UNIVERSIDADE TIRADENTES
CENTRO DE CIÊNCIAS FORMAIS E TECNOLOGIA – CCFT
CURSO DE TECNOLOGIA EM PROCESSAMENTO DE DADOS

INTEROPERABILIDADE ENTRE SERVIÇOS UTILIZANDO SOAP

Orientador: Methanias Colaço Rodrigues Júnior
Co-orientador: Marco Antônio Simões

Wilhelm de Araújo Rodrigues

Projeto Supervisionado em Processamento de Dados

Aracaju(SE)-Brasil
2001

UNIVERSIDADE TIRADENTES
CENTRO DE CIÊNCIAS FORMAIS E TECNOLOGIA – CCFT
CURSO DE TECNOLOGIA EM PROCESSAMENTO DE DADOS

INTEROPERABILIDADE ENTRE SERVIÇOS UTILIZANDO SOAP

Monografia apresentada ao Centro de Ciências Formais e Tecnologia da Universidade Tiradentes – UNIT, como requisito para conclusão do curso de graduação e obtenção do título de Tecnólogo em Processamento de Dados.

Orientador: Methanias Colaço Rodrigues Júnior

Co-orientador: Marco Antônio Simões

Wilhelm de Araújo Rodrigues

Projeto Supervisionado em Processamento de Dados

Aracaju(SE)-Brasil
2001

DATA _____ / _____ / _____

BANCA

(1° Examinador) _____

(2° Examinador) _____

(3° Examinador) _____

Dedico esta monografia à memória do meu pai, à minha mãe, pela força e confiança nas minhas decisões, à minha família que nunca deixou de acreditar no meu sucesso e à minha namorada pelo incentivo e paciência.

Wilhelm de Araújo Rodrigues

AGRADECIMENTOS

Aos colegas de curso, que se tornaram grandes amigos compartilhando comigo desta caminhada.

Aos professores, que se mostraram bastante sensíveis aos problemas da vida acadêmica.

Ao meu orientador, prof. Methanias Colaço, e ao meu co-orientador, prof. Marco Simões, pela coragem e disposição em participar desta monografia guiando cada passo deste trabalho, me concedendo parte do seu valioso tempo.

A todos que confiaram e torceram por mim.

Muito obrigado.

SUMÁRIO

LISTAS

LISTA DE FIGURAS

LISTAS DE TABELAS

RESUMO

As arquiteturas de desenvolvimento de sistemas distribuídos têm amadurecido bastante nos últimos anos. Este fato se deve principalmente pela adesão das grandes corporações, que perceberam na tecnologia de objetos distribuídos o novo paradigma no desenvolvimento de sistemas. Tais arquiteturas se baseiam na sua estrutura, nos serviços que oferecem e nos protocolos de comunicação que utilizam. Um fator que favorece a difusão de uma arquitetura de objetos distribuídos é a sua capacidade de interagir com uma outra arquitetura através de protocolos de comunicação apropriados. Um dos grandes desafios da atualidade para as arquiteturas de objetos distribuídos é permitir a interoperabilidade de sistemas disponíveis na Internet, através de estruturas de segurança (*firewalls*), independente da linguagem de programação na qual tenham sido escritos. *Web Services* tem sido considerada a tecnologia que representa a evolução das arquiteturas de objetos distribuídos e se baseia em um protocolo leve, de especificação não proprietária, baseado em XML (Extended Markup Language), chamado SOAP (*Simple Object Access Protocol*). Este protocolo, como os seus antecessores, possui uma série de vantagens e limitações, e tem como principal característica a capacidade de permitir a integração de sistemas distribuídos através da *Web*, formando serviços capazes de interagir de forma transparente para fornecer aos usuários funcionalidades específicas.

INTRODUÇÃO

O advento da internet indubitavelmente tocou de forma marcante a vida de uma parcela significante da população mundial, influenciando na comunicação, educação, lazer, comércio e finalmente no desenvolvimento de sistemas de informação, este último de forma particular. Informações privadas de pessoas e empresas passaram a ficar disponíveis para o acesso público de forma livre em conteúdo e padronizada na sua estrutura de exibição. Em poucos anos a festejada possibilidade inicial de acesso rápido a conteúdos estáticos em qualquer parte do mundo evoluiu rapidamente para um grande número de sistemas dinâmicos capazes de receber e produzir informações de e para os seus usuários, sendo denominados "*sistemas para internet*" ou "*via web*".

Recentemente uma nova revolução teve lugar na área de desenvolvimento de sistemas voltados para a internet. O termo "*web services*" surgiu a princípio para

denominar uma nova arquitetura de sistemas voltados para a internet, com especificação inicialmente proprietária da Microsoft, mas vem crescentemente sendo usado como sinônimo para denominar serviços disponíveis via internet, capazes de interagir entre si trocando informações de forma padronizada sem requerer para isso nenhuma intervenção humana. Para garantir a perfeita interação, ou interoperabilidade, entre dois ou mais *web services* a troca de informações entre estes deve obedecer a uma série de padrões.

Responsável por garantir a perfeita comunicação entre os sistemas, os protocolos de comunicação desempenham papel fundamental servindo de sustentação para a arquitetura de desenvolvimento. No caso dos *web services* o protocolo de comunicação a ser utilizado necessitaria de características particulares para lhe dar possibilidades de implementação capazes de superar as limitações de outros protocolos de função semelhante já existentes. Esse novo protocolo precisaria garantir entre outros recursos leveza, flexibilidade, facilidade de implementação e depuração. Para isso foi criado o SOAP (*Simple Object Access Protocol*). SOAP é o protocolo criado nestes moldes para, mais que outro protocolo de comunicação, ser uma nova tecnologia na qual se baseia um novo paradigma no desenvolvimento de sistemas distribuídos, garantindo a interoperabilidade de serviços heterogêneos, remotamente distribuídos, com todos os requisitos necessários.

Ao longo deste trabalho serão apresentadas algumas arquiteturas de objetos distribuídos, os protocolos de comunicação que utilizam, e finalmente, com destaque especial, será apresentado o protocolo SOAP, abordando-se também de maneira

introdutória a arquitetura de desenvolvimento na qual está inserido. Serão ressaltados seus recursos, indicadas algumas das suas principais formas de utilização e expostas algumas das principais ferramentas de desenvolvimento utilizando SOAP. Serão também indicados os principais projetos desenvolvidos por grandes empresas e alguns ambientes de desenvolvimento utilizando softwares comerciais e livres.

1 WEB SERVICES

1.1 O que são *web services*.

Hoje, as duas tecnologias mais populares usadas na Internet são correio eletrônico e www. Nos próximos anos provavelmente veremos uma outra tecnologia da Internet tornar-se muito importante: “*web services*". Alguns autores, como Matt Reynolds, costumam definir *web sites* como sendo estruturas projetadas basicamente para compartilhar informações e fornecer a interação entre uma pessoa e uma organização, e *web services* como sendo uma evolução onde a interação humana passa a não ser necessária no processo de obtenção das informações. [REYNOLDS 2001ª]. Porém, uma abordagem em um maior nível de detalhamento nos leva à idéia principal dos *web services*, que é fornecer aos desenvolvedores de sistemas distribuídos uma solução para o problema da comunicação entre os

componentes de uma aplicação distribuída através de redes distintas de computadores que possuem no ponto de interconexão, dispositivos de segurança, como *firewalls*, que permitem apenas o tráfego de informações em formatos e de maneiras previamente determinadas. Fornecer uma estrutura capaz de solucionar este problema no caso específico da Internet, utilizando padrões abertos extremamente conhecidos e de alta aceitação é o que torna os *web services* uma tecnologia com grande possibilidade de expansão.

O ato de buscar na Internet alguns tipos de informação pode ser uma tarefa extremamente cansativa e desestimulante. Estes dois últimos adjetivos ainda conseguem fazer com que os resultados obtidos nem sempre sejam os melhores. Podemos citar algumas situações: Obter informações sobre um produto, como por exemplo quais as companhias que fornecem o tal produto por qual preço e com quais condições de pagamento, ou ainda informações sobre serviços, como por exemplo quais hospitais existem dentro de uma determinada região geográfica e que sejam capazes de atender uma urgência em uma determinada especialidade médica. Esta tarefa pode ser facilitada pela utilização de *web services*, onde podemos fornecer simplesmente o nome do produto ou serviço e receber toda a sorte de informações a respeito do objeto da pesquisa. Ao invés do usuário pesquisar dezenas de *sites web* à procura da informação que precisa, o *web service* se encarrega dessa tarefa removendo a necessidade de interação humana.

Além de possibilitar que as informações fornecidas possam transpor sem dificuldade as barreiras dos *firewalls,* é preciso garantir que cada componente da aplicação que irá receber a informação saiba que tipo de informação está recebendo

e como essa informação pode ser lida. Para realizar essa tarefa os *web services* precisam fornecer além das informações, informações sobre essas informações. Isso é feito através do uso de XML (*Extended Markup Language*) [BRAY 2000], que é uma linguagem específica para a marcação de dados. Assim, encapsulado em um documento XML, além da informação enviada ou requerida, acompanham a mesma os seus *meta-dados,* que são as informações sobre a informação. Além disso, a própria definição das funcionalidades disponíveis em um *web service* fica registrada em XML.

"Os dados sobre dados, meta-dados, são uma das fundações-chave que tornam possível a criação dos web services"[REYNOLDS 2001[b]].

Segundo Joe Johnston, *web services* são uma maneira para programas em uma máquina usarem os recursos de uma outra máquina, por exemplo, através do uso de chamadas a procedimentos remotos (RPC). Embora o RPC binário esteja em evidência por décadas, os *web services* codificam suas conversações em XML e falam genericamente através do protocolo base da *Web*, HTTP [RAGGET 1998] (*Hyper Text Transfer Protocol*). Desta maneira, os *web services* são bem mais fáceis de compreender e eliminar erros do que o RPC tradicional. Como cada linguagem que quer se comunicar com um *web service* deve "falar" XML, os clientes escritos em uma linguagem podem fazer requisições aos *web services* onde os programas do lado servidor tenham sido escritos em uma linguagem diferente [JOHNSTON 2001]. Para prover a funcionalidade de disponibilizar serviços capazes de interagir entre si através da internet, além do XML, os *web services* utilizam outros padrões adicionais, para definição e registro dos recursos a serem disponibilizados. São o

WSDL(*Web Services Description Language*) [CHRISTENSEN 2001], SOAP (Simple Object Access Protocol) [GUDGIN 2001], UDDI (Universal Discovery and Description Integration) [UDDI 2000] e WSFL (Web Services Flow Language) [LEYMANN 2001], e serão abordados a seguir.

1.2 Utilização.

Segundo Matt Reynolds os *web services* podem ser usados para criar novas oportunidades de negócio, ou para oferecer alguns serviços a um custo-benefício superior ao de seus concorrentes [REYNOLDS 2001[b]].

Para ilustrar a utilização de *web services,* vamos considerar o cenário de uma grande empresa distribuidora de medicamentos, que possui um escritório central para administração e departamento comercial, e com depósitos nas principais capitais do país. A empresa em si possui uma grande rede subdividida nas unidades que possui. Cada um dos depósitos da empresa representa uma estrutura autônoma cuja funcionalidade é receber os produtos dos grandes laboratórios, armazená-los e providenciar a entrega dos pedidos feitos pelos clientes à unidade administrativa central, reportando toda a movimentação à mesma.

Quando um cliente deseja adquirir um conjunto de medicamentos, este faz uma busca através de servidores UDDI [UDDI 2000], que são responsáveis pelo registro de *web services*, procurando quais empresas são capazes de lhe fornecer os

medicamentos que deseja. Uma vez selecionado um *web service,* através da especificação WSDL daquele *web service,* é possível saber qual a maneira de interagir com o mesmo a fim de obter a informação que deseja. O cliente informa então os produtos que deseja adquirir e aguarda as informações relativas aos mesmos, como por exemplo, preço e condições de pagamento. O cliente então pode providenciar o seu cadastramento junto à empresa para que possa efetuar o seu pedido de compra. Sendo o cliente já previamente cadastrado junto à companhia, ele pode simplesmente enviar a sua identificação, os produtos e quantidades que deseja adquirir, recebendo então a informação de quando receberá no seu endereço os itens do pedido efetuado.

Na estrutura interna da empresa o mecanismo funciona de forma análoga. A empresa possui um servidor UDDI onde estão registrados, através de WSDL, os *web services* internos que possui. Esse servidor é acessado para descobrir quais os *web services* serão utilizados para atender a solicitação do cliente. Por exemplo, é necessário saber quais depósitos da empresa possui os medicamentos que o cliente solicitou e em quanto tempo é capaz de realizar a entrega, total ou parcial, uma vez que a entrega pode ser dividida entre dois depósitos, caso um deles não possua os produtos em quantidades suficientes. Essa entrega será ainda reportada ao setor administrativo para que sejam efetuados os procedimentos devidos, como por exemplo, o acompanhamento das entregas das unidades, para facilitar a avaliação do desempenho de cada uma delas num determinado período.

É importante lembrar que, como cada unidade é autônoma, ou seja, possui sistemas internos próprios, os *web services* que existem em cada uma das unidades

devem interagir quando necessário com os sistemas internos de cada unidade. Estes sistemas corporativos internos podem utilizar outras tecnologias de objetos distribuídos como, por exemplo, CORBA [OMG 2001] e RMI [RMI 1999]. Existem atualmente importantes pesquisas sendo feitas sobre esse tipo de integração.

Em se tratando de uma empresa pequena, onde todos os setores estão fisicamente próximos, esta teria apenas o seu sistema interno e este iria interagir com o seu *web service* responsável por atender os pedidos dos clientes.

Cenários mais simples podem ser citados. No ambiente comercial é muito comum e uma tendência definitiva o pagamento de contas através de cartões de crédito. Operadoras de cartões de créditos poderiam publicar em servidores UDDI os seus *web services*, através de WSDL, responsáveis por validar o número e a disponibilidade para compra dos cartões de crédito de seus clientes. Neste caso, um comerciante poderia usar um *web service* que recebesse um número de cartão do crédito e o valor da transação, validasse se este cartão pertence realmente ao cliente, e se tem permissão e limite para que a transação seja efetuada, retornando então "sim ou não", ou uma informação equivalente.

Para os desenvolvedores que desejam desenvolver programas capazes de interagir com *web services* existe a necessidade de buscar o *web service* através de servidores UDDI, obter as funcionalidades do *web service* desejado através da sua especificação WSDL, que servirá como interface entre o desenvolvedor e o *web service*, e escrever o código na linguagem com a qual estiver mais familiarizado.

1.3 Perspectivas.

Alguns autores afirmam que os *web services* têm potencial para ser uma tecnologia revolucionária, tanto para tradicionais fabricantes de softwares proprietários quanto para os desenvolvedores que optam pelo código aberto. Baseiam suas afirmações chamando a atenção para o início de algumas recentes mudanças que estão ocorrendo como as plataformas de hardware, onde os PC's tradicionais parecem estar sendo deixados um pouco de lado e as atenções estão se voltando para os dispositivos especializados em uma única finalidade que podem executar aplicações múltiplas, como por exemplo, *palmtops* ou dispositivos do tipo *handheld*. Como característica marcante entre as potencialidades destes dispositivos está o suporte a conexões de rede presente na esmagadora maioria destes dispositivos. Em função deste novo cenário que se forma toma força a idéia que, uma vez que ninguém gosta das despesas gerais administrativas para instalar e manter aplicações localmente hospedadas, alugar o software torna-se particularmente atraente para os negócios e também para os usuários ocasionais de computadores caseiros [JOHNSTON 2001].

Segundo Joe Johnston, um fato importante a considerar quando pensamos sobre as perspectivas dos *web services* é que em um mundo em que grandes empresas compram funcionalidades de software de "empresas virtuais" de *web services*, a marca do fornecedor está se tornando cada vez mais irrelevante enquanto o grau de funcionalidade e interoperabilidade dos serviços toma o papel principal. Os vendedores do sistema operacional, por exemplo, terão uma estadia

particularmente difícil neste ambiente, uma vez que é nas aplicações que funcionam naqueles sistemas que os consumidores estão interessados. Se uma aplicação funcionar igualmente bem em dois sistemas operacionais diferentes, provavelmente os fabricantes de hardware serão atraídos cada vez mais às variações de código aberto do UNIX [JOHNSTON 2001].

Todos estes argumentos são altamente relevantes, porém se observarmos os *web services* através da sua idéia inicial, veremos que no futuro o grande fator que firmará os *web services* de forma definitiva no cenário das tecnologias de sistemas distribuídos é a possibilidade de integração com outras tecnologias existentes e já consolidadas comercialmente, como CORBA e RMI. Tais tecnologias têm estado presentes no mercado por anos, representam atualmente considerável fração do parque de sistemas distribuídos instalados nas corporações e não devem ser substituídos em um futuro próximo. Assim sendo, as pesquisas atualmente em andamento que tratam da integração entre estas tecnologias através de *wrappers* - que são estruturas de código capazes de converter requisições feitas através de *web services,* para chamadas em CORBA ou RMI, ou o oposto - sem dúvida terão papel definitivo no futuro dos *web services*.

1.4 Estrutura dos *web services*

Algumas empresas como, por exemplo, a Microsoft e a IBM, tem empregado grandes esforços na consolidação da arquitetura para desenvolvimento dos *web*

services. Segundo Greg Flurry, a *figura 1* ilustra a visão da IBM sobre a pilha de um *web service* [FLURRY 2001]. Esta estrutura se torna difundida devido à utilização de padrões existentes e emergentes para facilitar e promover a interoperabilidade e disponibilidade dos serviços.

