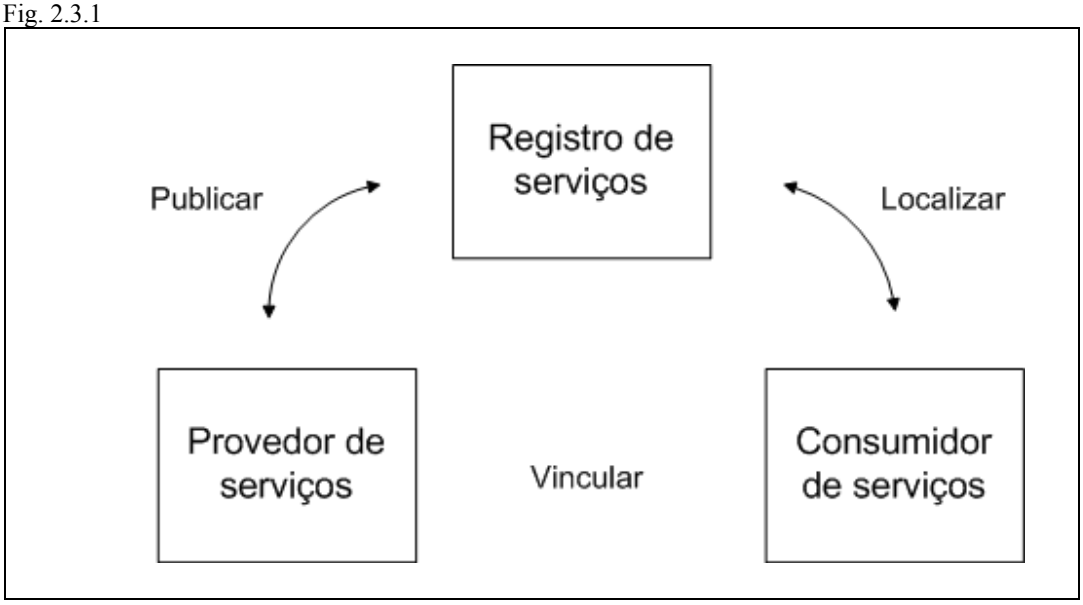
Para demonstrarmos a estrutura de um Web Service, usaremos um modelo conceitual composto de dois elementos básicos de qualquer sistema, seja web ou win32 ou ainda linux, papéis e operações.

- **Papéis**, são as entidades que formam o sistema;

- **Operações**, são os métodos chamados por estas entidades, ou seja, as funções que as mesmas podem executar e formam a funcionalidade do sistema como um todo.

A figura abaixo representa uma visão de alto nível de um modelo de Web Services típico, onde podemos observar os 3 papéis básicos de um Web Service e as operações fundamentais que os mesmos executam.

Fig. 2.3.1



- **Provedor de serviços:** É a entidade que cria o Web Service, geralmente o provedor de serviços oferece através de seu Web Service alguma funcionalidade comercial de interesse de outras empresas. O provedor de serviços para dispor seu Web Service a outras empresas possíveis interessadas, deve primeiramente descrever o Web Service em um formato padrão (WSDL tem sido o mais difundido atualmente mas não necessariamente seu Web Service tem de ser descrito neste padrão, existem diversas extensões deste padrão no mercado hoje, cada qual adaptada a um tipo de negócio), que seja compreensível por qualquer empresa que queira utilizar este Web Service. Em segundo lugar, para alcançar um grande público, o provedor de serviços deve publicar os detalhes sobre seu Web Service em um registro disponível publicamente para que chegue ao conhecimento de todos os possíveis interessados e desta forma gerar negócios.

- **Consumidor de serviços:** Toda empresa que utilize em seus sistemas ou em suas rotinas internas um Web Service desenvolvido por um provedor de serviços pode ser chamada de consumidor de serviços. O consumidor de serviços pode conhecer a funcionalidade de um Web Service, a partir da descrição WSDL disponibilizada pelo provedor de serviços. Para recuperar os detalhes técnicos como métodos disponíveis e parâmetros, o consumidor de serviços realiza uma pesquisa (num repositório UDDI, por exemplo) sobre o registro onde o provedor de serviços publicou sua descrição do Web

Service. O mais importante é que o consumidor de serviços pode obter, a partir da descrição do serviço, o mecanismo para vínculo com o Web Service do provedor de serviços, podendo então utilizar esse Web Service.

- **Registro de serviços:** Um registro de serviços é o local onde o provedor de serviços publica seus Web Services, e no qual um consumidor de serviços pode pesquisar Web Services. Os provedores de serviços normalmente publicam sua capacidade de Web Services no registro de serviço, para que dessa forma, os consumidores de serviço os encontrem e em seguida, vinculem-se ao seu Web Service. Tipicamente, informações como detalhes da empresa, Web Services por ela fornecidos e detalhes sobre cada Web Service, inclusive detalhes técnicos, são armazenadas no registro do serviço. Precisamos obter uma comunicação entre as aplicações, sem considerar o tipo de linguagem na qual a aplicação foi escrita, a plataforma onde a aplicação está sendo executada e outras considerações. Para que isso se torne possível, precisamos dos padrões de cada uma dessas três operações e de uma maneira padrão para um provedor de serviços descrever seu Web Service, sem considerar a linguagem em que ele foi escrito e é neste ponto que entra a WSDL como padrão para descrição de Web Services mais amplamente divulgado atualmente, e a UDDI como padrão para publicação e localização de Web Services nos registros de serviços.

2.3.2. Vantagens

Pela própria definição de Web Services que discutimos anteriormente podemos perceber várias vantagens de um modelo de desenvolvimento de aplicações baseado em Web Services, dentre elas:

- **Independência de plataforma**, promove interoperabilidade, não importa em qual plataforma ou linguagem tenha sido desenvolvido seu Web Service, ou em qual plataforma ele será consumido;

- **Utilização de padrões já consolidados e amplamente difundidos**, a comunicação é feita via HTTP; os protocolos de autenticação e encriptação são os mesmos; a manutenção de estado é feita da mesma forma; é normalmente implementado pelo próprio servidor Web;

- **Facilidade de transpor "firewalls" e roteadores**, pois para estes a chamada do Web Service é uma simples comunicação HTTP;

- **Tanto os dados como as funções são descritas em XML**, o que torna o protocolo não apenas fácil de usar como também muito robusto e altamente inteligível, além é claro de agregar todas as demais vantagens da XML;

- **B2B a baixo custo**, favorecido pela componentização e reuso dos mesmos.

2.3.3. Aplicações

Os Web Services mais simples são fontes de informação, que podem ser facilmente incorporados às aplicações como: preços de produtos, cotas de estoque, previsões de tempo e temperatura, resultados de partidas esportivas, etc.

É fácil imaginar uma grande classe de aplicações que podem ser feitas para analisar e agregar informações importantes, e apresentá-las de várias formas; como por exemplo, uma planilha que resume sua situação financeira, títulos, contas bancárias, empréstimos, etc. Se essa informação estiver disponível através de Web Services, a planilha pode ser atualizada continuamente.

Algumas dessas informações podem ser abertas ou podem requerer uma autorização de acesso para o serviço. Muita dessa informação já está disponível na Web, mas os Web Services tornam o acesso programático mais fácil e confiável.

A publicação de aplicações existentes através de Web Services permite a construção de aplicações novas e poderosas. Por exemplo, pode-se desenvolver uma aplicação de aquisições que automaticamente obtém os preços de vários fornecedores, possibilitando que o usuário selecione um fornecedor, faça o pedido e acompanhe o carregamento até o recebimento. A aplicação do fornecedor pode também utilizar um Web Service para checar o crédito do cliente, cobrar da conta e solicitar o transporte da carga para uma transportadora.

Para mostrar como o conceito de Web Services se aplica nos mais diversos tipos de negócios, podemos citar também como exemplo de Web Service, um sistema no contexto laboratorial, onde os resultados são recebidos e processados (analisados pelo laboratório) por um Web Service e posteriormente enviados a outro Web Service de banco de resultados, disponível na Internet onde os pacientes de uma clínica, por exemplo, podem consultar seus exames sem ter de sair de casa. Notamos nestes exemplos o quanto um Web Service pode representar em termos de economia e agilidade para os mais diversos tipos de empresas, e em praticamente todos os ramos de negócios.

2.4. WSDL (Web Service Description Language)

A primeira versão da WSDL foi proposta à W3C apenas a título de discussão (W3C Note) em março de 2001, pela empresas Ariba, IBM e Microsoft, com a proposta de se tornar um padrão para a descrição de Web Services (até agora ainda não é um padrão homologado pela W3C).

A WSDL é um padrão que utiliza o formato XML para descrever Web Services. Basicamente, o documento WSDL para um Web Service define os métodos que estão presentes no Web Service, os parâmetros de entrada/saída para cada um dos métodos, os

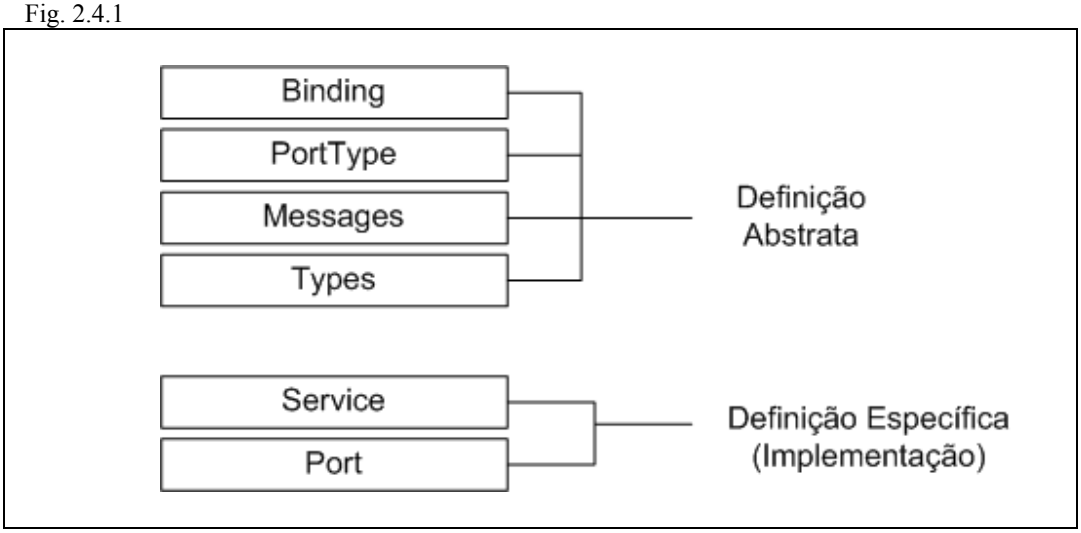
tipos de dados, o protocolo de transporte usado (geralmente SOAP) e a URL da extremidade onde o Web Service será hospedado.

A WSDL forma a camada de descrição dos serviços, fornece ao provedor de serviços uma forma padrão de descrever a funcionalidade de seus Web Services. A WSDL fornece este mecanismo, definindo a gramática de XML à descrição do Web Service. Um documento WSDL, descreve um Web Service como uma coleção de extremidades ou portas operando independentemente. A WSDL é para um Web Service o que o CORBA IDL é para o CORBA ou a Microsoft MIDL é para o os componentes COM, e assim por diante.

2.4.1. Definição estrutural de um documento WSDL

De acordo com a WSDL Specification podemos descrever uma especificação de Web Service, como vemos a seguir na figura 2.4.1:

Fig. 2.4.1



Um documento WSDL define um XML Schema para descrever um Web Service. A descrição WSDL é dividida em dois grupos, o primeiro apresenta uma definição abstrata independente do protocolo de transporte de alto nível, enquanto o segundo representa uma descrição específica para o transporte na rede.

Definição abstrata independente do protocolo de transporte:

- **Message:** Um elemento Message contém uma definição dos dados a serem transmitidos e pode consistir de uma ou mais partes. É semelhante ao parâmetro na chamada de um método ou função.

- **Types:** O elemento Types contém os tipos de dados que estão presentes na mensagem e são enviados e recebidos na mesma. Normalmente é usado XML Schema

para definir os elementos Types, visando garantir uma maior neutralidade na descrição dos mesmos.

- **PortType:** Os elementos PortType contêm um conjunto de operações representadas como elementos Operation, que estão presentes em um Web Service, como, por exemplo, RetiraDoEstoque, RetornaAoEstoque, tratamento de exceções e outros. Além disso, ela pode ter apenas uma mensagem de entrada ou saída, da mesma maneira que uma chamada de método normal. Comparativamente um elemento PortType se assemelha a uma classe, módulo ou biblioteca de funções presentes nas maiorias das linguagens de programação.

- **Binding:** O elemento Binding mapeia os elementos Operation em um elemento PortType, para um protocolo específico, ou seja faz o mapeamento da implementação com a interface descrita no XML descrevendo detalhes do protocolo. Um elemento Binding possui 2 atributos, o atributo Name que define a identificação do elemento Binding e o atributo Type que indica o elemento Port para a operação de vínculo.

Definição específica para o transporte na rede:

- **Services:** um service é uma coleção de elementos PortType.

- **ServiceType:** Define um conjunto de elementos PortType

Como pudemos observar para que uma operação possa ser executada em uma rede, devemos usar um protocolo de transporte específico para enviar os dados pela rede. É onde entra em cena o elemento Binding da WSDL. Um elemento binding captura o protocolo particular (SOAP, HTTP GET, HTTP POST, e outros) e os elementos PortType, Message e Types, mencionados acima. Podemos portanto, ligar a definição abstrata a vários protocolos de transporte, o que significa que uma definição de WSDL pode permitir que definamos um Web Service, independentemente do protocolo de transporte.

2.4.2. Exemplo de documento WSDL e seus elementos

Abaixo na figura temos um exemplo da estrutura de um documento WSDL.

Fig. 2.4.2

```
<message name="GetColecaoRequest">
   <part name="term" type="xs:string"/>
</message>

<message name="GetColecaoResponse">
   <part name="value" type="xs:string"/>
</message>
<portType name="ColecaoDeExames">
   <operation name="GetColecao">
     <input message="GetColecaoRequest"/>
     <output message="GetColecaoResponse"/>
```

```
        </operation>
    </portType>
```

No exemplo acima o elemento PortType define “ColecaoDeExames” como o nome de um elemento Port que pode ser visto como uma biblioteca de funções, se traçarmos um comparativo com as linguagens de programação mais comuns, e “GetColecao” como um elemento Operation, que corresponderia a uma função desta biblioteca de funções.

O elemento “GetColecao” comtém os elementos “GetColecaoRequest” e “GetColecaoResponse” que podem ser comparados aos parâmetros de entrada e retorno respectivamente .

Os elementos Message definem as partes de cada mensagem e os parametros de entrada e saida associados.

Resumindo, a descrição da WSDL de um Web Service contém uma coleção de portas cada porta se associa a uma extremidade na qual o Web Service pode aceitar comunicações (como um URL ou um endereço de e-mail) com um elemento Binding particular, o que descreve as mensagens aceitas para esta porta.

Sob o ponto de vista da arquitetura com a camada de descrição de serviços WSDL, o provedor de serviços descreve o Web Service através da criação e da publicação de um documento WSDL, que contém uma definição abstrata do Web Service e os detalhes de implementação do Web Service como localização, métodos e parâmetros.

A partir deste ponto as corporações, possíveis interessados a utilizar o Web Service deste provedor de serviços, devem de alguma forma localizar e recuperar este documento a fim de tomarem conhecimento dos tipos de dados, mensagens aceitas pelo Web Service e a própria localização (URL) do mesmo, para então poderem anexar as funcionalidades deste a suas rotinas internas ou sistemas (operação de vinculação).

2.4.3. Geração automática de documentos WSDL

Como já comentamos anteriormente um documento WSDL é baseado em XML, portanto pode ser escrito em qualquer simples editor de texto, mas este processo pode ser automatizado e com isso ganhar agilidade, já há no mercado diversos aplicativos que automatizam o processo de geração do documento WSDL e alguns incluem inclusive mecanismos de publicação dos mesmos.

Esse processo de geração automática facilita em muito o papel do desenvolvedor que deve se preocupar exclusivamente com a qualidade de seu aplicativo e após concluída uma versão de seu sistema seu documento de publicação (WSDL) estará pronto e gerado automaticamente. Dentre os aplicativos que possuem esta funcionalidade destacamos o Cape Clear's CapeConnect que é utilizado para geração de Web Services, o BEA's WebLogic Server 6.1 que permite a criação de EJB`s e gera automaticamente scripts Ant

para a criação do documento WSDL e diversas outras ferramentas disponíveis no mercado, que já trabalham com o conceito de WSDL, tais como Systinet WASP, The Mind Electic's GLUE, and IBM's Web Services Toolkit.

2.5. UDDI (Universal Description, Discovery, and Integration)

A UDDI é uma iniciativa em desenvolvimento no âmbito do consórcio industrial UDDI (http://www.uddi.org/), promovido originalmente pela IBM, Microsoft e Arriba, Atualmente se encontra na sua versão 3.0, cuja principal inovação foi o suporte a assinaturas digitais.

UDDI é atualmente a proposta mais aceita de protocolo padrão para publicar e/ou localizar Web Services, UDDI, permite que os provedores de serviços publiquem detalhes sobre suas empresas e os Web Services fornecidos pela mesma, em um registro central disponível publicamente. Esta é a parte relativa à "descrição" (Description) do UDDI. O protocolo UDDI Também fornece um padrão para permitir que os consumidores de serviços localizem provedores de serviços e detalhes sobre seus Web Services, esta é a parte relativa à "descoberta" (Discovery) do UDDI.

A partir de um documento WSDL, um consumidor de serviços pode determinar os detalhes do Web Service como as diferentes operações, tipos de dados, extremidades (vinculação), protocolos de vínculo (SOAP) e outros. Entretanto, existem outros fatores a serem considerados. Vamos examinar esses fatores, sob o ponto de vista tanto do provedor de serviços quanto do consumidor de serviços.

Primeiramente, em vez de publicar o documento WSDL para cada possível cliente, um provedor de serviços estará bem servido se publicar as informações sobre seu Web Service em um registro de forma que este esteja disponível publicamente para os consumidores de serviço interessados. Além de publicar apenas a descrição de seu Web Service, poderá ser uma boa idéia publicar também informações relacionadas ao negócio, como o nome dos seus negócios, os diferentes Web Services por eles oferecidos e outras informações relevantes.

Da mesma maneira, os consumidores de serviço podem querer encontrar os diferentes Web Services que sejam disponibilizados pelos provedores de serviços. Eles podem querer avaliá-los independentemente, antes de integrar esses Web Services às suas aplicações. Conseqüentemente, faz mais sentido, para eles, pesquisar em um "ambiente virtual" (uma rede P2P como demonstraremos em nossa proposta ou um repositório central como a proposta clássica da UDDI.org), onde os provedores de serviço já publicaram seus detalhes. Os consumidores de serviço também podem realizar uma pesquisa de nível mais amplo, com base em categorias comerciais, para encontrar as empresas nessas categorias e, a seguir, fazer o drill down em detalhes adicionais sobre os Web Services por elas oferecidos.

Existem muitas possibilidades além das apresentadas acima. Entretanto, o mais importante é a necessidade de um ambiente disponível publicamente, onde os provedores de serviços e os consumidores de serviços trabalham em conjunto para publicar e recuperar as informações apropriadas. Em outras palavras, precisamos de uma especificação para a publicação e descoberta de informações comerciais.

2.5.1. Definição estrutural da UDDI

Um diretório UDDI é um arquivo XML que descreve o negócio e os serviços. A forma como os serviços são definidos no documento UDDI é chamado Type Model ou tModel. Em muitos casos, o tModel contém o arquivo WSDL que descreve a interface SOAP do Web service, mas o tModel é flexível o suficiente para descrever quase todo tipo de serviço.

A especificação UDDI fornece uma estrutura comercial, usada para descrever um determinado negócio e esta estrutura de negócios, contém as seguintes informações:

- **Negócio:** (Páginas brancas) Contém as informações comerciais, como o nome do negócio, o contato comercial, e outras. Contém também uma ou mais instâncias da estrutura do serviço. A especificação UDDI também permite que uma definição comercial contenha um código de classificação que a identificará como pertencente a uma determinada categoria de negócios, como, por exemplo, instituições de empréstimo financeiro, esta característica é importante na busca de Web Services por classificação.

- **Serviço:** (Páginas amarelas) Uma estrutura de serviço captura os diferentes Web Services fornecidos por este negócio. Cada Web Service contém uma ou mais estruturas de especificação técnica. Uma empresa pode ter, por exemplo, dois Web Services: um Web Service representando o status do pedido de vendas e um representando o status do estoque. A especificação UDDI também permite que uma definição de serviço contenha um código de classificação que a identificará como pertencente a uma determinada categoria de serviços, como, por exemplo, a de verificação de crédito.

- **Especificação técnica:** (Páginas verdes) As especificações técnicas contêm detalhes técnicos sobre um Web Service. Uma das especificações técnicas é a especificação WSDL, à qual um consumidor de serviços pode fazer referência, e determinar os detalhes técnicos sobre a vinculação do Web Service.

2.6. Peer-To-Peer

O termo Peer-To-Peer (P2P) refere-se a uma classe de sistemas e aplicações que empregam recursos distribuídos para realizar tarefas críticas de forma descentralizada. Com o surgimento de técnicas de criptografia que tornam as redes mais confiáveis e tecnologias que as têm tornado mais estáveis, e a isso acrescido o fato do número de PC’s

(computadores pessoais) aumentar exponencialmente nos últimos anos, a tecnologia P2P está recebendo cada vez mais uma maior atenção por parte da comunidade científica, empresas desenvolvedoras de softwares e até de instituições militares que enxergam nesta nova tecnologia uma poderosa ferramenta para processamento de toda espécie, ou seja, alto poder de processamento (compartilhamento de recursos) a baixo custo.

Há diversas definições para P2P, dentre as mais adotadas pela comunidade científica:

- Intel P2P Working Group: "o compartilhamento de recursos e serviços computacionais através da troca direta entre vários sistemas";

- Kindberg [1]: "aqueles de expectativa de vida independente";

- Clay Shirky [1]: "P2P é uma classe de aplicações que tira vantagens dos recursos disponíveis nos nós da rede (armazenamento, conteúdo, hardware, software). Por acessar esses recursos de forma descentralizada, esta tecnologia precisa saber lidar com um ambiente de conectividade instável e endereços IP imprevisíveis, o que significa que necessita operar fora do sistema DNS e ter significativa senão total autonomia sobre os seus recursos".

O conceito adotado neste trabalho relaciona a tecnologia Peer-To-Peer ao termo Compartilhamento, ou seja, o modelo P2P permite aos nós oferecer e obter recursos dos *peers* pertencentes à topologia.

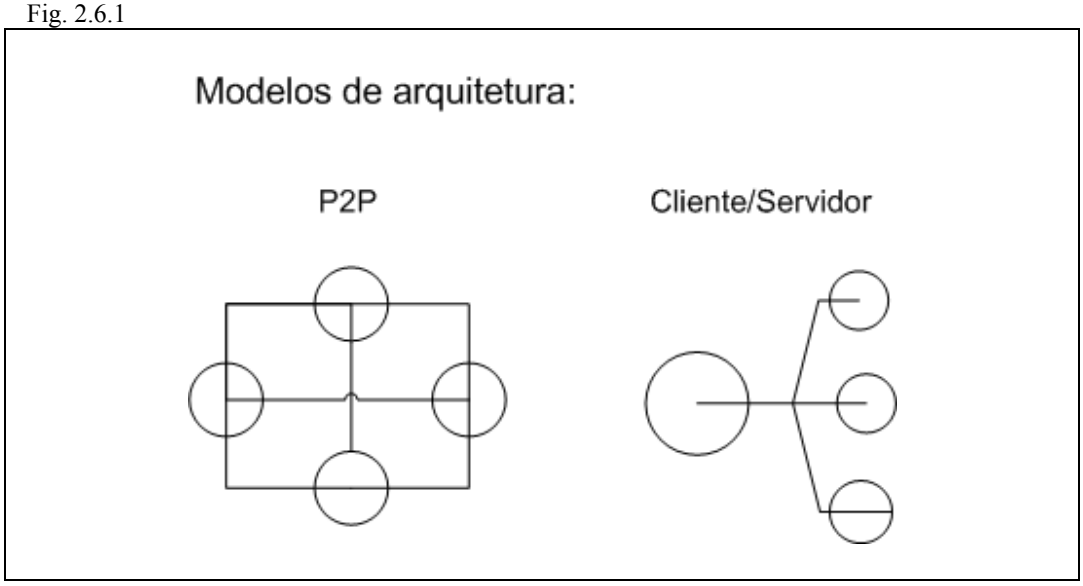
As tecnologias P2P ganharam popularidade a partir do surgimento do Napster e da sua posterior questão jurídica com as gravadoras sobre os direitos autorais das músicas, fato este que levou o sistema a sair do ar. Apesar disto, esta técnica encontra-se em constante crescimento em várias áreas, tais como na computação colaborativa em ambiente web, tendo recebido bastante foco das indústrias de softwares e do meio acadêmico. Dentre os investimentos mais importantes, pode-se citar: P2P Working Group; e o JXTA, uma iniciativa aberta da Sun, assunto de nosso próximo capítulo e utilizado como base na implementação de nossa proposta.

A tecnologia P2P propõe a interação direta entre os usuários da Internet e outras redes de diversos fins para um efetivo compartilhamento dos recursos geograficamente distribuídos. Esses recursos podem ser: capacidade de processamento, capacidade de armazenamento, banda de rede, entre outros.

Conceitualmente, computação P2P é uma alternativa ao modelo cliente/servidor, onde há tipicamente um único e pequeno cluster de servidores e muitos clientes, ou seja, o modelo P2P não possui o conceito de servidor: qualquer *peer* da rede pode exercer o papel tanto de cliente como de servidor, dependendo do contexto. Um *peer* é autônomo quando não é completamente controlado por outros ou pelo mesmo usuário. A dependência existente entre os *peers* diz respeito apenas à forma de obtenção/oferecimento de recursos, informações; encaminhamento de requisições, etc. Como conseqüência dessa autonomia, é

típica na arquitetura a existência de algoritmos de controle de replicação e de escalabilidade dos serviços a serem oferecidos, de forma a garantir a não eliminação da disponibilidade de um serviço na ocorrência de falha em alguns pontos.

