UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2585

WEB SERVICES TUTORIAL

Júlio Cézar Estrella
Regina Helena Carlucci Santana
Marcos José Santana

Nº 77

NOTAS DIDÁTICAS

São Carlos
Fev/2009

Web Services Tutorial

Júlio Cézar Estrella, Regina Helena Carlucci Santana, Marcos José Santana

Departamento de Sistemas de Computação
Grupo de Sistemas Distribuídos e Programação Concorrente
Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação
{jcezar, rcs, mjs}@icmc.usp.br
http://www.lasdpc.icmc.usp.br

Resumo O propósito deste documento é prover uma breve introdução da tecnologia de *Web Services*, seus protocolos e padrões, bem como descrever o processo de instalação e configuração de um motor de processamento de mensagens *SOAP* denominado Apache Axis2. *Web Services* é uma tecnologia relativamente recente que tem crescido rapidamente nos últimos anos em função da heterogeneidade da *Web* e do surgimento de novas aplicações distribuídas em que o foco é a interoperabilidade.

1 Introdução

O termo *Web Services* tem sido muito abordado nos últimos anos. Várias são as definições apresentadas pela literatura. Uma definição amplamente aceita apresenta *Web Services* como um componente, ou unidade lógica de aplicação, acessível através de protocolos padrões da *Internet*, possuindo uma funcionalidade que pode ser reutilizada sem a preocupação de como é implementada. Web Services são referenciados como a implementação de uma arquitetura orientada a serviços - *SOA* - (Service Oriented Architecture) [3].

A W3C define o termo como uma aplicação identificada por uma *URI* - (Uniform Resource Identifier), cujas interfaces e ligações são definidas, descritas e descobertas utilizando-se uma linguagem padrão como *XML* - *Web Services* ocorrem tipicamente como chamadas *SOAP* - (Simple Object Access Protocol), protocolo de comunicação baseado em *XML* (uma linguagem padronizada, adequada para descrever as interfaces dos *Web Services*, que são acessíveis por uma grande variedade de plataformas e linguagens de programação) para a interação de aplicações [8]. *SOAP* é apresentado como um *backbone* para uma nova geração de aplicações de computação distribuída, independente de plataforma e de linguagens. Além disso, as descrições de interfaces dos *Web Services* são expressas usando uma linguagem denominada *WSDL* - (Web Service Description Language) [10].

A pilha conceitual dos *Web Services* relaciona três camadas principais: camada física *(wire layer)*, camada de descrição *(description layer)* e camada de descoberta *(discovery layer)*, mostradas na figura 1.

Para a construção e utilização de *Web Services* é necessário considerar algumas especificações e tecnologias:

- Uma maneira de representar dados
- Um formato de mensagens comum e extensível
- Um mecanismo para localizar os serviços presentes em *web sites* específicos
- Um mecanismo para descobrir os provedores de serviços

Em resumo, um *Web Service* pode ser definido como um serviço de *software* publicado na *Web* através do protocolo *SOAP* [5], [6], descrito em uma interface *WSDL* e registrado num repositório denominado *UDDI* [2]. Os protocolos acima citados serão objetos de estudo nas próximas seções deste documento.

2 XML

XML é o acrônimo para *eXtensible Markup Language*. A informação armazenada em um documento XML é estruturada usando elementos e atributos, em que os elementos são os nomes das tags:

```
<Element> ... </Element>
```

Os atributos, por sua vez, são definidos dentro da tag *Element*.

```
<Element attribute= "..."> ... </Element>
```

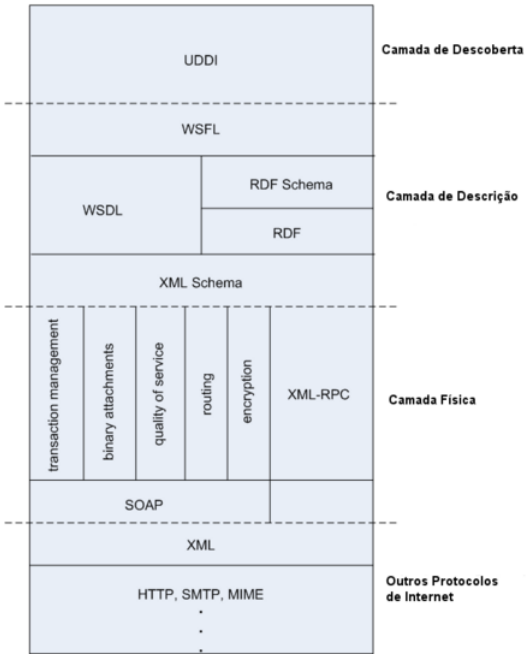

Figura 1. Pilha conceitual dos *Web Services* [1]

2.1 XML Schema

XML Schema é uma maneira de definir a estrutura de documentos *XML*. Ele pode ser utilizado para definir tipos de dados em um modelo de dados. Um documento *XML* pode ser uma instância de um *XML Schema* [11]. Tais documentos *XML* podem ser validados em oposição ao *Schema* para verificar que ele está em conformidade com o esquema. *XML Schema* pode ser comparado às definições de classes em programação orientada a objetos. Não apenas um tipo de dado em um modelo de dados pode ser expressado em uma linguagem de computador usando definições de classe, mas também tipos de dados podem ser expressados usando *XML Schema*. Isto permite a troca de tipos de dados entre diferentes linguagens de programação e plataformas. Esta importante característica do *XML* é utilizada intensamente em *Web Services*.