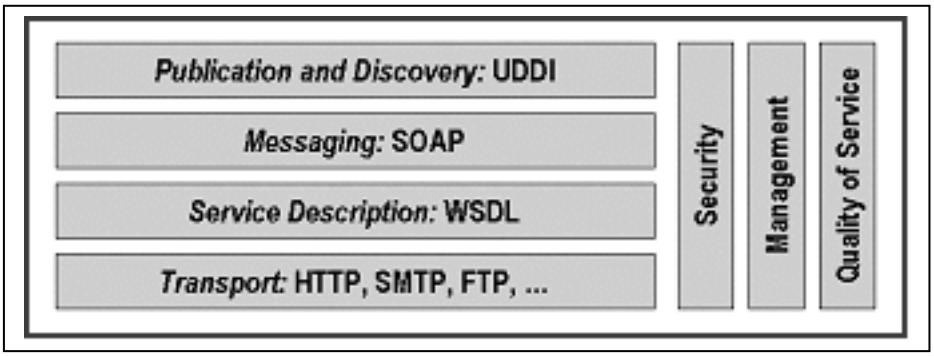

Figura 1: Visão da IBM sobre a pilha de um *web service*

Inicialmente é possível afirmar que os *web services* são a evolução natural das chamadas a procedimentos remotos. De fato, os *web services* em si não são uma tecnologia muito mais avançada do que o RPC(*Remote Procedure Call*) do passado, mas conceitos similares aplicados de novas maneiras.

Segundo Al Saganich [SAGANICH 2001], de um modo geral, os *web services* são uma tecnologia que poderá revolucionar a maneira como os serviços *business-to-business* e *business-to-consumer* podem ser fornecidos, pois usam uma variedade de tecnologias para permitir que duas aplicações comuniquem-se, sendo que, entretanto, nenhuma destas tecnologias é considerada uma inovação. O que faz os *web services* diferentes de outros mecanismos similares são as tecnologias que fornecem o serviço.

De fato, as tecnologias utilizadas para o fornecimento dos serviços internos que constituem a estrutura dos *web services* se baseiam em padrões extremamente conhecidos e difundidos, sendo que agora sendo usados de maneira diferente.

> "Os *web services* têm, em seu núcleo, XML como um mecanismo para comunicação e, finalmente, são baseados em três tecnologias específicas: um mecanismo para registar um serviço, um mecanismo para encontrar um serviço, e um mecanismo para que duas partes comuniquem-se" [SAGANICH 2001].

Qualquer linguagem que possua uma biblioteca ou ferramenta adequada pode ser utilizada para a produção de *web services*. Hoje, muitos desenvolvedores têm optado por utilizar as API's *Java 2 Enterprise Edition*(J2EE) [SHANNON 2001] e XML para disponibilizar *web services* na Internet. Estas ferramentas estimulam o desenvolvimento fornecendo métodos simples para estender, interconectar e publicar aplicações existentes baseadas em J2EE.

Fundamentalmente os padrões utilizados para o desenvolvimento de *web services* abrangem inicialmente, mas não unicamente, 3 (três) funcionalidades básicas:

- Uma maneira de encontrar e registar um serviço.
- Um mecanismo de transporte para alcançar um serviço.
- Uma maneira de definir quais os parâmetros de entrada e de saída para tal serviço.

A combinação dessas funcionalidades fornece uma nova estrutura para o ambiente da computação distribuída. Porém, o que torna os *web services* tão interessantes e atraentes é o fato da sua estrutura permitir que implementações em linguagens diferentes sendo executadas em servidores de fornecedores distintos podendo interagir sem precisar levar em conta como cada uma das implementações foi escrita.

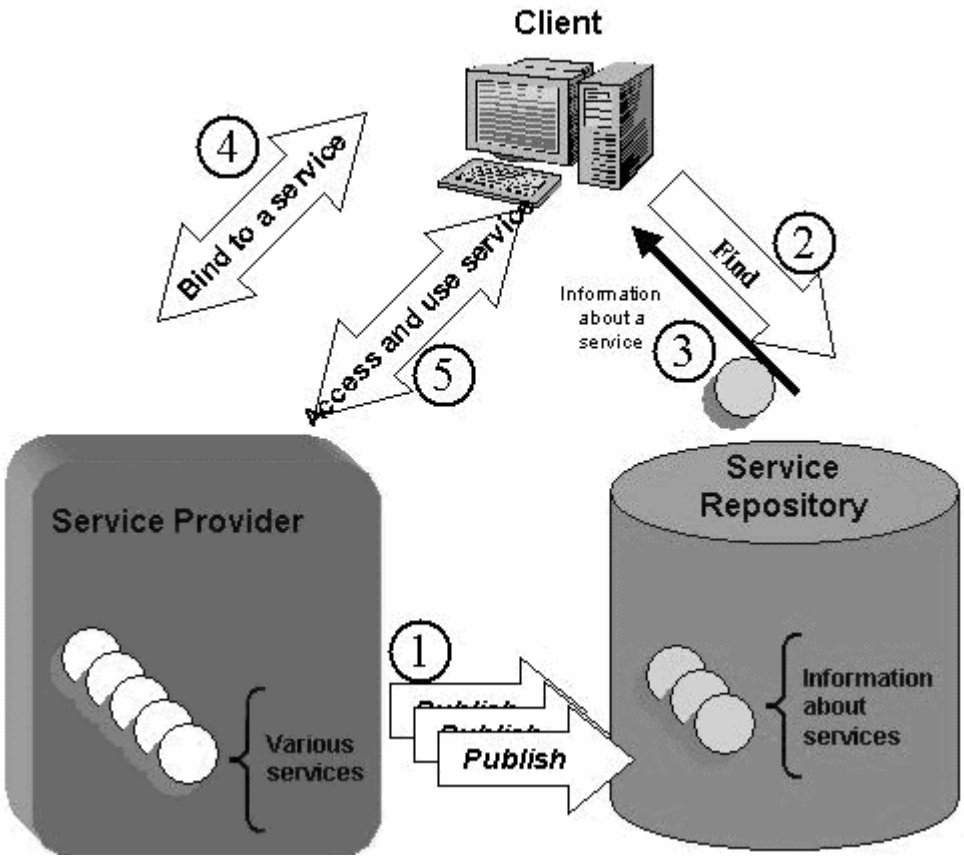

Figura 2: Visão geral de um *web service*

A *figura 2* mostra um diagrama que representa a estrutura de um *web service.*

O primeiro e mais importante componente é o fornecedor de serviço (*service provider).* Os fornecedores de serviço hospedam vários serviços, alguns dos quais são expostos como *web services*. O segundo componente é o repositório de

serviços (service *repository)* responsável pela publicação da informação que os clientes podem usar para encontrar a informação sobre serviços publicados na *Web*. Finalmente, são necessários vários mecanismos para encontrar e alcançar os serviços. Mais adiante serão abordados cada um dos padrões utilizados pelos *web services*. A seqüência de eventos ilustrada na *figura 2* representa:

A publicação de um serviço por um fornecedor em um repositório externo para que, uma vez registrado, este serviço possa ser utilizado por um cliente.

A busca por um serviço em um repositório com a obtenção das informações sobre o serviço desejado, tais como, por exemplo, o formato das chamadas de procedimento e o endereço do fornecedor de serviço.

A ligação do cliente ao serviço procurado para utilizar as funcionalidades que o mesmo oferece.

Toda esta estrutura é importante e faz com que os *web services* sejam o que são. Entretanto, é necessário olhar mais de perto os mecanismos subjacentes que fornecem a base para os *web services* e porque são fundamentalmente diferentes das tecnologias atuais utilizadas nos sistemas distribuídos. Estes mecanismos são representados pelos padrões abordados a seguir. São o XML, WSDL e UDDI.

1.5 A linguagem XML

Os *web services* têm no seu núcleo um conjunto fundamental de funcionalidades baseado em XML. XML é uma linguagem extensível cuja principal finalidade é a marcação de dados, e por isso é uma tecnologia amplamente utilizada hoje para a padronização e transmissão de mensagens. Uma necessidade básica para a comunicação entre dois sistemas é que seus dados ou mensagens sejam claramente definidos e delimitados. A transmissão de mensagens e dados entre componentes de sistemas distribuídos tem sido uma área em que existe uma grande dificuldade quando existe a necessidade de interoperabilidade entre sistemas diferentes. Dados precisam ser colocados no canal de comunicação de uma forma tal que ambas as extremidades da transmissão possam compreender e ler a informação. Quando os componentes de um sistema pertencem à mesma arquitetura (CORBA , por exemplo) e estão registrados dentro da mesma estrutura de rede, a transmissão de dados não é um problema considerável, uma vez que arquiteturas como CORBA e RMI possuem mecanismos que desempenham bem essa função. Porém quando aplicações de arquiteturas diferentes precisam “conversar” ou mesmo quando aplicações de mesma arquitetura precisam trocar dados através de estruturas de segurança, como *firewalls,* ou ainda quando aplicações de uma arquitetura precisam interagir com aplicações cuja estrutura é desconhecida pelo desenvolvedor, é fundamental que as informações trocadas entre as aplicações estejam padronizadas e em um formato muito conhecido e bem definido.

XML surgiu como uma boa solução fornecendo uma representação clara de dados, bem como um conjunto de validações e de regras bem definidas. Como resultado, XML tornou-se um veículo excelente para embalagem de dados de

maneira que ambos os lados da transmissão possam facilmente ler e compreender. XML tem seus inconvenientes; para o exemplo, é muito eloqüente, ou seja, transmitir informações contidas em um XML provoca uma utilização da rede muito grande, e essa característica apesar de ser um preço particularmente pequeno a pagar com as atuais redes de alta velocidade, deve ser considerada seriamente a depender do tipo de aplicação que se pretende desenvolver. Se a utilização da rede é um ponto crítico da aplicação, a utilização de XML na comunicação deve ser avaliada de forma extremamente criteriosa.

1.6 *Web Services* Description Language (WSDL)

WSDL é a linguagem que utiliza XML para servir como um meio para fornecedores de serviço disponibilizarem suas interfaces de acesso. Estas interfaces, representadas em XML, contém informações sobre todas as funcionalidades do serviço bem como os tipos de dados necessários para a sua utilização, ou seja, tudo que um cliente precisa para saber como utilizar um serviço. A representação dos dados se dá obedecendo a um dos objetivos dos *web services* que é a transparência da implementação. Assim essa forma de representação não está associada a nenhuma linguagem específica de programação. WSDL foi definida então como sendo a linguagem padrão para descrever a interface e a localização de um serviço.

Especificamente WSDL fornece um número de peças-chave de informação:

- A definição do formato das mensagens que são passadas entre dois pontos finais usando seus elementos < *types* > e < *message* > e definições apropriadas de esquemas (*schema*).

- A semântica do serviço: como ele deve ser chamado para fazer uma transmissão síncrona de solicitação e resposta, síncrona apenas de resposta ou comunicação assíncrona.

- O ponto final e o transporte do serviço através do elemento < *service* >: isto é, quem fornece o serviço.

- Uma ligação através do elemento < *binding* >, isto é, como o serviço é alcançado.

A *figura 3* representa um exemplo de um WSDL.

1.7 Universal Discovery and Description Integration (UDDI)

Atualmente para usar um *web service*, um cliente deve primeiro encontrar o serviço, obter a informação sobre como usar o serviço, e entender quem possivelmente disponibiliza o serviço. “A especificação UDDI define um número de serviços de busca destinados a permitir aos clientes encontrarem e obterem informações para acessar um *web service*” [SAGANICH 2001].

UDDI atualmente provê três serviços específicos:

- *Traditional White Pages* para procura de um *web service* pelo nome.
- *Traditional Yellow Pages* para procura de um *web service* pelo tópico.
- *Green Pages* para buscas mais genéricas baseadas nas características de um *web service.*

Fornecedores de serviços de UDDI tipicamente operam um serviço conhecido como UDDI *Business Registry*, ou UBR, que pode ser acessado para publicar e solicitar informações sobre um dado *web service*.

1.8 Simple Object Access Protocol (SOAP)

Uma vez que um *web service* já foi localizado, através de busca em servidores UDDI, e que já se sabe como um *web service* é definido, através do seu WSDL, é necessário entender como alcançar realmente o serviço e fazer uso de suas funcionalidades. Isso é feito através do SOAP

SOAP é o protocolo de comunicação padrão dos *web services*. A especificação SOAP define como podem ser representados diversos tipos de dados através de XML, define também o padrão de representação de mensagens, chamadas RPC e a ligação com o protocolo HTTP. Através do SOAP um cliente pode utilizar as

funcionalidades oferecidas por um *web service* através de RPC padrão ou troca de mensagens.

```
<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
    xmlns:tns="http://example.com/stockquote.wsdl"
    xmlns:xsd1="http://example.com/stockquote.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
       <complexType>
         <all>
           <element name="tickerSymbol" type="string"/>
         </all>
       </complexType>
      </element>
      <element name="TradePrice">
       <complexType>
         <all>
           <element name="price" type="float"/>
         </all>
       </complexType>
      </element>
    </schema>
  </types>
  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>
  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>
  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>
  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
      <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>
</definitions>
```

Figura3: Um exemplo de WSDL

Segundo Al Saganich, de maneira simples SOAP é um protocolo para encapsular uma requisição em um XML usando protocolos de comunicação padrão tais como o HTTP. O poder do SOAP vem do fato de ser simples, fácil de implementar, e bem suportado na indústria [SAGANICH 2001].

Tipicamente, uma chamada SOAP é empacotada no corpo de uma requisição HTTP. O exemplo abaixo é baseado na especificação SOAP da W3C, e mostra como um servidor HTTP deve receber uma chamada SOAP para alcançar um serviço conhecido como *GetLastTradePrice.*

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://example.org/2001/06/quotes"