Fig. 2.6.1



2.6.1. Vantagens

A arquitetura P2P propicia, através da agregação dos recursos disponíveis na rede:

- Interoperabilidade de baixo custo;

- Custo menor de compartilhamento dos recursos por utilizar a infra-estrutura de rede existente, e pela conseqüente distribuição dos custos de manutenção;

- Privacidade e anonimato - uma vez que incorpora esses requisitos em sua arquitetura e nos seus algoritmos, o P2P acaba por permitir que os *peers* tenham um controle autônomo sobre seus dados e recursos.

2.6.2 Aplicações

Podemos imaginar infinitas aplicações para esta tecnologia desde processamento distribuído, compartilhamento de mídias, comunicação, a sistemas colaborativos.

No caso do Napster, por exemplo, ele proporcionava a troca de músicas entre seus usuários. Mas a tecnologia em questão pode ser utilizada para diversos outros fins, tais como a busca de vida extraterrestre, ou a luta no combate ao câncer, aplicações estas cujo objetivo é ajudar na busca de benefícios a outras pessoas através da soma de recursos geograficamente distribuídos.

P2P pode ser visto, também, como uma forma de implementar sistemas baseados na noção de aumentar a descentralização de sistemas, aplicações ou simplesmente algoritmos. Levando-se em consideração que, em um futuro bem próximo, o mundo estará conectado de forma quase que totalmente distribuída, pode-se afirmar que P2P é uma maneira de impulsionar o intercâmbio de grandes capacidades de processamento, armazenamento e conectividade entre computadores pessoais distribuídos no mundo.

2.7. JXTA

2.7.1. Introdução

O Projeto JXTA iniciou como um projeto de pesquisa incubado na Sun Microsystems sob a liderança de Bill Joy e Mike Clary. Seu objetivo é explorar a visão da computação distribuída usando uma topologia P2P, e desenvolver blocos e serviços básicos que habilitariam aplicações novas para grupos de peers. Reconhecendo o esforço e a contribuição de pessoas de fora da Sun, o projeto foi publicado num site da Internet, mostrando sua especificação e o código de sua implementação, usando a licença da Apache Software, encorajando outros a se juntar em seu esforço de criar uma plataforma realmente útil e escalável.

Existem várias tecnologias P2P em uso hoje, nos níveis de aplicação e de plataforma. O Projeto JXTA é uma plataforma modular que provê blocos de construção essenciais para o desenvolvimento de uma gama de aplicações e serviços distribuídos. Ele especifica um conjunto de protocolos em vez de uma API. Desta forma ele pode ser implementado independentemente de sistema operacional ou linguagem de programação. O Projeto JXTA nos traz um simples wrapper baseado em protocolos que permitem a um pequeno dispositivo fazer parte de uma rede de peers para desenvolver uma aplicação completamente integrada que suporta medições, monitoração, segurança de alto nível e comunicação através de sistemas tipo servidor.

A plataforma do Projeto JXTA provê um ambiente descentralizado que minimiza falhas em pontos isolados e é independente de qualquer serviço centralizado. Apesar disso pode se desenvolver serviços centralizados e descentralizados sobre a plataforma do Projeto JXTA.

Neste capítulo apresentaremos inicialmente os conceitos principais envolvidos, passando em seguida a apresentar as características de uma aplicação JXTA. Continuamos apresentando a motivação para se utilizar o JXTA como modelo de implementação P2P e concluímos o capítulo mostrando uma visão um pouco mais detalhada dos protocolos que são a base do projeto JXTA.

2.7.2. Conceitos Iniciais

2.7.2.1. Peers

Um peer JXTA é qualquer dispositivo de rede (sensor, telefone, PDA, PC, servidor, supercomputador etc.) que implementa os protocolos centrais do JXTA. Cada peer é identificado por um ID único. O peer é autônomo e opera independentemente e assíncronamente, com relação aos outros peers. Alguns peer devem ter dependências em relação a outros peers, devido a exigências específicas como a necessidade de gateways, proxies, roteadores etc. Os peers devem publicar seus serviços recursos na rede (CPU, capacidade de armazenamento, bancos de dados, documentos, etc.) para que outros peers possam usá-los.

Os peers são configurados para descobrirem uns aos outros, espontaneamente, na rede, e formar relações passageiras ou persistentes com outros peers. Peers que provêem o mesmo conjunto de serviços tendem a ser interoperáveis; então os peers comumente só precisam interagir com um pequeno número de outros peers chamados "vizinhança de rede" ou "peers amigos". Peers não devem questionar a confiabilidade de outros peers. Podem entrar ou deixar a rede a qualquer momento. Um peer deve sempre prever que a conexão com qualquer peer pode cair durante a comunicação.

Um peer pode guardar anúncios de recursos JXTA, mas fazer isso é opcional. Peers podem ter armazenamento persistente.

Todo peer pode anunciar múltiplas interfaces de rede. Cada interface publicada é anunciada como um peer endpoint. Um peer endpoint é um URI que, exclusivamente, identifica uma interface de rede. Peer endpoints são usados pelo peer para estabelecer conexões diretas ponto-a-ponto entre dois peers.

A comunicação entre peers pode não ser ponto-a-ponto, ou devido à falta de conexões físicas de rede, ou configuração de rede (NATs, firewalls, proxies etc.). Um peer pode ter de usar um ou mais peers intermediários para rotear uma mensagem para outro peer.

2.7.2.1.1. Peers e Usuários

Freqüentemente um peer estará sob o controle de um operador humano, que seria a figura do usuário. O peer irá interagir com a rede no sentido de alcançar e servir o usuário. Mas nem todo peer tem um usuário. Muitos peers existem apenas para prover serviços e recursos para a rede, mas não é associado a qualquer usuário. Exemplos disso incluem dispositivos como sensores e impressoras, mas também serviços como bancos de dados.

O conceito de usuário existe principalmente para identificação na rede. Isto é necessário para aplicações que permitem aos usuários comunicarem-se com outros e administrarem suas interações. A identidade é freqüentemente ligada também a segurança e a permissões de uso.

Não há nenhuma relação explícita entre usuários e peers dentro do JXTA, mas para muitas aplicações os dois conceitos são associados a ponto de serem considerados como uma coisa só.

2.7.2.2. Grupos de Peers

Os peers se auto-organizam em grupos de peers (peergroups). Um grupo de peers é uma coleção de peers que tem interesses em comum. Cada grupo de peers é identificado exclusivamente por um PeerGroup Id, que é único. Os protocolos do JXTA não definem quando, onde, ou por que são criados os grupos. Eles só descrevem como um peer pode publicar, descobrir, tornar-se membro e monitorar grupos.

São três as principais motivações comuns para criar grupos de peers:

Criar um ambiente seguro. Os limites de um grupo permitem que o peer membro tenha acesso e publique conteúdos protegidos. Grupos de peers formam regiões lógicas cujos limites restringem o acesso para os recursos do grupo. Este grupo não reflete necessariamente os limites de rede física, como os impostos por roteadores e firewalls. Eles tornam virtuais as noções de roteadores e firewalls, subdividindo a rede em regiões seguras, não se preocupando com os limites físicos das redes atuais.

Criar um escopo de ambiente. Grupos de peers são geralmente formados e auto-organizados pelo interesse mútuo dos peers. Nenhuma regra particular é imposta nos grupos sobre o modo como ele é formado, mas peers com os mesmos interesses tenderão a unir-se a um mesmo grupo. Os grupos de peers servem para subdividir a rede em regiões abstratas, que provêem um mecanismo implícito de criação de escopo. Os limites de um grupo definem o âmbito de pesquisa quando se está procurando o conteúdo de um grupo.

Criar um ambiente de monitoração. Os grupos permitem aos peers monitorar um conjunto de peers com um propósito especial (verificação de status, introspecção de tráfico, análise de responsabilidade e etc.).

Alguns grupos de peers provêem uma gama de serviços denominados serviços de grupo. JXTA define um conjunto central destes serviços. Os protocolos do JXTA especificam o esqueleto destes serviços de grupo. Podem ser desenvolvidos serviços adicionais de grupo para servir serviços específicos. Por exemplo, um serviço de localização poderia ser implementado para achar serviços ativos (sendo executados em algum peer) e inativos (que não estão sendo executados no momento). O cerne destes serviços de grupo são:

Discovery Service (serviço de descoberta de informação): é usado por peer membro do grupo para procurar recursos de grupo de peers (outros peers, outros grupos, pipes e outros serviços).

Membership Service (serviço de organização de grupos): O Membership Service é usado pelos atuais peers membros para aceitar ou rejeitar uma nova aplicação que deseja tornar-se membro do grupo.

Espera-se que a maioria dos grupos de peers tenha pelo menos um Membership Service; entretanto pode ser simplesmente um serviço de autenticação de peers, que não impõe nenhuma real política de entrada de membros no grupo. O Membership Service é usado por um peer membro para permitir que um novo peer possa unir-se a um grupo de peers; para que um peer possa fazer isso, ele pode precisar ao menos descobrir um membro do grupo.

Peers que desejam entrar para um grupo de peers devem primeiro localizar um membro ativo do grupo, e então requisitar a entrada. O pedido de entrada é aceito ou rejeitado pelo conjunto de membros atuais. O Membership Service deve obrigar um peer a votar, ou então deve eleger um grupo representante designado para aceitar ou rejeitar os novos pedidos de entrada. Um peer pode pertencer, simultaneamente, a mais de um grupo de peers.

Access Service (serviço de acesso entre peers): serve para validar pedidos feitos por um peer a outro. O peer recebe o pedido; este pedido provê as credenciais do peer que fez a requisição, e ainda a informação sobre o pedido que é feito ao Access Service para determinar se o acesso é permitido.

Nem todas as ações dentro do grupo de peers precisam ser conferidas junto ao Access Service. Apenas as ações que são restritas a um subconjunto de peers membros devem fazê-lo.

Pipe Service (serviço de canais de comunicação): este serviço é usado para administrar e criar canais de conexão entre os diferentes peers membros do grupo.

Resolver Service (serviço de resolução): O Resolver Service é usado para enviar consultas a serviços que são executados em um peer do grupo e coletar as respostas.

Monitoring Service (serviço de monitoração): utilizado para permitir que um peer monitore outros peers membros do mesmo grupo de peers.

Nem todos os serviços acima precisam ser implementados por um grupo de peers. Cada serviço pode implementar um ou mais protocolos do JXTA. Um serviço implementará um protocolo por razões de simplicidade e modularidade, mas alguns serviços podem não implementar nenhum destes protocolos.

2.7.2.3. Serviços de Rede

Os peers cooperam e se comunicam para publicar, descobrir e invocar serviços de rede. Um peer pode publicar quantos serviços ele puder prover. Peers descobrem serviços de rede através do Peer Discover Protocol.

Os protocolos de JXTA reconhecem dois níveis de serviços de rede:

Serviços do peer: Um serviço de peer só é acessível no peer que está publicando o serviço. Caso este peer não esteja operável, então o serviço também não estará. Múltiplas instâncias do serviço podem ser executadas em diferentes peers, mas cada instância publica seu próprio anúncio.

Serviços do grupo: Um serviço de grupo é composto de uma coleção de instâncias (que podem estar cooperando entre si) do serviço que é executado em inúmeros membros do grupo de peers. Se um peer falhar, o serviço de grupo não é afetado, porque o serviço ainda está disponível em outro peer membro do grupo. São publicados serviços de grupo como parte do anúncio de um grupo de peers.

2.7.2.4. Identificadores

Dentro dos protocolos JXTA há várias entidades que precisam ser identificáveis através de um valor único. Estes são peers, grupos de peers, pipes e os conteúdos. Um identificador JXTA (JXTA ID) se refere a uma única entidade, servindo como meio canônico. São usadas URN's para a expressão de identificadores JXTA

2.7.2.5. Anúncios

Todos os recursos de rede, como peers, grupos de peers, pipes e serviços são representados através de anúncios. Anúncios são estruturas de meta-dados em linguagem neutra para descrever recursos. Os anúncios de JXTA utilizam XML para representar seus anúncios. Muitos dos protocolos JXTA dependem de anúncios para prover informações necessárias à sua utilização.

Os serviços podem definir novos tipos de anúncios ou utilizar sub-tipos de anúncios já existentes. Estes sub-tipos de anúncio permitem adicionar informação a ser provida, como também inserir meta-dados mais completos.

Anúncios são compostos de uma série de elementos organizados hierarquicamente. Os elementos podem aparecer em qualquer ordem dentro do anúncio. Cada elemento pode conter seus dados ou elementos adicionais. Um elemento também pode ter seus atributos; eles são pares do tipo nome-valor. Um atributo é usado para armazenar meta-dados que ajudam descrever os dados dentro do elemento.

A especificação dos padrões JXTA define, entre outros, os seguintes tipos de anúncio:

- Anúncio de Peer;
- Anúncio de Grupo de Peers;
- Anúncio de Pipe;
- Anúncio de Rendezvous.

Os protocolos JXTA fazem muitas referências a estes anúncios. Convém, portanto, estar familiarizado com anúncios antes de passar para os capítulos de especificação dos protocolos. Os anúncios são, sem dúvida, o documento mais comum trocado nas mensagens dos protocolos. Aqui vamos explicitar apenas os dois principais: o anúncio de peer e o de grupo de peers.

2.7.2.5.1. Anúncio de Peer

Um anúncio de peer descreve um peer e os recursos que ele provê ao grupo. O anúncio de peer guarda informações específicas sobre o peer como seu identificador, seu identificador no grupo e, opcionalmente, seu nome e informação descritiva. Também pode conter endereços de endpoint e qualquer outro atributo que serviços de um peer queiram publicar (como ser um peer Rendezvous para um grupo, por exemplo).

Fig. 2.7.1. Esquema de anúncio de peer

```
<xs:element name="PA" type="jxta:PA"/>
 <xs:complexType name="PA">
  <xs:sequence>
   <xs:element name="PID" type="JXTAID"/>
   <xs:element name="GID" type="JXTAID"/>
   <xs:element name="Name" type="xs:string" minOccurs="0"/>
   <xs:element name="Desc" type="xs:anyType" minOccurs="0"/>
   <xs:element name="Svc" type="jxta:serviceParams" minOccurs="0" maxOccurs="unbounded"/>
  <xs:sequence>
</xs:complexType>
```

<PID>: Este é um elemento obrigatório que identifica exclusivamente um peer.

<GID>: Este é um elemento obrigatório que identifica o grupo do peer ao qual este peer pertence.

<Name>: Este é uma string opcional que pode ser associada a um peer. O nome não é obrigatoriamente único, a menos que o nome seja obtido de um serviço de nomes centralizado que garanta a singularidade do nome.

<Desc>: Este é uma string opcional que pode ser usada para indexar e procurar um peer. Não é garantida sua unicidade. Dois peers podem ter as mesmas palavras-chave. A <Desc> pode conter espaços.

<Svc>: Qualquer número destes elementos pode existir. Cada um deles descreve a associação entre um serviço de grupo e os parâmetros arbitrários encapsulados. Em última instância, cada serviço é responsável pelo que é publicado sob seu identificador. Essa seção de serviço também pode, opcionalmente, conter um elemento "isOff " que significa que este serviço não está habilitado. Este elemento é usado para carregar uma escolha de configuração feita pelo dono do peer.

2.7.2.5.2. Anúncio de Grupo de Peers

Um anúncio de grupos de peers descreve recursos específicos do grupo tais como: nome, identificador de grupo, descrição, especificação e parâmetros de serviço.

Fig. 2.7.2. Esquema de anúncio de grupo de peers

```
<xs:element name="PGA" type="jxta:PGA"/>
<xs:complexType name="PGA">
  <xs:sequence>
    <xs:element name="GID" type="jxta:JXTAID"/>
    <xs:element name="MSID" type="jxta:JXTAID"/>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
    <xs:element name="Desc" type="xs:anyType" minOccurs="0"/>
    <xs:element name="Svc" type="jxta:serviceParam" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

<GID>: Este elemento provê o identificador do grupo. O identificador do grupo é o modo canônico de se referenciar a um grupo e identifica exclusivamente aquele grupo.

<MSID>: Especificação do identificador do grupo. Designa o módulo que provê mecanismos de funcionamento para o grupo de peers.

<Name>: Este é um nome opcional que pode ser associado a um grupo de peers. O nome não é obrigatoriamente único, a menos que o nome seja obtido de um serviço de nomes centralizado que garanta a singularidade do nome.

<Desc>: Este é um elemento opcional que provê informação descritiva, que pode ser usada para indexar e procurar um grupo de peers. O conteúdo deste elemento pode não ser único. Por exemplo, dois peergroups podem ter as mesmas palavras-chave.

<Svc>: Qualquer número de tais elementos pode existir. Cada deles descreve a associação entre um serviço de grupo e parâmetros arbitrários encapsulados. Este parâmetro opcional pode ser significativo só para alguns serviços. É usado para, especificamente, configurar um serviço em relação ao seu uso por este grupo. Por exemplo, um serviço de membros (Membership Service) pode achar uma lista de senhas encriptadas.

2.7.2.6. Pipes

Pipes são canais de comunicação virtuais que transmitem mensagens entre serviços ou aplicações sobre endpoints. Os pipes provêem uma abstração de rede sobre o transporte do endpoint. Os endpoints correspondem às interfaces de rede do peer, que podem ser usadas para enviar e receber dados de outro peer.

Diferentes tipos de serviços podem ser implementados por um pipe. Por exemplo:

Assíncrono unidirecional: O endpoint envia uma mensagem, nenhuma garantia de entrega existe.

Síncrona pergunta-resposta: O endpoint envia uma mensagem, e recebe uma resposta associada.

Transferência em grandes quantidades: Transfere dados binários em grandes quantidades de forma confiável.

Fluxo de dados (*streaming*): Transferência de dados com controle de fluxo eficiente.

Segurança: Fluxos de dados confiáveis e seguros.

O pipe assíncrono unidirecional é necessário para os protocolos JXTA. Podem ser implementadas outras variações de pipes para serem usados com serviços e com os seus protocolos associados.

Os pipes conectam um ou mais endpoints. A cada software de endpoint cabe enviar ou receber, como também cabe administrar filas de mensagens associadas ao pipe. Essas filas de mensagem são opcionais. Os endpoints do pipe são chamados input pipe (o receptor) e output pipe (o transmissor).

Endpoints do pipe são dinamicamente ligados a um outro endpoint em tempo de execução, através do Pipe Binding Protocol. O processo de ligação de pipes consiste em procurar e conectar dois ou mais endpoints de um pipe.

Quando uma mensagem é enviada em um output pipe, a mensagem é enviada pelo endpoint local para um ou vários endpoints (de um ou mais peers) onde o input pipe associado é localizado. O conjunto de endpoints de peers atualmente associados com o input pipe é descoberto usando o Pipe Binding Protocol.

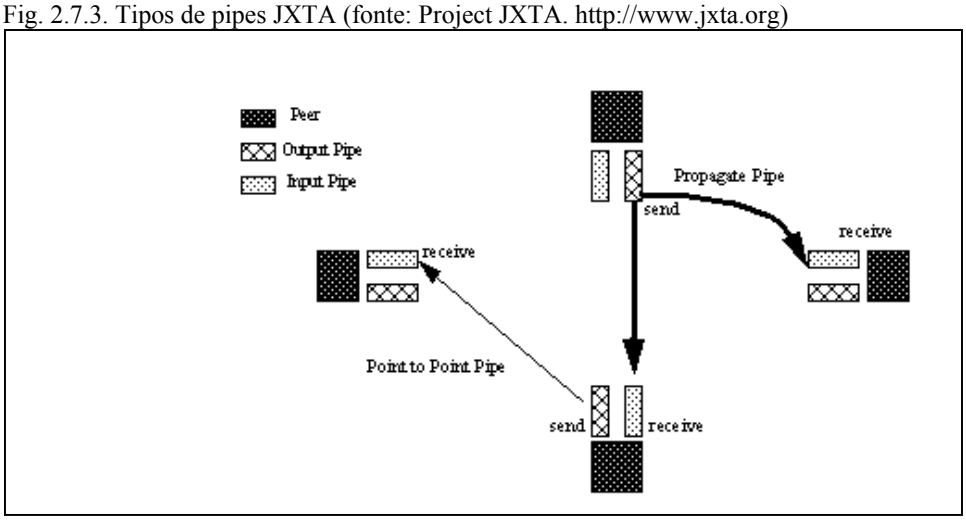
Um pipe oferece dois modos de comunicação:

Ponto-a-ponto: Um pipe ponto-a-ponto conecta exatamente dois endpoints de pipe (um input pipe que recebe mensagens enviadas por um output pipe). Nenhuma resposta ou operação de reconhecimento é suportada. Pode ser necessária a adição de informações na mensagem, como um identificador único, para enviar sucessivas mensagens. O corpo da mensagem também pode conter um anúncio de pipe, que pode ser usado para abrir um pipe para responder ao remetente (caracterizando a seqüência pergunta/resposta).

Pipe de propagação: Um pipe de propagação conecta um output pipe a múltiplos input pipes. As mensagens seguem para os input pipes do output pipe (fonte de propagação). A mensagem propagada é enviada a todos os input pipes associados. Este processo pode criar múltiplas cópias da mensagem a ser enviada. Em TCP/IP, quando o

escopo de propagação é compatível com uma sub-rede física subjacente de 1 para 1, podem ser usados IP's multicast como uma implementação para a propagação. Ela pode ser implementada usando ponto-a-ponto em transportes que não provêem multicast, como o HTTP.

Fig. 2.7.3. Tipos de pipes JXTA (fonte: Project JXTA. http://www.jxta.org)



Os pipes podem conectar dois peers que não tem um vínculo físico direto. Um ou mais peers intermediários são usados para rotear mensagens entre dois endpoints de pipes.

2.7.2.7. Mensagens

A informação transmitida usando pipes e entre endpoints é encapsulada como mensagens. Mensagens definem um envelope para transferir qualquer tipo de dados. Uma mensagem pode conter um número arbitrário de sub-seções nomeadas, que podem ter qualquer forma de dados.

A idéia é que os protocolos JXTA suportem padrões de protocolos do XML (W3C); desta forma os protocolos JXTA podem ser implementados em transportes de XML como SOAP, XML-RPC etc.

Os protocolos JXTA são especificados como um conjunto de mensagens XML trocadas entre peers. Cada plataforma de software descreve como uma mensagem é convertida para (e convertida de) estruturas de dados nativas como objetos de Java ou structures de “C”.

O uso de mensagens XML para definir protocolos permite que muitos tipos diferentes de peers participem em um protocolo. Cada peer é livre para implementar melhor o protocolo do jeito que melhor servir às suas habilidades e necessidades.

2.7.3. Os protocolos

O conjunto de protocolos JXTA é formado por seis protocolos que foram especificamente projetados para uma computação em rede P2P de múltiplos saltos. Usando estes protocolos, os peers podem cooperar para formar grupos auto-organizáveis e auto-configuráveis independente de suas posições na rede (sob firewalls, NAT's, endereçamento público de rede *versus* privado) e sem a necessidade de uma infra-estrutura centralizada.

O projeto JXTA segue a idéia de criar protocolos que tenham baixo overhead, que busquem poucas informações sobre a camada de transporte e imponham poucas características obrigatórias ao ambiente dos peers, e também podem ser usados para publicar uma grande variedade de aplicações e serviços P2P, num ambiente de rede que está sempre em constante mudança, mas que possui alta confiabilidade.

Os peers usam os protocolos JXTA para anunciar seus recursos e encontrar recursos na rede (serviços, pipes etc.) que outros peers tornaram disponíveis. Eles criam grupos e entram em grupos já existentes para criar relações especiais. Os peers cooperam entre si para rotear as mensagens, permitindo uma maior conectividade entre os peers. Estes protocolos permitem também aos peers comunicarem-se sem a necessidade de entender ou administrar a dinâmica e potencialmente complexa topologia de rede, que tende normalmente a aumentar de complexidade.

Os protocolos JXTA dão aos peers a possibilidade de, dinamicamente, rotear mensagens, através de múltiplos saltos, para qualquer destino na rede (podendo inclusive alcançar peers que estejam sob firewalls). Cada mensagem carrega consigo uma lista (ordenada ou parcialmente ordenada) de peers que servem como gateways, pelos quais a mensagem deve ser roteada. Peers intermediários neste caminho devem auxiliar o roteamento utilizando rotas conhecidas para reduzir e otimizar a rota da mensagem.

Há dois protocolos, os quais cada peer deve implementar para poder ser alcançável na rede: o Peer Resolver Protocol e o Endpoint Routing Protocol. Esses dois protocolos, mais os anúncios, os serviços e as definições formam a especificação que serve como base dos protocolos JXTA. Essa especificação estabelece a infra-estrutura básica utilizada por outros serviços e aplicações.

Cada um dos protocolos JXTA é independente dos outros. Não é necessário que um peer implemente todos os protocolos JXTA para ser um peer JXTA. Um peer só implementa os protocolos que ele necessita usar. Exemplos:

Um dispositivo deve ter todos os anúncios necessários usados por ele pré-armazenados em memória, de modo que o peer não necessita implementar o Peer Discovery Protocol.

Um peer deve usar um conjunto pré-configurado de peers roteadores para rotear todas as suas mensagens; conseqüentemente o peer não necessita implementar completamente o

Endpoint Routing Protocol. Ele somente envia as mensagens aos roteadores conhecidos para serem transmitidas.

Um peer pode não necessitar, nem desejar, fornecer informações de status para outros peers. Assim, o peer não precisaria implementar o Peer Information Protocol.

Os protocolos JXTA foram projetados para permitir que o JXTA seja facilmente implementado com links unidirecionais (nos quais apenas um dos peers envia dados) e transportes assimétricos. JXTA permite que qualquer link unidirecional seja utilizado quando necessário, melhorando a performance global e a conectividade da rede no sistema. A intenção é que os protocolos JXTA sejam tão eficientes quanto possível, e fáceis de implementar sobre qualquer transporte. Implementações em transportes bidirecionais e confiáveis, tais como TCP/IP ou HTTP, deveriam produzir eficientes comunicações bidirecionais.

O transporte JXTA unidirecional e assimétrico também funciona bem em ambientes de rede multi-saltos, onde a latência de mensagem pode ser difícil de prever. Além disso, peers em uma rede P2P tendem a ter comportamentos não-determinísticos. Eles podem entrar ou sair da rede com muita freqüência.

A seguir apresentaremos os seis protocolos JXTA, suas características, imposições, exemplos e interoperabilidade entre os protocolos da pilha do JXTA.

2.7.3.1. Peer Resolver Protocol

O Peer Resolver Protocol (PRP) permite a disseminação de consultas genéricas para um ou mais handlers dentro de um grupo de peers e a identificação de respostas. Cada consulta é enviada a um nome de handler específico. Este nome define a semântica particular da questão e de suas respostas, mas não é associado a nenhum peer específico. Uma determinada consulta pode ser recebida por qualquer número de peers no grupo, possivelmente todos, e processada de acordo com o nome do handler, se esse handler tiver o nome definido.

A intenção é que o PRP venha prover a infra-estrutura genérica de consulta/resposta essencial para construir serviços de resolução de alto nível. Em muitas situações um serviço de mais alto nível pode ter um maior conhecimento da topologia do grupo.

O PRP usa o Rendezvous Service para disseminar uma consulta para múltiplos peers ou mensagens unicast para enviar consultas a determinados peers.

Os peers também podem participar no SRDI (Shared Resource Distributed Index), ou seja, Índice Distribuído de Recursos Compartilhados. O SRDI provê um mecanismo geral onde serviços do JXTA podem utilizar um índice distribuído de recursos compartilhados com outros peers que são agrupados como um conjunto de peers mais robustos como peers rendezvous. Estes índices podem ser usados para rotear consultas na direção para onde elas

serão mais provavelmente respondidas, e mensagens que foram propagadas novamente para peers interessados nestas mensagens.

2.7.3.1.1. Mensagem de Resolução de Consulta

A mensagem de resolução de consulta é usada para enviar uma consulta ao handler nomeado em um ou mais peers que são membros do grupo. A consulta é enviada na forma de uma cadeia de caracteres a um handler específico. Cada consulta possui um identificador singular. A string da consulta pode ser qualquer string que será interpretada pelo handler escolhido.

Fig. 2.7.4. Esquema da Resolução de Consulta

```
<xs:element name="ResolverQuery" type="jxta:ResolverQuery"/>
<xs:complexType name="ResolverQuery">
  <xs:all>
    <xs:element ref="jxta:Cred" minOccurs="0"/>
    <xs:element name="SrcPeerID" type="jxta:JXTAID"/>
    <!--Isto poderia ser estendido com uma restrição de padrão-->
    <xs:element name="HandlerName" type="xs:string"/>
    <xs:element name="QueryID" type="xs:string"/>
    <xs:element name="HC" type="xs:unsignedInt"/>
    <xs:element name="Query" type="xs:anyType"/>
  </xs:all>
</xs:complexType>
```

<jxta:Cred>: A credencial do remetente.

<HandlerName>: Uma string que especifica o destino desta consulta.

<SrcPeerID>: O identificador do peer que originou a consulta (como uma URN).

<QueryID>: Um identificador ininteligível a ser usado pelo remetente para comparar respostas. O <QueryID> deveria ser incluído nas respostas a esta consulta.

<HC>: Especifica o número de saltos que a consulta teve de dar até a sua resposta. O <HC> deveria ser incrementado por cada peer que reenviasse a consulta.

<Query>: Contém a consulta.

2.7.3.1.2. Mensagem de Resolução de Resposta

Uma mensagem de resposta é usada para enviar uma resposta a uma mensagem de resolução de consulta.

Fig. 2.7.5. Esquema de Resolução de Resposta

```
<xs:element name="ResolverResponse" type="ResolverResponse"/>

<xs:complexType name="ResolverResponse">
  <xs:all>
    <xs:element ref="jxta:Cred" minOccurs="0"/>
    <xs:element name="HandlerName" type="xs:string"/>
    <xs:element name="QueryID" type="xs:string"/>
    <xs:element name="Response" type="xs:anyType"/>
  </xs:all>
</xs:complexType>
```

<jxta:Cred>: A credencial do peer que responde a consulta.

<HandlerName>: Especifica como rotear a resposta.

<QueryID>: O identificador da consulta para a qual esta é uma resposta.

<Response>: As respostas.

2.7.3.1.3. Mensagem de Resolução de SRDI

A mensagem de resolução de SRDI é enviada para um handler nomeado em um ou mais peers que são membros do peergroup. A mensagem é enviada a um handler específico. O conteúdo da mensagem pode ser qualquer um, que será interpretado pelo handler escolhido.

Fig. 2.7.6. Esquema de Resolução de SRDI

```
<xs:element name="ResolverSRDI" type="jxta:ResolverSRDI"/>

<xs:complexType name="ResolverSRDI">
  <xs:all>
    <xs:element name="HandlerName" type="xs:string"/>
    <xs:element ref="jxta:Cred" minOccurs="0"/>
    <xs:element name="Payload" type="xs:anyType"/>
  </xs:all>
</xs:complexType>
```

<HandlerName>: Uma string que especifica o destino desta mensagem.

<jxta:Cred>: A credencial do remetente.

<Payload>: Contém a carga da mensagem.

2.7.3.1.4. Políticas e Qualidade de Serviço

O PRP não garante aos peers que, uma vez definido um nome de handler de consulta, o handler receberá aquela consulta, nem obriga que todos os peers que definem este nome de handler irão recebê-la. É feito o melhor esforço para disseminar a consulta de modo que maximize as chances de se obter uma resposta, se algum peer tiver essa resposta.

Não há nenhuma garantia de que uma resposta para um pedido de consulta de resolução seja conseguida. É importante notar que a resposta para um pedido de consulta de resolução é opcional. Um peer não é obrigado a responder.

Não há também nenhuma garantia de que uma mensagem de resolução SRDI será honrada. Uma vez mais é importante mostrar que aceitar uma mensagem de resolução SRDI é opcional. Um peer não é obrigado a aceitar a mensagem.

O PRP não obriga a presença de uma entrega de mensagem confiável. Múltiplas mensagens de consulta podem ser enviadas e nenhuma, uma, múltiplas ou redundantes respostas podem ser recebidas.

O PRP provê um mecanismo genérico para que os serviços enviem consultas, recebam respostas e mensagens de SRDI. Como um serviço, a implementação de referência ajuda outros serviços, ao tomar cuidado sobre todos os aspectos de troca de mensagens, respostas armazenadas de consultas e mensagens de SRDI, e envio de consultas, baseados na decisão do remetente. O PRP executa a autenticação e a verificação de credenciais e elimina as mensagens incorretas.

A tarefa atual de propagar uma consulta para o próximo conjunto de peers é delegada ao Rendezvous Protocol. O serviço Rendezvous é responsável por determinar o conjunto de peers que devem receber uma mensagem que é propagada, mas nunca deve passar à frente, de forma automática, uma mensagem propagada que acabou de chegar.

Cabe ao serviço (o handler de consulta) determinar se a propagação deve continuar a ser executada. A política do PRP é a seguinte: se o handler de consulta não diz ao PRP que ele descarte a consulta, e se o peer local é um peer rendezvous, então a consulta é propagada novamente (dentro dos limites das regras de loop e de TTL, forçadas pelo serviço Rendezvous). Além disso, se instruída pelo handler de consulta, uma consulta idêntica pode ser emitida com o peer local como o originador da mesma.

2.7.3.2. Endpoint Routing Protocol

A rede JXTA é adaptável por natureza. As conexões na rede podem ser temporárias, o que causa o não-determinismo no roteamento das mensagens. Rotas podem ser unidirecionais e podem mudar rapidamente. Os peers podem se juntar a um grupo e freqüentemente podem sair dele. Um peer atrás de um firewall pode enviar uma mensagem diretamente para um peer fora do firewall. Mas um peer fora do firewall não pode estabelecer uma conexão diretamente com um peer atrás do firewall.

O Endpoint Routing Protocol define um conjunto de mensagens de consultas e respostas que são processadas por um serviço de roteamento para ajudar um peer a rotear mensagens até o seu destino.

Quando um peer precisa enviar uma mensagem para um determinado endereço de endpoint de um peer, ele olha em seu armazenamento local (cache) para ver se há uma rota até este peer. Se não encontra uma rota, envia uma mensagem de consulta de resolução de rotas para os peers roteadores disponíveis, pedindo informação de roteamento. Um peer pode ter tantos peers roteadores quanto possa encontrar, ou então o peer pode tê-los pré-configurados. Roteadores pré-configurados são opcionais.

Os peers roteadores provêem a infra-estrutura de baixo nível para rotear mensagens entre dois peers na rede. Alguns peers em um grupo podem se eleger para se tornarem roteadores para outros peers. Peers roteadores oferecem a habilidade de armazenar a informação de roteamento, como também atravessar diferentes redes físicas ou lógicas. Um peer pode dinamicamente encontrar seu peer roteador através de uma busca qualificada. Um peer pode descobrir se outro peer é um peer roteador verificando o elemento <Parms> do seu anúncio.

Quando um peer roteador recebe uma consulta de rota, ele verifica se conhece uma rota até o destino, em caso afirmativo, responde a consulta devolvendo a informação sobre a rota como uma lista de saltos da rede. Uma vez que uma rota foi descoberta, uma mensagem pode ser enviada ao primeiro roteador e aquele roteador usará a informação da rota para rotear a mensagem para o peer de destino. A rota é ordenada do próximo salto ao peer que é o destino final. Se, a qualquer momento, a informação sobre a rota tornar-se obsoleta, o roteador atual deve descobrir uma rota nova para completar a entrega da mensagem.

O Endpoint Routing Protocol (ERP) provê um recurso de roteamento para um peer. Roteamentos mais inteligentes podem ser implementados através de serviços de roteamento mais sofisticados, em lugar do serviço básico de roteamento. Serviços de roteamento de alto nível podem administrar e aperfeiçoar rotas de forma mais eficaz que o serviço original. O que o JXTA pretende é prover o suporte necessário para que o usuário defina serviços de roteamento para manipular e atualizar a informação da tabela de roteamento (anúncios das rotas) usada pelo peer roteador. A intenção é termos melhores serviços de análise e descoberta de rotas sendo executados sobre o serviço básico por serviços de roteamento de alto nível, e assim esses serviços de roteamento podem prover sugestões inteligentes para o peer roteador no que diz respeito ao roteamento das mensagens.

O ERP é usado para encontrar as rotas disponíveis para se enviar uma mensagem a um peer destino. Isto é feito através de trocas de mensagens entre peers roteadores. Às vezes podem ser necessários peers de roteamento para habilitar a comunicação entre dois peers, dependendo da localização deles na rede. Por exemplo, os dois peers podem estar usando transportes diferentes; podem estar separados por um *firewall*; ou podem estar usando um IP privado de espaços incompatíveis. Quando necessário, um ou mais peers roteadores

podem ser usados para entregar uma mensagem do endpoint do peer original para o endpoint do peer de destino.

2.7.3.2.1. Endereços de endpoint

Endpoints são identificados em JXTA usando URI's conhecidos como Endereços de Endpoint. Esses endereços podem descrever localizações de rede físicas e endereços virtuais para peers e grupos.

Fig. 2.7.7. Descrição do URI de endereços de endpoint no formato ABNF

```
<ENDPOINTADDRESS> ::=
   (<URISCHEME> <PROTOCOLADDR> 0*1("/" <RECIPIENT> "/"
0*1(<RECIPIENTPARAM>))) |
    (<URNSCHEME> <PROTOCOLADDR> 0*1("#" <RECIPIENT> "/"

0*1(<RECIPIENTPARAM>)))


<PROTOCOL> ::= <URISCHEME> | <URNSCHEME>
<URISCHEME> ::= 1 (<upper> | <lower>) 0* <URICHARS> <SCHEMESEP>
<SCHEMESEP> ::= ":" | "://"
```

2.7.3.2.2. Informação de Rota

A informação de rota é representada como segue:

Fig. 2.7.8. Anúncio de Rota

```
<xs:element name="APA" type="jxta:APA"/>
<xs:complexType name ="jxta:APA">
  <xs:sequence>
    <xs:element name="EA" type="jxta:JXTAID"
            minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="RA" type="jxta:RA"/>
<xs:complexType name ="jxta:RA">
  <xs:sequence>
    <xs:element name="DstPID" type="xs:anyURI"/>
    <xs:element ref="jxta:RA"/>
    <xs:element name="Hops" minOccurs="0">
      <xs:sequence>
        <xs:element ref="jxta:APA" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

<DstPID>: O identificador do peer que é descrito por este anúncio.

<APA>: Um anúncio de ponto de acesso que contém uma lista de endereços de endpoint associados com o identificador do peer especificado.

<Hops>: Uma coleção semi-ordenada de anúncios de ponto de acesso que descrevem uma rota para o peer indicado por <DstPID>.

O parâmetro time-to-live é medido em saltos e especifica por quanto tempo esta rota é válida. O criador da rota pode decidir quanto tempo esta rota será válida. Os gateways são definidos como uma seqüência ordenada de identificadores de peers que definem a rota do peer origem até o peer destino. A seqüência pode não estar completa, mas pelo menos o primeiro gateway deve estar presente. O primeiro gateway já é suficiente para iniciar o roteamento das mensagens. A sucessão de gateways restantes é opcional.

Os peers roteadores geralmente armazenam informações de rotas. Qualquer peer pode consultar um peer roteador para obter informações de rota. Qualquer peer em um grupo pode se tornar um peer roteador.

2.7.3.2.3. Mensagem de Consulta de Rota

Esta mensagem é enviada por um peer para pedir informação de rota a outro peer. São transmitidas mensagens de consulta de rota como consultas dentro da mensagem de consulta de resolução.

Fig. 2.7.9. Consulta de rota de endpoint

```
<xs:element name="ERQ" type="jxta:ERQ"/>

<xs:complexType name ="jxta:ERQ">
  <xs:sequence>
    <xs:element name="Dst" type="jxta:JXTAID"/>
    <xs:element name="Src">
      <xs:element ref="jxta:RA"/>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

<Dst>: O identificador do peer de destino.

<Src>: Anúncio de rota do peer pedindo informação de rota. Esta informação de rota é necessária para assegurar uma rota de retorno para as respostas.

2.7.3.2.4. Mensagem de Resposta de Rota

Esta mensagem é enviada pelo peer em resposta às mensagens de consulta de rota. Esta mensagem contém um anúncio de rota para o peer destino. Mensagens de resposta de rota são transmitidas como respostas dentro da mensagem de resposta de resolução.

Fig. 2.7.10. Resposta de rota de endpoint

```
<xs:element name="ERR" type="jxta:ERR"/>
<xs:complexType name ="jxta:ERR">
  <xs:sequence>
    <xs:element name="Dst">
      <xs:element ref="jxta:RA"/>
    </xs:element>
    <xs:element name="Src">
      <xs:element ref="jxta:RA"/>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

<Dst>: O anúncio de rota do peer que requisitou a consulta de rota de endpoint.

<Src>: O anúncio de rota do peer destino.

2.7.3.3. Peer Discovery Protocol

O Peer Discovery Protocol é usado para descobrir qualquer recurso publicado. Os recursos são representados através de anúncios. Um recurso pode ser um peer, um grupo de peers, um pipe, um módulo, ou qualquer recurso que tenha um anúncio. Cada recurso deve ser representado por um anúncio.

O Peer Discovery Protocol (PDP) habilita um peer para achar anúncios em seu grupo. O PDP é o protocolo usado pelo WorldPeergroup. Serviços de descoberta específicos podem escolher estender o PDP. Se um grupo de peers não precisar definir seu próprio protocolo de descoberta, ele pode usar o PDP do WorldPeergroup.

A intenção do PDP é prover uma infra-estrutura de descoberta básica para construir serviços de descoberta de mais alto nível. Em muitas situações, a informação de descoberta é mais bem reconhecida por um serviço de alto nível, porque o serviço pode ter mais conhecimento sobre a topologia do grupo de peers.

2.7.3.3.1. Discovery Query Message

A Discovery Query Message é usada por um peer para enviar uma requisição de descoberta quando estiver procurando por anúncios.

Fig. 2.7.11. Esquema da Consulta de Descoberta de Informação

```
<xs:element name = DiscoveryQuery " type = jxta:DiscoveryQuery " />
 <xsd:simpleType name = DiscoveryQueryType ">
  <xsd:restriction base = xsd:string ">
    <!--peer-->
    <xsd:enumeration value = 0 " />
    <!--grupo-->
    <xsd:enumeration value = 1 " />
    <!--anúncio-->
    <xsd:enumeration value = 2 " />
  </xsd:restriction>
</xsd:simpleType>
<xs:complexType name="DiscoveryQuery">
  <xs:sequence>
    <xs:element name="Type" type="jxta:DiscoveryQueryType"/>
    <xs:element name="Threshold" type="xs:unsignedInt" minOccurs="0"/>
    <xs:element name="Attr" type="xs:string" minOccurs="0"/>
    <xs:element name="Value" type="xs:string" minOccurs="0"/>
    <xs:element name="PeerAdv" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

<type>: Apenas anúncios do tipo pedido serão retornados como resposta. Os possíveis valores são:

0 : anúncio de peer;
1 : anúncio de grupo de peers;
2 : anúncio qualquer

<Threshold>: Especifica o número máximo de anúncios que cada peer deve prover ao dar uma resposta. O número total de resultados recebidos depende do número de peers que respondem à consulta e dos anúncios que eles têm. Se <type> é 0 (um anúncio de peer) e o <Threshold> é 0, então a consulta tem um significado especial: seu objetivo é coletar anúncios de peers que estão respondendo. Então, qualquer peer deve responder a consulta, embora nenhum resultado será incluído.

<PeerAdv>: Se existir, é o anúncio do peer que requisitou a consulta.

<Attr>, <Value>: Ambos podem estar presentes ou ausentes. Se ausentes, então cada peer que responder a consulta deve prover um conjunto aleatório de anúncios do tipo apropriado até o valor de <Threshold>.

Somente anúncios que contêm um elemento que é cujo nome é <Attr> e que também contém um valor igual a <Value> é dito como encontrado. <Value> pode começar ou terminar com " *", ou ambos. Neste caso <Value> comparará todos os valores encontrados com os que terminam ou começando com, ou contém o resto da cadeia de caracteres. Se <Value> só contém "*", o resultado pode ser qualquer um. Algumas implementações podem escolher não encontrar nenhum anúncio para <Value>="*".

2.7.3.3.2. Discovery Response Message

Uma Discovery Response Message é usada por um peer para responder a uma mensagem de consulta de descoberta de informação.

Fig. 2.7.12. Esquema da Resposta de Descoberta de Informação

```
<xs:element name="DiscoveryResponse" type="jxta:DiscoveryResponse"/>
<xs:complexType name="DiscoveryResponse">
 <xs:sequence>
  <xs:element name="Type" type="jxta:DiscoveryQueryType"/>
  <xs:element name="Count" type="xs:unsignedInt" minOccurs="0"/>
  <xs:element name="Attr" type="xs:string" minOccurs="0"/>
  <xs:element name="Value" type="xs:string" minOccurs="0"/>
  <xs:element name="PeerAdv" minOccurs="0">
   <xs:complexType>
    <xs:simpleContent>
   <xs:extension base="xs:string">
     <xs:attribute name="Expiration" type="xs:unsignedLong"/>
   </xs:extension>
   </xs:simpleContent>
   </xs:complexType>
  </xs:element>
  <xs:element name="Response" maxOccurs="unbounded">
   <xs:complexType>
    <xs:simpleContent>
     <xs:extension base="xs:string">
      <xs:attribute name="Expiration" type="xs:unsignedLong"/>
     </xs:extension>
    </xs:simpleContent>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
```

<Type>: O tipo de todos os anúncios devolvidos no(s) elemento(s) <Response>.

<Count>: Se estiver presente, o número de elemento(s) <Response> é incluído nesta resposta.

<PeerAdv>: Se existir, é o anúncio do peer que responde. O atributo Expiration é o tempo de vencimento em milissegundos.

<Attr>, <Value>: Se presente, devolve os conteúdos vindos na consulta de descoberta de informação para a qual esta é a resposta.

<Response>: Um anúncio. O atributo Expiration é o tempo de vencimento em milissegundos.

2.7.3.3.3. Políticas e Qualidade de Serviço

O PDP não garante que os peers, que receberem uma consulta responderão a ela, nem faz com que o número de anúncios que foram pedidos seja retornado. Apenas é feito o melhor esforço a fim de encontrar resultados para a consulta no peer que responderá.

Não há nenhuma garantia de que uma resposta será encontrada para um pedido de consulta de descoberta de informação. É importante salientar que responder a um pedido de consulta é opcional. Não é obrigatório aos peers responder a um pedido de consulta.

O PDP provê um mecanismo para que os serviços possam explorar a rede atrás de recursos e receber respostas. Como um serviço, a implementação da referência ajuda outros serviços por cuidar de todos os aspectos de envio de mensagens, armazenamento, e expiração de anúncios.

A tarefa real de propagar continuamente uma consulta é função do serviço de resolução.

2.7.3.4. Rendezvous Protocol

O Rendezvous Protocol (RVP) é responsável por propagar mensagens dentro de um grupo de peers. Enquanto que diferentes grupos podem possuir diferentes formas de propagar as mensagens, o Rendezvous Protocol define um protocolo simples que permite aos peers:

- Habilitar a conexão aos serviços da rede (propaga mensagens suas para outros peers e recebe mensagens propagadas de outros peers);

- Controlar a propagação da mensagem (através de TTL, detecção de loopbacks etc.)

2.7.3.4.1. Anúncios de peers rendezvous

Um anúncio de um peer rendezvous descreve como age este peer num determinado grupo. Esses anúncios podem ser publicados e também recuperados, desta forma os peers que estão buscando peers rendezvous podem encontrá-los com facilidade.

Fig. 2.7.13. Esquema de anúncios de peers rendezvous

```
<xs:element name="RdvAdvertisement" type="jxta:RdvAdvertisement"/>
<xs:complexType name="RdvAdvertisement">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
    <xs:element name="RdvGroupId" type="jxta:JXTAID"/>
    <xs:element name="RdvPeerId" type="jxta:JXTAID"/>
  </xs:sequence>
</xs:complexType>
```

<Name>: Este é um nome opcional, que pode estar associado ao peer rendezvous. Freqüentemente, ele é igual ao nome do peer.

<RdvGroupId>: Este é um elemento obrigatório que contém o JxtaID do grupo para qual o peer é um peer rendezvous.

<RdvPeerId>: Este é um elemento obrigatório que contém o JxtaID do peer rendezvous.

2.7.3.4.2. Conexão de peers

O RVP introduz a noção de peers especiais, chamados peers rendezvous, que podem ser usados para continuar a propagar as mensagens que eles receberam. Um peer pode se tornar dinamicamente um peer rendezvous e/ou pode dinamicamente conectar-se a um peer rendezvous. A conexão entre um peer comum e um peer rendezvous é feita através de uma conexão explícita, associada a uma locação.

Esta conexão é realizada enviando mensagens que usam o Endpoint Protocol. Cada RVP está escutando um endereço de endpoint com os seguintes nomes e parâmetros:

- Nome do Serviço: JxtaPropagate
- Parâmetro do Serviço: PeerGroup ID

Um conjunto de consultas e de respostas é definido pelo Rendezvous Protocol para estabelecer conexões:

LeaseRequest. Esta requisição é enviada por um peer que deseja se conectar a um peer rendezvous. Não há nenhuma indicação do quanto será locado: o peer rendezvous determinará o quanto achar apropriado. Uma locação sempre pode ser cancelada por ambas as partes, a qualquer momento, se necessário. Um peer rendezvous que concede uma locação devolve um LeaseGranted.

LeaseGranted. Esta mensagem é enviada por um peer rendezvous que concedeu uma locação a um determinado cliente. A quantidade de tempo para a qual a locação é concedida é incluída na mensagem.

LeaseCancelRequest. Esta mensagem é enviada por um cliente a seu peer rendezvous para cancelar uma locação existente. Espera-se que o peer rendezvous responda com LeaseCancelled.

OBS: O Peer Resolver Protocol não é usado para enviar essas mensagens: o Rendezvous Protocol está imediatamente acima do Endpoint Routing Protocol. A razão disto é o layout da pilha dos protocolos: o Peer Resolver Protocol está acima do Rendezvous Protocol.

2.7.3.4.3. Controle de Propagação

O Rendezvous Protocol é responsável por controlar a propagação de mensagens. Ele propagará a mensagem, a menos que uma das condições seguintes seja encontrada:

Loop: se uma mensagem propagada já foi processada em um peer, ela é descartada.

TTL: mensagens propagadas têm um tempo de vida associado a elas (TTL). Cada vez que uma mensagem propagada é recebida por um peer, seu TTL é decrementado em 1 unidade. Quando o TTL de uma mensagem chega a zero, a mensagem é descartada.

Duplicata: cada mensagem propagada é associada a um identificador único. Quando uma mensagem propagada foi duplicada, e um peer já a recebeu anteriormente, as duplicatas são descartadas.

Este controle é feito ao se embutir um elemento de mensagem dentro de cada mensagem propagada, como a que é definida:

Fig. 2.7.14. Esquema de Propagação de Mensagem de Peers Rendezvous

```
<xs:element name="RendezVousPropagateMessage"
type="jxta:RendezVousPropagateMessage"/>
<xs:complexType name="RendezVousPropagateMessage">
   <xs:element name="MessageId" type="xs:string"/>
   <xs:element name="DestSName" type="xs:string"/>
   <xs:element name="DestSParam" type="xs:string"/>
   <xs:element name="TTL" type="xs:unsignedInt"/>
   <xs:element name="Path" type="xs:anyURI" maxOccurs="unbounded"/>
```

2.7.3.5. Peer Information Protocol

Uma vez que um peer foi localizado, podem ser consultados seu estado e suas capacidades. O PIP (Peer information Protocol) provê uma coleção de mensagens para se obter uma informação sobre o estado de um peer. Este é um protocolo opcional. Os peers não obrigatoriamente precisam responder a pedidos de informação.

Um mecanismo de transporte confiável também é opcional para o PIP. Múltiplas mensagens de informação sobre peers podem ser enviadas. Podem ser recebidas: nenhuma, uma ou várias respostas para uma consulta qualquer.

O PIP é encapsulado no Peer Resolver Protocol. O elemento <QueryID> é usado para comparar consultas do PIP contendo elementos <request> com as PIP Response Messages (mensagens de respostas) que contêm as respostas encontradas.

2.7.3.5.1. Obtendo respostas do PIP

Uma PIP Query Message (mensagem de consulta) provê um campo request que pode ser usado para codificar um pedido específico. O PIP não determina o formato do campo request e deixando a cargo do implementador. Serviços de alto nível podem utilizar o campo request para oferecer novas possibilidades de consulta.

2.7.3.5.2. Mensagem de Consulta

A mensagem de consulta é enviada por um peer para verificar o estado atual de outro peer, e obter outras informações pertinentes sobre o outro peer. Uma consulta sem um campo de pedido definido devolve um conjunto padrão de informações de um peer (por exemplo: tempo no grupo, quantidade de mensagens etc.).

2.7.3.5.3. Mensagem de Resposta

A mensagem de resposta do PIP provê informação específica sobre o estado atual de um peer, como tempo conectado à rede, quantidade de mensagens enviadas e recebidas, tempo desde a última mensagem que recebeu, e tempo desde a última mensagem enviada.

2.7.3.6. Pipe Binding Protocol

O Pipe Binding Protocol (PBP) é usado por aplicações e serviços para comunicar-se com outros peers. Um pipe é um canal virtual entre dois endpoints descritos em um anúncio de pipe. Existem duas extremidades em um pipe: o Input pipe (receptor de dados) e o Output Pipe (transmissor de dados).

O Pipe Binding Protocol é encapsulado no Endpoint Protocol, e utiliza uma variedade de meios de transporte de mensagens como o JXTA HTTP Transport, o JXTA TCP/IP Transport, e o seguro JXTA TLS Transport para enviar as mensagens.

Um pipe pode ser visto como um fila de mensagens rotuladas que suporta as operações de criação, resolução, destruição, envio e recebimento de mensagens, entre outras operações. A implementação real do pipe pode diferir, mas todas as implementações usam o PBP para ligar o pipe a um endpoint.

Um transporte confiável de mensagens é opcional. Podem ser enviadas múltiplas mensagens de consulta. Pode ser recebida nenhuma, uma ou várias respostas.

2.7.3.6.1. Anúncio de pipe

Um anúncio de pipe descreve um pipe. O anúncio de pipe é usado pelo serviço de pipe para criar input e output pipes, criando um endpoint local. Cada anúncio de pipe inclui um PipeID, que é o nome canônico para o pipe.

Todo anúncio de pipe deve incluir um tipo. Existem atualmente três tipos diferentes de pipes:

JxtaUnicast: Unicast, inseguro e não confiável. Este tipo de pipe é usado para enviar mensagens um-para-um.

JxtaUnicastSecure: Unicast e seguro (usando TLS). Uma extensão do JxtaUnicast, a não ser pelo fato de que os dados usados em uma conexão virtual de TLS entre os endpoints são protegidos.

JxtaPropagate: Pipes de difusão de mensagens. Este tipo de pipe é usado para enviar muitas mensagens. Qualquer peer que habilitou um input pipe em um pipe deste tipo recebe mensagens que são enviadas sobre ele.

Um anúncio de pipe pode incluir um nome simbólico opcionalmente.

Fig. 2.7.15. Esquema de um anúncio de pipe

```
<xs:element name="PipeAdvertisment" type="jxta:PipeAdvertisment"/>
<xs:complexType name="PipeAdvertisement">
  <xs:sequence>
    <xs:element name="Id" type="jxta:JXTAID"/>
    <xs:element name="Type" type="xs:string"/>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

<Id>: Este é um elemento obrigatório que identifica o pipe. Cada pipe tem um identificador singular.