<env:Envelope xmlns:env="http://www.w3.org/2001/06/soap-envelope" >
<env:Body>
 <m:GetLastTradePrice
    env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"
    xmlns:m="http://example.org/2001/06/quotes">
  <symbol>DIS</symbol>
 </m:GetLastTradePrice>
</env:Body>
</env:Envelope>
```

Figura 4**:** Um exemplo SOAP

O SOAP será abordado de maneira mais detalhada a seguir, no capítulo 3.

1.9 WSFL – Web Services Flow Language

WSFL é uma linguagem baseada em XML para descrição de composições de *web services*. WSFL considera dois tipos de composições de *web services*: o primeiro tipo (modelos de fluxo) especifica o modelo apropriado para uso de uma coleção de *web services*, de modo que a composição resultante descreve uma maneira para alcançar uma meta de negócios particular; tipicamente, o resultado é uma descrição de um processo de negócios. O segundo tipo (modelos globais) especifica o modelo de interação de uma coleção de *web services*; neste caso,o resultado é uma descrição das interações entre os parceiros globais [LEYMANN 2001]. WSFL define através da estrutura de um arquivo XML um padrão de utilização de um conjunto de *web services* tendo como resultado o procedimento para se conseguir utilizar determinada funcionalidade, ou ainda um padrão de interação entre vários *web services*.

2 PROTOCOLOS UTILIZADOS NA CHAMADA A PROCEDIMENTOS REMOTOS

No paradigma cliente-servidor, um servidor, através de mecanismos apropriados, anuncia um conjunto de serviços que fornecem acesso para alguns recursos, tais como, por exemplo, o acesso uma base de dados. O código da implementação desses serviços fica armazenado localmente pelo servidor. É o servidor quem executa o serviço, tendo então a capacidade de processamento. Quando um cliente precisa acessar um recurso fornecido pelo servidor, isso será feito através de requisições para os serviços registrados e oferecidos pelo servidor. O que o cliente precisa é apenas possuir um mecanismo para decidir quais dos serviços ele deve utilizar. O servidor possui então o conhecimento, os recursos e processamento.

A programação com objetos distribuídos estende um sistema de programação orientado a objetos para permitir que objetos possam ser distribuídos através de

uma rede heterogênea, de modo que estes possam interoperar. Esses objetos podem ser distribuídos em computadores diferentes por toda uma rede, existindo fora do espaço local de endereço de uma aplicação, e ainda aparecer como se eles estivessem locais para a aplicação. A distribuição de objetos através da Internet tem merecido atenção especial

A maioria dos sistemas distribuídos tem se baseado no paradigma cliente-servidor. Ele é suportado por tecnologias como: Chamadas Remotas de Procedimento (RPC), *Object Request Brokers* (CORBA [OMG 2001]), da OMG, e a Invocação de Métodos Remotos (RMI [RMI 2001]) da linguagem Java. Recentemente os desenvolvedores têm utilizado uma nova arquitetura de desenvolvimento de sistemas distribuídos, *web services*, baseados no protocolo SOAP[GUDGIN 2001]. Este novo protocolo será abordado no capítulo 3.

A *figura 5* representa o paradigma cliente-servidor.

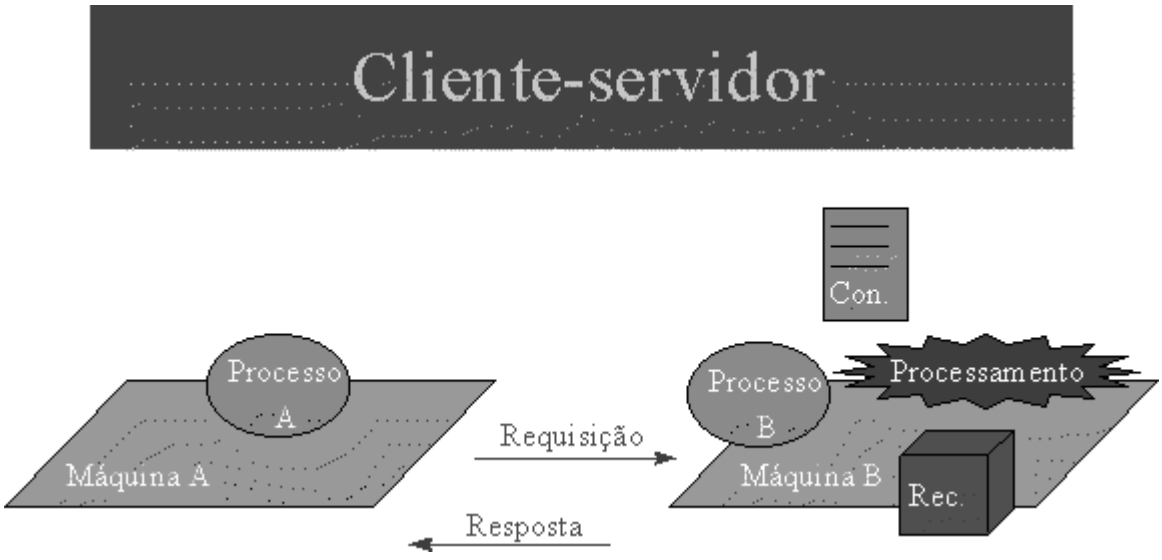

Figura 5: Paradigma Cliente-Servidor

2.1 CORBA.

2.1.1 Arquitetura - visão geral

2.1.1.1 Introdução

CORBA (*Common Object Request Broker Architecture*) é um ambiente para aplicações distribuídas orientadas a objeto. O padrão CORBA foi introduzido em 1991 (versão 1.1) pela OMG - *Object Management Group* (um consórcio de empresas sem fins lucrativos com o objetivo de estabelecer normas e especificações para gerenciamento de objetos para a indústria).

> "A especificação CORBA foi criada pelo *Object Management Group* com o intuito de possibilitar o desenvolvimento de sistemas distribuídos que possam se integrar com os demais sistemas legados da empresa. Isso é obtido pelo fato do CORBA ser uma arquitetura que não é baseada em nenhuma linguagem de programação, diferente de outras tecnologias de objetos distribuídos onde deve-se usar uma solução proprietária de um fabricante" [COSTA 1998].

Segundo Tânia Pagnussat, o principal objetivo da arquitetura CORBA é garantir para o desenvolvedor a idéia de transparência. Essa idéia é formada pelas noções de *Location Transparency* e *Programming Language Transparency*, ou seja, o programador não necessita saber onde o objeto remoto está localizado e em que linguagem de programação foi escrito para poder acessá-lo. As aplicações clientes

que usam um objeto CORBA devem se preocupar apenas em obter uma referência a um desses objetos para que possam usá-lo como se fosse um objeto da sua própria linguagem [PAGNUSSAT 2001]. Podemos assim considerar como principais características desta arquitetura a transparência de linguagem, localização e plataforma , e a interoperabilidade entre ORBs de diferentes fabricantes

CORBA é um padrão para objetos desenvolvidos em sistemas distribuídos. A arquitetura deste padrão possibilita mecanismos pelos quais um objeto pode enviar requerimentos ou receber respostas, para/de outro objeto em sistemas distribuídos heterogêneos, de uma forma transparente como definido pelo ORB padrão do OMG. Utiliza um protocolo chamado o *Internet Inter-Orb Protocol* (IIOP) para objetos remotos e, uma vez que CORBA é somente uma especificação, esta pode ser utilizada em diversas plataformas de sistema operacional, desde *mainframes*, passando por estações UNIX até máquinas com Windows ou dispositivos do *handheld* desde que haja uma implementação do ORB para aquela plataforma.

Os serviços CORBA são especificações definidas pela OMG, através de interfaces IDL (*Interface Definition Language*), que definem serviços em nível de sistema que ajudam a gerenciar e manter objetos. Uma vez implementados, estes serviços podem ser facilmente acoplados às definições de classes dos objetos distribuídos via mecanismos de herança, ou delegação.

2.1.1.2 O Object Request Broker (ORB)

ORB (*Object Request Broker*) - comercialmente conhecido como CORBA - é o "coração" da arquitetura que provê os mecanismos de comunicação entre os objetos. O ORB fornece uma estrutura que permite aos objetos conversarem entre si, independente de aspectos específicos da plataforma e técnicas usadas para implementá-los, ou seja, um cliente pode invocar, transparentemente, um método num objeto servidor, o qual pode estar na mesma máquina ou em qualquer lugar da rede. Aplicações típicas cliente/servidor usam *designs* próprios ou um padrão reconhecido para definir o protocolo a ser usado entre os dispositivos. Definições de protocolos dependem da linguagem de implementação, transporte na rede, localização de objetos e de outros fatores. ORBs simplificam este processo. Com um ORB, o protocolo de rede é abstraído, o usuário se preocupa apenas com a interface com os objetos, como se estes "fossem locais". É claro que os usuários podem definir o protocolo de comunicação entre seus objetos, durante a especificação de suas interfaces. Seguir os padrões da arquitetura CORBA garante portabilidade e interoperabilidade de objetos sobre uma rede de sistemas heterogêneos.

A figura 6 representa graficamente o ORB e seus serviços:

Para que os objetos de linguagens de programação distintas possam se comunicar, é necessário que cada linguagem tenha seu próprio ORB, assim, os objetos terão um meio comum de comunicação. “Quando uma aplicação cliente requisita uma referência para um objeto remoto é responsabilidade do ORB cliente encontrar o objeto remoto no sistema distribuído” [SILBERSCHATZ 2000]. Ainda segundo Silberschatz, o ORB cliente é também responsável por encaminhar

chamadas a métodos remotos para o servidor apropriado e por receber as respostas do servidor.

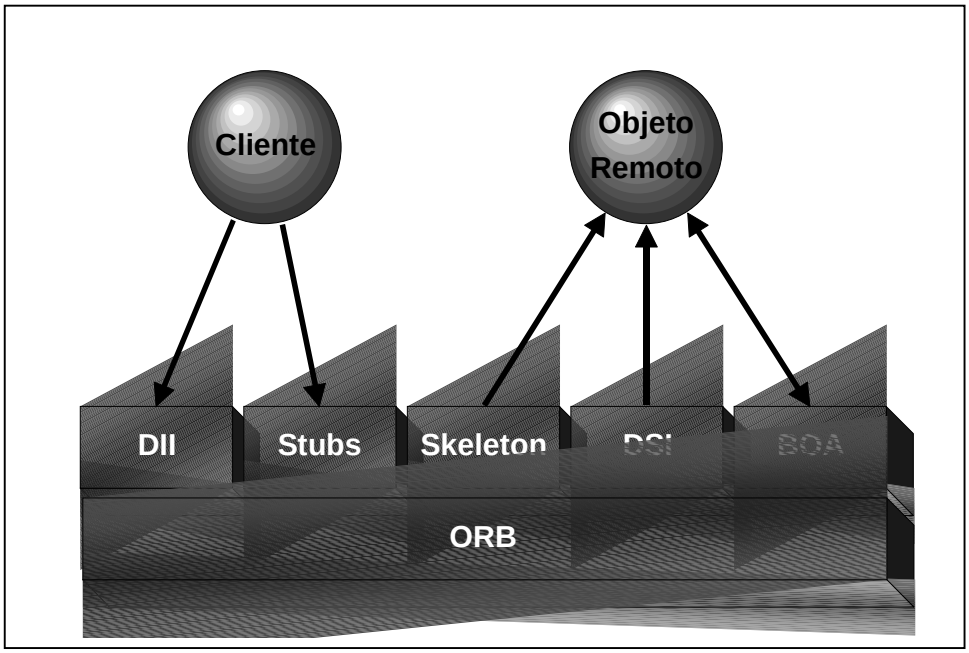

Figura 6: Representação gráfica do ORB e seus serviços

A *figura 7* ilustra a comunicação entre ORBs através do protocolo IIOP, que será abordado ainda neste capítulo. CORBA, porém, suporta outros protocolos além do IIOP. Isto permite a integração com outras arquiteturas de sistemas distribuídos.

Os *stubs* e *skeletons* servirão para fazer o intercâmbio de mensagens entre o cliente e o objeto remoto. Quando uma aplicação faz uma chamada a um método de um objeto remoto, a requisição é passada diretamente e de forma transparente ao objeto *stub* desse objeto que irá transformar a chamada do método e os argumentos em um formato apropriado para que possam ser transmitidos para o espaço de endereço do objeto remoto. Antes do objeto remoto receber a chamada, o seu objeto *skeleton* se encarrega de receber e recompor a requisição para o formato original,

além de refazer todos os passos no sentido inverso para que o retorno do método chegue ao cliente [TECHMETRIX 2000].

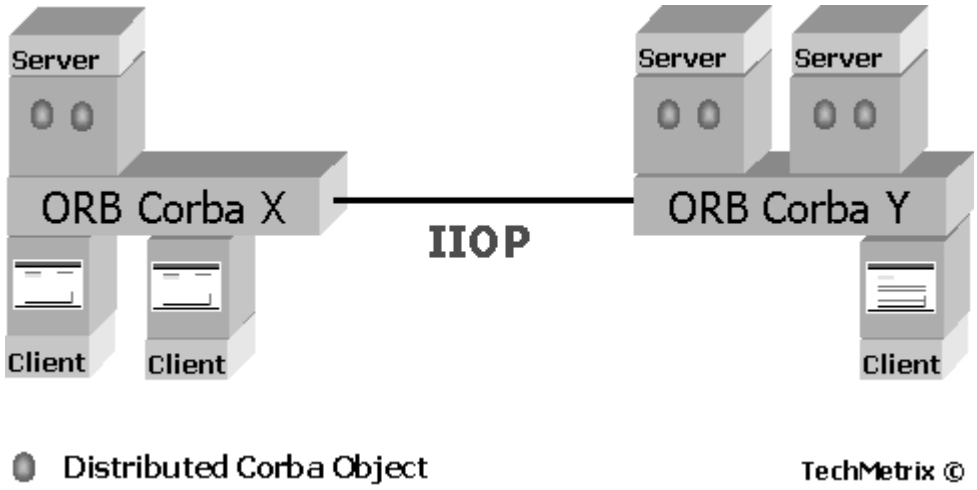

Figura 7: Interação entre ORBs

2.1.1.3 As Interfaces CORBA

CORBA é baseado, dentre outros aspectos, na construção e armazenamento de interfaces para os objetos. A interface especifica o conjunto de operações possíveis de serem requisitadas por clientes a um determinado objeto. Estas interfaces são armazenadas em repositórios de interfaces. No repositório de interface os objetos podem fazer requerimentos para obter informações sobre determinada interface, ou mesmo criar uma nova interface nele.

2.1.3.1 Interface Definition Language (IDL)

“A IDL é uma linguagem genérica de programação; ela permite a descrição de um serviço independente de uma linguagem específica” [SILBERSCHATZ 2000].

Antes de se implementar um objeto remoto, é necessário declarar qual será a sua *interface*, ou seja, quais serão os métodos que poderão ser chamados remotamente, seus argumentos e valores de retorno. CORBA usa a IDL (*Interface Definition Language*) para que um cliente possa saber os métodos públicos e atributos de um servidor. O uso dessa linguagem declarativa permite que clientes e servidores escritos em linguagens diferentes possam cooperar.

Essa linguagem oferece tipos de dados pré-definidos como *boolean, char* e *float* que serão usados nos métodos das *interfaces* como parâmetros e valores de retornos. Além dos tipos primitivos, é possível também passar objetos definidos pelo desenvolvedor como argumento e retorno de métodos, desde que esses objetos estejam devidamente declarados na IDL para que possam ser reconhecidos tanto no lado do objeto remoto quanto no do cliente. Com a IDL, também é possível criar exceções e especificar quais serão lançadas em um método para que, quando forem geradas no objeto remoto possam ser tratadas na aplicação que realizou a chamada. Com a *interface* definida na IDL, os desenvolvedores de aplicações clientes não precisarão nem mesmo saber em que linguagem o objeto remoto está implementado.

Cada ORB oferece um compilador IDL para transformar as definições das *interfaces* em estruturas nativas da sua linguagem. Cada *interface* gera pelo menos uma classe *stub*, uma *skeleton* e algumas outras classes que servirão para acessar o ORB e disponibilizar os serviços CORBA como *Dynamic Invocation Interface* (DII)e *Dynamic Skeleton Interface* (DSI). Apesar das classes geradas variarem em função do ORB, o mapeamento dos tipos da IDL serão sempre os mesmos para uma mesma linguagem independente do ORB. Ou seja, o int na IDL será sempre o int do C++ assim como o string da IDL será o *java.lang.String* do Java. Além dos tipos primitivos, outros elementos da IDL variam de acordo com a linguagem como o *module* e a *interface* que em C++ torna-se um arquivo *header* com a declaração da classe remota baseada na *interface* e em Java torna-se um *package* contendo uma interface Java. Um exemplo de um arquivo IDL seria algo do tipo:

```
Interface Automobile
{
  long color();
  long price();
};
```

2.1.1.4 GIOP e IIOP

Para permitir que diferentes implementações de ORB possam se comunicar, o formato das mensagens deve estar bem definido. O CORBA usa o *General Inter-Orb*

Protocol (GIOP) para definir o formato destas mensagens. No interior de uma mensagem do GIOP há informações sobre a operação a ser executada no servidor e os parâmetros. Parâmetros e valores de retorno são enviados em mensagens do GIOP usando o Protocolo de Representação de Dados Comuns (CDR). Este protocolo define o processo de representação das primitivas do IDL em um formato binário serializado. Isto possibilita a qualquer aplicação CORBA receber e compreender dados no formato CDR de qualquer outra aplicação CORBA.

O Protocolo *Inter-Orb Internet* (IIOP) define uma camada de transporte para enviar mensagens do GIOP sobre TCP/IP. IIOP estabelece um mapeamento de mensagens do GIOP para mensagens TCP/IP, e o protocolo de comunicação a ser utilizado quando do envio de mensagens do GIOP via Internet.

A *figura 8* mostra as interações entre um servidor CORBA e um cliente:

2.1.2 Principais vantagens

Uma das grandes vantagens da utilização da tecnologia CORBA é a necessidade crescente de adoção de um padrão para integração de aplicativos, principalmente no ambiente *middleware*. Criar padrão de interoperabilidade entre sistemas, mesmo que desenvolvido por diferentes fabricantes, é uma das responsabilidades dos fornecedores.

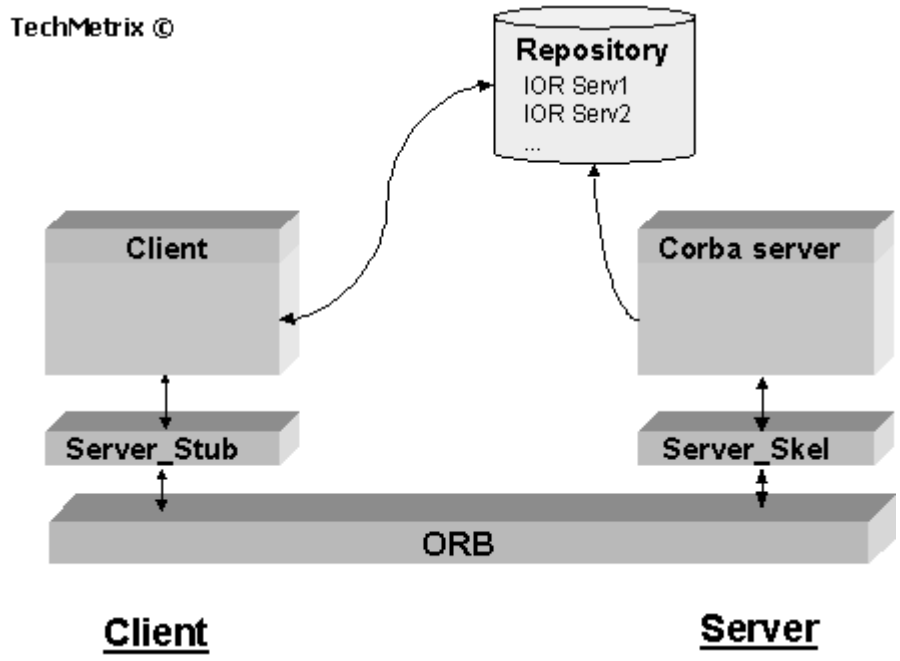

Figura 8: Interações entre um servidor e um cliente CORBA

O fato da independência de linguagem possibilita que os sistemas já desenvolvidos pela empresa sejam aproveitados e integrados com novas aplicações mesmo que esses sistemas sejam implementados em linguagens mais antigas como o COBOL. Além disso, possibilita que para uma determinada tarefa seja usada a linguagem mais apropriada e que o conhecimento dos desenvolvedores em uma tecnologia seja plenamente usado. Uma importante vantagem oferecida pelo CORBA refere-se ao dinamismo que será obtido na criação de um sistema distribuído que use uma nova linguagem, uma vez que os desenvolvedores que implementarão a parte cliente não precisarão conhecer a linguagem na qual os objetos que desejam acessar foram implementados [COSTA 1998].

A solução CORBA proporciona escalabilidade corporativa, uma base instalada e uma abordagem baseada em padrões.

Como foi mostrado anteriormente, para referenciar um objeto remoto não é necessário saber sua localização na rede o que irá possibilitar a criação de sistemas mais flexíveis onde os objetos remotos podem mudar de *host* sem alteração no código das aplicações clientes, ou seja, os desenvolvedores terão a abstração de que o objeto está simplesmente por toda a rede.

2.1.3 Limitações

Por se tratar de uma tecnologia que não é baseada em linguagem, o uso do CORBA obriga a um mapeamento pela IDL de todo objeto que for passado de uma linguagem para outra. Além disso, nem todos os recursos de uma determinada linguagem poderão ser suportados por outra, fazendo com que esses recursos não sejam utilizados.

A especificação CORBA também é um pouco criticada por não definir com precisão certos serviços como BOA. Porém, com a definição do POA na versão CORBA 2.3 foi possível solucionar muitos dos problemas de interoperabilidade entre ORBs que existiam. Ainda assim, alguns problemas ainda persistem com relação ao serviço de nomes.

2.1.4 Perspectivas Tecnológicas

O escopo de execução das aplicações distribuídas tem ultrapassado os ambientes locais, cruzando fronteiras administrativas e tecnológicas, provocando uma procura nas características de interoperabilidade e portabilidade. Para suprir essas necessidades, CORBA surge como uma padronização, constituída de especificações abertas, facilitando a portabilidade e os aspectos de interoperabilidade.

O CORBA foi criado para aplicações de propósito geral que desejam obter transparências de distribuição e na forma de gestão de seus recursos. Esses requisitos em aplicações de tempo real, ou não são desejáveis ou não são necessários. Pesquisas recentes têm apresentado grandes vantagens, objetivando estender as propriedades de portabilidade, interoperabilidade, flexibilidade e redução de custos nos sistemas de tempo real. Um dos estudos feitos é a abordagem de escalonamento adaptativo para aplicações distribuídas em tempo real.

Uma importante pesquisa é a utilização de CORBA, em conjunto com JAVA. O conjunto JAVA/CORBA, em comparação com aplicações baseadas em CGI, tem uma série de vantagens como: separação de interfaces e operações remotas, aceita estruturação de dados, um objeto remoto pode ser criado para manuseio de operações remotas e gerenciar várias chamadas remotas, controle pleno da interface pelo usuário e o desenvolvedor pode criar novos componentes a partir de já existentes, o *applet* gerencia direto as novas interfaces, uma invocação remota pode ser gerenciada por um *applet* já sendo executada e o objeto servidor remoto pode carregar os recursos antes do pedido. De maneira análoga, o estudo da integração CORBA/SOAP tem se mostrado uma grande perspectiva para o futuro. A

integração entre estas arquiteturas tem como principais objetivos fornecer ao CORBA a flexibilidade da comunicação entre objetos distribuídos através da internet, que é característica do SOAP, e por outro lado, procura fornecer ao SOAP camadas que o CORBA tem e que são bastante consistentes como, por exemplo, a de segurança, bem como os serviços do CORBA e as velocidade de comunicação em redes privadas e possibilidade de integração com outros sistemas, também características do CORBA.

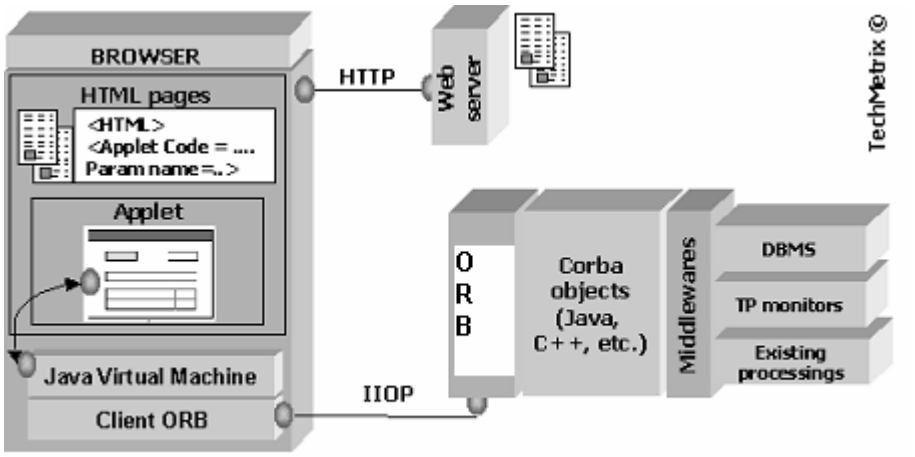

Figura 9: Java – IIOP. Princípios da interação entre um applet e objetos remotos

A *figura 9* ilustra um *applet* Java sendo executado na máquina virtual Java de um *browser* , comunicando-se com objetos remotos através do protocolo IIOP. Assim como em RMI a arquitetura de comunicação entre os objetos é baseada no princípio de *stubs and skeletons*. O *Stub* carregado no cliente contém as regras do *proxy* do servidor. Através dele é feita a chamada a um método remoto e a localização do servidor se torna transparente para o desenvolvedor. Uma chamada a um método remoto terá a mesma sintaxe que uma chamada a um método local. O

lado servidor possui o parceiro do *Stub*, chamado *Skeleton*, que representa o cliente para o servidor.

2.2 RMI

2.2.1 Arquitetura - visão geral

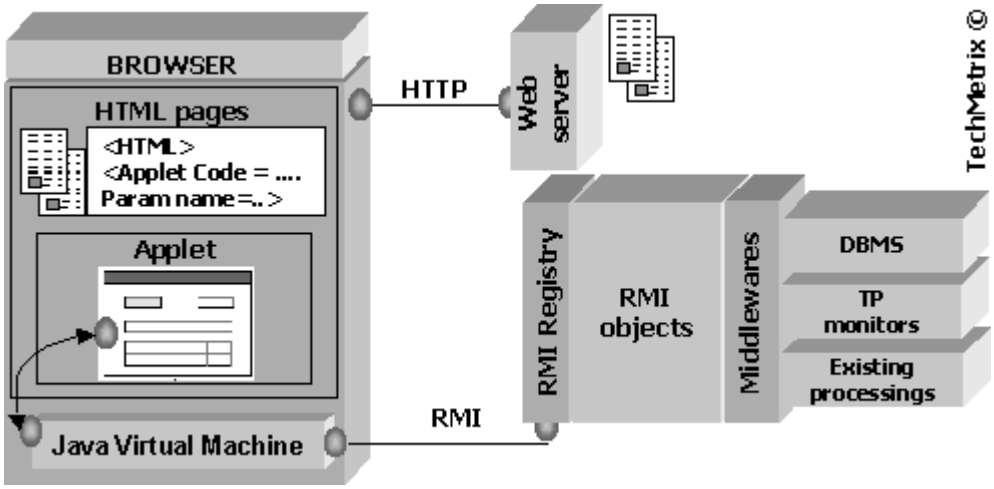

Figura 10: Interação entre um applet e objetos remotos na arquitetura RMI

Presente na especificação do JDK 1.1 da Sun, RMI (*Remote Method Invocation*) é a alternativa totalmente em Java para a criação de aplicações com objetos distribuídos. Essa tecnologia se propõe a estender o objetivo do Java de “Escreva uma vez, execute em qualquer lugar” com “Escreva uma vez, execute em todo lugar”. Nesse modelo, o objeto remoto é aquele que possui métodos que podem ser chamados de uma outra Máquina Virtual Java através da rede. Mesmo

estando o objeto em um ponto de rede totalmente remoto, as chamadas aos métodos são feitas como se o objeto fosse local, além de se poder usar todos os recursos da Orientação a Objeto. Nesse sentido, a criação de classes locais e remotas torna-se bastante semelhante, levando em consideração que não há preocupação com aspectos técnicos ligados à rede, ou seja, não há necessidade de uso direto de *sockets* ou mensagens. RMI provê um grau de abstração mais elevado que as técnicas tradicionais de comunicação entre objetos.

RMI disponibiliza uma estrutura de comunicação permitindo que aplicações ou *applets* Java possam estabelecer um diálogo. A figura 10 representa a interação entre *applets* e objetos remotos nesta arquitetura. Por ser totalmente ligado ao Java, RMI pode ser implementado de forma relativamente simples. A arquitetura é baseada no princípio de *stubs* e *skeletons* (já utilizados na comunicação RPC), conforme ilustrado na *figura 11* [TECHMETRIX 2001], e será detalhado ainda neste capítulo. Estas duas classes podem ser geradas pelo programador através de um aplicativo (rmic), ou geradas automaticamente por um mecanismo próprio da arquitetura.

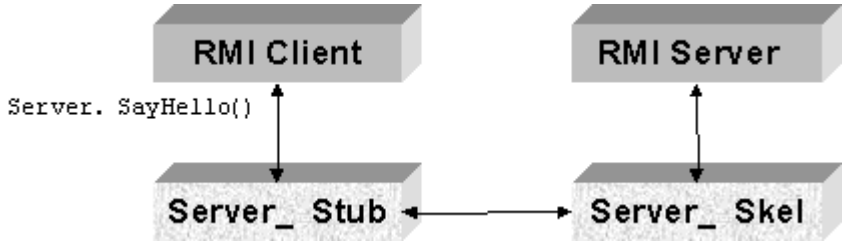

Figura 11: Esquema básico STUB / SKELETON para comunicação entre objetos

2.2.1.1 - Definição da Interface Remota

Uma *interface* Java é uma estrutura que contém apenas declarações de constantes e de métodos que deverão ser implementados por uma ou várias classes. Este conjunto de informações define um comportamento definido pela interface. Sempre que uma classe especifica que irá implementar uma *interface*, ela deverá conter o código da implementação de todos os métodos contidos na *interface* que está implementado.

Para uma *interface* ser remota basta que ela seja derivada da *interface* *java.rmi.Remote*. Com isso, qualquer método que ela declarar poderá ser chamado remotamente, estes métodos devem lançar exceções relacionadas a problemas que podem ocorrer na chamada do método, como por exemplo, falha na conexão entre cliente e servidor. A exceção básica a ser lançada é do tipo *java.rmi.RemoteException*.

Um exemplo de código de uma interface é algo do tipo:

```
public interface Hello extends java.rmi.Remote {
String sayHello() throws java.rmi.RemoteException;
}
```

As *interfaces* Java permitem a definição de um padrão comportamental na comunicação entre cliente e servidor, de forma semelhante ao papel da IDL CORBA.

2.2.1.2 - Implementação da Classe Remota

A implementação da classe remota deve conter os métodos que estão declarados na sua *interface* além dos seus próprios métodos, sendo que estes últimos só poderão ser chamados da mesma Máquina Virtual Java da classe. Uma importante característica dos métodos remotos é que os objetos de retorno e os que servem como argumento serão passados por referência se forem remotos e por cópia caso contrário, além disso podem ser de qualquer classe Java desde que esta classe implemente a *interface* java.io.Serializable. Mesmo assim, pode-se usar todos os tipos primitivos do Java como também classes mais freqüentemente usadas contidas nos pacotes java.util e java.lang.

O processo de serialização de objetos consiste em gerar uma seqüência de bytes (*streams*) a partir de uma instância de uma classe e restaurar a instância a partir da *stream* que pode até mesmo ser um arquivo em disco. Quando um objeto cliente faz uma requisição a um objeto remoto e o tipo de retorno do método é uma classe não remota, é feita uma cópia da instância dessa classe para a forma de *stream* para que essa instância possa ser transportada via TCP/IP para a Máquina Virtual do objeto cliente, onde a instância será reconstituída resultando em uma referência local à classe. Ou seja, o objeto é copiado para o endereço de quem está solicitando para que este possa referenciá-lo e chamar um de seus métodos [COSTA 1998].

Caso deva ser retornada uma referência a uma classe remota, a classe não será copiada para o cliente, mas sim a sua classe *stub* será serializada, transportada, carregada e referenciada. Essa classe *stub* é criada automaticamente a partir da classe remota e funciona como um *proxy* no lado cliente, ou seja, quando o cliente faz a requisição para um objeto remoto, internamente, a requisição é feita a classe *stub* correspondente que se encarrega de [RMI 1999]:

- Receber os dados passados na chamada do método remoto;
- Fazer a chamada ao objeto remoto passando os dados necessários sob a forma de *stream*;
- Receber de volta o resultado do método repassando-o para o objeto cliente.

No *host* do objeto remoto, há a sua classe *skeleton* que funciona de forma similar a *stub* recebendo as requisições para passá-las ao objeto remoto. Esse sistema está representado na *figura 12*:

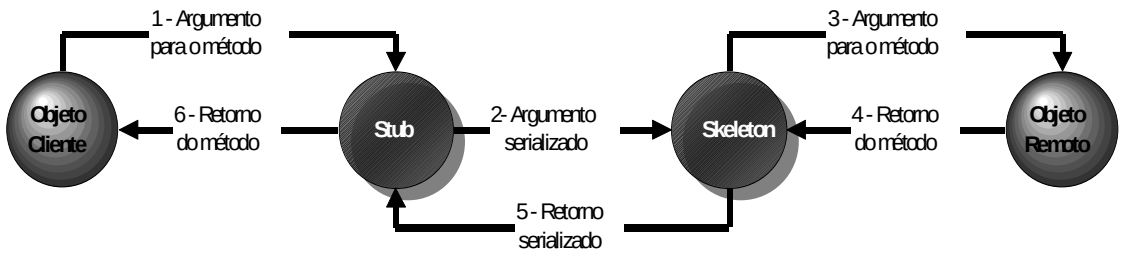

Figura 12: Funcionamento do *stub* e do *skeleton*

Um exemplo da implementação de uma classe remota seria algo do tipo:

```
public class HelloImpl
extends UnicastRemoteObject implements Hello
{
private String name;
public String sayHello() throws RemoteException {
return "How do you do";
}...
```

2.2.1.3 - Chamada de Métodos Remotos

Antes de ser feita uma chamada remota a um método, é necessário que o cliente possua uma referência à classe que contem o método. Deve-se levar em consideração que na Máquina Virtual do cliente apenas a *interface* remota está presente, portanto o operador *new* não pode ser usado uma vez que não se pode instanciar uma *interface*.

Uma maneira de se obter a referência é a partir da chamada de um método que retorna um objeto da classe remota. Como nem sempre existe essa possibilidade, o RMI disponibiliza uma maneira direta de referenciar uma determinada classe remota que consiste no uso do *Registry* através da classe *Naming*. O *RMIRegistry* responde por requisições de conexão aos servidores RMI, os quais registram a si próprios retornando suas interfaces para o cliente. A classe *Naming* possui o método *lookup* que retorna um objeto do tipo *Remote* que pode ser convertido para a

interface da classe remota desejada. Na chamada desse método basta fornecer a *string* que identifica o objeto no *Registry*, sendo que essa *string* deve ter o seguinte formato: //host/identificador. Onde [RMI 1999]:

- host é o número IP ou nome da máquina onde o objeto remoto está em execução;
- identificador é o nome com o qual o objeto remoto foi registrado no método bind.

O *Naming.lookup* usa uma *stub* padrão para realizar uma chamada remota ao *Registry* que irá devolver a *stub* do objeto associado ao identificador passado como parâmetro. Ou seja, para uma aplicação obter uma referência a um objeto remoto basta que se conheça onde ele está e qual o seu identificador. Com a referência, já é possível usar todos os métodos declarados na *interface* do objeto remoto da mesma maneira que são usados os métodos de objetos locais.

Um exemplo da execução de uma chamada seria:

```
...
try {
Hello server = (Hello) Naming.lookup("//" + getCodeBase().getHost()
+ "/HelloServer");
message = server.sayHello();
} ...
```

Como a carga do *Stub* é completamente transparente, a comunicação é mantida sem qualquer programação específica. A única parte que necessita ser desenvolvida é o registro do servidor no *RMIRegistry* assim que este é iniciado.

Um exemplo da implementação responsável por este registro basicamente é:

```
public static void main(String args[])
{
// O Security manager é obrigatório
System.setSecurityManager(new RMISecurityManager());
try {
HelloImpl obj = new HelloImpl("HelloServer");
Naming.rebind("HelloServer", obj);
System.out.println("server registered");
}
catch (Exception e) {
}
}
```

Convém lembrar que o *RMIRegistry* é uma aplicação que executa em memória, e que é carregada da mesma maneira que qualquer outro módulo executável. A figura 13 mostra a interação entre um servidor, um cliente e o *RMIRegistry*.

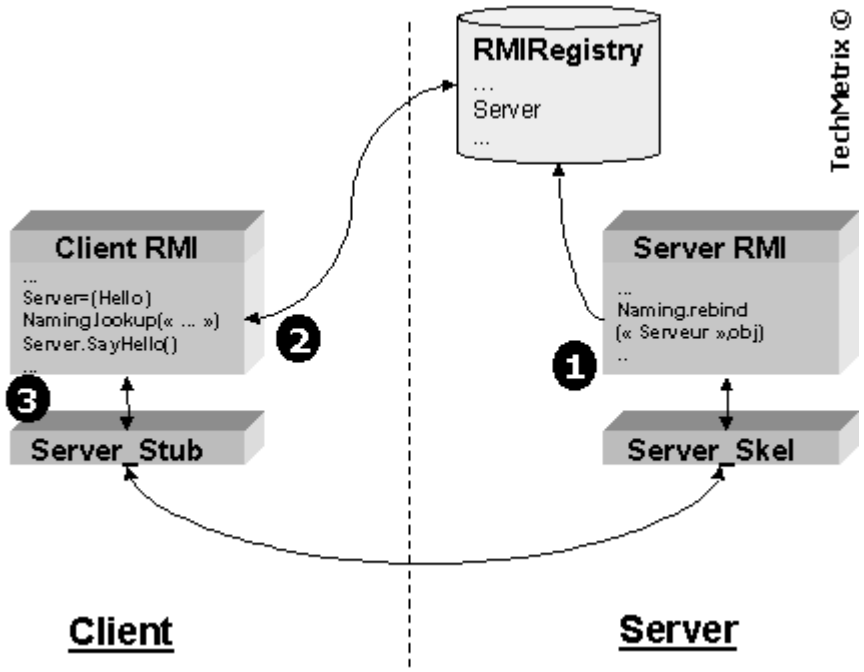

Figura 13: Interação cliente-servidor-RMIRegistry

2.2.1.4 - Distributed Garbage Collection

O *Garbage Collector* é um recurso automático oferecido pela linguagem Java cuja finalidade é remover da memória objetos instanciados pela aplicação e que não estão mais sendo referenciados. Isso é feito efetuando periodicamente uma varredura da memória. É um recurso que retira do programador a preocupação em remover da memória objetos instanciados, que existe em outras linguagens.

No ambiente distribuído, a arquitetura RMI disponibiliza um mecanismo mais complexo chamado *Distributed Garbage Collection.* Este mecanismo estende a

funcionalidade do *Garbage Collection* padrão da linguagem Java adicionando o controle sobre os objetos remotos.

> "Esse sistema usa o algoritmo de contagem de referências e uma relação dos objetos com as Máquinas Virtuais de onde se originam as chamadas aos objetos. Na primeira referência, é enviada uma mensagem de "referenciado" ao servidor do objeto, a partir daí o contador é incrementado a cada nova referência e decrementado sempre que o *runtime* do RMI detectar que uma das referências é finalizada. Quando o contador de um objeto remoto chega a zero, uma mensagem de "não-referenciado" é enviada e o *runtime* do RMI faz uma "referência fraca" a esse objeto para que o *Garbage Collector* da sua Máquina Virtual possa retirá-lo da memória"[COSTA 1998].

2.2.1.5 - Camadas do RMI

A arquitetura RMI é composta por três camadas distintas: a do *Stub/Skeleton*; a de Referência Remota e a de Transporte. Cada uma delas é independente e, por isso, a implementação de uma delas pode ser alterado sem afetar o funcionamento das outras. No topo das três camadas ficam as aplicações baseadas em RMI que se comunicam através da camada do *Stub/Skeleton*. A representação desse sistema de camadas está mostrado na figura 14:

A primeira camada é formada por objetos *stubs* e *skeletons* que são criados automaticamente a partir da classe remota usando o utilitário *rmic* do JDK. O objeto *stub* funciona do lado do cliente como um substituto do objeto remoto recebendo e serializando as requisições do cliente para poderem ser transportadas para a Máquina Virtual do objeto remoto onde estará o objeto *skeleton* correspondente que recebe os dados sob a forma de *stream* para reconstituir a requisição e os

argumentos que serão processados no objeto remoto. Após o método ter sido finalizado, o resultado (valor de retorno ou exceção) será passado ao *skeleton* para que o *stub* possa receber e retornar ao cliente que fez a requisição.

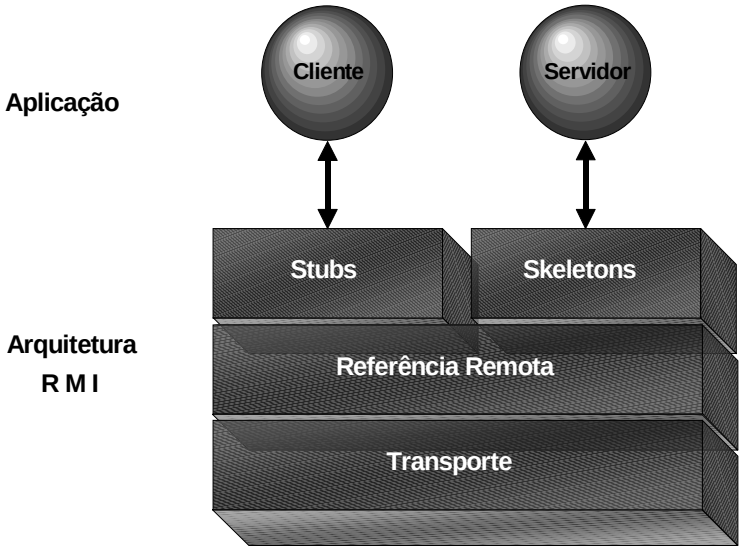

Figura 14: Camadas do RMI

A comunicação entre os componentes da primeira camada é feita através da camada de Referência Remota. Ela é a responsável por distinguir se o objeto remoto está replicado ou não e se ele precisa ser ativado para aceitar requisições, por isso o seu funcionamento vai depender da implementação do objeto chamado. Após determinar o tipo de objeto remoto, a camada de Referência Remota dará inicio à transferência dos dados usando um protocolo próprio do RMI chamado JRMP (*Java Remote Method Protocol*) por meio da camada de Transporte, uma vez que essa camada trata dos detalhes de baixo nível da comunicação como a inicialização e o gerenciamento das conexões. Essa camada é baseada em TCP e *sockets* padrões Java, mas poderia ser substituída por outra que usasse UDP [RMI 1999].

Atualmente, RMI evidencia a RRL (*Remote Reference Layer* – Camada de Referência Remota) para interface com as camadas de transporte. Porém, a versão 1.3 do JDK (Java Development Kit) já suporta a interoperabilidade entre RMI e CORBA, o que a Sun chamou de RMI-IIOP. Um exemplo deste tipo de interoperabilidade está na *figura 8* [TECHMETRIX 2001].

2.3 Vantagens

Apesar de algumas limitações, que serão citadas a seguir, RMI é uma alternativa bastante interessante se forem levadas em consideração as seguintes características:

- O uso de objetos distribuídos é totalmente integrado a linguagem, uma vez que o desenvolvimento de classes locais e remotas é bastante semelhante, o que torna a criação das aplicações bastante simples;
- O RMI usa da portabilidade e dos mecanismos de segurança oferecidos pelo ambiente Java;
- A presença do *Distributed Garbage Collection* é completamente transparente;
- A possibilidade de se passar todo um objeto como parâmetro possibilita o desenvolvimento de aplicações distribuídas totalmente orientada a objetos;
- Além dos dados o código também pode ser transportado;

- Como o RMI faz parte do *Java Development Kit* a partir da versão 1.1.x, toda Máquina Virtual Java compatível com essa versão está apta a executar aplicações distribuídas.

2.4 Limitações

Atualmente, o uso do RMI limita-se somente pelo fato de seu uso ser restrito ao ambiente Java, ou seja, não é possível que uma aplicação escrita em uma outra linguagem chame um método de um objeto remoto Java através de RMI, o que inviabiliza o aproveitamento de sistemas legados da empresa. Apesar do Java oferecer recursos como o JNI (*Java Native Interface*) que possibilita a integração com objetos de outras linguagens, o uso desses meios sempre se torna uma solução inadequada porque os recursos de uma linguagem nem sempre poderão ser passados para outra, ficando restritos pelo nível de compatibilidade das interfaces.

2.5 Comparativo

Segundo Gopalan Raj, a *tabela T1* representa um comparativo entre os principais mecanismos e características das arquiteturas CORBA e RMI [RAJ 2001]. Este comparativo é importante para que se possa perceber as semelhanças entre as arquiteturas, uma vez que tanto CORBA quanto RMI fornecem mecanismos

transparentes de chamadas remotas a objetos distribuídos. Estes mecanismos são diferentes e se baseiam na estrutura de cada arquitetura, mas não impedem a existência de tais semelhanças.

CORBA	RMI
Suporta herança múltipla em nível de interface	Suporta herança múltipla em nível de interface
Toda interface herda de CORBA.Object	Todo objeto servidor implementa java.rmi.Remote
Identifica de maneira individual objetos servidores remotos através de referências(objref), que servem para manipulação de objetos em tempo de execução.	Identifica, de maneira individual, objetos servidores remotos com Object ID's, que servem para manipulação de objetos em tempo de execução.
Identifica de forma individual uma interface usando seu nome e identifica de forma individual uma implementação específica de um objeto servidor pelo mapeamento para o seu nome correspondente no Implementation Repository.	Identifica de forma individual uma interface usando o seu nome e identifica de forma individual uma implementação específica de um objeto servidor pelo seu mapeamento para uma URL correspondente no Registry.
A geração de uma referência para o objeto servidor remoto é feita em conjunto com o protocolo pelo Object Adapter.	A geração de uma referência para o objeto servidor remoto é feita através de chamada ao método UnicastRemoteObject.exportObject(this)
O construtor implícito executa tarefas comuns como registro de objetos, instanciação dos skeletons e outras.	O RMIRegistry executa tarefas comuns como registro de objetos atraves da classe Naming. O método UnicastRemoteObject.exportObject(this) Executa a instanciação dos skeletons e é chamado implicitamente pelo construtor do objeto.
Utiliza o Internet Inter-ORB Protocol(IIOP) como protocolo de chamadas remotas	Utiliza o Java Remote Method Protocol(JRMP) como protocolo de chamadas remotas
Quando um objeto cliente precisa ativar um objeto servidor, ele se liga ao serviço de nomes (naming) ou a um serviço negociante (trader).	Quando um objeto cliente precisa obter uma referência para um objeto servidor ele deve fazer um lookup() na URL do objeto servidor remoto.
O manipulador de objetos que o cliente usa é a referência a um objeto (Object Reference)	O manipulador de objetos que o cliente usa é a referência a um objeto (Object Reference)
O mapeamento do nome do objeto para a sua implementação é conduzido pelo repositório de implementações (Implementation Repository)	O mapeamento do nome do objeto para a sua implementação é conduzido pelo RMIRegistry.
A informação dos tipos para os métodos é guardada pelo repositório de interfaces(Interface Repository)	Qualquer informação de tipos é guardada pelo objeto em si e pode ser consultada usando Reflexão e introspecção.
A responsabilidade de localizar a implementação de um objeto recai sobre o Object Request Broker (ORB)	A responsabilidade de localizar a implementação de um objeto recai sobre a Java Virtual Machine (JVM), ou máquina virtual Java.
A responsabilidade de ativar a implementação de um objeto recai sobre o Object Adapter (OA) – da mesma maneira sobre o Basic Object Adapter (BOA) ou o Portable Object Adapter (POA)	A responsabilidade de ativar a implementação de um objeto é da máquina virtual Java.

O stub do lado cliente é chamado proxy ou stub.	O stub do lado cliente é chamado proxy ou stub.
O stub do lado servidor é chamado skeleton.	O stub do lado servidor é chamado skeleton.
Na passagem de parâmetros entre o cliente e o objeto servidor remoto, todos os tipos de interface são passados por referência. Todos os outros objetos são passados por valor, inclusive tipos de dados de alta complexidade.	Na passagem de parâmetros entre o cliente e o objeto servidor remoto, todos os objetos que implementam interfaces que herdem de java.rmi.Remote são passados por referência remota. Todos os outros objetos são passados por valor.
Não se propõe a implementar garbage collection distribuído**.**	Propõe-se a implementar garbage collection distribuído dos objetos servidores remotos através dos mecanismos contidos na máquina virtual.
Tipos complexos que ultrapassem as fronteiras da interface devem ser declarados na IDL.	Qualquer objeto java serializável pode ser passado como parâmetro através dos processos.
Pode ser executado em qualquer plataforma desde que haja uma implementação de ORB para a mesma.	Pode ser executado em qualquer plataforma desde que haja uma implementação da máquina virtual Java para a plataforma.
Uma vez que é uma especificação, várias linguagens de programação podem ser usadas para escrever o código dos objetos desde que a linguagem escolhida possua bibliotecas ORB.	Uma vez que se baseia extremamente na serialização de objetos Java, o código destes objetos só pode ser escrito em Java.
O tratamento de exceções é cuidado pelos objetos de exceção. Quando um Objeto distribuído lança um objeto de exceção o ORB transparentemente serializa e empacota o mesmo para que possa trafegar no meio.	Permite o lançamento de exceções que são então serializadas e empacotadas para que possam trafegar no meio.

Tabela 1: Comparativo entre características das arquiteturas CORBA e RMI

2.6 Enterprise Java Beans (EJB)

Enterprise Java Beans define uma arquitetura para desenvolvimento de aplicações de objetos distribuídos baseada em componentes *"server-side"* que encapsulam a lógica de negócios de uma aplicação. A lógica de negócios é o código que satisfaz o propósito da aplicação. A lógica de negócios é implementada em métodos que podem ser chamados por clientes remotos para que possam ter acesso aos serviços fornecidos pela aplicação [SUN 2001]. Estes componentes,

chamados de EJBs, são executados em uma ambiente especial, chamado de container, desenvolvido exclusivamente para controlar, em tempo de execução, os aspectos relacionados aos componentes, além da tarefa de abrigar, gerenciar acesso, segurança, persistência, controle de transações, concorrência e acesso a recurso reservados. Grande parte desse gerenciamento é feita automaticamente, sem a interferência do programador.

Por serem escritos na linguagem Java os EJBs fazem uso dos recursos da tecnologia J2EE (*Java 2 Enterprise Edition*), como utilização de RMI para chamada a métodos remotos e JNDI como padrão de acesso no serviço de nomes, além da possibilidade de utilização do CORBA. Da mesma forma, se utiliza a estrutura *stub/skeleton* e mecanismo de serialização, porém tendo como particularidade um ciclo de vida diferenciado, uma vez que são capazes de representar persistência de dados.

3 SOAP (Simple Object Access Protocol)

SOAP [BOX 2000] é um protocolo de comunicação leve, baseado na linguagem XML e concebido para utilização na troca de informações entre sistemas em um ambiente descentralizado, distribuído. Consiste basicamente de três partes: um envelope que define uma estrutura para descrever o conteúdo de uma mensagem e como o mesmo pode ser processado, um conjunto de regras de controle para expressar instâncias de tipos de dados de aplicações, e uma convenção para representar chamadas a procedimentos remotos e respostas.

SOAP representa uma nova realidade no sentido em que fornece uma nova base no processo de padronização dos procedimentos de troca de informações via Internet, através de estruturas de segurança (*firewalls*).

3.1 Visão geral do protocolo

SOAP é um protocolo simples, comercialmente independente, e que não define a semântica da aplicação onde será utilizado como, por exemplo, o modelo de programação; ao invés disto, define uma estrutura simples para expressar a semântica da aplicação por meio de um mecanismo modular de empacotamento de informações e codificação dos tipos de dados contidos nos pacotes. Esta característica permite a utilização do SOAP em uma grande variedade de sistemas, desde aqueles baseados em troca de mensagens até os baseados em RPC, independente da linguagem de programação utilizada.

Sempre é importante lembrar que SOAP em si é apenas um protocolo e não uma arquitetura completa de objetos distribuídos. Por outro lado, arquiteturas completas de objetos distribuídos são projetadas em torno dos seus protocolos para maior eficiência. Algumas tecnologias de objetos distribuídos se preocupam com os próprios aspectos de segurança, por exemplo, e carregam informações de segurança dentro seus pacotes de dados. Elas têm mecanismos para eficientemente codificar a informação de segurança para aumentar a velocidade de recepção dos dados.

A principal característica do SOAP é o objetivo de manter a simplicidade e extensibilidade. Isto faz com que vários recursos das arquiteturas atuais de objetos distribuídos não estejam incluídos na especificação do SOAP, e torna o SOAP um

protocolo atrativo a um grande número de projetos. Por exemplo, alguns destes recursos são:

- Garbage collection distribuído.
- Passagem de objetos por referência (que requer *garbage collectiion* distribuído)

Também, diferente das arquiteturas contemporâneas de objetos distribuídos, SOAP usa abertamente tecnologias disponíveis que, quando combinadas, especificam um protocolo consistente, que pode ser utilizado para auxiliar, principalmente, sistemas distribuídos através da Internet. O fato de o SOAP utilizar padrões bem definidos e aceitos no mercado, como por exemplo a linguagem XML na sua estrutura, também é estendido aos protocolos de comunicação. SOAP permite a sua utilização em combinação com uma variedade de outros protocolos de comunicação, como por exemplo HTTP, SMTP e FTP; entretanto, na maioria das aplicações, normalmente ele é utilizado em combinação com o HTTP .

O SOAP geralmente usa o protocolo HTTP para transportar, encapsulados em arquivos XML, dados serializados de argumentos de métodos, de sistema para sistema. Estes dados serializados de argumentos são utilizados no lado remoto para executar as chamadas aos procedimentos que o cliente fez àquele sistema.

3.2 Funcionalidades

SOAP é fundamentalmente uma estrutura que consiste de uma combinação de informações textuais compartilhada via a Internet. A informação textual é codificada na forma de arquivo XML, que tem regras específicas para codificação e processamento. A transmissão real dos dados do XML é administrada pelo protocolo de transporte, que é hoje geralmente HTTP servido por um servidor *Web*. A combinação do estilo aberto da estrutura da linguagem XML e a penetração do protocolo HTTP tornam o SOAP um protocolo de elevado grau de interoperabilidade.

Uma vez que se pode utilizar HTTP como protocolo de transporte para o SOAP então o processamento do SOAP está muito ligado com a Internet, que especifica um modelo de programação *stateless*. Isto é, objetos clientes solicitam serviços de uma entidade remota, que responde com a informação pertinente. Depois que a entidade remota responde, toda informação de estado referente àquela chamada é destruída a menos que sejam utilizadas medidas para persistir esta informação para posterior uso.

Servidores SOAP simples seguem este *stateless* básico com request/response. Porém o protocolo em si não impede a implementação de servidores mais complexos, capazes de administração de estado. Esta facilidade é possível tanto para simples scripts quanto para aplicações mais complexas utilizando SOAP.

Utilizando SOAP para a interoperabilidade de serviços a estrutura das mensagens encapsuladas em arquivos XML é a informação chave e que deve ser conhecida entre as partes que se comunicam. Isto permite que serviços disponíveis na Web em servidores de plataformas diferentes e que foram escritos utilizando linguagens de programação diferentes possam interagir sem a que seja necessária a intervenção humana.

Pelo fato de utilizar protocolos de comunicação extremamente conhecidos e difundidos, e por isso registrados nas estruturas de segurança dispostas na internet (*firewalls*), as mensagens SOAP conseguem transpor tais estruturas facilmente, sem que seja necessário nenhum esforço adicional por parte de desenvolvedores e administradores de redes.

3.3 Estrutura

A estrutura do protocolo SOAP é baseada em XML, contendo elementos de definição estrutural e elementos de definição do tipo de conteúdo. Estes elementos serão identificados de acordo com o exemplo ilustrado na Figura 15.Namespaces XML fornecem um método simples de qualificação de elementos e nomes de atributos utilizados em documentos XML associando-os com namespaces identificados por referências URI (Uniform Resource Identifiers) [RFC2396]. Um namespace XML é uma coleção de nomes, identificados por uma referência URI que são utilizados em documentos XML como tipos [RFC2396].

ANEXO I - Código Fonte do protótipo

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Http Request

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 15: Pacotes SOAP dentro de um request e um response HTTP

Uma aplicação SOAP deve incluir o namespace SOAP adequado em todos os elementos e atributos definidos pelo SOAP nas mensagens geradas. Uma aplicação de SOAP deve ser capaz de processar namespaces SOAP nas mensagens que receber, descartar mensagens que têm namespaces incorretos e processar mensagens SOAP sem namespaces SOAP como se tivessem namespaces SOAP corretos.

SOAP define dois namespaces:

O envelope de SOAP tem como identificador namespace "<http://schemas.xmlsoap.org/soap/envelope/>"

O serializador SOAP tem como identificador namespace "<http://schemas.xmlsoap.org/soap/encoding/>"

Os prefixos de namespace "SOAP-ENV" e " SOAP-ENC" estão associados com os namespaces SOAP "<http://schemas.xmlsoap.org/soap/envelope/>" e "<http://schemas.xmlsoap.org/soap/encoding/>" respectivamente. O prefixo de namespace "xsi" está associado com o URI "<http://www.w3.org/1999/XMLSchema-instance>" e, analogamente, o prefixo do namespace "xsd" está associado ao URI "<http://www.w3.org/1999/XMLSchema>".. O prefixo de namespace "tns" é utilizado para indicar qual é o namespace de alvo do documento corrente. O indicador xmlns indica a presença de uma referência a um namespace (xmlns = XML namespace). Os outros namespaces encontrados na Figura 15 são meramente exemplos.

Uma mensagem SOAP é um documento do XML que consiste de um envelope de SOAP obrigatório, um cabeçalho opcional, e um corpo obrigatório. O namespace que identifica estes atributos da mensagem é "<http://schemas.xmlsoap.org/soap/envelope/>". Os componentes da mensagem SOAP podem ser assim definidos:

O Envelope é o elemento principal do documento XML representando a mensagem. O atributo encodingStyle, do Envelope indica o mecanismo de serialização a ser utilizado e está definido no URI http://schemas.xmlsoap.org/soap/encoding/".

- O Cabeçalho é um mecanismo genérico para adicionar recursos a uma mensagem de SOAP, de maneira descentralizada, sem acordo prévio entre as partes que estão se comunicando. SOAP define alguns atributos que podem ser utilizados para indicar se as partes devem lidar com um recurso e se este é opcional ou obrigatório. As regras de definição do cabeçalho são as seguintes:

    - Uma entrada no cabeçalho é identificada pela sua referência completa, que consiste do URI do seu namespace e o seu nome local. Todos os elementos contidos no cabeçalho devem possuir um namespace válido especificado.
    - O atributo encodingStyle pode ser utilizado para identificar o tipo de serialização das entradas do cabeçalho.
    - O atributo *mustUnderstand* e *actor* podem ser usados para indicar como a entrada do cabeçalho deve ser processada e por quem, conforme pode ser verificado na Figura 16. O pacote SOAP ilustrado na figura contém a informação sobre uma transação cujo valor é “5”. A presença do atributo *mustUnderstand* faz com que esta informação deve ser compreendida pelo receptor da mensagem sob pena da obrigatoriedade de se indicar uma exceção caso isso não aconteça. O atributo global *actor* pode ser utilizado para indicar num elemento do cabeçalho o receptor final da mensagem. Uma vez que uma mensagem SOAP pode ser repassada por várias aplicações até chegar no destino onde será processada, o valor do atributo *actor* é um URI especial, "<http://schemas.xmlsoap.org/soap/actor/next>", que indica se o elemento de cabeçalho se destina à primeira aplicação SOAP que processar a mensagem. Omitir o atributo *actor* indica que o receptor é o destino final da mensagem SOAP.

- O Corpo é um recipiente para informação obrigatória destinada ao receptor final da mensagem. SOAP define no corpo um elemento de exceção para que

possa ser expressa a ocorrência de erros. Os elementos de erro SOAP se dividem nos seguintes subelementos: *faultcode*, indicando o código do erro ocorrido*; faultstring*, indicando uma explicação textual do erro ocorrido; *faultactor*, indicando quem causou o erro durante o caminho percorrido pela mensagem; *detail*, cuja presença indica que o conteúdo do Corpo (Body) não foi corretamente processado, só devendo estar presente neste caso particular. Outros elementos de erro podem estar presentes, desde que identificados por *namespaces* válidos.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Header>
       <t:Transaction
           xmlns:t="some-URI"
           SOAP-ENV:mustUnderstand="1">
               5
       </t:Transaction>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DEF</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 16: Pacote SOAP contendo cabeçalho com informações obrigatórias

Os tipos de dados suportados pela especificação padrão e que permitem interoperabilidade entre as várias implementações SOAP são os citados a seguir:

O suporte do protocolo a valores escalares é válido conforme a tabela T2.

Type value	Type	Example
xsd:int	32-bit signed integer	-12
xsd:Boolean	a boolean value, 1 or 0	1
xsd:string	string of characters	hello world
xsd:float or xsd:double	signed floating point number	-12.214
xsd:timeInstant	date/time	2001-03-27T00:00:01-08:00
SOAP-ENC:base64	base64-encoded binary	eW91IGNhbid0IHJlYWQgdGhpcyE=

Tabela 2: Valores escalares suportados pelo protocolo.

Um valor também pode ser um struct, que é definido por um XML que contém sub elementos. Structs podem ser aninhados e conter qualquer outros tipos, incluindo um array. Um exemplo de um struct pode se representado da seguinte forma, onde o nome dos elementos do struct são significativos, enquanto a posição, não.

```
<param>
  <lowerBound xsi:type="xsd:int">18</lowerBound>
  <upperBound xsi:type="xsd:int">139</upperBound>
 </param>
```

Um valor pode também ser um array, que é definido por um elemento XML que é o atributo SOAP-ENC:arrayType, o qual inicia com ur-type[, seguido do número de elementos do array, seguido de]. Um array de quatro elementos pode ser representado dessa forma:

```
<parametro SOAP-ENC:arrayType="xsd:ur-type[4]" xsi:type="SOAP-ENC:Array">
```

```
<item xsi:type="xsd:int">12</item>
<item xsi:type="xsd:string">Egypt</item>
<item xsi:type="xsd:boolean">0</item>
<item xsi:type="xsd:int">-31</item>
</param>
```

A ordem dos elementos do array é significativa enquanto os nomes dos elementos não. Se os elementos do array são do mesmo tipo o valor do elemento SOAP-ENC:arrayType determina o tipo dos sub elementos do array. Por exemplo, SOAP-ENC:arrayType="xsd:int[4]" representa um array de quatro elementos do tipo xsd:int.

Para arrays de tipos mistos o atributo SOAP-ENC:arrayType sempre deve especificar xsd:ur-type. Para arrays de tipos simples, o atributo xsi:type é opcional para os sub elementos, mas sua inclusão é recomendada.

Um valor também pode ser um null , que é especificado por um elemento XML com um atributo, xsi:null, cujo valor é 1 e é representado da seguinte forma:

<parametro xsi:null="1"/>.

3.4 Principais vantagens e limitações

Ao tratarmos de vantagens e limitações de protocolos de comunicação o mais importante é saber qual o objetivo a ser atingido. Inúmeros pontos podem ser abordados, mas sempre deve ser considerado que a natureza do sistema, a sua finalidade e o tipo do processamento a ser realizado é que serão os fatores determinantes do sucesso ou não da utilização de determinado protocolo de comunicação.

Considerando o exemplo de uma aplicação científica responsável por coletar dados de sensores, realizar uma série de cálculos e que utiliza um banco de dados apenas como repositório dos resultados obtidos, nesta aplicação um simples arquivo de log pode substituir um poderoso SGBD. Já o mesmo não é possível quando trata-se de uma aplicação onde o acesso a dados é o pronto crítico do problema.

O mesmo raciocínio deve ser estendido à utilização do protocolo SOAP. Serão abordados a seguir alguns quesitos-chave na utilização de protocolos de comunicação:

3.4.1 Tamanho da mensagem

Por encapsular os dados da aplicação em pacotes de arquivos XML as mensagens SOAP tendem a ser maiores que as mensagens de outros protocolos, como por exemplo IIOP, uma vez que, além dos dados, o pacote XML contém em si a sua própria estrutura que define a si e a informação que contém. Em aplicações

onde o consumo de rede ou o tamanho da mensagem é o ponto crítico, como apliações wireless, por exemplo, o uso do SOAP deve ser meticulosamente avaliado. Já em aplicações que visam funcionar normalmente na web e com isso utilizando o protocolo HTTP, que é baseado em texto, a utilização do protocolo SOAP não deve representar uma perda significativa na performance da aplicação.

3.4.2 Complexidade de empacotamento

Comparado com a de outros protocolos, por exemplo, IIOP, a complexidade de empacotamento requerida pelo SOAP é bem maior devido à estrutura dos seus pacotes e apesar de mesmo sendo a linguagem XML tão difundida e esse fato possibilitar o desenvolvimento de excelentes mecanismos de empacotamento de arquivos XML, este processo para o SOAP ainda será mais dispendioso em termos de performance que para outros protocolos. Com isso, aplicações distribuídas de tempo real sofrerão drásticos prejuízos caso se decida utilizar o SOAP, pois são mais sensíveis ao grau de complexididade de empacotamento do protocolo. O mesmo problema não será tão sentido em aplicações onde entrada e saída (I/O) é o ponto crítico (*I/O bound*).

3.4.3 Facilidade de depuração

Facilidade de depuração é uma das características das mensagens SOAP mais apregoadas pelos defensores do protocolo. Sem dúvida possibilita uma facilidade na correção de erros do sistema. Porém, em sistemas em produção, a depuração é um processo que ocupa uma fração mínima do tempo total de execução do sistema e que envolve poucos e específicos responsáveis.

Penalizar o desempenho do sistema com a decisão de usar um protocolo de fácil depuração pode ser sem dúvida uma falha grave de projeto, a ser sentida por toda a vida útil do sistema. Esta técnica poder utilizada em se tratando de prototipação do sistema, mas em tempo de execução na produção pode representar um ônus considerável em termos de performance. E, a depender do sistema, torná-lo impraticável.

3.4.4 Interoperabilidade entre sistemas

Estruturas de segurança estabelecidas na Internet (firewall), tendem a ser por natureza bastante seletivas no que diz respeito ao tráfego de informações através das mesmas. A utlização de padrões abertos e comercialmente difundidos é um fator que possibilita aos pacotes SOAP trafegarem normalmente através de tais estruturas. Tal funcionalidade, aliada a mecanismos de segurança, como criptografia, representa uma economia significativa à medida que dispensam canais especiais de comunicação para a interoperabilidade de sistemas. O próprio protocolo de transporte, responsável por conduzir os pacotes SOAP, geralmente HTTP,

representa uma estrutura cômoda e consistente de transporte. Quando desejamos obter interação entre sistemas corporativos dentro da própria estrutura da empresa, uma vez que temos acesso aos *firewalls* internos (se existirem) com a possibilidade de alterações das suas configurações, pode ser mais rentável em termos de performance do sistema utilizar um protocolo com melhor sintaxe de transferência, como por exemplo JRMP(RMI) ou IIOP (CORBA). Em se tratando de sistemas distribuídos através da Internet , SOAP será sempre um candidato com grande vantagem.

3.4.5 Interoperabilidade com outros protocolos

A interação com outros protocolos é uma possibilidade que resulta da natureza da estrutura do protocolo. Existem atualmente estudos avançados na integração entre SOAP e protocolos de outras arquiteturas, como RMI e CORBA. Por exemplo sobre RMI existe inclusive um projeto desenvolvido pela Universidade de Indiana , nos Estados Unidos que é uma implementação SOAP baseada em RMI, chamada XSOAP [SLOMINSKI 2001].

A interoperabilidade entre protocolos pode trazer funcionalidades extras ao sistema e uma economia considerável. Por exemplo, através da interoperabilidade IIOP/SOAP é possível fazer com que objetos CORBA possam interagir com serviços na Internet através de *firewalls,* e da mesma maneira permitir que sistemas baseados em CORBA possam interoperar com sistemas de outras arquiteturas.

3.5 Problemas de interoperabilidade

Para garantir a definição de um protocolo mais leve e comercialmente mais atraente, a última especificação do protocolo SOAP submetida à W3C contou com a colaboração de representantes de grandes corporações, como Microsoft, IBM e Lotus, mas também de pequenas empresas, como por exemplo a UserLand. Este fato foi decisivo para o surgimento de uma especificação bem mais simples e enxuta que a anterior e cujo objetivo é manter simples a implementação do protocolo.

Este fato deu origem a várias implementações comerciais e livres do protocolo SOAP, causando o surgimento de alguns problemas de interoperabilidade entre plataformas. Um fator adicional complicante é a participação das grandes companhias no mercado, que tentam impor sobre os menores desenvolvedores, padrões para o desenvolvimento de implementações SOAP. Por exemplo, a definição pela IBM do WSDL (*Web Services Description Language*). Estas camadas adicionais não têm sido bem aceitas por alguns desenvolvedores e algumas implementações de *frameworks* SOAP simplesmente não se destinam à utilização do WSDL, como por exemplo o APACHE SOAP [APACHE 2001], enquanto outras são totalmente orientadas ao uso do mesmo, como é o caso do GLUE [GLUE 2001]. Isto faz com que, por exemplo, a APACHE possua dois *frameworks* para utilização do SOAP: um orientado a WSDL, chamado AXIS [AXIS 2001], além do APACHE SOAP que não utiliza e atualmente não pretende utilizar WSDL, segundo a documentação disponível no *site* do projeto (http://xml.apache.org/soap/docs/index.html).

Segundo Dave Winer, membro da empresa UserLand e colaborador da especificação SOAP 1.1, como ele, muitos desenvolvedores não aceitam a imposição dos padrões das grandes empresas sobre o desenvolvimento com SOAP e procuram alternativas para um aprimoramento da especificação SOAP de forma a prover soluções robustas de interoperabilidade entre *frameworks* de fornecedores diferentes. Seja padronizando um modelo de envelope SOAP específico para permitir a utilização do modelo XML-RPC tradicional com SOAP, seja documentando as principais implementações SOAP e mecanismos específicos para interoperabilidade entre as mesmas [WINER 2001].

Em contrapartida aos problemas de interoperabilidade entre os *frameworks* existem estudos em andamento sobre a interoperabilidade entre *web services* e objetos CORBA. Tais estudos visam possibilitar total interoperabilidade entre sistemas legados que utilizam CORBA com serviços disponíveis na internet, com o objetivo de redução de custos da base instalada de equipamento e software nas empresas por intermédio da interação dos sistemas corporativos com *web services*. Esta interoperabilidade está sendo testada através de *wrappers* que são estruturas capazes de converter requisições recebidas via SOAP/HTTP em requisições ao ORB e vice-versa.

A figura 17 representa um objeto corba utilizando um ou mais *web services*, enquanto que a figura 18 representa um *web service* efetuando chamada a objetos CORBA. Em ambos os casos a estrutura intermediária, chamada *wrapper,* é responsável por converter requisições SOAP em requisições CORBA e vice-versa.

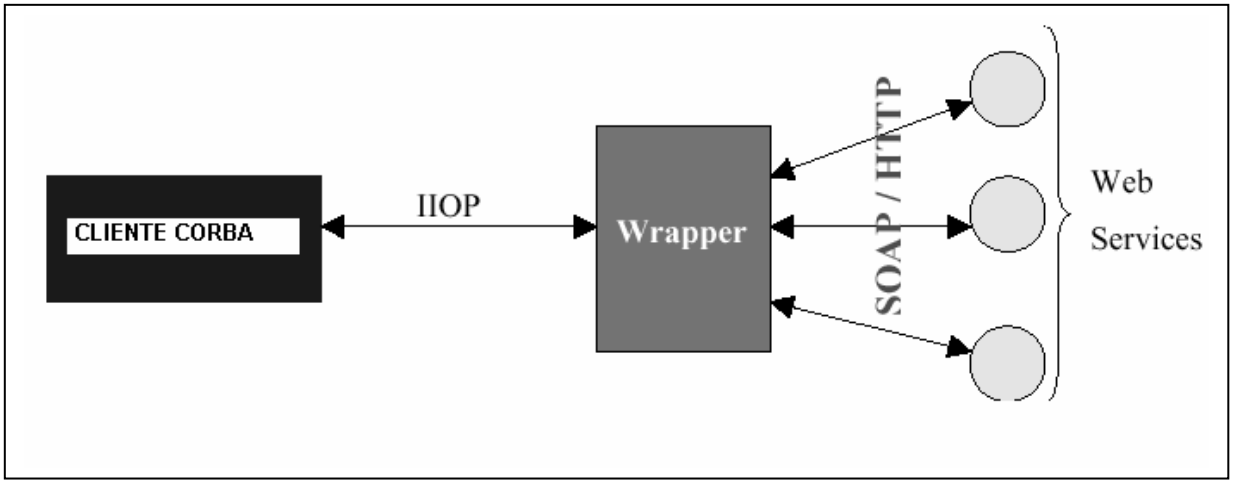

Figura 17 Objetos CORBA utilizando *web services*

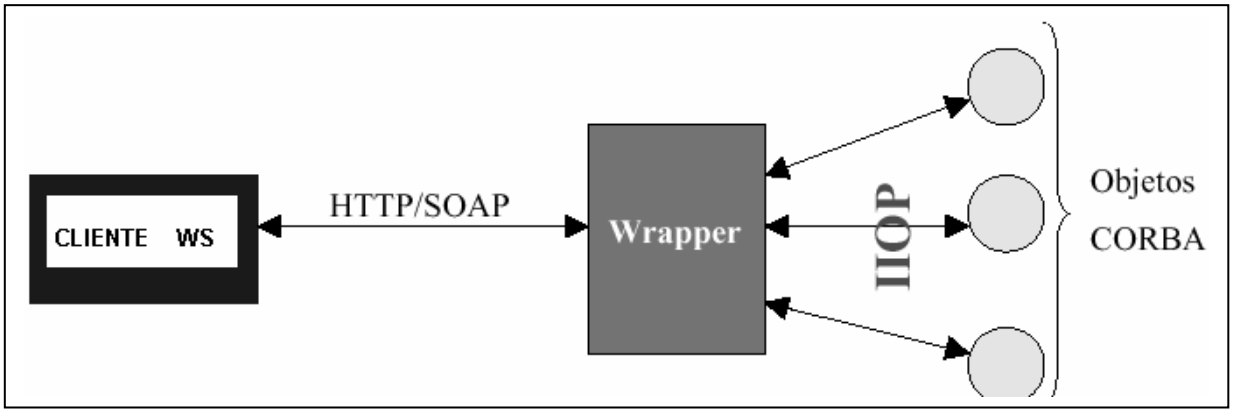

Figura 18 *Web services* utilizando objetos CORBA

3.6 Principais projetos e empresas envolvidas

A primeira especificação do SOAP (1.0) submetida à W3C era bastante complexa, contendo a definição de vários recursos e serviços agregados ao núcleo do protocolo. Esta característica fez com que poucas empresas, como por exemplo,

Microsoft e IBM manifestassem interesse no mesmo. Posteriormente, com a submissão da especificação SOAP 1.1, que definia um protocolo bem mais leve, cujo objetivo era permitir a sua extensibilidade, SOAP se tornou um protocolo particularmente atraente Atualmente outras grandes empresas possuem projetos utilizando o SOAP, como a HP e a Sun Microsystems. Este fato provocou um aumento significativo no número de projetos

Os projetos de utilização do SOAP são representados pelas várias implementações SOAP já disponíveis. Nem todas estão completas e a maioria ainda não está madura, uma vez que a arquitetura para a qual foram originalmente concebidos, os web *services*, ainda encontra-se em fase de amadurecimento. Algumas das implementações SOAP disponíveis são de domínio público, outras são comerciais. Mas, as implementações que hoje se encontram em estado mais consistente possuem uma versão gratuita e outra comercial, como por exemplo o GLUE, da Electric Mind, e esta é uma tendência que deve ser adotada pela maioria das implementações que conseguirem transpor este período inicial, conseguindo a simpatia dos desenvolvedores. A seguir, uma relação com as principais, dentre as mais de 60 (sessenta) implementações SOAP disponíveis:

PROJETO	RESPONSÁVEL	CATEGORIA
CapeConnect SOAP	Cape Clear Software	COMERCIAL
Glue	The Mind Electric	COMERCIAL
HP Web Services Beta	HP	COMERCIAL
Jbroker	SilverStream	COMERCIAL
WebLogic	Bea	COMERCIAL
WASP	Systinet	COMERCIAL
TRLSOAP	IBM	COMERCIAL
Web Services Toolkit	IBM	COMERCIAL
Microsoft SOAP toolkit	Microsoft	COMERCIAL
Visual Studio .NET	Microsoft	COMERCIAL
.NET Framework SDK	Microsoft	COMERCIAL
X-Soap	xWareSoft	LIVRE
XSOAP	Extreme! Computing Lab	LIVRE
Apache Soap	Apache Fundation	LIVRE
JAXM	Sun Microsystems	LIVRE
AXIS	Apache Project	LIVRE
ESOAP	Embedding.net	LIVRE
ZOAP	jBoss.org	LIVRE
SOAP::Lite	Paul Kulchenko	LIVRE

Tabela 3: Principais implementações SOAP.

4 Exemplo de aplicação

A aplicação de exemplo tem por objetivo demonstrar a utilizaçao do protocolo SOAP para proporcionar a interoperabilidade de sistemas feitos em arquiteturas diferentes de objetos distribuídos. Consiste de um simples cadastro de clientes e cujo código foi escrito utilizando o GLUE como implementação SOAP, a linguagem de programação Java e tecnologias da sua plataforma J2EE (Java *Enterprise Edition),* sendo elas: JSP (*Java Server Pages*) e EJB (*Enterprise Java Beans*), e RMI.

Nesta aplicação de cadastro consideramos a necessidade de se fazer uma pequena validação prévia de uma das informações fornecidas pelo cliente antes de cadastrá-lo. No exemplo em questão trata-se do CPF. A validação é feita através da utilização de um *web service* simples, feito exclusivamente para esta finalidade e utilizando a linguagem de programação Java. As aplicações, serviço de cadastro e serviço de validação de CPF, se comunicam utilizando o protocolo SOAP onde o

serviço de cadastramento envia ao validador uma mensagem contendo o CPF do cliente a ser cadastrado e recebe de volta uma mensagem contendo a informação sobre a validade ou não do número do CPF informado.

Na estrutura dos serviços são utilizadas as seguintes ferramentas:

- Tom Cat : Servidor web e servlet engine.
- JBOSS : Servidor de aplicação (Application Server).
- GLUE : Implementação SOAP.
- Como SGBD foi utilizado o ORACLE.

O cenário é composto por dois ambientes distintos:

No primeiro ambiente (A) funciona uma aplicação desenvolvida em camadas, utilizando a linguagem de programação Java e as tecnologias de sua plataforma de desenvolvimento J2EE. Esta aplicação é responsável por receber e armazenar em banco de dados as informações dos clientes.

No segundo ambiente (B) funciona o *web service*, escrito na linguagem de programação Java e utilizando o GLUE como implementação SOAP.

A estrutura do primeiro ambiente é formada por uma estação servidora (1) contendo um servidor *web / servlet engine* (Tom Cat), onde ficam as *home pages, JSPs* e um servidor de aplicação(JBOSS), onde ficam as classes de transação que representam as regras de negócio, e por uma segunda estação servidora (2)

contendo um servidor de aplicação (JBOSS), onde ficam os objetos responsáveis pela persistência dos dados dos usuários.

A estrutura do segundo ambiente é composta apenas por uma estação servidora (3) onde funciona o *web service* responsável pela validação dos números de CPF.

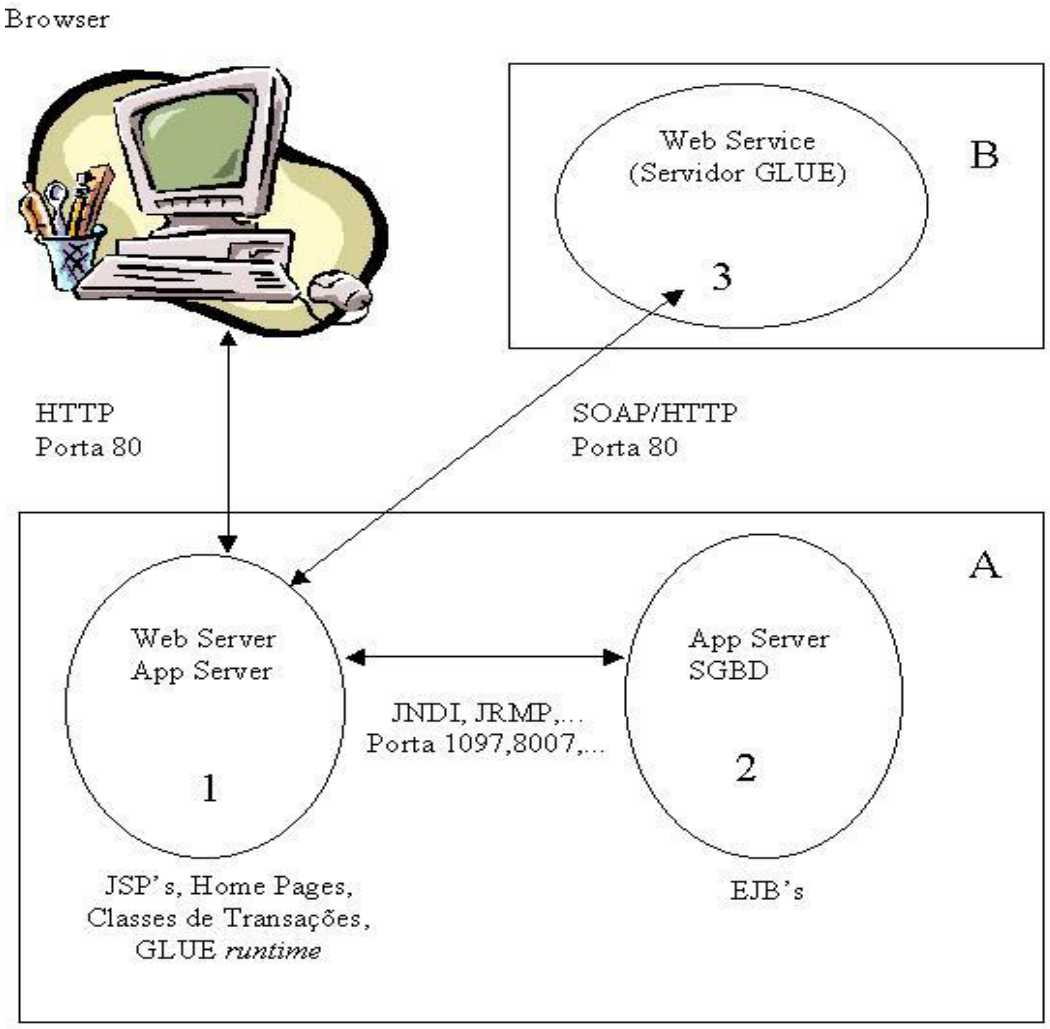

Figura 19: Cenário de execução do exemplo de aplicação