<Type>: Este é um elemento obrigatório que define o tipo do pipe. Os tipos seguintes são definidos atualmente:

JxtaUnicast: pode não chegar ao destino, pode ser entregue mais de uma vez ao mesmo destino, pode chegar em ordem diferente.

JxtaUnicastSecure: pode não chegar ao destino, pode ser entregue mais de uma vez ao mesmo destino, pode chegar em ordem diferente, mas é codificado usando TLS.

JxtaPropagate: de um para muitos pipes.

<Name>: Este é um nome opcional que pode ser associado a um pipe. O nome não precisa ser único, a menos que o nome seja obtido de um serviço de nomes centralizado, o que garantiria a singularidade de dele.

2.7.3.6.2. Mensagem do Pipe Resolver

Esta mensagem é usada pelo Pipe Resolver para achar um input pipe de endpoint para o mesmo anúncio de pipe. O mesmo esquema de mensagem é usado para a resolução e para a resposta.

3. Proposta P2P para Publicação e Descoberta de Web Services

Neste capítulo, trataremos sobre a proposta deste trabalho em questão, trataremos das idéias iniciais que nos levaram a desenvolvê-la e nossos objetivos em relação à mesma. Demonstraremos uma das possíveis implementações de nossa proposta, detalhando as dificuldades encontradas para os diferentes paradigmas que tratamos, ao decorrer da pesquisa e desenvolvimento desta proposta, e as soluções que encontramos para os mesmos.

Nossa Proposta consiste em realizar a publicação, localização e vínculo de Web Services, através de uma rede P2P, apresentando desta forma uma alternativa à proposta atual mais aceita para "Registro de Serviços", a UDDI, que se baseia em um repositório central. Tentamos desta forma, propor uma descentralização para todo o processo tornando os documentos de especificação de Web Services (WSDL), disponíveis em qualquer repositório, que venha a se tornar disponível publicamente na rede, e não envolvendo mais um repositório central, passando o controle à toda a comunidade de desenvolvedores que fica livre para buscar os melhores caminhos para esta tecnologia, baseando-se unicamente nos fatores relevantes à mesma tais como eficiência, reusabilidade e otimização dos processos envolvidos.

O conceito básico de nossa proposta é democratizar o conceito de "Registro de Serviços", fazendo com que qualquer sistema capaz de se conectar à rede P2P de "Registro de Serviços", se torne um repositório de serviços em potencial, capaz de realizar a localização e publicação de documentos WSDL e consequentemente ter acesso a todas as informações necessárias à operação de vínculo a Web Services.

3.1. Objetivos

Propor uma alternativa baseada em uma tecnologia P2P aberta para a publicação e localização de Web Services, que leve a uma maior democratização deste processo, visando favorecer os detalhes que a comunidade formada pelos desenvolvedores achar interessante.

Notamos como principal vantagem o maior dinamismo da arquitetura P2P em relação à cliente-servidor, tornando possível que se publique e/ou descubra novos Web Services na rede sem que seja necessário consultar um sistema central que, em caso de falhas, inviabiliza a consulta e/ou publicação dos serviços.

3.2. Desenvolvimento

Tivemos como principal preocupação ao buscar as tecnologias que utilizaríamos em nossa proposta e na implementação da mesma, utilizar tecnologias que fossem:

- **De domínio público**, visando fácil difusão entre a comunidade de desenvolvimento;

- **Tecnologias já existentes**, sem reinventar a "roda", facilitando a aceitação da proposta e conseqüente implementação da mesma, já que os conceitos utilizados já estão bem difundidos no meio;

- **Aberta**, fazendo com que seu aprimoramento seja constante e que se faça pela própria comunidade disposta a utilizá-la, buscando assim uma constante evolução da mesma.

- **Independente de plataforma**, promovendo a interoperabilidade, não importando em qual plataforma ou linguagem tenha sido desenvolvido o Web Service a ser disponibilizado, ou em qual plataforma ele será consumido;

- **Baixo custo de desenvolvimento**, favorecido pela constante criação de bibliotecas de funções disponibilizadas pela própria comunidade.

Pelo fato de que JXTA apresenta uma proposta de implementação dos protocolos básicos de comunicação de uma rede P2P e, como citado no capítulo anterior, ter diversas características vantajosas, como sua implementação sobre XML que lhe agrega diversas das vantagens da mesma, se mostrou como o mais indicado para ser utilizado em nossa implementação. Em relação à linguagem utilizada para desenvolvimento utilizamos Java, por já haver uma implementação de JXTA sendo desenvolvida em Java, que se encontra na sua versão 2.0, que apesar de possuir alguns problemas constatados durante o desenvolvimento que detalharemos na conclusão deste trabalho, apresentou-se estável o suficiente para ser trabalhada.

Com relação ao formato do repositório de Web Services, utilizamos o formato proposto pela UDDI.org, por ser atualmente o mais aceito se apresentando abrangente o suficiente para nossa proposta, por já ser de conhecimento público, com o qual os desenvolvedores de Web Services já estão habituados a trabalhar, e por ser baseada em XML, o que facilita sua utilização sobre a plataforma JXTA.

Durante o desenvolvimento encontramos diversas propostas de desenvolvimento de bibliotecas e projetos sobre JXTA, inclusive bibliotecas com função de diminuir a curva de aprendizado do JXTA (devido à troca de modelo de desenvolvimento de uma plataforma cliente/servidor, atualmente mais difundida, para o modelo P2P), tais como a JAL – JXTA Abstraction Layer – com a função de ser uma camada de abstração sobre a plataforma JXTA, que faz parte do projeto Easy Entry Library (EZEL) for JXTA.

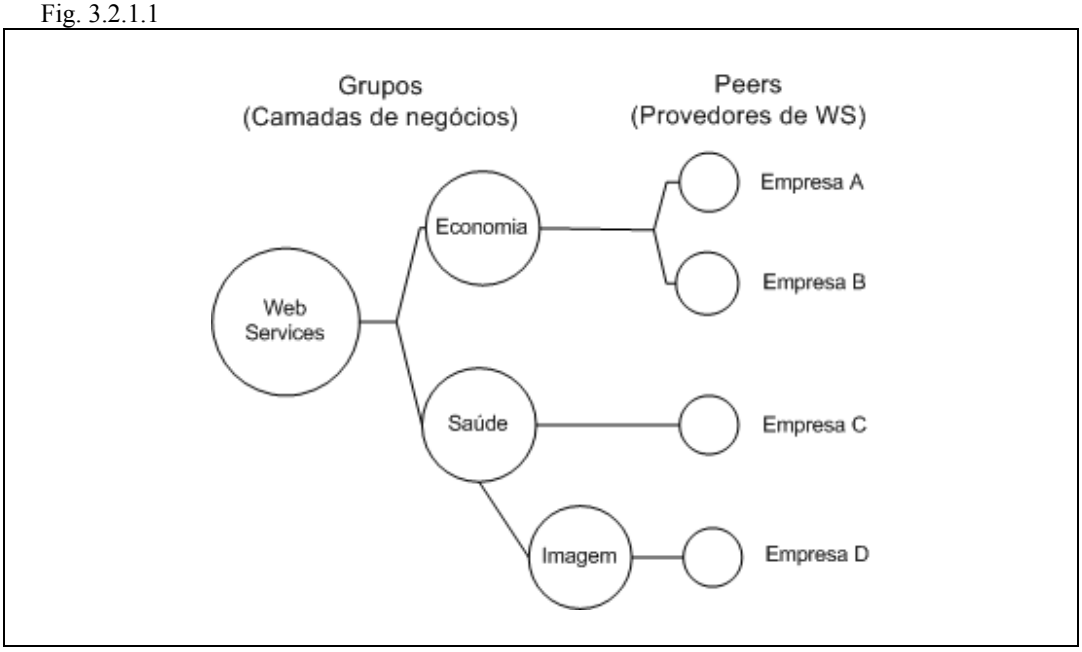
3.2.1. O sistema

A qualquer sistema que se conecte à rede P2P de "Publicação e Descoberta de Web Services" cabe implementar os métodos necessários ao funcionamento da própria rede, seguindo os padrões presentes dentro da nossa proposta, que são totalmente baseados em

padrões já existentes na proposta UDDI para publicação e descoberta de Web Services e no Projeto JXTA.

Basicamente o sistema deve ser capaz de se conectar à rede P2P utilizando os padrões propostos pelo Projeto JXTA, acessar os grupos (neste caso tratamos Grupos como PeerGroups que tem a função de agregar Peers) de interesse, grupos estes que devem ser hierarquizados representando as camadas de negócio, em que os Peers formadores da rede disponibilizam Web Services. Em nossa proposta, sobre uma abstração de alto nível, os grupos representam as camadas de negócios do mundo real, onde os peers se organizam de forma a facilitar a publicação e descoberta de Web Services e a diminuir o tráfego na rede, umas vez que o anúncio se limita ao grupo em questão (conforme Figura 3.2.1). Ao acessar a rede, nos tornamos um Peer de rede, ou seja, um ponto de presença na mesma, partilhando recursos entre os peers vizinhos em seu grupo.

Fig. 3.2.1.1



Cabe aqui ressaltar, uma vez mais, que existe uma diferenciação entre Peer e Usuário: o peer é o provedor de um determinado serviço e/ou recurso, para servir usuários ou outros peers. Mas nem todo peer tem um usuário. Um exemplo disso é o Peer RendezVous. Ele serve apenas como intermediário entre peers, fazendo com que estes troquem informações. O usuário é o operador humano associado em determinado momento a um peer. Desta forma este peer guarda a identificação do usuário, bem como armazena também permissões de uso da rede e informações de segurança do próprio usuário.

Caso estejamos tratando um Peer que represente uma empresa e/ou profissional que seja um Provedor de Web Services, para que este venha a disponibilizar seus Web Services publicamente, deve ingressar no grupo que representar a sua camada de negócio e através

de um anúncio (Advertisement) de Web Services informar aos demais Peers de seu grupo os Web Services que dispõem.

Um consumidor de serviços para buscar Web Services, deve acessar os grupos cuja camada de negócio atenda a seus interesses e buscar os anúncios com os Web Services de seu interesse, anúncios estes que seguem o padrão de repositórios de Web Services proposto pela UDDI.org, conforme figura abaixo:

Fig. 3.2.1.1

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE UDDIAdvertisement>
<UDDIAdvertisement>
  <businessServices>
    <name> Empresa A </name>
    <description xml:lang="br">Empresa A de exemplo</description>
    <discoveryURL useType=”businessEntity”>
       http://www.exemplo.com.br
    </discoveryURL>
    <businessService>
      <name> Web Service A </name>
      <description xml:lang="br">Um exemplo de Web Service</description>
      <bindingTemplates>
        <bindingTemplate>
          <accessPoint useType="endPoint"> http://www.exemplo.com.br/webserviceA/
          </accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo tModelKey="uddi:uddi.org:transport:http">
            </tModelInstanceInfo>
          </tModelInstanceDetails>
          <categoryBag>
             <keyedReference tModelKey="uddi:uddi.org:categorization:exemplos"
                             keyName="uddi_Name" keyValue="000001">
            </keyedReference>
          </categoryBag>
        </bindingTemplate>
      </bindingTemplates>
    </businessService>
  </businessServices>
</UDDIAdvertisement>
```

Na verdade, na plataforma JXTA, se descermos na camada de abstração, perceberemos que tudo é representado por arquivos XML. Um Peer fica sabendo da presença de outro Peer em seu grupo através dos anúncios que são na verdade arquivos XML; os próprios grupos são criados através do envio de arquivos XML, chamados anúncios (ou Advertisements), aos Peers RendezVous de seu grupo, que funcionam como uma espécie de “cache” para o grupo, armazenando os anúncios publicados pelo Peers presentes em seu grupo e gerenciando a propagação dos mesmos aos demais peers.

Nesta característica da plataforma JXTA é que baseamos nossa proposta. Por ser expansível, podemos modelar um novo tipo de anúncio, o “anúncio de Web Services”, que é baseado no formato XML para repositório de Web Services proposto pela UDDI.org, e utilizamos a propriedade de propagação de anúncios, característica básica da plataforma JXTA, para que este chegue ao conhecimento de todos os Peers de seu grupo. O Peer monta o anúncio baseado nos Web Services que estão disponíveis e registrados neste Peer.

Os anúncios são propagados pela rede até o Peer RendezVous, de onde podem ser descobertos por outros peers. Estes analisam o anúncio, podendo armazená-lo em cache local ou não. Uma vez analisados, eles podem ser apresentados ao usuário, de forma a poder tomar decisões quanto ao seu uso ou não.

Cada grupo criado deve ter ao menos um Peer RendezVous, que a princípio é o Peer inicial deste grupo e o responsável por sua manutenção, mas nada impede que haja mais de um Peer RendezVous por grupo. Na verdade isto apenas agiliza a propagação de mensagens dentro do grupo, encurtando a "distância" entre os Peers e diminuindo o delay na rede, que se mostrou um fator importante a ser observado.

Os Peers RendezVous também são responsáveis pelo número de "saltos" (retransmissão de informação de um Peer ao seu vizinho) que um anúncio dará antes de se extinguir, ou seja, eles determinam o alcance dos anúncios. Os peers não alcançados por estarem além do alcance determinado pelos saltos também podem utilizar os serviços disponíveis, pois os vizinhos destes peers

3.2.2. Definição estrutural

Nossa implementação foi desenvolvida em Java mas poderia ter sido desenvolvida em qualquer outra plataforma que implemente as funções básicas da plataforma JXTA. O próprio Projeto JXTA tem duas implementações disponíveis: uma em Java e outra em C. A implementação referência Utilizamos como base a implementação Java e desenvolvemos, em nossa implementação, as seguintes classes com funções bem definidas:

- Classe "**Arquivo**", responsável pelas funções de IO do sistema;

- Classe "**Janela_info**", interface utilizada para mostrar funções gerais do sistema como Advertisement de Peer, PeerGroup, RendezVous, recebidos pelo Peer, de outros Peers da rede ou geradas pelo próprio Peer;

- Classe "**Janela_principal**", interface principal do sistema fornece ao usuário todas as funcionalidades do mesmo, tais como publicação, descoberta e criação de Peer Groups e outras passando a classe "Principal" as requisições do usuário;

- Classe "**Msg_Interface**", gerencia as mensagens de erro e demais do sistema antes de chegarem ao usuário, facilitando o entendimento das mesmas;

- Classe "**Peer_p2p**", principal classe do sistema é a camada de mais baixo nível do mesmo, ligada diretamente a plataforma JXTA, tem suas funções baseadas nas funções mínimas que um nó de rede P2P (um Peer como é chamado na plataforma JXTA) deve possuir para criar novos grupos, se conectar aos mesmo, prover serviços aos demais peers vizinhos, sejam estes presentes em seu grupo ou não.

- Classe “**Principal**”, é a classe gerente do sistema, funciona como camada de abstração entre a interface do usuário e a classe “Peer_p2p”, agrupa as funções presentes na classe “Peer_p2p” em macro funções inteligíveis ao usuário.

- Classe “**Advertisement_p2p**”, classe com as funções de manipulação, do repositório de Web Services. Realiza as transformações na árvore formada pelo documento XML, como a inserção e deleção de documentos de especificação de Web Services

3.2.3. Principais métodos

Detalharemos nesta seção os principais métodos de nosso sistema sob uma abstração de alto nível para facilitar o entendimento dos mesmos.

- **Iniciar_Root:** iniciação da plataforma JXTA em si, cria uma nova instância da plataforma através dos valores pré-configurados para conexão (tais como relay, proxy e outro já usuais a sistemas que trabalhem sobre protocolo HTTP), e com a instância da plataforma inicia-se o PeerGroup Root, ou seja, o grupo padrão do JXTA, ao qual todo Peer deve se conectar para entrar na rede JXTA, pois é com a instância desse grupo que se realiza e se obtém todos as outras operações essenciais a um Peer, tais como acessar outros grupos, ou buscar os demais serviços fornecidos pela plataforma JXTA (Discovery Services, Peer Info Services, Peer RendezVous Services, MemberShip Services e demais serviços);

- **Buscar_PeerGroups**, **Buscar_Peers**, **Buscar_WebServices**, **Buscar_RendezVous**: estes métodos realizam tarefas semelhantes com apenas a diferença na abrangência. Eles são responsáveis por buscar o Discovery Service do grupo em questão para poder consultar o Peer RendezVous deste grupo e obter para o seu cache local todos os anúncios publicados pelos demais Peers. Após isto se obtém, através de uma pesquisa em seu cache local, todos os anúncios desejados, sejam eles anúncios de Peers, de Grupos, de RendezVous, ou os demais tipos como os anúncios de Web Services.

- **Criar_PeerGroup:** a tarefa de criar um grupo dá-se enviando ao Peer RendezVous o anúncio da criação deste grupo, ou seja, primeiro cria um anúncio da criação deste grupo, mas o mesmo só se efetiva ao ser enviado, se tornando portanto conhecido de todos os Peers do grupo “pai”. O tempo pelo qual um grupo exitirá será dado pelo tempo de expiração de seu anúncio e deve ser informado na sua criação.

Outro detalhe importante é com relação ao cache local de um Peer. Ao realizarmos buscas por anúncios, é sempre importante efetuarmos limpezas do cache local caso tenhamos como objetivo enxergar o estado atual da rede, ou podemos estar vendo Peers e Grupos que não mais existem.

- **Entrar_PeerGroup**, método que nos permite ingressar nos grupos que representem as camadas de negócios de nosso interesse, facilitando assim a busca pelos consumidores de serviço. Consiste basicamente em buscar o Discovery Service, para obter

o anúncio do grupo que se quer ingressar e, através deste, obter a instância do grupo em questão para que possamos através do MemberShip Service concretizar a autenticação deste Peer no grupo. A autenticação só será dada se o Peer tiver autorização para ingressar no grupo em questão.

- **Publicar_WSUDDI**, através deste disponibilizamos um repositório de Web Services aos demais Peers de nosso Grupo ou de toda rede. Para diminuir o tráfego na rede propomos que a publicação se dê apenas dentro do grupo em questão. Ele consiste do envio ao Peer RendezVous, do anúncio já previamente montado do repositório de Web Services que, seguindo o padrão UDDI, contém dados de uma empresa, especificações técnicas sobre os Web Services deste peer e dados para que se faça a vinculação aos mesmos.

3.2.4. Arquiteturas centralizadas versus P2P

A principal contribuição da tecnologia P2P no campo da computação é a mudança de paradigma que ela trouxe em relação aos modelos existentes de processamento distribuído e compartilhamento de recursos.

Pode-se afirmar também que o grande catalisador desta mudança foi a Internet. Os principais conceitos e tecnologias que hoje constituem o P2P são antigos, mas só puderam ser colocados em prática de forma efetiva com o surgimento de uma rede de dados confiável, aberta e de baixo custo.

No aspecto econômico, o P2P fomentou o surgimento de aplicações que permitiram um grau inédito de interação e intercâmbio de dados. Aliando-se isto à digitalização crescente de toda informação que nos permeia, tivemos pela primeira vez condições de distribuição de conteúdo altamente eficientes e descentralizadas, e que passam por cima de todas as formas de controle atuais.

O surgimento do Napster e a conseqüente reação da indústria fonográfica em seguida evidenciam que há um descompasso entre os modelos atuais de negócio de alguns setores e a nova tecnologia.

Mas as mudanças introduzidas parecem ser irreversíveis, e ao mesmo tempo em que algumas indústrias tentam proteger-se através de leis contra as “ameaças” do P2P, outras descobriram nela possibilidades comerciais muito interessantes.

Como o UDDI apresenta uma estrutura centralizada de distribuição de Web Services, este não se apresentou adequado à implementação em uma arquitetura P2P. Outra desvantagem do serviço UDDI é a obrigatoriedade do uso de SOAP para a comunicação com o servidor, o que limita uma vez mais seu uso. Logo, a solução encontrada foi utilizar a estrutura do projeto JXTA para a distribuição dos Web Services, pois esta apenas troca mensagens e anúncios baseados em documentos XML, padrão já adotado e consolidado no mercado de desenvolvimento de sistemas.

No geral, a arquitetura centralizada é mais indicada a ambientes restritos, cujo controle é mais simples e direto. Já a arquitetura P2P distribui melhor informações e recursos, simplificando sua manutenção e seu gerenciamento podendo, contudo, trazer uma certa dose de não-determinismo na sua estrutura.

4. Considerações finais

Consideramos que a arquitetura de redes P2P se mostrou adequada para o desenvolvimento de sistemas de publicação e descoberta de Web Services e notamos como principal vantagem o maior dinamismo da arquitetura P2P em relação à cliente-servidor, tornando possível que se publique e/ou descubra novos Web Services na rede, sem que seja necessário consultar um sistema central, que em caso de falhas, inviabiliza a consulta e/ou publicação dos serviços. Mesmo que haja um sistema de replicação ou redundância de servidores, se o problema for por exemplo uma queda no link de minha sub-rede à internet, numa arquitetura P2P, minha sub-rede continua operante, apenas aguardando o retorno da conexão para realizar o processo de sincronização de maneira transparente ao usuário, enquanto que na arquitetura cliente/servidor, como na empregada na proposta UDDI as operações em questão se tornam inviáveis.

Podemos citar ainda como pontos positivos deste modelo de desenvolvimento:

- O emprego de forma eficiente do poder computacional e da confiabilidade crescente das redes nos últimos anos;

- Simplificação do processo de publicação e descoberta, com conseqüente redução dos custos de manutenção e maior dinamismo ao mesmo, uma vez que cada vez mais empresas estão ingressando nesta camada de mercado, a de produção de sistemas baseados no modelo de Web Services o processo de colocá-los disponíveis aos possíveis consumidores, deve ser o mais ágil possível, a fim de gerar negócios.

Entretanto notamos também alguns pontos negativos, que consideramos devem ser superados com o tempo e com o crescente interesse das empresas neste modelo de desenvolvimento.

- P2P ainda é visto como tecnologia do futuro e não do presente por muitas empresas;

- Descrença em desenvolvimento sobre plataformas P2P por parte dos consumidores;

- Falta de investimento em desenvolvimento para P2P;

- Modelo cliente/servidor já esta consolidado no mercado.

5. Lista de abreviaturas

CORBA – Common Object Request Broker Architecture

DOM – Document Object Model

DTD – Document Type Definition

EZEL – Projeto Easy Entry Library

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

IPC – Inter-Process Communication

JAL – JXTA Abstraction Layer

OASIS - Organization for Advancement in Structured Information Standards

P2P – Peer-To-Peer

RPC – Remote Procedure Call

SGML – Standard Generalized Markup Language

SOAP – Simple Object Access Protocol

UDDI - Universal Description, Discovery, and Integration

W3C – World Wide Web Consortium

WSDL - Web Service Description Language

XLink – Linguagem que representa ligações entre documentos XML

XML – Extensible Markup Language

XPointer – Linguagem que usa expressões XPath como identificadores URI, para permitir a referência de elementos em documentos externos

XSL – Extensible Style Language

6. Referências bibliográficas

1. MILOJICI, D.; KALOGERAKI, V.; et al. **Peer-To-Peer Computing**. Disponível em: http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf. Acesso em: 11 abr. 2002.

2. Mullender S. **Distributed Systems**. New York: ACM Press, 1989. 458p.

3. Simon E. **Distributed Information Systems**. Berkshire, England: McGraw-Hill, 1996. 412p.

4. BENDER, M. **Desenvolvendo Sites com XML**. Florianópolis: Editora Advanced Books, 2001. 170 p.

5. **ESTUDO DE CASOS**. Disponível em: http://www.xml.com.br. Acesso em: 23 mar. 2003.

6. **O QUE HÁ DE NOVO**. Disponível em: http://www.xml.com.br. Acesso em: 11 mar. 2003.

7. **SOAP FAQ**. Disponível em: http://www.xml.com.br/index.cfm?fuseaction=VeTexto&CD_Texto=63. Acesso em: 30 mar. 2002.

8. DÉCIO, O. C. **XML Guia de Consulta Rápida**. São Paulo: Novatec Editora, 2000. 96p.

9. COVER PAGES: **XML-RPC**. Disponível em: http://xml.coverpages.org/xml-rpc.html Acesso em: 22 ago. 2002.

10. DERTOUZO, Michael. **A Revolução Inacabada.** São Paulo. Editora Futura, 2002.

11. **Easy Entry Library (EZEL) for JXTA**. Disponível em: http://ezel.jxta.org/servlets/ProjectHome Acesso em: 10 Jul. 2003.

12. **Project JXTA**. Disponível em: http://jxta.org. Acesso em: 10 Dez. 2002.

7. Anexos

7.1. Anexo 1

Tipos de dados primitivos disponíveis em um XML Schema.

xsd:string	strings em XML
xsd:boolean	valores lógico (true, false)
xsd:decimal	números decimais
xsd:float	tipo ponto flutuante precisão simples
xsd:double	tipo ponto flutuante precisão dupla
xsd:duration	uma duração de tempo
xsd:datetime	um instante de tempo específico
xsd:time	um instante de tempo que ocorre todo dia
xsd:date	uma data do calendário
xsd:gYearMonth	mês e ano gregoriano específicos
xsd:gYear	ano gregoriano específico
xsd:gMonthDay	uma data gregoriana que ocorre cada ano
xsd:gDay	um dia gregoriano que ocorre cada mês
xsd:gMonth	um mês gregoriano que ocorre todo ano
xsd:hexBinary	dado binário em base hexa
xsd:base64Binary	dado binário em base 64
xsd:anyURI	uma URI (*Uniform Resource Identifier*)
xsd:Qname	um nome XML qualificado
xsd:NOTATION	tipo de atributo NOTATION da XML 1.0

7.2. Anexo 2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

TÍTULO: UMA PROPOSTA P2P PARA PUBLICAÇÃO E DESCOBERTA DE WEB SERVICES

AUTORES: **Khaue R. Rodrigues**
Gustavo de Souza

RESUMO

Este trabalho consiste em apresentar uma proposta alternativa à proposta UDDI para a publicação e descoberta de Web Services, demonstrando sua viabilidade no que diz respeito a desempenho e demais fatores, através da implementação de um sistema baseado nesta proposta. Utilizamos para isto os protocolos propostos pelo projeto JXTA para redes P2P e as especificações da proposta UDDI para repositórios de Web Services.

ABSTRACT

This work consists of presenting a proposal alternative to the proposal UDDI for the publication and discovery of Web Services, demonstrating its viability in that says respect the performance and too much factors, through the implementation of a system based on this proposal. We use for this the protocols considered for project JXTA for nets P2P and the specifications of proposal UDDI for repositories of Web

Services.

INTRODUÇÃO

Conforme veremos neste trabalho várias tecnologias discutidas aqui já existiam a muito tempo, apenas eram apresentadas com outras simbologias, ou sobre outras óticas, mas sempre há pontos em comum entre as tecnologias utilizadas no passado para desenvolvimento de sistemas distribuídos e as emergentes atualmente, oque fez a diferença entre estas novas tecnologias tem sido a forma como elas se apresentam, buscando a simplicidade, utilização de tecnologias já existentes, e largamente adotadas mundialmente (HTTP, XML, SOAP), sem haver a "reinvenção da roda" mas adotando tecnologias robustas e independentes de plataforma.

Com certeza dentre as tecnologias discutidas neste trabalho, a que iniciou esta revolução no modelo de desenvolvimento de sistemas distribuídos que veio a culminar nos Web Services, foi a XML. A XML surgiu como um padrão para formatação de dados, e se tornou a base da comunicação dos Web Services e de praticamente todo tipo de comunicação moderna seja ela Internet ou não. Seguiram-se a ela a criação de formatos de arquivos e de documentos baseados na XML (WSDL, UDDI), assim como de esquemas de formatação e validação de seu conteúdo (DTD, XML Schema).

Este trabalho tem como objetivo apresentar uma proposta de software peer-to-peer para a publicação, a descoberta e a disseminação da tecnologia dos Web Services, além da implementação de um sistema que demonstre a viabilidade desta proposta. Nossa principal motivação

para este trabalho é oferecer uma alternativa às propostas centralizadoras do repositório central de Web Services, como no caso da UDDI.

Da mesma forma que buscamos por uma alternativa ao repositório central, também foi necessário analisar um novo modelo de arquitetura de rede que fosse mais próprio para a nossa proposta. Devido às características desejáveis das redes peer-to-peer, como a auto-organização, a independência de mecanismo de transporte de rede, a capacidade para replicar espontaneamente informações para usuários e o fomento na construção de serviços e aplicações interoperáveis, nos direcionamos para uma plataforma reconhecida por sua estrutura simples e ao mesmo tempo completa em termos de expansibilidade e versatilidade que é a plataforma proposta pelo projeto JXTA. A plataforma do Projeto JXTA provê um ambiente descentralizado que minimiza falhas em pontos isolados e é independente de qualquer serviço centralizado.

Nos próximos capítulos apresentaremos as tecnologias estudadas por nós para o desenvolvimento do nosso sistema, bem como consideraremos suas vantagens e desvantagens. Apresentaremos em seguida os passos do desenvolvimento da nossa proposta, encerrando este trabalho com as nossas conclusões finais e com a indicação de possíveis trabalhos futuros baseados neste tema.

1. Web Services

Historicamente, os desenvolvedores construíam aplicativos integrando serviços em um sistema local, este modelo garantia acesso a uma ampla gama de recursos de desenvolvimento e o controle preciso de como o aplicativo se comportaria. Este modelo, porém, já está superado, hoje os desenvolvedores constroem sistemas complexos que unificam o uso dos mais diversos aplicativos (n-tier), através de intranets ou Internet.

”Na prática, as máquinas do amanhã não serão puras máquinas de rede que adquirem suas funções on-line nem puros PCs, recheados de softwares de fábrica. Eles usarão uma mistura de recursos locais e remotos por meio de softwares nômades, fluentes, porque isso atenderá melhor às necessidades das pessoas” [DERT, 2002].

Com o aumento da complexidade dos sistemas devido à própria evolução dos recursos computacionais surgiram novos paradigmas como concorrência, e desafios característicos da interoperação entre sistemas heterogêneos tais como:

- Grande diversidade de componentes (EJB, CORBA, DCOM);
- Diversidade de linguagens: Java, C/C++, C#, ASP;
- Firewalls;
- Ausência de padrões de desenvolvimento amplamente adotados.

O desenvolvimento baseado no modelo n-tier resulta em menor tempo de acesso ao mercado, maior produtividade no .

A necessidade de superar os desafios citados, a combinação dos processos coesos e altamente produtivos do modelo n-tier com a flexibilidade e os conceitos orientados para a mensagem da Web, levaram ao surgimento deste estilo de computação chamado Web Service. Um Web Service é um aplicativo que expõe suas propriedades programaticamente dentro da Internet ou de uma intranet usando protocolos padrão, como HTTP (Hypertext Transfer Protocol) e XML. É um sistema publicado, localizado e chamado através da Internet de forma

síncrona ou assíncrona, que encapsula funções e objetos remotos via um protocolo padrão conhecido.

Apontado por muitos como o próximo estágio evolutivo da Internet (no que diz respeito ao desenvolvimento de sistemas) e consequentemente dos sistemas distribuídos, Web Services são oque há, de mais recente em desenvolvimento de sistemas distribuídos. Grandes empresas como Microsoft, IBM e Sun já possuem implementações das principais tecnologias utilizadas em Web Services como SOAP (Simple Object Access - Protocolo padrão de aplicações, baseado em XML, para o vínculo a Web Services), XML e das outras camadas constituintes dos Web Services e estão investindo fortunas nestas novas tecnologias.

O conceito fundamental é simples, os Web Services nos permitem compor as RPC's (Remote Procedure Calls) de um objeto pela Internet ou por uma rede, seguindo as especificações (WSDL - Web Service Description Language) que encontramos em repositórios (UDDI - Universal Description, Discovery, and Integration) por exemplo. Os Web Services não são a primeira tecnologia a nos permitir isto, mas são diferentes das tecnologias existentes pelo fato de usarem padrões comuns e já existentes, como HTTP, XML, SOAP. Em um Web Service ocultamos os detalhes relativos à implementação do cliente, ao qual precisa conhecer apenas a URL do serviço, os métodos disponíveis neste e os tipos de dados usados para as chamadas do método, mas não precisa saber se o serviço foi construído em Java e se está sendo executado em uma plataforma Linux, ou se é um Web Service escrito em ASP.NET, que é executado em uma plataforma Windows. Notamos neste ponto então uma das grandes vantagens dos Web Services, a Independência de plataforma.

Definição estrutural

Para demonstrarmos a estrutura de um Web Service, usaremos um modelo conceitual composto de dois elementos básicos de qualquer sistema, seja web ou win32 ou ainda linux, papéis e operações.

- Papéis, são as entidades que formam o sistema;

- Operações, são os métodos chamados por estas entidades, ou seja, as funções que as mesmas podem executar e formam a funcionalidade do sistema como um todo.

Para exemplificar melhor as operações que envolvem os Web Services podemos imaginar 3 papéis básicos.

- Provedor de serviços: É a entidade que cria o Web Service, geralmente o provedor de serviços oferece através de seu Web Service alguma funcionalidade comercial de interesse de outras empresas. O provedor de serviços para dispor seu Web Service a outras empresas possíveis interessadas, deve primeiramente descrever o Web Service em um formato padrão (WSDL tem sido o mais difundido atualmente mas não necessariamente seu Web Service tem de ser descrito neste padrão, existem diversas extensões deste padrão no mercado hoje, cada qual adaptada a um tipo de negócio), que seja compreensível por qualquer empresa que queira utilizar este Web Service. Em segundo lugar, para alcançar um grande público, o provedor de serviços deve publicar os detalhes sobre seu Web Service em um registro disponível publicamente para que chegue ao conhecimento de todos os possíveis interessados e desta forma gerar negócios.

- Consumidor de serviços: Toda empresa que utilize em seus sistemas ou em suas rotinas internas um Web Service desenvolvido por

um provedor de serviços pode ser chamada de consumidor de serviços. O consumidor de serviços pode conhecer a funcionalidade de um Web Service, a partir da descrição WSDL disponibilizada pelo provedor de serviços. Para recuperar os detalhes técnicos como métodos disponíveis e parâmetros, o consumidor de serviços realiza uma pesquisa (num repositório UDDI, por exemplo) sobre o registro onde o provedor de serviços publicou sua descrição do Web Service. O mais importante é que o consumidor de serviços pode obter, a partir da descrição do serviço, o mecanismo para vínculo com o Web Service do provedor de serviços, podendo então utilizar esse Web Service.

- Registro de serviços: Um registro de serviços é o local onde o provedor de serviços publica seus Web Services, e no qual um consumidor de serviços pode pesquisar Web Services. Os provedores de serviços normalmente publicam sua capacidade de Web Services no registro de serviço, para que dessa forma, os consumidores de serviço os encontrem e em seguida, vinculem-se ao seu Web Service. Tipicamente, informações como detalhes da empresa, Web Services por ela fornecidos e detalhes sobre cada Web Service, inclusive detalhes técnicos, são armazenadas no registro do serviço. Precisamos obter uma comunicação entre as aplicações, sem considerar o tipo de linguagem na qual a aplicação foi escrita, a plataforma onde a aplicação está sendo executada e outras considerações. Para que isso se torne possível, precisamos dos padrões de cada uma dessas três operações e de uma maneira padrão para um provedor de serviços descrever seu Web Service, sem considerar a linguagem em que ele foi escrito e é neste ponto que entra a WSDL como padrão para descrição de Web Services mais amplamente divulgado atualmente, e a UDDI como padrão para publicação e localização de Web Services nos registros de serviços.

2. UDDI

A UDDI é uma iniciativa em desenvolvimento no âmbito do consórcio industrial UDDI (http://www.uddi.org/), promovido originalmente pela IBM, Microsoft e Arriba, Atualmente se encontra na sua versão 3.0, cuja principal inovação foi o suporte a assinaturas digitais.

UDDI é atualmente a proposta mais aceita de protocolo padrão para publicar e/ou localizar Web Services, UDDI, permite que os provedores de serviços publiquem detalhes sobre suas empresas e os Web Services fornecidos pela mesma, em um registro central disponível publicamente. Esta é a parte relativa à “descrição” (Description) do UDDI. O protocolo UDDI Também fornece um padrão para permitir que os consumidores de serviços localizem provedores de serviços e detalhes sobre seus Web Services, esta é a parte relativa à “descoberta” (Discovery) do UDDI.

A partir de um documento WSDL, um consumidor de serviços pode determinar os detalhes do Web Service como as diferentes operações, tipos de dados, extremidades (vinculação), protocolos de vínculo (SOAP) e outros. Entretanto, existem outros fatores a serem considerados. Vamos examinar esses fatores, sob o ponto de vista tanto do provedor de serviços quanto do consumidor de serviços.

Primeiramente, em vez de publicar o documento WSDL para cada possível cliente, um provedor de serviços estará bem servido se publicar as informações sobre seu Web Service em um registro de forma que este esteja disponível publicamente para os consumidores de serviço interessados. Além de publicar apenas a descrição de seu Web Service, poderá ser uma boa idéia publicar também informações relacionadas ao negócio, como o nome dos seus negócios, os diferentes Web Services por eles oferecidos e outras informações relevantes.

Da mesma maneira, os consumidores de serviço podem querer encontrar os diferentes Web Services que sejam disponibilizados pelos provedores de serviços. Eles podem querer avaliá-los independentemente, antes de integrar esses Web Services às suas aplicações. Conseqüentemente, faz mais sentido, para eles, pesquisar em um "ambiente virtual" (uma rede P2P como demonstraremos em nossa proposta ou um repositório central como a proposta clássica da UDDI.org), onde os provedores de serviço já publicaram seus detalhes. Os consumidores de serviço também podem realizar uma pesquisa de nível mais amplo, com base em categorias comerciais, para encontrar as empresas nessas categorias e, a seguir, fazer o drill down em detalhes adicionais sobre os Web Services por elas oferecidos.

Existem muitas possibilidades além das apresentadas acima. Entretanto, o mais importante é a necessidade de um ambiente disponível publicamente, onde os provedores de serviços e os consumidores de serviços trabalham em conjunto para publicar e recuperar as informações apropriadas. Em outras palavras, precisamos de uma especificação para a publicação e descoberta de informações comerciais.

Definição estrutural

Um diretório UDDI é um arquivo XML que descreve o negócio e os serviços. A forma como os serviços são definidos no documento UDDI é chamado Type Model ou tModel. Em muitos casos, o tModel contém o arquivo WSDL que descreve a interface SOAP do Web service, mas o tModel é flexível o suficiente para descrever quase todo tipo de serviço.

3. P2P

O termo Peer-To-Peer (P2P) refere-se a uma classe de sistemas e aplicações que empregam recursos distribuídos para realizar tarefas críticas de forma descentralizada. Com o surgimento de técnicas de criptografia que tornam as redes mais confiáveis e tecnologias que as têm tornado mais estáveis, e a isso acrescido o fato do número de PC's (computadores pessoais) aumentar exponencialmente nos últimos anos, a tecnologia P2P está recebendo cada vez mais uma maior atenção por parte da comunidade científica, empresas desenvolvedoras de softwares e até de instituições militares que enxergam nesta nova tecnologia uma poderosa ferramenta para processamento de toda espécie, ou seja, alto poder de processamento (compartilhamento de recursos) a baixo custo.

Há diversas definições para P2P, dentre as mais adotadas pela comunidade científica:

- Intel P2P Working Group: "o compartilhamento de recursos e serviços computacionais através da troca direta entre vários sistemas";

- Kindberg [1]: "aqueles de expectativa de vida independente";

- Clay Shirky [1]: "P2P é uma classe de aplicações que tira vantagens dos recursos disponíveis nos nós da rede (armazenamento, conteúdo, hardware, software). Por acessar esses recursos de forma descentralizada, esta tecnologia precisa saber lidar com um ambiente de conectividade instável e endereços IP imprevisíveis, o que significa que necessita operar fora do sistema DNS e ter significativa senão total autonomia sobre os seus recursos".

O conceito adotado neste trabalho relaciona a tecnologia Peer-To-Peer ao termo Compartilhamento, ou seja, o modelo P2P permite aos nós oferecer e obter recursos dos peers pertencentes à topologia.

A tecnologia P2P propõe a interação direta entre os usuários da Internet e outras redes de diversos fins para um efetivo compartilhamento dos recursos geograficamente distribuídos. Esses recursos podem ser: capacidade de processamento, capacidade de armazenamento, banda de rede, entre outros.

Conceitualmente, computação P2P é uma alternativa ao modelo cliente/servidor, onde há tipicamente um único e pequeno cluster de servidores e muitos clientes, ou seja, o modelo P2P não possui o conceito de servidor: qualquer peer da rede pode exercer o papel tanto de cliente como de servidor, dependendo do contexto. Um peer é autônomo quando não é completamente controlado por outros ou pelo mesmo usuário. A dependência existente entre os peers diz respeito apenas à forma de obtenção/oferecimento de recursos, informações; encaminhamento de requisições, etc. Como conseqüência dessa autonomia, é típica na arquitetura a existência de algoritmos de controle de replicação e de escalabilidade dos serviços a serem oferecidos, de forma a garantir a não eliminação da disponibilidade de um serviço na ocorrência de falha em alguns pontos.

Vantagens

A arquitetura P2P propicia, através da agregação dos recursos disponíveis na rede:

- Interoperabilidade de baixo custo;

- Custo menor de compartilhamento dos recursos por utilizar a infra-estrutura de rede existente, e pela conseqüente distribuição dos custos de manutenção;

- Privacidade e anonimato - uma vez que incorpora esses requisitos em sua arquitetura e nos seus algoritmos, o P2P acaba por permitir que os peers tenham um controle autônomo sobre seus dados e recursos.

Aplicações

Podemos imaginar infinitas aplicações para esta tecnologia desde processamento distribuído, compartilhamento de mídias, comunicação, a sistemas colaborativos.

4. JXTA

O Projeto JXTA iniciou como um projeto de pesquisa incubado na Sun Microsystems. Hoje é um projeto aberto e independente desta empresa. Seu objetivo é explorar a visão da computação distribuída usando uma topologia P2P, e desenvolver blocos e serviços básicos que habilitariam aplicações novas para grupos de peers.

Os blocos de construção são a essência para o desenvolvimento de uma gama de aplicações e serviços distribuídos. JXTA especifica um conjunto de protocolos em vez de uma API. Desta forma ele pode ser implementado independentemente de sistema operacional ou linguagem de programação. Esta plataforma provê um ambiente descentralizado que minimiza falhas em pontos isolados e é independente de qualquer serviço centralizado. Apesar disso pode se desenvolver tanto serviços centralizados como descentralizados.

Conceitos Iniciais: Peers e Grupos

Peer: é qualquer dispositivo de rede (sensor, telefone, PDA, PC

etc.) que implementa os protocolos centrais do JXTA. Cada peer é identificado por um ID único. O peer é autônomo e opera independentemente e assincronamente em relação aos outros peers. Freqüentemente um peer estará sob o controle de um operador humano, que seria a figura do usuário. O peer irá interagir com a rede no sentido de alcançar e servir o usuário. Mas nem todo peer tem um usuário. Muitos peers existem apenas para prover serviços e recursos para a rede, mas não são associados a qualquer usuário. Exemplos disso incluem dispositivos como sensores e impressoras, mas também serviços como bancos de dados.

Grupos de Peers: Os peers se auto-organizam em grupos de peers (peergroups). Um grupo de peers é uma coleção de peers que tem interesses em comum. Cada grupo de peers é identificado exclusivamente por um PeerGroupId, que é único. Os protocolos do JXTA não definem quando, onde, ou por que são criados os grupos. Eles só descrevem como um peer pode publicar, descobrir, tornar-se membro e monitorar grupos.

Protocolos

São seis os protocolos JXTA; formam um conjunto que foi especificamente projetado para a computação em rede P2P de múltiplos saltos. Usando estes protocolos, os peers podem cooperar entre si para criar grupos auto-organizáveis e autoconfiguráveis independentes de suas posições na rede (sob firewalls, NAT etc.) e sem a necessidade de uma infra-estrutura centralizada.

Os seis protocolos JXTA trabalham juntos para permitir a descoberta, a organização, a monitoração e a comunicação entre peers. São eles:

- Peer Resolver Protocol (PRP): é o mecanismo pelo qual um peer pode enviar uma consulta a um ou mais peers, e receber uma resposta (ou várias respostas) a esta consulta. O PRP implementa um protocolo do tipo pergunta/resposta. A resposta é associada à consulta usando um identificador único incluído no corpo da mensagem. As consultas podem ser enviadas a todos os membros do grupo ou só a peers específicos.

- Peer Discovery Protocol (PDP): é o protocolo que permite ao peer anunciar seus recursos e descobrir os recursos de outros peers (por exemplo: grupos de peers, serviços, etc.). Cada recurso de um peer é descrito e publicado usando um anúncio. Estes anúncios são estruturas de meta-dados escritas em uma linguagem neutra e, neste caso, XML foi a linguagem escolhida.

- Peer Information Protocol (PIP): um peer pode obter informação sobre o estado de outros peers (como tráfego, capacidades, quantidade de peers conectados etc.) utilizando este mecanismo.

- Pipe Binding Protocol (PBP): é o protocolo com o qual um peer pode estabelecer um canal de comunicação (ou pipe) entre um ou mais peers. O PBP é usado para ligar dois ou mais pontos de conexão (pipe endpoints). Pipes provêem o mecanismo fundamental de comunicação entre os peers.

- Endpoint Routing Protocol (ERP): é o modo que o peer tem para descobrir uma rota (seqüência de saltos) usada para enviar uma mensagem para outro peer. Se um peer “A” quer enviar uma mensagem para o peer “C”, e não é conhecida uma rota que ligue diretamente “A” a “C”, então o peer “A” precisa encontrar peer(s) intermediário(s) que irão rotear a mensagem para “C”. O ERP é utilizado para determinar informações sobre a rota. Quando a topologia de rede for alterada de

modo que a rota para “C” não possa ser mais usada (por causa de uma ligação no meio do caminho que não funciona), o peer pode usar o ERP para encontrar alguma rota conhecida por outros peers para formar uma nova rota para “C”.

• Rendezvous Protocol (RVP): é o mecanismo pelo qual peers podem assinar ou cancelar um serviço. Dentro de um grupo, os peers podem ser peers rendezvous, ou peers que estão esperando por peers rendezvous. O Rendezvous Protocol permite a um peer mandar mensagens para todos que estão esperando instâncias do serviço. O RVP é usado pelo Peer Resolver Protocol e pelo Pipe Binding Protocol, em ordem, para a propagação de mensagens.

5. Proposta P2P para Publicação e Descoberta de Web Services

Aqui apresentaremos a proposta deste trabalho, as idéias iniciais que nos levaram a desenvolvê-la e nossos objetivos com relação à mesma. Demonstraremos uma das possíveis implementações de nossa proposta, detalhando as dificuldades encontradas para os diferentes paradigmas que tratamos, ao decorrer da pesquisa e desenvolvimento desta proposta, e as soluções que encontramos para os mesmos.

Nossa proposta consiste em realizar a publicação, localização e vínculo de Web Services, através de uma rede P2P, apresentando desta forma uma alternativa à proposta atual mais aceita para “Registro de Serviços”, a UDDI, que se baseia em um repositório central. Tentamos desta forma, propor uma descentralização para todo o processo tornando os documentos de especificação de Web Services (WSDL), disponíveis em qualquer repositório, que venha a se tornar disponível publicamente na rede, e não envolvendo mais um repositório central, passando o controle a toda a comunidade de desenvolvedores que fica livre para buscar os

melhores caminhos para esta tecnologia, baseando-se unicamente nos fatores relevantes à mesma tais como eficiência, reusabilidade e otimização dos processos envolvidos.

O conceito básico de nossa proposta é democratizar o conceito de “Registro de Serviços”, fazendo com que qualquer sistema capaz de se conectar a rede P2P de “Registro de Serviços”, se torne um repositório de serviços em potencial, capaz de realizar a localização e publicação de documentos WSDL e conseqüentemente ter acesso a todas as informações necessárias à operação de vínculo a Web Services.

Como objetivo, neste trabalho propomos uma alternativa baseada em uma tecnologia P2P aberta para a publicação e localização de Web Services, que leve a uma maior democratização deste processo, visando favorecer os detalhes que a comunidade formada pelos desenvolvedores achar interessante.

Por JXTA apresentar uma proposta de implementação dos protocolos básicos de comunicação de uma rede P2P com diversas características vantajosas, e como XML é a linguagem na qual é baseada sua troca de mensagens, agregando diversas das vantagens da mesma, se mostrou como o mais indicado para ser utilizado em nossa implementação. Utilizamos Java por já haver uma implementação de JXTA desenvolvida nesta linguagem, que se encontra na sua versão 2.0, que apesar de possuir alguns problemas constatados durante o desenvolvimento, apresentou-se estável o suficiente para ser trabalhada.

Com relação ao formato do repositório de Web Services utilizamos o formato proposto pela UDDI.org, por ser atualmente o mais aceito, ser suficiente para nossa proposta e por ser de conhecimento público.

A qualquer sistema que se conecte à rede P2P de “Publicação e Descoberta de Web Services”, alvo do nosso trabalho, cabe implementar os métodos necessários ao funcionamento da própria rede, seguindo os padrões propostos pelo Projeto JXTA, para acessar os grupos (neste caso tratamos grupos como agregadores de peers) de seu interesse, grupos estes que devem ser hierarquizados representando as camadas de negócio, em que os peers formadores da rede disponibilizam Web Services.

Em nossa proposta, através de uma abstração de alto nível, os grupos representam as camadas de negócios do mundo real, onde os peers se organizam de forma a facilitar a publicação e descoberta de Web Services e a diminuir o tráfego na rede, uma vez que o anúncio se limita ao grupo em questão. Ao acessar a rede, nos tornamos um peer de rede, ou seja, um ponto de presença na mesma, partilhando recursos entre os peers vizinhos em seu grupo.

Na plataforma JXTA a troca de informações é feita manipulando-se arquivos XML. Por exemplo, um peer fica sabendo da presença de outro peer em seu grupo através dos anúncios que são na verdade arquivos XML. É nessa característica que baseamos nossa proposta. Por ser expansível, podemos modelar um novo tipo de anúncio, o "anúncio de Web Services", que é baseado no formato XML para repositório de Web Services proposto pela UDDI.org, e utilizamos a propriedade de propagação de anúncios, característica básica da plataforma JXTA, para que este chegue ao conhecimento de todos os peers de seu grupo. O peer monta estes anúncios baseado nos Web Services que estão disponíveis e registrados no próprio peer. Estes anúncios são propagados pela rede até os outros peers. Estes analisam o anúncio, podendo armazená-lo em cache local ou não. Uma vez analisados, eles podem ser apresentados ao usuário, de forma a poder tomar decisões quanto ao seu uso ou não.

6. Considerações finais

Consideramos que a arquitetura de redes P2P se mostrou adequada para o desenvolvimento de sistemas de publicação e descoberta de Web Services e notamos como principal vantagem o maior dinamismo da arquitetura P2P em relação à cliente-servidor, tornando possível que se publique e/ou descubra novos Web Services na rede, sem que seja necessário consultar um sistema central, que em caso de falhas, inviabiliza a consulta e/ou publicação dos serviços, como na proposta UDDI que, mesmo que haja um sistema de replicação ou redundância de servidores, ainda pode falhar caso haja uma queda no link da sub-rede local à Internet.

Em nossa implementação, houve uma simplificação do processo de publicação e descoberta, com conseqüente redução dos custos de manutenção e maior dinamismo ao mesmo, uma vez que cada vez mais empresas estão ingressando nesta fatia de mercado – a de produção de sistemas baseados no modelo de Web Services. O processo de colocá-los disponíveis aos possíveis consumidores, deve ser o mais ágil possível, a fim de gerar cada vez mais negócios.

Apesar disto, percebemos também alguns pontos negativos que consideramos superáveis com o passar do tempo e com o crescente interesse das empresas neste modelo de desenvolvimento. Citamos, para ilustrar estes pontos, o fato de que a arquitetura P2P ainda é vista como tecnologia do futuro e não do presente por muitas empresas. A descrença no desenvolvimento sobre a plataforma leva à redução nos investimentos por parte das corporações mais influentes no mercado de softwares. Estas se baseiam numa arquitetura consolidada e conhecida por sua estrutura simples e objetiva (o modelo cliente/servidor) para apresentar projetos a

seus clientes, não buscando em novas tecnologias como o P2P um possível fator vantajoso para seus produtos.

Entretanto, acreditamos nos novos ramos de tecnologia, como as redes wireless, para incentivar as indústrias de software a desenvolverem aplicações distribuídas que utilizem melhor as crescentes capacidades das máquinas, no que diz respeito a armazenamento e processamento.

7. Referências bibliográficas

MILOJICI, D.; KALOGERAKI, V.; et al. Peer-To-Peer Computing. Disponível em: http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf. Acesso em: 11 abr. 2002.

WILSON, B. J. JXTA. Indiana, EUA: NewRiders, 2002. 527p.

Mullender S. Distributed Systems. New York: ACM Press, 1989. 458p.

Simon E. Distributed Information Systems. Berkshire, England: McGraw-Hill, 1996. 412p.

DERTOUZO, Michael. A Revolução Inacabada. São Paulo. Editora Futura, 2002.

Easy Entry Library (EZEL) for JXTA. Disponível em: http://ezel.jxta.org/servlets/ ProjectHome. Acesso em: 10 Jul. 2003.

Project JXTA. Disponível em: http://www.jxta.org. Acesso em: 10 Dez. 2002.

7.3. Anexo 3

Código fonte