Na estação cliente o usuário solicita o seu cadastramento fornecendo seus dados através do seu navegador *web*. Fornecendo a URL do *web site* onde está disponível o serviço de cadastramento será exibida na tela do usuário a *home page* enviada pelo servidor *web* sob a forma de JSP. Após o cliente informar seus dados, o *browser* do cliente os envia ao servidor *web* onde está o serviço de cadastramento de clientes utilizando protocolo HTTP e a porta de comunicação 80 (padrão dos servidores *web*). Ao receber os dados do cliente o servidor *web* repassa o arquivo JSP para o *servlet engine*, que o processa, extraindo as informações fornecidas pelo usuário e dando início às validações.

Para efetuar a validação do CPF é feita a localização e ligação ao serviço de validação (*web service*) através da Internet. É efetuada então, a chamada ao método responsável por receber o CPF a ser validado. O retorno do método representa se o CPF é válido ou não. Caso o *web service* retorne a informação que o CPF informado é válido, o *servlet engine* se comunica com o servidor de aplicação onde se encontra o objeto responsável pela persistência das informações do cliente. É feita a busca pelo objeto remoto (*lookup*) e, de posse da interface remota, é feita a chamada para o método responsável por gravar em banco de dados as informações do usuário cujo CPF foi validado com sucesso anteriormente.

A aplicação possui a opção de listar os clientes efetivamente cadastrados na base de dados.

O *web service* que fornece o serviço de validação de CPF é composto por uma classe (CpfImp) que disponibiliza os métodos utilizados pela validação, conforme

ilustrado no apêndice II. Esta classe contém a implementação dos métodos definidos pela interface (Cpf) que implementa, conforme ilustrado no apêndice I e utiliza uma classe (CpfCnpj) que contém as regras de validação de CPF, conforme ilustrado no apêndice III. O diagrama de classes deste *web service* está representado na *figura 20.*

Finalmente o serviço é registrado e inicializado utilizando como implementação SOAP o GLUE, Conforme ilustrado no apêndice IV.

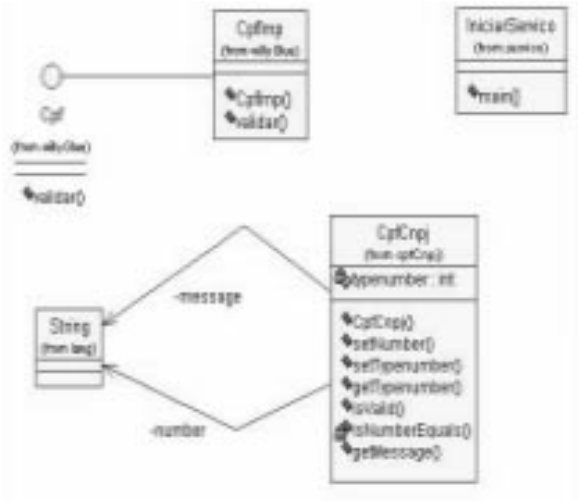

Figura 20: Diagrama de classes do web service.

O cadastro de clientes é implementado pela classe CadastrarCliente, que contém a implementação dos métodos necessários para as tarefas de localização e utilização dos objetos e métodos envolvidos no procedimento de cadastramento de clientes, cuja aplicação tem o seu diagrama de classes demonstrado na figura 21.

O método *execute()* está ilustrado no apêndice V*,* pertence à classe de transação “CadastrarCliente” e contém o procedimento principal do cadastramento

de clientes efetuando a chamada para os métodos responsáveis por cada etapa do processo.

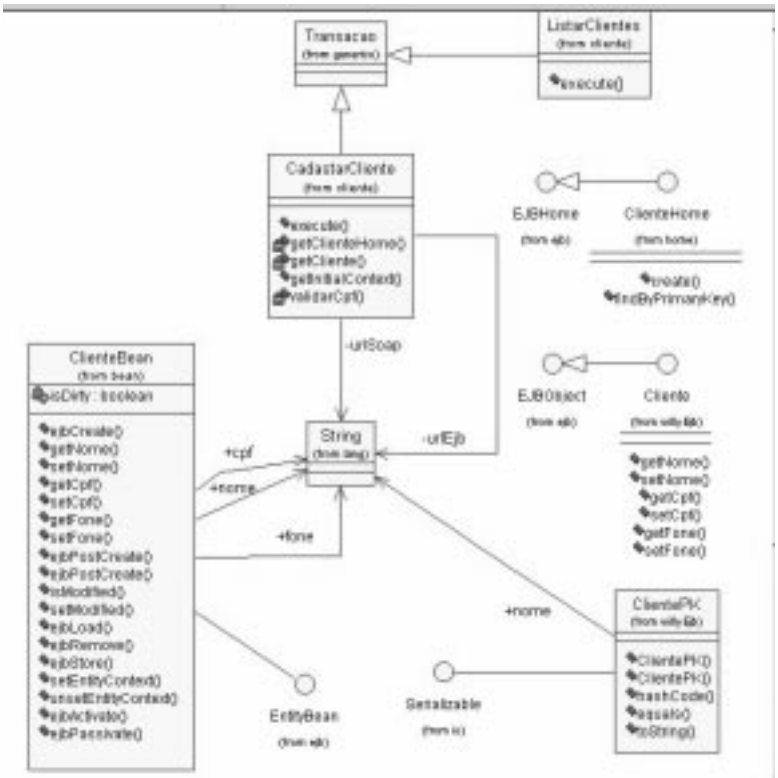

Figura 21: Diagrama de classes da aplicação de cadastramento de clientes

A partir de um documento XML a transação obtém a informação de onde encontrar o *web service* de validação de CPF e onde encontrar os objetos de persistência dos dados do cliente (EJBs). Um trecho deste documento XML está ilustrado no apêndice VI.

O método validarCPF(), ilustrado no apêndice VII, é responsável pela ligação a um *web service* específico para validação de CPF, tendo como um dos parâmetros a URL do *web service*, obtida do arquivo XML, ilustrado no apêndice VI, que contém

as definições da aplicação. No exemplo em questão esta URL é http://wally/glue/urn:cpf.wsdl.

Uma vez que o CPF informado foi validado com sucesso, através do método getInitialContext() será feita a ligação ao servidor de aplicação que contém os objetos remotos. A implementação deste método vai sofrer algumas modificações a depender do servidor de aplicação utilizado. No exemplo em questão a implementação é direcionada ao JBoss e utiliza a informação do servidor de aplicação obtida do arquivo XML que contém as definições da aplicação, conforme ilustrado no apêndice VI que, neste exemplo tem valor "wally:1097". Após a ligação ao servidor é necessário fazer a pesquisa para obtenção da interface remota (ClienteHome) do EJB responsável (ClienteBean) e executar a persistência dos dados dos clientes através do método create(). Este processo é efetuado pelos métodos ilustrados nos apêndices IX e X, respectivamente.

5 CONCLUSÃO

Apesar do SOAP ser uma solução para uma série de problemas de integração entre sistemas distribuídos, uma visão crítica do protocolo envolvendo todos os aspectos da sua utilização nos permite facilmente perceber que, embora possa trazer grandes benefícios de imediato, ele não é a solução para todos os problemas e a decisão de se utilizar o SOAP como protocolo de comunicação do sistema distribuído deve ser tomada somente após uma cuidadosa avaliação da natureza e finalidade da aplicação, ambiente de execução, tipos de processamentos envolvidos e recursos de equipamentos disponíveis. SOAP não deve ser utilizado por se tratar de uma nova tendência comercial suportada por várias empresas e sim se os recursos que disponibiliza representarem um ganho efetivo em relação aos objetivos que se pretende atingir e ao cenário já existente. Decisões desse tipo devem ser tomadas em nível de engenharia de software e devem contar com o envolvimento de todos os setores envolvidos no problema como, por exemplo, *web design,*

desenvolvimento, gerencia de dados e gerencia de redes. Uma decisão errada a esse respeito pode acarretar sérios prejuízos à empresa.

Com o estudo feito podemos também concluir que SOAP é um protocolo que representa uma tecnologia definitiva como suporte à interoperabilidade de sistemas distribuídos. Porém, a sua maior aplicabilidade e o ponto onde seus recursos e sua funcionalidade se tornam evidentes, são na integração de sistemas através da Internet. Por ter como características marcantes a simplicidade e extensibilidade o protocolo uma especificação ainda bastante indefinida em vários pontos referentes ao quesito interoperabilidade. Este fato faz com que empresas comerciais tentem submeter o protocolo a camadas superiores definidas pelas empresas com o objetivo de criar uma arquitetura tão bem definida quanto de caráter extremamente comercial, fato que atualmente provoca divergência entre os vários segmentos de desenvolvedores, inclusive aqueles que colaboram com a especificação do projeto. Este ambiente provoca um impasse pois ao mesmo tempo em que o SOAP se firma como uma boa solução para problemas antigos de interação entre diferentes sistemas, a especificação da estrutura do protocolo tende a sofrer alterações a depender da reação do mercado e desenvolvedores, e assim cria-se a possibilidade de integração entre diferentes sistemas distribuídos porém nem sempre válida para todos as implementações de *framework* possíveis no mercado.

Sem dúvida, a posição do protocolo SOAP na arquitetura dos *web services* e a própria definição da sua estrutura interna poderá vir a sofrer mudanças motivadas pelo avanço das discussões sobre interoperabilidade. Os investimentos em interoperabilidade certamente passarão a ocupar posição de destaque dentro da

evolução do protocolo, merecendo especial atenção de grandes e pequenos desenvolvedores.

Por este trabalho abordar um tema relativamente novo e ainda em amadurecimento no cenário do desenvolvimento de aplicações distribuídas, alguns itens mencionados nesta pesquisa podem ser objeto de trabalhos futuros. Podemos citar entre os principais: uma estrutura de segurança aplicada ao protocolo, independente da arquitetura de desenvolvimento e das camadas que vierem a compor esta arquitetura; um estudo específico sobre a performance obtida com a utilização do SOAP sobre TCP, UDP e protocolos superiores; a definição de um padrão de compatibilidade, em nível de protocolo, entre as várias implementações SOAP existentes e que possam vir a surgir ; e a descrição de um estudo de caso de uma empresa cujos serviços estejam integrados e disponíveis através da internet com a utilização exclusiva de software livre, os benefícios obtidos e as dificuldades encontradas.

6 REFERÊNCIAS BIBLIOGRÁFICAS

[APACHE 2001] Apache SOAP v2.2 Documentation. 2001. The Apache Software Foundation. Disponível on-line em : < **http://xml.apache.org/soap/docs/index.html** >. Acessado em 20 de Outrubro de 2001.

[AXIS 2001] NELSON, C., DAVIS, D., MATKOVITS, G., DANIELS, G., KOPECKY, J., SNELL, J., MITCHELL, K., DUFTLER, M., JELLINGHAUS, R., NEYAMA, R., RUBY, S., WEERAWARANA, S., GRAHAM, S., AMIRBEKYAN, V., CLOETENS, W., NAKAMURA, Y., NAGY, B., BUTEK, R., JORDAHL, T., LORITSCH, B., SRINIVAS, D., KUMAR, R., **Axis User's Guide.** Disponível on-line em: < http://xml.apache.org/axis/index.html >. Acessado em 2 de Novembro de 2001.

[BELL 2000] **An Overview of the UNIX* Operating System**.Disponível on-line em: < http://unix.about.com/gi/dynamic/offsite.htm?site=http://www.bell%2Dlabs.com/history/unix/tutorial.html>. Acessado em: 10 de Novembro de 2001

[BOX 2000] BOX, D., EHNEBUSKE, D., KAKYVAYA, G., LAYMAN, A., MENDELSOHN, N. , NIELSEN, H. F. , THATTE, S. , WINER, D., **Simple Object**

Access Protocol (SOAP) 1.1, 8 de Maio de 2000. Disponível on-line em: < http://www.w3.org/TR/SOAP/>. Acessado em 19 de Novembro de 2001

[BRAY 2000] BRAY, T., PAOLI, J., McQUEEN C. M. S. ,MALER, E., **Extensible Markup Language**. 6 de Outubro 2000. Disponível on-line em: <http://www.w3.org/TR/2000/REC-xml-20001006>

[COSTA 1998] A. A., Desenvolvimeto de Sistemas com Objetos Distribuídos**. 1998**, **47p. Monografia de Conclusão de Curso. Universidade Tiradentes -UNIT.**

[CHRISTENSEN 2001] Erik Christensen & Greg Meredith & Francisco Curbera & Sanjiva Weerawarana. ***Web services*** **Description Language (WSDL) 1.**1. Microsoft Corporation & IBM Research, 2001. Disponível on-line em: <http://msdn.microsoft.com/library/en-us/dnwebsrv/html/wsdl.asp>

[JOHNSTON 2001] JOHNSTON, J., **Why Open Source Developers Should Care About *Web services***. 25 de Maio de 2001. Disponível on-line em: <http://www.webservicesarchitect.com/content/articles/johnston01.asp>. Acessado em: 20 de Agosto de 2001.

[FLURY 2001] FLURY, G., ***Web services*** **and IBM**. 25 de Maio de 2001. Disponível on-line em: <www.theserverside.com>. Acessado em 20 de Agosto de 2001.

[GLUE 2001] **Glue User's Guide**. 2001. The Mind Electric. Disponível on-line em : < http://www.themindelectric.com/products/glue/releases/GLUE-1.3/docs/guide/userguide.pdf >. Acessado em : 22 de Outubro de 2001.

[GROSSO 2001] GROSSO, W., **Java RMI**,Outubro de 2001. ISBN 1-56592-452-5, 572p. Disponível on-line em:

<http://www.oreilly.com/catalog/javarmi/chapter/ch10.html>. Acessado em 20 de Outubro de 2001.

[GUDGIN 2001] GUDGIN, M., HADLEY, M., MOREAU, J., NIELSEN, H. F. , **SOAP Version 1.2 Messaging Framework**. 2 de Outubro de 2001.Disponível on-line em: < http://www.w3.org/TR/2001/WD-soap12-part1-20011002/>

[JAVA 2001] **The Source for Java(TM) Tecnology**. Disponível on-line em:< http://java.sun.com/ >. Acessado em 15 de julho de 20001.

[JAVAWORLD 2001] Java World July 2001. Disponível on-line em:http://www.javaworld.com/. Acessado em 25 de julho de 2001.

[LEYMANN 2001] LEYMANN F., ***Web services*** **Flow Language - WSFL1.0** , Maio de 2001.187p. Disponível on-line em: < http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf >. Acessado em 6 de Novembro de 2001.

[McLAUGHLIN 2001] McLAUGHLIN, B., **Java and XML, 2nd Ed.** 2a. Edição , Setembro de 2001, ISBN 0-596-00197-5 , 528p. Disponível on-line em: <http://www.oreilly.com/catalog/javaxml2/chapter/ch12.html> . Acessado em 15 de Setembro de 2001

[NIELSEN 2001] NIELSEN, H. F., **Introduction to SOAP**, Disponível on-line em: <http://www.w3.org/2000/xp/Group/Admin/minutes-oct1100/soap-xp-wg_files/frame.htm#slide0188.htm>.Acessado em 08 de agosto de 2001

[OMG 2001] OMG – Object Management Group. **The Common Object Request Broke**r: Architeture and Specification. Revision 2.5, 2001. Disponível on-line em: <http://www.omg.org/cgi-bin/doc?formal/01-09-01 >.

[ONJAVA 2001] ***Web services*** . Disponível on-line em: <http://www.oreillynet.com/pub/a/onjava/2001/08/07/webservices.html>. Acessado em 14 de Agosto de 2001

[PAGNUSSAT 2001] PAGNUSSAT,T. M., **Modelo CORBA.** 2000, 10p, Monografia (Conclusão de Disciplina). Disponível on-line em: **<**http://www.sit.com.br/SeparataDIV0007.htm>. Acessado em 24 de Outubro de 2001.

[RAJ 2001] **RAJ,** G. S., A Detailed Comparison of CORBA, DCOM and Java/RMI. Disponível on-line em: <http://www.execpc.com/~gopalan/misc/compare.html**>.** Acessado em 25 de Outubro de 2001

[RAGGET 1998] RAGGET,D. , Le HORS A. , JACOBS, I. **HTML 4.0 Specification** , 24 de Abril de 1998**.** Disponível on line em: <http://www.w3.org/TR/1998/REC-html40-19980424>

[REYNOLDS 2001[a]]REYNOLDS. M., **Microsoft and *Web services***. 4 de Julho de 2001. Disponível on-line em: <http://www.webservicesarchitect.com/content/articles/reynolds01.asp>. Acessado em: 2 de setembro de 2001.

[REYNOLDS 2001[b]] REYNOLDS. M., **What Are *Web services***. 25 de Maio de 2001. Disponível on-line em: <http://www.webservicesarchitect.com/content/articles/reynolds02.asp>. Acessado em: 2 de setembro de 2001.

[RMI 1999] **Java Remote Method Invocation**. RMI Architecture and Functional Specification**.** Sun Microsystems, Inc.Disponível on-line em: <http://java.sun.com/j2se/1.4/docs/guide/rmi/spec/rmiTOC.html>

[SAGANICH 2001] SAGANICH, A. , **Java and *Web services*, Part I**. 7 de Agosto de 2001. Disponível on-line em: <http://www.onjava.com/pub/a/onjava/2001/08/07/webservices.html>. Acessado em 21 de Agosto de 2001.

[SHANNON 2001] SHANNON, B., **Java™ 2 Platform Enterprise Edition Specification, v1.3**,174p. 27 de Julho de 2001. Disponível on-line em: < http://java.sun.com/j2ee/j2ee-1_3-fr-spec.pdf >. Acessado em 6 de Novembro de 2001.

[SHARKEY 2001] SHARKEY, K., SEELY, S., SOAP: **Cross Platform Web Service Development Using XML**, Ed. Prentice Hall , Agosto de 2001,ISBN: 0130907634

[SILBERSCHATZ 2000] SILBERSCHATZ, A., GALVIN, P., GAGNE, G., **Applied Operating System Concepts**, 2000, 840p., John Wiley & Sons Inc., ISBN: 0-471-36508-4.

[SLOMINSKI 2001] SLOMINSKI, A., GOVINDARAJU, M., GANNON, D., BRAMLEY, R., Design of an XML based Interoperable RMI System: SoapRMI C++/Java 1.1. 29 de Março de 2001.Disponível on-line em: < http://www.extreme.indiana.edu/soap/rmi/design/ >.Acessado em: 16 de Novembro de 2001

[SUN 2001] Sun Microsystems Inc. The J2EE Tutorial.2001.Disponível on-line em: < http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html>. Acessado em 19 de Novembro de 2001

[TECHMETRIX 2001] **Modelos das arquiteturas RMI e CORBA**. Disponível on-line em: <http://www.techmetrix.com/lab/labindex.shtml>. Acessado em 26 de Outubro de 2001

[UDDI 2000] **UDDI Technical White Paper.** Ariba, Inc. IBM Corporation & Microsoft Coporation, 2000. Disponível on-line em: <http://www.uddi.org>. Acessado em 5 de Novembro de 2001.

[W3C 2001] **World Wide Web Consortium**. Disponível on-line em: <http://www.w3c.org>. Acessado em 25 de julho de 2001

[WINER 2001] WINER D., **Unstalling SOAP.** 29 de Março de 2001. Disponível on-line em: < http://davenet.userland.com/2001/03/29/unstallingSoap > . Acessado em 19 de Novembro de 2001.

7 ANEXOS

7.1 ANEXO I - Código Fonte da aplicação

```
package br.com.willyGlue;

/**
 * Title:        Projeto Soap Glue
 * Description:  Validação de Cpf
 * Copyright:    Copyright (c) 2001
 * Company:      EasyNet Cia
 * @author Willy
 * @version 1.0
 */

public interface Cpf {

  /**
   * Valida o cpf
   * @param nrCpf Número do Cpf
   * @return true|false
   */
  public boolean validar(String nrCpf);
}
```

Quadro I: Interface que define os métodos para validação de CPF

```
package br.com.willyGlue;

import br.com.willyGlue.cpfCnpj.CpfCnpj;

public class CpfImp implements Cpf {

  public CpfImp() {
  }

  /**
   * Validação de Cpf
   * @param nrCpf Número do Cpf
   * @return true|false
   */
  public boolean validar (String nrCpf) {
    System.out.println("Executando o Método Validar()");
    String cpfAux = nrCpf;
    if (nrCpf.length() == 14) {
      nrCpf = cpfAux.substring(0, 3);
      nrCpf += cpfAux.substring(4, 7);
      nrCpf += cpfAux.substring(8, 11);
      nrCpf += cpfAux.substring(12);
    }
    CpfCnpj cpfCnpj = new CpfCnpj();
    cpfCnpj.setTypenumber(1);
    cpfCnpj.setNumber(nrCpf);
    return cpfCnpj.isValid();
  }
}
```

Quadro II: Classe que fornece o serviço de validação de CPF

```
package br.com.willyGlue.cpfCnpj;

public class CpfCnpj
{
   private  String  number;
   private  int     typenumber;  // 1 - CPF  2 - CNPJ
   private  String  message;

...
            enumber == 1) {           // CPF
        if(number.length() == 11 &&
!isNumberEquals(number)) {
            for (int i=0; i < 9; i++)

               soma += (10 - i) * (number.charAt(i) - '0');
            soma = 11 - (soma % 11);
            if (soma > 9) soma = 0;
            if (soma == (number.charAt(9) - '0')) {
               soma = 0;
               for (int i=0; i < 10; i++)
                  soma += (11 - i) * (number.charAt(i) - '0');
               soma = 11 - (soma % 11);
               if (soma > 9) soma = 0;
               if (soma == (number.charAt(10) - '0')) {
                  message = "CPF Válido";
                  return true;
               }
            }
         }
         message = "CPF Inválido";
```

Quadro III: Trecho da classe que implementa a lógica de validação do CPF

```
package br.com.willyGlue.servico;


// Classes Soap Glue
import electric.registry.Registry;
import electric.server.http.HTTP;

// Classes Cpf
import br.com.willyGlue.CpfImp;

public class IniciarServico {

  public static void main(String[] args) {
   try {
     // Iniciando o serviçõ na porta 80, aceita mensagens via/glue
     HTTP.startup( "http://localhost:80/glue" );
     // Registrando a instancia do CpfImp no servidor
     Registry.publish( "urn:cpf", new CpfImp() );
   } catch ( Exception e) {
     e.printStackTrace();
   }
  }
}
```

Quadro IV: Classe que inicia e registra o serviço de validação de CPF

```
  /**
   * Executa a operacao de cadastro de cliente
   */
  public void execute () throws Exception {
    Documento docEntrada = getDocEntrada();
    // Lendo os dados da interface
    String nome = (String)docEntrada.getObject("nome");
    String cpf = (String)docEntrada.getObject("cpf");
    String fone = (String)docEntrada.getObject("fone");
    urlEjb = getPropriedade("UrlEjb");
    urlSoap = getPropriedade("UrlSoap");
    try {
      // Verificando se o cpf é válido
      if (validarCpf(cpf)) {
        // Cadastrando o cliente
        getCliente(nome, cpf, fone);
      } else {
        throw new DadoException("Cpf invalido!");
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
```

Quadro V: Método execute()

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE projeto SYSTEM "file://localhost/D:/EasyNet/dtd/GPEN_Enterprise.dtd">
<projeto>
  <informacao id="Willy" nome="Monografia Willy" autor="Willy" descricao="ApresentaÃ§Ã£o da Monograifa"/>
  <usuario login="U_willy" nome="Willy" fone="246-5021" senha="1234"/>
  <!-- Propriedades do Projeto -->
  <propriedade nome="UrlEjb" valor="wally:1097"/>
  <propriedade nome="UrlSoap" valor="http://wally/glue/urn:cpf.wsdl"/>
  <propriedade nome="NomePool" valor="java://WillyPool"/>
```

Quadro VI: Trecho do arquivo XML que contém as definições do projeto

```
private boolean validarCpf (String nrCpf) throws Exception {
  Cpf cpf = (Cpf) Registry.bind( urlSoap, Cpf.class );
  boolean isValido = cpf.validar( nrCpf );
  System.out.println( "Validando cpf = " + isValido );
  return isValido;
}
```

Quadro VII: Método que valida um CPF através de um *web service*

```
/**
 * Obtendo o Contexto para pesquisa do objeto remoto
 */
public Context getInitialContext()
throws Exception {
  Properties p = new Properties();
  p.put(Context.INITIAL_CONTEXT_FACTORY,
    "org.jnp.interfaces.NamingContextFactory");
  p.put(Context.PROVIDER_URL,urlEjb);
  return new InitialContext(p);
}
```

Quadro VIII: Método getInitialContext()

```
/**
 * Efetua um lookup no bean "cliente"
 * @return Obtém a interface remota do cliente
 */
private ClienteHome getClienteHome () throws Exception {
  return (ClienteHome) getInitialContext().lookup( "Cliente" );
}
```

Quadro IX: Método que pesquisa determinado objeto no servidor de aplicação

```
/**
 * Cria um novo cliente
 * @param nome Nome do Cliente
 * @param nome cpf do Cliente
 * @param nome fone do Cliente
 */
private Cliente getCliente (String nome, String cpf, String fone)
                    throws Exception  {
  return getClienteHome().create(nome, cpf, fone);
}
```

Quadro X: Persistência dos dados no SGBD através do objeto remoto