```
package tcc;

import net.jxta.document.Advertisement;
import net.jxta.document.Document;
import net.jxta.document.MimeMediaType;
import net.jxta.id.ID;
import net.jxta.protocol.PipeAdvertisement;

public abstract class UDDIAdvertisement extends Advertisement
{
   /**
   * O elemento raiz do documento XML.
   */
   private static final String tipoAdv = "UDDIAdvertisement";

   /**
   * Um nome para identificar o peer.
   */
   private String nome = null;

   /**
   * O Peer ID deste peer na rede P2P.
   */
   private String peerID = null;

   /**
   * Retorna o tipo de advertisement para o documento do advertisement.
   *
   * @return o tipo de advertisement em String.
   */
   public static String getTipoAdv()
   {
      return tipoAdv;
   }

   /**
   * Retorna um identificador único para este documento.
   * Como não existe nenhum para este tipo de advertisement, então
   * este método retorna um ID nulo.
   *
   * O null ID diz ao Cache Manager para usar um hash do advertisement
```

```
* para servir de índice no cache; isto é suficiente para este propósito.
*
* @return o null ID.
*/
public ID getID()
{
   return ID.nullID;
}

/**
* Retorna o nome do usuário descrito no advertisement.
*
* @return o nome do usuário.
*/
public String getNome()
{
   return nome;
}

/**
* Retorna o Peer ID do usuário descrito no advertisement.
*
* @return o Peer ID do usuário.
*/
public String getPeerID()
{
   return peerID;
}

/**
* Coloca o nome do usuário dentro do advertisement
*
* @param nome o nome do usuário.
*/
public void setNome(String nome)
{
   this.nome = nome;
}

/**
* Coloca o Peer ID identificando a localização do peer
* na rede P2P.
*
* @param peerID o Peer ID do advertisement.
*/
public void setPeerID(String peerID)
```

```
    {
        this.peerID = peerID;
    }
}

package tcc;

import java.io.InputStream;
import java.io.IOException;
import java.io.StringWriter;
import java.util.Enumeration;
import net.jxta.document.Advertisement;
import net.jxta.document.AdvertisementFactory;
import net.jxta.document.Element;
import net.jxta.document.Document;
import net.jxta.document.MimeMediaType;
import net.jxta.document.StructuredDocument;
import net.jxta.document.StructuredDocumentFactory;
import net.jxta.document.StructuredDocumentUtils;
import net.jxta.document.StructuredTextDocument;
import net.jxta.document.TextElement;
import net.jxta.protocol.PipeAdvertisement;

import tcc.UDDIAdvertisement;

public class UDDIAdv extends UDDIAdvertisement
{

    private String tipoMime = "text/xml";
    private String tagNome = "Nome";
    private String tagPeerID = "PeerID";

    /* ..... */

    /**
    * Um Instantiator usado pela AdvertisementFactory para instanciar
    * esta classe em uma classe abstrata.
    */
    public static class Instantiator
            implements AdvertisementFactory.Instantiator
    {

        public String getAdvertisementType()
        {
            return UDDIAdvertisement.getAdvertisementType();
        }
```

```
    //public Advertisement newInstance()
    public UDDIAdvertisement newInstance()
    {
        return new UDDIAdv();
    }

    /**
    * Instancia um novo UDDIAdvertisement a partir de um dado elemento raiz.
    *
    * @param raiz a a raiz da árvore do objeto onde será
    * colocado o advertisement.
    * @return uma nova instância do UDDIAdvertisement.
    */
    //public Advertisement newInstance(Element root)
    public UDDIAdvertisement newInstance(Element root)
    {
        return new UDDIAdv(root);
    }
};

/* ..... */

/**
* Cria um novo UDDIAdvertisement.
*/
public UDDIAdv()
{
    super();
}

/* ..... */

/**
* Cria um novo UDDIAdvertisement analisando o objeto stream.
*
* @param stream o InputStream dos dados do advertisement.
* @exception IOException se o advertisement não puder ser
* analisado a partir do stream.
*/
public UDDIAdv(InputStream stream) throws IOException
{
    super();
    StructuredTextDocument doc = (StructuredTextDocument)
        StructuredDocumentFactory.newStructuredDocument(
            new MimeMediaType(tipoMime), stream);
```

```
   readAdvertisement(doc);
}

/* ..... */

/**
* Cria um novo UDDIAdvertisement analisando o objeto document.
*
* @param document o Element que contém os dados do UDDIAdvertisement.
*/
public UDDIAdv(Element document) throws IllegalArgumentException
{
   super();
   readAdvertisement((TextElement) document);
}

/* ..... */

/**
* Retorna um objeto Document contendo
* a representação em árvore do UDDIAdvertisement.
*
* @param oTipoMime o MIME type para o UDDIAdvertisement.
* @return um objeto Document contendo a representação
* em árvore do UDDIAdvertisement.
* @exception IllegalArgumentException se Peer ID é nulo.
*/
public Document getDocument(MimeMediaType oTipoMime)
        throws IllegalArgumentException
{
   // Verifica se todos os elementos necessários estão presentes.
   if (null != getPeerID())
   {
      PipeAdvertisement pipeAdv = null;
      StructuredDocument doc = (StructuredTextDocument)
         StructuredDocumentFactory.newStructuredDocument(
            oTipoMime, getAdvertisementType());
      Element elemento;
      // Adiciona o Peer ID.
      elemento = doc.createElement(tagPeerID, getPeerID());
      doc.appendChild(elemento);
      // Adiciona o nome associado, caso exista.
      if (null != getNome())
      {
         elemento = doc.createElement(tagNome, getNome());
         doc.appendChild(elemento);
```

```
        }
        return doc;
    }
    else
    {
        throw new IllegalArgumentException("Falta peer ID!");
    }
}

/* ..................................................................... */

/**
* Lê o documento em forma de árvore do UDDIAdvertisement.
*
* @param doc o objeto que contém as informações do UDDIAdvertisement.
* @exception IllegalArgumentException se o documento
* não é um UDDIAdvertisement como é esperado.
*/
public void readAdvertisement(TextElement doc)
            throws IllegalArgumentException
{
    if (doc.getName().equals(getAdvertisementType()))
    {
        Enumeration lista = doc.getChildren();
        while (lista.hasMoreElements())
        {
            TextElement elemento = (TextElement) lista.nextElement();
            // Verifica a presença do elemento nome.
            if (elemento.getName().equals(tagNome))
            {
                this.setNome(elemento.getTextValue());
                continue;
            }
            // Verifica a presença do elemento Peer ID.
            if (elemento.getName().equals(tagPeerID))
            {
                this.setPeerID(elemento.getTextValue());
                continue;
            }
        }
    }
    else
    {
        throw new IllegalArgumentException("Nao eh um UDDIAdvertisement!");
    }
}
```

```
/* ..... */

/**
* Retorna uma representação String XML do advertisement.
*
* @return a representação String XML do advertisement.
*/
public String toString()
{
   try
   {
      StringWriter out = new StringWriter();
      StructuredTextDocument doc = (StructuredTextDocument)
      getDocument(new MimeMediaType(tipoMime));
      doc.sendToWriter(out);
      return out.toString();
   }
   catch (Exception e)
   {
      return "";
   }
}

/* ..... */

/**
* Abre um arquivo com o conteúdo indicado
*
* @param nomeArquivo o nome do arquivo.
* @param conteudo o conteúdo do arquivo.
*/
public String abrir(String nomeArquivo)
{
   String strArquivo = "";
   try
   {
      FileReader leitor = new FileReader(nomeArquivo);
      BufferedReader entrada = new BufferedReader(leitor);
      String linha;
      while ((linha = entrada.readLine()) != null)
      {
         strArquivo = strArquivo + "\n" + linha;
      }
      entrada.close();
      //System.out.println(strArquivo);
```

```
    }
    catch (IOException e)
    {
       //Erro ao abrir o arquivo
       System.out.println("Erro ao abrir o arquivo");
    }
    return strArquivo;
}

/* ..................................................................... */

/**
* Salva um arquivo com o conteúdo indicado
*
* @param nomeArquivo o nome do arquivo.
* @param conteudo o conteúdo do arquivo.
*/
public void salvar(String nomeArquivo, String conteudo)
{
    try {
       FileWriter store = new FileWriter(nomeArquivo);
       store.write(conteudo);
       store.flush();
       store.close();
       System.out.println("Salvo o arquivo " + nomeArquivo);
    }
    catch (IOException ex)
    {
       //Erro ao salvar o arquivo
       System.out.println("Erro ao salvar o arquivo");
    }
}

/* ..................................................................... */

/**
* Monta um UDDIAdvertisement como os parâmetros passados.
*
* @return StructuredTextDocument a representação do UDDIAdvertisement
* de forma estruturada.
*/
public void montaUDDIAdvertisement(String documentation)
{
    try
    {
       StringWriter out = new StringWriter();
```

```
            StructuredTextDocument stDoc = (StructuredTextDocument)
               retornaDocument(new MimeMediaType("text/xml"));
            stDoc.sendToWriter(out);
         }
         catch (Exception e)
         {
            System.out.println("Não foi possível montar o UDDIAdvertisement.");
         }
      }

      /* ..................................................................... */

      /**
      * Retorna um objeto Document representando a resposta.
      *
      * @param asMimeType o tipo MIME da resposta.
      * @return o objeto Document, representando a resposta no tipo MIME.
      * @exception Exception se o documento não puder ser criado.
      */
       public Document retornaDocument(MimeMediaType asMimeType) throws
Exception
      {
         Element element;
         StructuredDocument document = (StructuredTextDocument)
         StructuredDocumentFactory.newStructuredDocument(
         asMimeType, "UDDIAdvertisement");
         element = document.createElement("raiz", "este é o raiz");
         document.appendChild(element);
         return document;
      }

      /* ..................................................................... */

      /**
      * Insere um atributo em um elemento
      *
      * @param elemento um objeto tipo TextElement.
      * @param chave o nome do atributo.
      * @param valor o valor do atributo.
      */
       public static boolean insereAtributo( TextElement elemento, String chave, String
valor )
      {
         // cria atributo
         Attribute atributo;
         atributo = new Attribute( chave, valor );
```

```
   // insere o atributo no elemento
   LiteXMLElement novo = (LiteXMLElement) elemento;
   novo.addAttribute(atributo);
   return true;
}

/* ..................................................................... */

/**
* Cria um novo elemento em um elemento raiz,
* adicionando um novo valor ao elemento
*
* @param raiz o documento onde deve ser inserido.
* @param pai o elemento pai do novo elemento.
* @param nome o nome do novo elemento.
* @param valor o valor do atributo.
*/
public static boolean criaElemento( StructuredTextDocument raiz,
                                    TextElement pai,
                                    String nome,
                                    String valor )
{
   // cria elemento de texto
   TextElement textElement = raiz.createElement( nome, valor );
   pai.appendChild( textElement );
   return true;
}

/* ..................................................................... */

/**
* Cria um novo elemento em um elemento raiz
*
* @param raiz o documento onde deve ser inserido.
* @param pai o elemento pai do novo elemento.
* @param nome o nome do novo elemento.
*/
public static boolean criaElemento( StructuredTextDocument raiz,
                                    TextElement pai,
                                    String nome )
{
   // cria elemento de texto
   TextElement textElement = raiz.createElement( nome );
   pai.appendChild( textElement );
   return true;
```

```
}

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: */

/**
* Mostra um texto na linha de comando
*
* @param texto a string a ser mostrada.
*/
public static boolean mostra( String texto )
{
   System.out.println(texto);
   return true;
}

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: */

/**
* Retorna o primeiro filho de um elemento pai
* OBS.: Precisa ser adaptado para pegar vários filhos
*       do mesmo pai, com o mesmo nome...
*
* @param elemento o elemento-pai.
* @param nome o nome do elemento filho.
*/
public static TextElement retornaFilho( TextElement elemento, String nome )
{
   Enumeration lista = elemento.getChildren(nome);
   TextElement filho = (TextElement)lista.nextElement();
 /* while ( lista.hasMoreElements() )
    {
      filho = (TextElement) lista.nextElement();
      mostra("Filho de dentro eh: " + filho.getName());
      if (!filho.getName().equals(nome))
      {
         retornaFilho( filho, nome );
      }
    }
 */
   return filho;
}

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: */

/**
* Gera o UDDIAdvertisement a partir dos dados de entrada
```

```
        *
        * @param v_nameBS cria o businessServices.
        * @param v_descriptionBS o nome do repositório.
        * @param v_xmllangBS língua utilizada na descrição dos serviços.
        * @param v_discoveryURL a URL para descobrir os serviços.
        * @param v_useTypeBS o tipo da URL businessEntity.
        * @param v_name o nome do businessService.
        * @param v_description a descrição do businessService.
        * @param v_xmllang a língua da descrição do businessService.
        * @param v_accessPoint ponto de informação sobre o serviço.
        * @param v_useType tipo do serviço.
        * @param v_tModelInstanceInfo_tModelKey informação sobre a instância do
tModel.
        * @param v_keyedRef_tModelKey chave do tModel.
        * @param v_keyedRef_keyName nome do atributo do tModel.
        * @param v_keyedRef_keyValue valor do tModel.
        */
        public static String geraUDDI(String v_nameBS,
                                       String v_descriptionBS,
                                       String v_xmllangBS,
                                       String v_discoveryURL,
                                       String v_useTypeBS,
                                       String v_name,
                                       String v_description,
                                       String v_xmllang,
                                       String v_accessPoint,
                                       String v_useType,
                                       String v_tModelInstanceInfo_tModelKey,
                                       String v_keyedRef_tModelKey,
                                       String v_keyedRef_keyName,
                                       String v_keyedRef_keyValue)
        {
           Attribute atributo;
           TextElement textElement;

           // inicia um documento
           StructuredTextDocument doc = (StructuredTextDocument)
              StructuredDocumentFactory.newStructuredDocument(
                new MimeMediaType("text/xml"), "UDDIAdvertisement");

           /*... CRIA BUSINESS SERVICES .....*/
           criaElemento( doc, doc, "businessServices" );
           TextElement businessServices = retornaFilho( doc, "businessServices" );

           // só inicializa o textElement...
           textElement = (TextElement) doc.getParent();
```

```
        /*... CRIA NAME BS .....*/
        // pede para o doc criar o elemento de texto e depois insere no lugar devido
        criaElemento( doc, businessServices, "name", v_nameBS );

        /*... CRIA DESCRIPTION BS .....*/
        criaElemento( doc, businessServices, "description", v_descriptionBS);
        TextElement descriptionBS = retornaFilho( businessServices, "description" );
        insereAtributo( descriptionBS, "xml:lang", v_xmllangBS );

        /*... CRIA DISCOVERYURL BS .....*/
        criaElemento( doc, businessServices, "discoveryURL", v_discoveryURL);
        TextElement discoveryURL = retornaFilho( businessServices, "discoveryURL" );
        insereAtributo( discoveryURL, "useType", v_useTypeBS );


        /*... CRIA BUSINESS SERVICE .....*/
        criaElemento( doc, businessServices, "businessService" );
        TextElement businessService = retornaFilho( businessServices,
                                        "businessService" );

        /*... CRIA NAME .....*/
        // pede para o doc criar o elemento de texto e depois insere no lugar devido
        criaElemento( doc, businessService, "name", v_name);

        /*... CRIA DESCRIPTION .....*/
        criaElemento( doc, businessService, "description", v_description);
        TextElement description = retornaFilho( businessService, "description" );
        insereAtributo( description, "xml:lang", v_xmllang );

        /*... CRIA BINDING TEMPLATES .....*/
        criaElemento( doc, businessService, "bindingTemplates" );
        TextElement bindingTemplates = retornaFilho( businessService,
                                        "bindingTemplates" );

        /*... CRIA BINDING TEMPLATE .....*/
        criaElemento( doc, bindingTemplates, "bindingTemplate" );
        TextElement bindingTemplate = retornaFilho( bindingTemplates,
"bindingTemplate" );
        //insereAtributo( bindingTemplate, "bindingKey", "uddi:exemplo.org:p2p_java" );

        /*... CRIA ACCESS POINT .....*/
        criaElemento( doc, bindingTemplate, "accessPoint", v_accessPoint );
        TextElement accessPoint = retornaFilho( bindingTemplate, "accessPoint" );
        insereAtributo( accessPoint, "useType", v_useType );
```

```
        /*... CRIA TMODELINSTANCEDETAILS .....*/
        criaElemento( doc, bindingTemplate, "tModelInstanceDetails" );
        TextElement tModelInstanceDetails = retornaFilho( bindingTemplate,
"tModelInstanceDetails" );

        /*... CRIA TMODELINSTANCEINFO .....*/
        criaElemento( doc, tModelInstanceDetails, "tModelInstanceInfo" );
        TextElement tModelInstanceInfo = retornaFilho( tModelInstanceDetails,
"tModelInstanceInfo" );
        insereAtributo( tModelInstanceInfo, "tModelKey",
v_tModelInstanceInfo_tModelKey );

        /*... CRIA CATEGORYBAG .....*/
        criaElemento( doc, bindingTemplate, "categoryBag" );
        TextElement categoryBag = retornaFilho( bindingTemplate, "categoryBag" );

        /*... CRIA KEYEDREFERENCE .....*/
        criaElemento( doc, categoryBag, "keyedReference" );
        TextElement keyedReference = retornaFilho( categoryBag, "keyedReference" );
        insereAtributo( keyedReference, "tModelKey", v_keyedRef_tModelKey );
        insereAtributo( keyedReference, "keyName", v_keyedRef_keyName );
        insereAtributo( keyedReference, "keyValue", v_keyedRef_keyValue );

        String str = doc.toString();

        mostra(str);
        return str;
      }

      /* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: */
      /* ::::::::::::::::: fim do arquivo UDDIAdv.java :::::::::::::::::: */
      /* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: */

    }


    /* UTILIZAR ESTA CHAMADA PARA TESTAR O GERADOR */
    /* ESTA É A SAÍDA-MODELO DO TESTE:
    <?xml version="1.0"?>
    <!DOCTYPE UDDIAdvertisement>
    <UDDIAdvertisement>
      <businessServices>
        <name> Empresa A </name>
        <description xml:lang="br"> Empresa A de exemplo </description>
        <discoveryURL useType="businessEntity">
            http://www.exemplo.com.br
```

```
  </discoveryURL>
  <businessService>
   <name> Web Service A </name>
   <description xml:lang="br"> Um exemplo de Web Service </description>
    <bindingTemplates>
     <bindingTemplate>
      <accessPoint useType="endPoint">
        http://www.exemplo.com.br/webserviceA/
      </accessPoint>
      <tModelInstanceDetails>
       <tModelInstanceInfo tModelKey="uddi:uddi.org:transport:http">
       </tModelInstanceInfo>
      </tModelInstanceDetails>
      <categoryBag>
       <keyedReference tModelKey="uddi:uddi.org:categorization:exemplos"
                  keyName="uddi_Name" keyValue="000001">
       </keyedReference>
      </categoryBag>
     </bindingTemplate>
    </bindingTemplates>
  </businessService>
 </businessServices>
</UDDIAdvertisement>
*/


  public static void main(String[] args)
  {
    geraUDDI(
          "Empresa A",                        // nameBS
          "Empresa A de exemplo",             // descriptionBS
          "br",                               // xmllangBS
          "http://www.exemplo.com.br",        // discoveryURL
          "businessEntity",                   // useTypeBS
          "Web Service A",                    // name
          "Um exemplo de Web Service",        // description
          "br",                               // xmllang
          "http://www.exemplo.com.br/webserviceA/",// accessPoint
          "endPoint",                         // useType
          "uddi:uddi.org:transport:http",     // v_tModelInstanceInfo_tModelKey
          "uddi:uddi.org:categorization:exemplos", // v_keyedRef_tModelKey
          "uddi_Name",                        // v_keyedRef_keyName
          "000001");                          // v_keyedRef_keyValue
    System.exit(1);
  }
```

```
package p2p_java;

import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.util.*;
import java.lang.Integer;
import javax.swing.tree.*;
import javax.swing.*;

public class janela_info extends JFrame {
  JScrollPane jScrollPane1 = new JScrollPane();
  JButton jButton1 = new JButton();
  JTextArea txtInfo = new JTextArea();
  String texto;
  public janela_info(String txt) {
    texto=txt;
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  private void jbInit() throws Exception {
    jScrollPane1.setBorder(BorderFactory.createLineBorder(Color.black));
    jScrollPane1.setBounds(new Rectangle(5, 4, 541, 300));
    this.setTitle("ADV Info");
    this.getContentPane().setLayout(null);
    jButton1.setBounds(new Rectangle(482, 312, 64, 31));
    jButton1.setText("OK");
    txtInfo.setText(texto);
    jButton1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton1_actionPerformed(e);
      }
    });
    this.setResizable(false);
    txtInfo.setEditable(false);
    this.getContentPane().add(jScrollPane1, null);
    jScrollPane1.getViewport().add(txtInfo, null);
    this.getContentPane().add(jButton1, null);
  }

  void jButton1_actionPerformed(ActionEvent e) {
     this.setVisible(false);
```

```
  }
}

package p2p_java;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.util.*;
import java.lang.Integer;
import javax.swing.tree.*;

import p2p_java.*;

public class janela_principal extends JFrame {
  JLabel jLabel1 = new JLabel();
  JButton jButton3 = new JButton();
  JTextField jTextField1 = new JTextField();
  JButton btnPublicar = new JButton();
  JLabel jLabel2 = new JLabel();
  JLabel jLabel3 = new JLabel();
  JButton btnDescobrir = new JButton();
  JButton btn_ngrupo = new JButton();
  JButton btn_buscar = new JButton();
  JLabel jLabel4 = new JLabel();
  JTextArea txtGrupo = new JTextArea();
  static Principal gerente=new Principal();
  static janela_principal JPrincipal;
  JButton btn_Info = new JButton();
  JTextArea txtDescricao = new JTextArea();
  JLabel jLabel6 = new JLabel();
  JButton btnExcluirGrupo = new JButton();
  JLabel jLabel7 = new JLabel();
  JLabel lbGrupoAtual = new JLabel();
  JButton btnAcessarGrupo = new JButton();
  JScrollPane teste = new JScrollPane();
  JEditorPane lsStatus = new JEditorPane();
  JScrollPane CArvore = new JScrollPane();
  JButton btnADV2 = new JButton();
  JButton btnADV3 = new JButton();
  JButton btnADV1 = new JButton();
  JButton btnFlush = new JButton();
  JTree tree = new JTree();

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

```
    Vector vec=gerente.IniciarRoot();
   //inicia janela principal

   JPrincipal = new janela_principal();

   JPrincipal.abre_janela_principal(vec);
 }

public void abre_janela_principal(Vector vec) {
 // janela_principal janela_principal = new janela_principal();

  JPrincipal.setSize(566,557);
  JPrincipal.setVisible(true);
  lbGrupoAtual.setText(gerente.getPeerGroupName());
  mostraStatus((Vector)vec.elementAt(0));
  montaArvore((Vector)vec.elementAt(1));
}

public janela_principal() {
  try {
     jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}

private void jbInit() throws Exception {
  this.setDefaultCloseOperation(3);
  this.setResizable(false);
  this.setTitle("P2P WS");

  this.getContentPane().setLayout(null);
  jLabel1.setToolTipText("");
  jLabel1.setText("Status:");
  jLabel1.setBounds(new Rectangle(5, 415, 228, 13));
  jButton3.setBounds(new Rectangle(417, 95, 136, 30));
  jButton3.setEnabled(false);
  jButton3.setToolTipText("");
  jButton3.setText("Publicar");
  jTextField1.setBounds(new Rectangle(5, 98, 406, 24));
  btnPublicar.setBounds(new Rectangle(6, 25, 112, 28));
  btnPublicar.setEnabled(false);
  btnPublicar.setMaximumSize(new Dimension(85, 27));
  btnPublicar.setMinimumSize(new Dimension(85, 27));
  btnPublicar.setText("Publicar WS");
```

```
btnPublicar.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnPublicar_actionPerformed(e);
  }
});
jLabel2.setText("Repositório de Web Services:");
jLabel2.setBounds(new Rectangle(5, 77, 199, 19));
jLabel3.setText("Plataforma JXTA:");
jLabel3.setBounds(new Rectangle(8, 4, 147, 22));
btnDescobrir.setBounds(new Rectangle(123, 25, 112, 28));
btnDescobrir.setEnabled(false);
btnDescobrir.setText("Descobrir WS");
btn_ngrupo.setBounds(new Rectangle(421, 221, 125, 27));
btn_ngrupo.setText("Novo Grupo");
btn_ngrupo.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btn_ngrupo_actionPerformed(e);
  }
});
btn_buscar.setBounds(new Rectangle(422, 318, 125, 28));
btn_buscar.setText("Buscar Grupos");
btn_buscar.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btn_buscar_actionPerformed(e);
  }
});
jLabel4.setToolTipText("");
jLabel4.setText("Grupo:");
jLabel4.setBounds(new Rectangle(421, 139, 127, 20));
txtGrupo.setBounds(new Rectangle(421, 158, 125, 19));
btn_Info.setBounds(new Rectangle(422, 352, 125, 27));
btn_Info.setText("Inf. Grupos");
btn_Info.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btn_Info_actionPerformed(e);
  }
});
txtDescricao.setBounds(new Rectangle(422, 196, 123, 18));
jLabel6.setText("Descrição:");
jLabel6.setBounds(new Rectangle(422, 178, 92, 17));
btnExcluirGrupo.setBounds(new Rectangle(422, 253, 125, 27));
btnExcluirGrupo.setText("Deixar Grupo");
btnExcluirGrupo.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnExcluirGrupo_actionPerformed(e);
  }
```

```
        });
        jLabel7.setToolTipText("");
        jLabel7.setText("Grupo atual:");
        jLabel7.setBounds(new Rectangle(7, 140, 74, 17));
        lbGrupoAtual.setBounds(new Rectangle(79, 139, 235, 19));
        btnAcessarGrupo.setBounds(new Rectangle(422, 285, 126, 26));
        btnAcessarGrupo.setText("Acessar Grupo");
        btnAcessarGrupo.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
            btnAcessarGrupo_actionPerformed(e);
          }
        });

teste.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

teste.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        teste.setBorder(BorderFactory.createLineBorder(Color.black));
        teste.setBounds(new Rectangle(6, 430, 542, 95));
        lsStatus.setForeground(Color.gray);
        lsStatus.setBorder(null);
        lsStatus.setEditable(false);
        CArvore.setBorder(BorderFactory.createLineBorder(Color.black));
        CArvore.setBounds(new Rectangle(7, 158, 404, 254));
        btnADV2.setBounds(new Rectangle(264, 25, 93, 28));
        btnADV2.setText("ADV Rdv");
        btnADV2.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
            btnADV2_actionPerformed(e);
          }
        });
        btnADV3.setBounds(new Rectangle(362, 25, 93, 28));
        btnADV3.setText("ADV PG");
        btnADV3.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
            btnADV3_actionPerformed(e);
          }
        });
        btnADV1.setBounds(new Rectangle(460, 25, 93, 28));
        btnADV1.setText("ADV Peer");
        btnADV1.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
            btnADV1_actionPerformed(e);
          }
        });
        btnFlush.setBounds(new Rectangle(423, 385, 125, 27));
        btnFlush.setText("Esvaziar Cache");
```

```
    btnFlush.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        btnFlush_actionPerformed(e);
      }
    });
    this.getContentPane().add(jLabel1, null);
    this.getContentPane().add(jLabel3, null);
    this.getContentPane().add(jLabel2, null);
    this.getContentPane().add(jTextField1, null);
    this.getContentPane().add(jButton3, null);
    this.getContentPane().add(teste, null);
    this.getContentPane().add(CArvore, null);
    CArvore.getViewport().add(tree, null);
    this.getContentPane().add(btnPublicar, null);
    this.getContentPane().add(btnADV1, null);
    teste.getViewport().add(lsStatus, null);
    this.getContentPane().add(jLabel4, null);
    this.getContentPane().add(txtDescricao, null);
    this.getContentPane().add(jLabel6, null);
    this.getContentPane().add(txtGrupo, null);
    this.getContentPane().add(btn_ngrupo, null);
    this.getContentPane().add(btnExcluirGrupo, null);
    this.getContentPane().add(btnAcessarGrupo, null);
    this.getContentPane().add(btnFlush, null);
    this.getContentPane().add(btn_Info, null);
    this.getContentPane().add(btn_buscar, null);
    this.getContentPane().add(jLabel7, null);
    this.getContentPane().add(btnADV3, null);
    this.getContentPane().add(btnADV2, null);
    this.getContentPane().add(btnDescobrir, null);
    this.getContentPane().add(lbGrupoAtual, null);
  }

/* public void btnConectar_actionPerformed(ActionEvent e) {
      // inicializando JXTA
      Vector vec=new Vector();
      vec=gerente.conectar();
      if (vec.elementAt(0)!=null){
        btnConectar.setEnabled(false);
        btnDesconectar.setEnabled(true);
      }
      this.mostraStatus(vec);
  }

  public void btnDesconectar_actionPerformed(ActionEvent e) {
     // parando Jxta
```

```
    Vector vec=new Vector();
    vec=gerente.desconectar();
    if (vec.firstElement()!=null){
       btnConectar.setEnabled(true);
       btnDesconectar.setEnabled(false);
    }
    this.mostraStatus(vec);
 }
*/
 void btn_Info_actionPerformed(ActionEvent e) {
    //busca informacoes gerais
    Vector vec = gerente.getInfoPeerGroup();
    this.mostraStatus(vec);
 }

 void btn_ngrupo_actionPerformed(ActionEvent e) {
    Vector vec=new Vector();

    if (!txtGrupo.getText().equalsIgnoreCase("") && txtGrupo.getText()!=null){
      vec = gerente.criarPeerGroup(txtGrupo.getText(),txtDescricao.getText());
      if (vec.firstElement()!=null){
        this.lbGrupoAtual.setText(gerente.getPeerGroupName());
        txtGrupo.setText(""); //limpa caixa de texto
        txtDescricao.setText("");
      }
      this.mostraStatus(vec);
      this.buscarGroups();
    }
 }

 void btn_buscar_actionPerformed(ActionEvent e) {
    //busca ADV de grupos e peers e monta arvore
    this.buscarGroups();
 }
 private void  buscarGroups(){
 Vector v=new Vector();
    v = gerente.getPeerGroups();
    if (v!=null){
       montaArvore(v);
       v=null;
    }
    this.lbGrupoAtual.setText(gerente.getPeerGroupName());
 }

 private void montaArvore(Vector obj){
  // monta a arvore de grupos e peers
```

```
  DefaultMutableTreeNode root = processaNosArvore(obj);
  JTree treeTemp = new JTree(root);
  CArvore.remove(tree);
  tree=treeTemp;
  treeTemp=null;
  CArvore.getViewport().add(tree,null);
  CArvore.updateUI();
}

void mostraStatus(Vector v){
   //preenche lista de status com as acoes retornadas
   String Acao="";
   int i;
   for (i=1;i<v.size();i++){
     //  Acao = Acao + v.get(i).toString() + "\n";
       Acao = Acao + v.elementAt(i) + "\n";
  }
  lsStatus.setText(lsStatus.getText() + Acao);
}

static DefaultMutableTreeNode processaNosArvore(Vector v){
  DefaultMutableTreeNode node = new DefaultMutableTreeNode(v.elementAt(0));
  DefaultMutableTreeNode child;
  for(int i=1; i<v.size(); i++) {
     Object nodeSpecifier = v.elementAt(i);
     if (nodeSpecifier instanceof Vector){
       child = processaNosArvore((Vector)nodeSpecifier);
       child.setUserObject(v.elementAt(i));
     }else{
       child = new DefaultMutableTreeNode(nodeSpecifier);
       child.setUserObject(v.elementAt(i));}
     node.add(child);
  }
  return(node);
}

void btnAcessarGrupo_actionPerformed(ActionEvent e) {
 //acessa um PeerGroup
   Vector vec=new Vector();
   if (!txtGrupo.getText().equalsIgnoreCase("") ){
     vec = gerente.acessarPeerGroup(txtGrupo.getText());
     Vector vStatus = (Vector)vec.elementAt(1);
     Vector vRetorno = (Vector)vec.elementAt(0);

      try{
        if (vRetorno!=null){
```

```
            if (vRetorno.firstElement()!=null){ //sucesso
              this.lbGrupoAtual.setText(gerente.getPeerGroupName());
              this.buscarGroups();
              txtGrupo.setText(""); //limpa caixa de texto
              txtDescricao.setText("");
            }
           }
      }catch(Exception er){
        er.printStackTrace();
      }
      this.mostraStatus(vStatus);
      vec=null;
      vStatus=null;
      vRetorno=null;
    }
}

void btnADV2_actionPerformed(ActionEvent e) {
    String texto=gerente.buscarADV(1);
    if (texto.trim()!=""){
        janela_info jInfo = new janela_info(texto);
        jInfo.setSize(560,375);
        jInfo.setVisible(true);
    }
}

void btnPublicar_actionPerformed(ActionEvent e) {

}

void btnADV3_actionPerformed(ActionEvent e) {
    String texto=gerente.buscarADV(2);
    if (texto.trim()!=""){
        janela_info jInfo = new janela_info(texto);
        jInfo.setSize(560,375);
        jInfo.setVisible(true);
    }
}

void btnADV1_actionPerformed(ActionEvent e) {
    String texto=gerente.buscarADV(3);
    if (texto.trim()!=""){
        janela_info jInfo = new janela_info(texto);
        jInfo.setSize(560,375);
        jInfo.setVisible(true);
    }
```

```
}

void btnFlush_actionPerformed(ActionEvent e) {
 // gerente.flush();
 Vector a = new Vector();
 a.add("NetPeerGroup");
 a.add("PeerGroup Economia");
 a.add("PeerGroup Saude");
 a.add("PeerGroup Etc");
 a.add("Peer Empresa A");
 a.add("Peer Empresa B");
 montaArvore(a);
}



void btnExcluirGrupo_actionPerformed(ActionEvent e) {
   //deixar um PeerGroup
   Vector vec=new Vector();
   if (!txtGrupo.getText().equalsIgnoreCase("") ){
     vec = gerente.deixarPeerGroup(txtGrupo.getText());
     Vector vStatus = (Vector)vec.elementAt(1);
     Vector vRetorno = (Vector)vec.elementAt(0);

      try{
        if (vRetorno!=null){
          if (vRetorno.firstElement()!=null){ //sucesso
            this.lbGrupoAtual.setText(gerente.getPeerGroupName());
            this.buscarGroups();
            txtGrupo.setText(""); //limpa caixa de texto
            txtDescricao.setText("");
          }
        }
     }catch(Exception er){
       er.printStackTrace();
     }
     this.mostraStatus(vStatus);
     vec=null;
     vStatus=null;
     vRetorno=null;
   }
}
```

```
}

package p2p_java;

import java.io.*;
import java.util.*;
import java.lang.Integer;

import p2p_java.*;

public class msg_Interface{

  public msg_Interface() {
  }

  public String TrataErro(int erro,String msgerro){
    String vRetorno=null;
    switch(erro){
       case 88153: //falha ao cria NetPeerGroup ROOT
         vRetorno = "Erro(" + erro + "): Falha ao criar Peer Group Root. ";
         break;
       case 10001: //NetPeerGroup ROOT é nulo
         vRetorno = "Erro(" + erro + "): Falha ao criar Peer Group, retorno nulo. ";
         break;
       case 10002:
         vRetorno = "Erro(" + erro + "): Falha ao criar Peer Group. ";
         break;
       case 10003:
         vRetorno = "Erro(" + erro + "): Não foi possível iniciar o Peer Group: " +
msgerro;
         break;
       case 10004:
         vRetorno = "Erro(" + erro + "): Não foi possível publicar Advertisement. ";
         break;
       case 10005:
         vRetorno = "Erro(" + erro + "): Não foi possível criar Advertisement Group. ";
         break;
       case 10006:
         vRetorno = "Erro(" + erro + "): Não foi possível criar Peer Group. ";
         break;
       case 10008:
         vRetorno = "Erro(" + erro + "): Não foi possível criar Advertisement de grupo. ";
         break;
       case 100010:
```

```
        vRetorno = "Erro(" + erro + "): Falha ao autenticar acesso ao grupo. ";
        break;
       default:
        vRetorno = "Erro(" + erro + "): " +  msgerro;
        break;
    }
    return vRetorno;
  }

  public String mostraMsg(String msg){
    String vRetorno = "MSG: " +  msg;
    return vRetorno;
  }


}

package p2p_java;

import net.jxta.platform.*;
import net.jxta.platform.Module;
import net.jxta.document.*;

import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;

import net.jxta.rendezvous.RendezVousService;
import net.jxta.exception.*;
import net.jxta.endpoint.*;
import net.jxta.pipe.*;
import net.jxta.protocol.*;
import net.jxta.discovery.*;
import net.jxta.credential.AuthenticationCredential;
import net.jxta.membership.Authenticator;
import net.jxta.membership.*;
import net.jxta.membership.MembershipService;
import net.jxta.impl.*;
import net.jxta.impl.peergroup.* ;
import net.jxta.impl.protocol.*;
import net.jxta.impl.id.UUID.UUID;
import net.jxta.impl.id.UUID.UUIDFactory;
import java.util.*;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

import net.jxta.protocol.ModuleImplAdvertisement;
```

```
import net.jxta.discovery.DiscoveryService;
import net.jxta.exception.PeerGroupException;
import net.jxta.protocol.PipeAdvertisement;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;
import net.jxta.pipe.PipeService;
import net.jxta.protocol.PeerGroupAdvertisement;
import net.jxta.document.AdvertisementFactory;
import net.jxta.document.MimeMediaType;
import net.jxta.exception.PeerGroupException;
import net.jxta.protocol.ModuleClassAdvertisement;
import net.jxta.document.Element;
import net.jxta.document.StructuredDocument;
import net.jxta.document.StructuredDocumentFactory;
import net.jxta.document.StructuredDocumentUtils;
import net.jxta.document.StructuredTextDocument;
import net.jxta.endpoint.Message;
import net.jxta.endpoint.MessageElement;
import net.jxta.id.ID;
import net.jxta.id.IDFactory;
import net.jxta.pipe.InputPipe;
import net.jxta.platform.ModuleClassID;
import net.jxta.protocol.ModuleClassAdvertisement;
import net.jxta.protocol.ModuleSpecAdvertisement;

import net.jxta.platform.*;
import net.jxta.document.*;
import net.jxta.peergroup.*;
import net.jxta.exception.*;
import net.jxta.endpoint.*;
import net.jxta.pipe.*;
import net.jxta.protocol.*;
import net.jxta.discovery.*;
import net.jxta.credential.AuthenticationCredential;
import net.jxta.membership.Authenticator;
import net.jxta.membership.MembershipService;
import net.jxta.impl.peergroup.* ;
import net.jxta.impl.protocol.*;
import net.jxta.impl.id.UUID.UUID;
import net.jxta.impl.id.UUID.UUIDFactory;
import java.util.*;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.io.*;
import java.util.*;
import java.lang.Integer;
```

```
import java.lang.*;


import p2p_java.*;

public class Peer_p2p {
   private msg_Interface msgInterface=new msg_Interface();
   public DiscoveryService discovery;

//-------------------------------------------------------------------------------------
   public Vector pararPlataform(PeerGroup netPeerGroup){
      Vector vec=new Vector();
      vec.add(null);
      vec.add("Parando Plataform JXTA...");
      try{
         netPeerGroup.stopApp();
      }catch(Exception e){
         vec.add(msgInterface.TrataErro(e.hashCode(),e.getMessage()));
         vec.set(0,null);
         return vec;
      }
      vec.add("Plataform JXTA parada com sucesso.");
      vec.set(0,netPeerGroup);
      return vec;
   }
//-------------------------------------------------------------------------------------
   public PeerGroup buscarPeerGroup(String Nome,PeerGroup PGRoot){
     //retorna um PeerGroup em especifico
     PeerGroupAdvertisement pGroupADV=null;
     PeerGroup PGAcessado=null;
     Object obj;
     int achou=0;
     try{
       DiscoveryService disco = PGRoot.getDiscoveryService();
       Enumeration enum = disco.getLocalAdvertisements(disco.GROUP,null,null);
       while (enum.hasMoreElements()){
          obj = (Object) enum.nextElement();
          if (obj.getClass().getName()=="net.jxta.impl.protocol.PeerGroupAdv"){
               try{
                  pGroupADV=(PeerGroupAdvertisement) obj;
                  System.out.println(pGroupADV.getName());
                  if(pGroupADV!=null){
                    if (pGroupADV.getName().equalsIgnoreCase(Nome)){
                      System.out.println("achou!!!");
                      achou=1;
                      break;
```

```
                    }
                  }else{System.out.println("nullo");}
                }catch(Exception e){
                      e.printStackTrace();
                      System.out.println("erro PGADV");
                      return null;
                }
            }
          }
        }catch(Exception e){
          e.printStackTrace();
          return null;
        }
        if (pGroupADV!=null){
          try{
            if (achou==1){
               PGAcessado=PGRoot.newGroup(pGroupADV);
            }
          }catch(Exception e){
            e.printStackTrace();
            return null;
          }
        }
        return PGAcessado;
    }
//-------------------------------------------------------------------------------------------
    public Vector RootPeerGroup() {
        //iniciando o peergroup root
        PeerGroup netPeerGroup  = null;
        PeerAdvertisement PeerAdv=null;
        PeerGroupAdvertisement PeerGroupAdv=null;
        Vector vec=new Vector();
        vec.add(null);
        vec.add("Iniciando Peer Group Root...");
        try {

            netPeerGroup = PeerGroupFactory.newNetPeerGroup();
            this.flush(netPeerGroup);
            PeerAdv = netPeerGroup.getPeerAdvertisement();
            PeerGroupAdv = netPeerGroup.getPeerGroupAdvertisement();

        //    DiscoveryService discovery = netPeerGroup.getDiscoveryService();

          //  discovery.publish(PeerAdv,discovery.PEER);
         //   discovery.remotePublish(PeerAdv,discovery.PEER);
          //  discovery.publish(PeerGroupAdv,discovery.GROUP);
```

```
            //  discovery.remotePublish(PeerGroupAdv,discovery.GROUP);

          } catch ( PeerGroupException e) {
              vec.add(msgInterface.TrataErro(e.hashCode(),e.getMessage()));
              vec.set(0,null);
              return vec;
          } catch (NullPointerException e){
              vec.add(msgInterface.TrataErro(e.hashCode(),e.getMessage()));
              vec.set(0,null);
              return vec;
          }catch (Exception e){
              vec.add(msgInterface.TrataErro(e.hashCode(),e.getMessage()));
              vec.set(0,null);
              return vec;
          }
          vec.add("Peer Group Root iniciado com sucesso.");
          vec.set(0,netPeerGroup);
          return vec;
        }
      //-------------------------------------------------------------------------------------------
       public void flush(PeerGroup GrupoAtual){
        //limpando cache local
        DiscoveryService disco = GrupoAtual.getDiscoveryService();
        PeerGroupAdvertisement pGroupADV;
        PeerAdvertisement pADV;
        Enumeration enum;
        Object obj;
        try{
           // flush GROUPs
           enum = disco.getLocalAdvertisements(disco.GROUP,null,null);
           while (enum.hasMoreElements()){
              obj = (Object) enum.nextElement();
               if (obj.getClass().getName()=="net.jxta.impl.protocol.PeerGroupAdv"){
                    try{
                       pGroupADV=(PeerGroupAdvertisement) obj;

disco.flushAdvertisements(pGroupADV.getPeerGroupID().toString(),disco.GROUP);
                    }catch(Exception e){
                          System.out.println("erro PGADV");
                    }
               }
           }
            // flush PEERS
           enum = disco.getLocalAdvertisements(disco.PEER,null,null);
           while (enum.hasMoreElements()){
              obj = (Object) enum.nextElement();
```

```
                if (obj.getClass().getName()=="net.jxta.impl.protocol.PeerAdv"){
                   try{
                      pADV=(PeerAdvertisement) obj;
                      disco.flushAdvertisements(pADV.getPeerID().toString(),disco.PEER);
                   }catch(Exception e){
                         System.out.println("erro PADV");
                   }
                }
          }

        }catch(Exception e){
           System.out.println("erro");
        }
       }
     //-------------------------------------------------------------------------------------
        public Vector entrarPeerGroup(PeerGroup netPeerGroup,Vector vec){
           // entrando peer Group
           StructuredDocument creds = null;
           MembershipService memberShip=null;
          //   Vector vec=new Vector();
           if (vec==null){ vec.add(null);}
           vec.add("Acessando Peer Group...");
           System.out.println("Entrando: " + netPeerGroup.getPeerGroupName());
           try{
              AuthenticationCredential authCred = new AuthenticationCredential(
netPeerGroup, null, creds );
              memberShip = (MembershipService) netPeerGroup.getMembershipService();

              Authenticator auth = memberShip.apply( authCred );
              completeAuth( auth );
              if( !auth.isReadyForJoin() ) {
                 System.out.println("not join");
                 vec.add(msgInterface.TrataErro(100010,null));
                 vec.set(0,null);
                 return vec;
              }
              memberShip.join( auth );
              System.out.println("join");
           }catch(Exception e){
              e.printStackTrace();
              System.out.println("not join2");
              vec.add(msgInterface.TrataErro(e.hashCode(),e.getMessage()));
              vec.set(0,null);
              return vec;
           }
```

```
            vec.add("Peer Group '" + (String) netPeerGroup.getPeerGroupName() + "'
acessado com sucesso.");
            vec.set(0,netPeerGroup);
            return vec;
           }

    //-------------------------------------------------------------------------------------
            private void completeAuth (Authenticator auth) throws Exception{
            Method [] methods = auth.getClass().getMethods();
            Vector authMethods = new Vector();
            for( int eachMethod = 0; eachMethod < methods.length; eachMethod++ ) {
                if( methods[eachMethod].getName().startsWith("setAuth") ){
                  if( Modifier.isPublic( methods[eachMethod].getModifiers())){
                  for( int doInsert = 0; doInsert <= authMethods.size();
                                                    doInsert++ ) {
                       int insertHere = -1;
                       if( doInsert == authMethods.size() )
                          insertHere = doInsert;
                       else {
                          if(methods[eachMethod].getName().compareTo(
     ((Method)authMethods.elementAt( doInsert )).getName()) <= 0 )
                             insertHere = doInsert;
                       } // end else
                       if(-1!= insertHere ) {
                          authMethods.insertElementAt(
                                   methods[eachMethod],insertHere);
                          break;
                       } // end if ( -1 != insertHere)
                    } // end for (int doInsert=0
                  } // end if (modifier.isPublic
                } // end if (methods[eachMethod]
            } // end for (int eachMethod)
               Object [] AuthId = {"rootpeers"};
               Object [] AuthPasswd = {"JLXH"}; // encrypted RULE
            for( int eachAuthMethod=0;eachAuthMethod<authMethods.size();
                                              eachAuthMethod++ ) {
              Method doingMethod = (Method) authMethods.elementAt(eachAuthMethod);
              String authStepName = doingMethod.getName().substring(7);
              if (doingMethod.getName().equals("setAuth1Identity")) {
                 // Found identity Method, providing identity
                 doingMethod.invoke( auth, AuthId);
               }
               else if (doingMethod.getName().equals("setAuth2Password")){
                 // Found Passwd Method, providing passwd
                 doingMethod.invoke( auth, AuthPasswd );
               }
```

```
        }
    }
//-------------------------------------------------------------------------------------------
    public Vector deixarPeerGroup(PeerGroup netPeerGroup,Vector vec){
        // abandonando Peer Group
        MembershipService memberShip=null;
       //Vector vec=new Vector();
        if (vec==null){ vec.add(null);}
        vec.add(null);
        vec.add("Abandonando Peer Group...");
        try{
          memberShip = (MembershipService) netPeerGroup.getMembershipService();
          memberShip.resign();
        }catch(Exception e){
          vec.add(msgInterface.TrataErro(e.hashCode(),e.getMessage()));
          vec.set(0,null);
          return vec;
        }
        vec.add("Peer Group abandonado com sucesso.");
        vec.set(0,netPeerGroup);
        return vec;

    }
//-------------------------------------------------------------------------------------------
    public Vector buscarPeerGroups(PeerGroup netPeerGroup){
      //advertisement de busca de peerGroups
      PeerGroupAdvertisement pGrouADV=null;
      DiscoveryService disco=null;
      Object obj;
      Vector vec=new Vector();
      vec.add(netPeerGroup.getPeerGroupName());
      System.out.println("asdfasdf");
      try{
         disco = netPeerGroup.getDiscoveryService();
         disco.getRemoteAdvertisements(null,disco.GROUP,null,null,90);
         Enumeration v = disco.getLocalAdvertisements(disco.GROUP,null,null);
       //  System.out.println("ROOT? " +  netPeerGroup.getPeerGroupName());
         while (v.hasMoreElements()){
               obj = (Object) v.nextElement();
               if (obj.getClass().getName()=="net.jxta.impl.protocol.PeerGroupAdv"){
                   try{
                     System.out.println("asdasd");
                     pGrouADV=(PeerGroupAdvertisement) obj;
                     vec.add("PeerGroup " + pGrouADV.getName());
                   }catch(Exception e){}
               }
```

```
            }
            }catch( Exception e){
          //  vec.add(msgInterface.TrataErro(e.hashCode(),e.getMessage()));
            e.printStackTrace();
            return null;
          }
       //  vec.add("Lista de Peer Group's recuperada com sucesso.");
      //   vec.set(0,null);
       //  Object[] objRetorno = { vec,vecPG };
        return vec;
     }
//-------------------------------------------------------------------------------------------
     public Vector buscarPeers(PeerGroup netPeerGroup){
        //advertisement de busca de peers
        PeerAdvertisement pGrouADV=null;
        DiscoveryService disco=null;
        Object obj;
        Vector vec=new Vector();
         try{
            disco = netPeerGroup.getDiscoveryService();
            disco.getRemoteAdvertisements(null,disco.PEER,null,null,90);
            Enumeration v = disco.getLocalAdvertisements(disco.PEER,null,null);
             while (v.hasMoreElements()){
                  obj = (Object) v.nextElement();
                  if (obj.getClass().getName()=="net.jxta.impl.protocol.PeerAdv"){
                      try{
                        pGrouADV=(PeerAdvertisement) obj;
                        vec.add("Peer " + pGrouADV.getName());
                      }catch(Exception e){
                        e.printStackTrace();
                      }
                  }
            }
            }catch( Exception e){
          //  vec.add(msgInterface.TrataErro(e.hashCode(),e.getMessage()));
            e.printStackTrace();
            return null;
          }
       //  vec.add("Lista de Peer Group's recuperada com sucesso.");
      //   vec.set(0,null);
       //  Object[] objRetorno = { vec,vecPG };
        return vec;
     }
//-------------------------------------------------------------------------------------------
```

```
        public Vector criarPeerGroup(String Nome,String descricao,PeerGroup
netPeerGroup){
          //Criando Peer Group
          PeerGroup newGroup=null;
          Vector vec=new Vector();
          vec.add(null);
          vec.add("Criando novo Peer Group...");
          try{

            ModuleImplAdvertisement implAdv =
netPeerGroup.getAllPurposePeerGroupImplAdvertisement();

            PeerGroupID groupID = IDFactory.newPeerGroupID();
            newGroup = netPeerGroup.newGroup(groupID, implAdv, Nome, descricao);
            if (newGroup == null){
              vec.add(msgInterface.TrataErro(10001,null));
              vec.set(0,null);
              return vec;
            }
            // só é necessario publicar remotamente pois metodo newGroup() publica
localmente.
            PeerGroupAdvertisement groupAdv = newGroup.getPeerGroupAdvertisement();
            discovery = netPeerGroup.getDiscoveryService();
            discovery.publish(groupAdv,discovery.GROUP);
            discovery.remotePublish(groupAdv,discovery.GROUP);
          }catch (Exception e){
            vec.add(msgInterface.TrataErro(10002,e.getMessage()));
            vec.set(0,null);
            return vec;
          }
          vec.add("Grupo '"  + Nome + "' criado com sucesso.");
          vec.set(0,newGroup);
          return vec;
        }
      //-------------------------------------------------------------------------------------
        private PeerGroupAdvertisement criarGroupAdvertisement (PeerGroupAdvertisement
parentPeerGroupAdvertisement) {
            PeerGroupAdvertisement GroupAdv=null;
            //funcao nao consolidada :) só pra teste

            try {
                GroupAdv= (PeerGroupAdvertisement)
                            AdvertisementFactory.newAdvertisement(
                            PeerGroupAdvertisement.getAdvertisementType());
            }
            catch (Exception e){
```

```
                msgInterface.TrataErro(10006,e.getMessage());
                return null;
            }

            GroupAdv.setPeerGroupID(parentPeerGroupAdvertisement.getPeerGroupID());
            // Set PeerGroupId
           //  PeerGroupId = ();
            //  GroupAdv.setPeerGroupID (new net.jxta.impl.id.UUID.PeerGroupID (new
UUID( 0x4d6172676572696eL, 0x204272756e6f2030L)));

            GroupAdv.setName ("UDDIP2P");
            return GroupAdv ;
        }
    //-------------------------------------------------------------------------------------
     public Vector getInfoPeerGroup(PeerGroup pGroup){
        Vector vec=new Vector();
        vec.add(null);
        vec.add("Informações: ");
        vec.add("Peer atual: " + pGroup.getPeerName());
        vec.add("Peer Group atual: " + pGroup.getPeerGroupName());
       //  vec.add("Peer ADV: \n" + pGroup.getPeerAdvertisement());
      //   vec.add("Peer Group ADV: \n" + pGroup.getPeerGroupAdvertisement());
       //  vec.add("RendezVous ADV: \n" + buscarADVRDVZ(pGroup));
        // vec.add("ResolverService: " + pGroup.getResolverService());
        // vec.add("Peer Info Service: " + pGroup.getPeerInfoService());
        vec.set(0,new Boolean(true));
        return vec;
     }
    //-------------------------------------------------------------------------------------
     public String buscarADVRDVZ(PeerGroup netPeerGroup){
          //busca ADV de RDZ
          net.jxta.impl.protocol.RdvAdv pGrouADV=null;
          DiscoveryService disco=null;
          Object obj;
          String ADV="";
          try{
             disco = netPeerGroup.getDiscoveryService();
             disco.getRemoteAdvertisements(null,disco.GROUP,null,null,30,null);
             Enumeration v = disco.getLocalAdvertisements(disco.GROUP,null,null);
             while (v.hasMoreElements()){
                   obj = (Object) v.nextElement();
                   if (obj.getClass().getName()=="net.jxta.impl.protocol.RdvAdv"){
                       pGrouADV=(net.jxta.impl.protocol.RdvAdv) obj;
                       ADV = ADV + "\n" + pGrouADV.toString();
                   }
             }
```

```
    }catch( Exception e){
      return null;
    }
    return ADV;
 }

//-------------------------------------------------------------------------------------

  public String buscarADV(PeerGroup netPeerGroup,int tipoADV){
    //busca ADV
    DiscoveryService disco=null;
    Enumeration enum=null;
    Object obj;
    String tADV="";
    String ADV="";
    try{
       disco = netPeerGroup.getDiscoveryService();
       switch(tipoADV){
          case 1:
            tADV="net.jxta.impl.protocol.RdvAdv";
            disco.getRemoteAdvertisements(null,disco.ADV,null,null,30,null);
            enum = disco.getLocalAdvertisements(disco.ADV,null,null);
            break;
          case 2:
            tADV="net.jxta.impl.protocol.PeerGroupAdv";
            disco.getRemoteAdvertisements(null,disco.GROUP,null,null,30,null);
            enum = disco.getLocalAdvertisements(disco.GROUP,null,null);
            break;
          case 3:
            tADV="net.jxta.impl.protocol.PeerAdv";
            disco.getRemoteAdvertisements(null,disco.PEER,null,null,30,null);
            enum = disco.getLocalAdvertisements(disco.PEER,null,null);
            break;
       }
       while (enum.hasMoreElements()){
             obj = (Object) enum.nextElement();
             if (obj.getClass().getName()==tADV){
                 //pGrouADV=(net.jxta.impl.protocol.RdvAdv) obj;
                 ADV = ADV + "\n" + obj.toString(); //pGrouADV.toString();
             }
       }
    }catch( Exception e){
      e.printStackTrace();
      return null;
    }
    return ADV;
```

```
  }




}


package p2p_java;

import net.jxta.platform.*;
import net.jxta.platform.Module;
import net.jxta.document.*;

import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;

import net.jxta.exception.*;
import net.jxta.endpoint.*;
import net.jxta.pipe.*;
import net.jxta.protocol.*;
import net.jxta.discovery.*;
import net.jxta.credential.AuthenticationCredential;
import net.jxta.membership.Authenticator;
import net.jxta.membership.*;
import net.jxta.membership.MembershipService;
import net.jxta.impl.*;
import net.jxta.impl.peergroup.* ;
import net.jxta.impl.protocol.*;
import net.jxta.impl.id.UUID.UUID;
import net.jxta.impl.id.UUID.UUIDFactory;
import java.util.*;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

import net.jxta.discovery.DiscoveryService;
import net.jxta.exception.PeerGroupException;
import net.jxta.protocol.PipeAdvertisement;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;
import net.jxta.pipe.PipeService;
import net.jxta.protocol.PeerGroupAdvertisement;
import net.jxta.document.AdvertisementFactory;
import net.jxta.document.MimeMediaType;
```

```
import net.jxta.exception.PeerGroupException;
import net.jxta.protocol.ModuleClassAdvertisement;
import net.jxta.document.Element;
import net.jxta.document.StructuredDocument;
import net.jxta.document.StructuredDocumentFactory;
import net.jxta.document.StructuredDocumentUtils;
import net.jxta.document.StructuredTextDocument;
import net.jxta.endpoint.Message;
import net.jxta.endpoint.MessageElement;
import net.jxta.id.ID;
import net.jxta.id.IDFactory;
import net.jxta.pipe.InputPipe;
import net.jxta.platform.ModuleClassID;
import net.jxta.protocol.ModuleClassAdvertisement;
import net.jxta.protocol.ModuleSpecAdvertisement;

import net.jxta.platform.*;
import net.jxta.document.*;
import net.jxta.peergroup.*;
import net.jxta.exception.*;
import net.jxta.endpoint.*;
import net.jxta.pipe.*;
import net.jxta.protocol.*;
import net.jxta.discovery.*;
import net.jxta.credential.AuthenticationCredential;
import net.jxta.membership.Authenticator;
import net.jxta.membership.MembershipService;
import net.jxta.impl.peergroup.* ;
import net.jxta.impl.protocol.*;
import net.jxta.impl.id.UUID.UUID;
import net.jxta.impl.id.UUID.UUIDFactory;
import java.util.*;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.io.*;
import java.util.*;
import java.lang.Integer;

import p2p_java.*;

public class PeerGroup_p2p {
  private PeerGroupAdvertisement groupAdvertisement = null;
  private PeerGroupAdvertisement groupAdvertisemen = null;
 // private PeerGroup netPeerGroup  = null;
 // private PeerGroup PeerGroup_atual = null;
  private DiscoveryService discovery;
```

```
    private msg_Interface msgInterface=new msg_Interface();

    public PeerGroup_p2p(){

    }

    public PeerGroup CriarPeerGroup(String Nome,PeerGroup parentGroup,Hashtable
userPasswdHastable){
        PeerGroupAdvertisement parentGroupAdv, GroupAdv=null;
        PeerGroup WSPeerGroup;
         //Criando Peer Group
         WSPeerGroup=PeerGroupFactory.newPeerGroup();
         if (WSPeerGroup == null){
            msgInterface.TrataErro(10001,null);
            return null;
         }

         parentGroupAdv=parentGroup.getPeerGroupAdvertisement();

         //chama metodo
         GroupAdv=criarGroupAdvertisement(parentGroupAdv, userPasswdHastable);
         if (GroupAdv == null){
            msgInterface.TrataErro(10002,null);
            return null;
         }
         // associar o grupo novo com o grupo root
         try {
            WSPeerGroup.init(parentGroup,WSPeerGroup.getPeerGroupID(),GroupAdv);
         }catch (Exception e){
            msgInterface.TrataErro(10003,e.getMessage());
            return null;
         }

         //enviar advertisement de grupo
         discovery = parentGroup.getDiscoveryService();
         try {
            // publica localmente
            discovery.publish((PeerGroupAdv) GroupAdv, discovery.GROUP);
            // publica para o peer group
            discovery.remotePublish((PeerGroupAdv) GroupAdv, discovery.GROUP);
         }catch (Exception e) {
            msgInterface.TrataErro(10004,e.getMessage());
            return null;
         }

         msgInterface.mostraMsg("Grupo '"  + Nome + "'Criado com sucesso.");
```

```
        return (WSPeerGroup);
    }

    private PeerGroupAdvertisement criarGroupAdvertisement (PeerGroupAdvertisement
parentPeerGroupAdvertisement,
                    Hashtable userPasswdHastable) {
        PeerGroupAdvertisement GroupAdv=null;
       //  net.jxta.id.ID PeerGroupId;

        try {
            GroupAdv= (PeerGroupAdvertisement)
                    AdvertisementFactory.newAdvertisement(
                    PeerGroupAdvertisement.getAdvertisementType());
        }
        catch (Exception e){
            msgInterface.TrataErro(10006,e.getMessage());
            return null;
        }

        // Set Pid
        GroupAdv.setPeerGroupID(parentPeerGroupAdvertisement.getPeerGroupID());
        // Set PeerGroupId
       //  PeerGroupId = ();
        GroupAdv.setPeerGroupID (new net.jxta.impl.id.UUID.PeerGroupID (new
UUID( 0x4d6172676572696eL, 0x204272756e6f2030L)));

        GroupAdv.setName ("UDDI-P2P");
        // Set the Keywords associated with the Peer Group
        //GroupAdv.ssetKeywords ("UDDI WSDL Web Services Sharing P2P Peer to
Peer");
        // Set EndPoints.
       // GroupAdv.setEndpointAdvertisements(
       //       parentPeerGroupAdvertisement.getEndpointAdvertisements());
       // GroupAdv.setIsRendezvous(true);

        // coloca os servicos disponiveis no ADV
       //  ServiceAdvertisement serviceAdv1 = (ServiceAdvertisement)
       //  AdvertisementFactory.newAdvertisement("jxta:ServiceAdvertisement");
        // serviceAdv1.setName("Web Service de automação laboratorial");
      /*   serviceAdv1.setProvider("http://www.inf.ufsc.br/~khaue/ws/webservice.jar");
        serviceAdv1.setSecurity("TBD");
        serviceAdv1.setVersion("0.3");
        serviceAdv1.setParams(new Vector( ));
        serviceAdv1.setCode(this.getClass( ).getName( ));
        GroupAdv.setApp(serviceAdv1);
        //Set the group implementation
```

```
        ServiceAdvertisement serviceAdv2 = (ServiceAdvertisement)
        AdvertisementFactory.newAdvertisement("jxta:ServiceAdvertisement");
        serviceAdv2.setName("Group");
        serviceAdv2.setProvider("http://www.jxta.org/download/jxta.jar");
        serviceAdv2.setSecurity("TBD");
        serviceAdv2.setVersion("1.0");
        serviceAdv2.setParams(new Vector( ));
        serviceAdv2.setCode(this.getClass( ).getName( ));
        GroupAdv.setGroupImpl(serviceAdv2);

        Hashtable
GroupAdvServiceHashtable,grpAdvServicesHashtable=null;Enumeration keyEnumeration
= null;
                Object keyObject=null;
                Object membershipServiceKey=null;
                Object valueOject=null;
                Object dumbObject=null;
                ServiceAdvertisement GroupMembershipServiceAdv=null;
                ServiceAdvertisement grpMembershipServiceAdv=null;
                Vector paramsLoginsAndPasswdsVector;
                Enumeration loginsEnumeration;
                String login, passwd, loginsandPasswdString="";
        GroupAdvServiceHashtable=new Hashtable();

grpAdvServicesHashtable=parentPeerGroupAdvertisement.getServiceAdvertisements();
        keyEnumeration = grpAdvServicesHashtable.keys();

        while (keyEnumeration.hasMoreElements()) {
            keyObject=keyEnumeration.nextElement();
            if (!(keyObject.toString().equals("jxta.service.membership"))) {
                dumbObject = GroupAdvServiceHashtable.put
                        (keyObject, grpAdvServicesHashtable.get(keyObject));
            }else {
                membershipServiceKey=keyObject;
                grpMembershipServiceAdv=(ServiceAdvertisement)
                grpAdvServicesHashtable.get(keyObject);
            }
         }
         try {
            GroupMembershipServiceAdv =(ServiceAdvertisement)
                    AdvertisementFactory.newAdvertisement(
                            ServiceAdvertisement.getAdvertisementType());
         }
         catch (Exception e) {
            msgInterface.TrataErro(1.0008,e.getMessage());
            return null;
```

```
            }
            paramsLoginsAndPasswdsVector=new Vector();
            loginsEnumeration=userPasswdHastable.keys();
            while (loginsEnumeration.hasMoreElements()) {
              login=loginsEnumeration.nextElement().toString();
              passwd=userPasswdHastable.get(login).toString();
              loginsandPasswdString=loginsandPasswdString+login+":"+passwd+":";
            }
            paramsLoginsAndPasswdsVector.addElement (loginsandPasswdString);

GroupMembershipServiceAdv.setName(grpMembershipServiceAdv.getName());

GroupMembershipServiceAdv.setCode("net.jxta.impl.membership.PasswdMembership");

GroupMembershipServiceAdv.setKeywords(grpMembershipServiceAdv.getKeywords());
            GroupMembershipServiceAdv.setParams(paramsLoginsAndPasswdsVector);
            GroupMembershipServiceAdv.setPipe(grpMembershipServiceAdv.getPipe());
            GroupMembershipServiceAdv.setProvider
(grpMembershipServiceAdv.getProvider());
            GroupMembershipServiceAdv.setSecurity
(grpMembershipServiceAdv.getSecurity());
            GroupMembershipServiceAdv.setUri (grpMembershipServiceAdv.getUri());

GroupMembershipServiceAdv.setVersion(grpMembershipServiceAdv.getVersion());
            dumbObject                                                    =
GroupAdvServiceHashtable.put(membershipServiceKey,GroupMembershipServiceAdv);
            GroupAdv.setServiceAdvertisements(GroupAdvServiceHashtable);*/
            return GroupAdv ;
        }

        private void completeAuth (Authenticator auth) throws Exception{
            Method [] methods = auth.getClass().getMethods();
            Vector authMethods = new Vector();
            for( int eachMethod = 0; eachMethod < methods.length; eachMethod++ ) {
                if( methods[eachMethod].getName().startsWith("setAuth") ){
                  if( Modifier.isPublic( methods[eachMethod].getModifiers())){
                  for( int doInsert = 0; doInsert <= authMethods.size();
                                                  doInsert++ ) {
                        int insertHere = -1;
                        if( doInsert == authMethods.size() )
                            insertHere = doInsert;
                        else {
                            if(methods[eachMethod].getName().compareTo(
      ((Method)authMethods.elementAt( doInsert )).getName()) <= 0 )
                                insertHere = doInsert;
                        } // end else
```

```
                if(-1!= insertHere ) {
                    authMethods.insertElementAt(
                            methods[eachMethod],insertHere);
                    break;
                } // end if ( -1 != insertHere)
              } // end for (int doInsert=0
            } // end if (modifier.isPublic
          } // end if (methods[eachMethod]
        } // end for (int eachMethod)
          Object [] AuthId = {"rootpeers"};
          Object [] AuthPasswd = {"JLXH"}; // encrypted RULE
        for( int eachAuthMethod=0;eachAuthMethod<authMethods.size();
                                        eachAuthMethod++ ) {
          Method doingMethod = (Method) authMethods.elementAt(eachAuthMethod);
          String authStepName = doingMethod.getName().substring(7);
          if (doingMethod.getName().equals("setAuth1Identity")) {
             // Found identity Method, providing identity
             doingMethod.invoke( auth, AuthId);
          }
          else if (doingMethod.getName().equals("setAuth2Password")){
             // Found Passwd Method, providing passwd
             doingMethod.invoke( auth, AuthPasswd );
          }
        }
}
```

```
}


package p2p_java;

import net.jxta.discovery.DiscoveryService;
import net.jxta.exception.PeerGroupException;
import net.jxta.protocol.PipeAdvertisement;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;
import net.jxta.pipe.PipeService;
import net.jxta.protocol.PeerGroupAdvertisement;
import net.jxta.document.AdvertisementFactory;
import net.jxta.document.MimeMediaType;
import net.jxta.exception.PeerGroupException;
import net.jxta.protocol.ModuleClassAdvertisement;
import net.jxta.document.Element;
import net.jxta.document.StructuredDocument;
import net.jxta.document.StructuredDocumentFactory;
import net.jxta.document.StructuredDocumentUtils;
import net.jxta.document.StructuredTextDocument;
import net.jxta.endpoint.Message;
import net.jxta.endpoint.MessageElement;
import net.jxta.id.ID;
import net.jxta.id.IDFactory;
import net.jxta.pipe.InputPipe;
import net.jxta.platform.ModuleClassID;
import net.jxta.protocol.ModuleClassAdvertisement;
import net.jxta.protocol.ModuleSpecAdvertisement;

import java.io.IOException;

import java.io.*;
import java.util.*;
import java.lang.Integer;

import p2p_java.*;
```

```
public class Principal {

private PeerGroup pGroup = null;
private PeerGroup pGroupRoot = null;
private Peer_p2p peer_p2p=new Peer_p2p();

//-----------------------------------------------------------------------------------------
  public Vector conectar(){
      //inicializando JXTA

     Vector vec=new Vector();
     vec=this.IniciarRoot();
     return vec;
 }
//-----------------------------------------------------------------------------------------

   public Vector desconectar(){
      //parando jxta
      Vector vec=new Vector();
      vec = peer_p2p.pararPlataform(pGroupRoot);
      if (vec.elementAt(0)!=null){
         pGroupRoot.stopApp();
         pGroupRoot=null;
         return vec;
      }else{
         return vec;
      }
   }
//-----------------------------------------------------------------------------------------
   public Vector deixarPeerGroup(String Nome){
     //abandonando grupo
      //acessando peer Group
     Vector vStatus=new Vector();
     Vector vRetorno=new Vector();
    // vRetorno.add(null);
    //  vRetorno.add(null);
     vStatus.add(null);
     vStatus.add("Buscando PeerGroup '" + Nome + "'...");
     PeerGroup PG=peer_p2p.buscarPeerGroup(Nome,pGroupRoot);
     if (PG!=null){
       try{
          Vector v=peer_p2p.deixarPeerGroup(PG,vStatus);
          vRetorno.add(v);
          if (v.elementAt(0)!=null){
             pGroup=pGroupRoot;
          }else{
```

```
                vStatus.add("Falha ao tentar abandonar PeerGroup '" + Nome + "'.");
            }
          }catch(Exception e){
            vStatus.add("Falha ao tentar abandonar PeerGroup '" + Nome + "'.");
          }
          vRetorno.add(vStatus);
          return vRetorno;
       }else{
          System.out.println("falha ao buscarrr");
          vStatus.add("Falha ao localizar PeerGroup '" + Nome + "'.");
          vRetorno.add(null);
          vRetorno.add(vStatus);
          return vRetorno;
       }

    }
//-------------------------------------------------------------------------------------------
    public Vector entrarPeerGroup(PeerGroup peerGroup,Vector vec){
        //entrando peer Group
        vec=peer_p2p.entrarPeerGroup(peerGroup,vec);
        pGroup=(PeerGroup)vec.elementAt(0);
        return vec;
    }
//-------------------------------------------------------------------------------------------
    public Vector acessarPeerGroup(String Nome){
        //acessando peer Group
       Vector vStatus=new Vector();
       Vector vRetorno=new Vector();
       vStatus.add(null);
       vStatus.add("Buscando PeerGroup '" + Nome + "'...");
       PeerGroup PG=peer_p2p.buscarPeerGroup(Nome,pGroupRoot);
       if (PG!=null){
          try{
             System.out.println("retornou valor e vou entrar");
             Vector v=peer_p2p.entrarPeerGroup(PG,vStatus);
             vRetorno.add(v);
             if (v.elementAt(0)!=null){
                System.out.println("coloquei novo peergroup");
                pGroup=(PeerGroup)v.elementAt(0);
             }else{
                vStatus.add("Falha ao tentar acessar PeerGroup '" + Nome + "'.");
             }
          }catch(Exception e){
             vStatus.add("Falha ao tentar acessar PeerGroup '" + Nome + "'.");
          }
          vRetorno.add(vStatus);
```

```
        return vRetorno;
      }else{
        vStatus.add("Falha ao localizar PeerGroup '" + Nome + "'.");
        vRetorno.add(null);
        vRetorno.add(vStatus);
        return vRetorno;
      }
    }
//-------------------------------------------------------------------------------------
    public Vector criarPeerGroup(String NGroup,String descGroup){
      Vector vec=new Vector();
      vec = peer_p2p.criarPeerGroup(NGroup,descGroup,pGroup);
      return vec;
    }
//-------------------------------------------------------------------------------------
    public Vector getPeerGroups(){
        //buscando ADV de grupos
        int i;
        Vector vec=peer_p2p.buscarPeerGroups(pGroup);
        Vector v=peer_p2p.buscarPeers(pGroup);
        for (i=0;i<v.size();i++){
            vec.add(v.elementAt(i));
        }
        return vec;
    }

//-------------------------------------------------------------------------------------
   public Vector IniciarRoot(){
       //iniciando o sistema e criando o peergroup root
       Vector vec=new Vector();

       Vector v=peer_p2p.RootPeerGroup();
       pGroupRoot=(PeerGroup)v.elementAt(0);
       pGroup=pGroupRoot;
       if (pGroupRoot!=null){
         //acessando peer group root
         vec.add(this.entrarPeerGroup(pGroupRoot,v));
         vec.add(getPeerGroups());
         return vec;
       }else{
         vec.add(v);
         vec.add("null");
         return vec;
       }
   }
//-------------------------------------------------------------------------------------
```

```
  public String getPeerGroupName(){
    return pGroup.getPeerGroupName();
  }
//-------------------------------------------------------------------------------------------

  public String buscarADV(int tipoADV){
     return peer_p2p.buscarADV(pGroup,tipoADV);
  }
//-------------------------------------------------------------------------------------------
 public void flush(){
     peer_p2p.flush(pGroup);
 }
//-------------------------------------------------------------------------------------------
 public void teste(){

    //  peer_p2p.teste(pGroup);

 }
//-------------------------------------------------------------------------------------------

  public Vector getInfoPeerGroup(){
      Vector vec=new Vector();
      vec = peer_p2p.getInfoPeerGroup(pGroup);
      return vec;
   }

}
```