Um dos usos mais comuns para schemas é verificar que um documento XML é válido de acordo com um conjunto de regras definidas. Um schema pode ser utilizado para validar [11]:

- A estrutura de elementos e atributos
- A ordem de elementos
- Os valores de dados de atributos e elementos, baseado em intervalos enumerações e padrões de correspondência
- A singularidade dos valores em uma instância

2.2 Namespaces

XML Namespace é um método para evitar conflito entre elementos com o mesmo nome. Eles são definidos por uma recomendação da W3C chamado Namespaces em XML. Uma instância XML pode conter elemento ou nomes de atributo de mais de um vocabulário XML [14]. Se cada vocabulário possui um namespace, então a ambigüidade entre elementos ou atributos com mesmo nome pode ser resolvida, como mostrado a seguir:

```
<namespace:Element> ... </namespace:Element>
```

Para evitar repetir uma declaração de *namespace* para cada elemento que lhe pertence, a declaração de um prefixo pode ser feita em um elemento pai [9]. Se, por exemplo, houver dois elementos com o nome *"element"* e quisermos usar dois *namespaces* separados para esses, é preciso escrever:

```
<ParentElement xmlns:ns1=
      "http://meunamespace.com.br/NS1 xmlns:ns1=
      "http://meunamespace.com.br/NS2>
     <ns1:Element> ... </ns1:Element>
     <ns2:Element/>
</ParentElement>
```

O primeiro *Element* no exemplo anterior pertence ao *namespace*,

- ***http://meunamespace.com.br/NS1***,

enquanto o segundo pertence ao *namespace*,

- ***http://meunamespace.com.br/NS2***.

O uso do endereço http para a declaração do *namespace* é apenas uma convenção. Qualquer string pode ser utilizada.

3 Web Services

Um *Web Service* é um sistema de software construído para suportar interoperabilidade na interação entre máquinas sobre uma rede de computadores [13]. *Web Services* são auto-contidos, auto-descritivos, modulares e independentes de plataformas e sistemas operacionais. Outros sistemas interagem com o *Web Service* em uma maneira prescrita pelo próprio *Web Service*, usando mensagens *SOAP*, normalmente transferida usando o protocolo HTTP. Devido ao alto nível de padronização que os *Web Services* requerem, desenvolvedores consideram que seja fácil integrá-los e reutilizá-los. Combinado com sua utilização generalizada, isso os fazem uma tecnologia adequada para a integração de serviços. As plataformas de desenvolvimento mais populares e linguagens de programação de alto nível provêem atualmente funcionalidades que tornam fácil acessar os *Web Services*.

3.1 SOAP

SOAP, originalmente definido como (Simple Object Access Protocol) é um protocolo usado para a troca de documentos *XML* entre Web Services [12]. *SOAP* introduz alguma estrutura adicional aos documentos *XML*. Uma mensagem *SOAP* é um elemento *Envelope* que consiste de um elemento *Header* e este por sua vez consiste de um elemento *Body* [6].

```
<Envelope>
     <Header> ... </Header>
     <Body> ... <Body/>
</Envelope>
```

Esta estrutura permite a separação de metadados contido no *Header* do *payload* da mensagem presente no elemento *Body*. Este metadado pode ser a informação de roteamento, informação de expiração ou outra mensagem qualquer. Um motor de processamento *SOAP* também define:

- Um modelo de processamento
- Um mecanismo de manipulação de erros
- Um modelo extensível
- Um mecanismo para representação de dados
- Uma convenção para *RPC (Remote Procedure Call)*
- Um framework para o *binding* de protocolo

3.2 Motores de processamento SOAP

Um motor de processamento *SOAP* é um framework utilizado em clientes e provedores de *Web Services*, e apresenta como funções principais:

- A serialização de objetos de uma linguagem de programação em mensagens *SOAP*
- A deserialização de mensagens *SOAP* em objetos em uma linguagem de programação, ou seja, criar tipos de dados apropriados e populá-los com o conteúdo da mensagem.

Para atuar como um motor de processamento *SOAP*, o Axis2 possui características importantes que facilitam a geração automática do código cliente e servidor do *Web Service*. Essa geração automática pe feita baseado em classes Java ou arquivos *WSDL*.

3.3 WSDL - Web Services Description Language

A interface de um *Web Service* é definida usando o formato *XML* denominado *WSDL*. A interface é independente da implementação do Web Service. Um documento *WSDL* pode não necessariamente parecer complexo à primeira vista, mas eles não são adequados para aleitura de seres humanos. Enquanto provêem um alto nível de flexibilidade em termo de interface para *Web Services*, os documentos *WSDL* devem também permitir que o mapeamento *WSDL-SOAP* não

seja ambíguo. Em outras palavras, a interface do *Web Service* deve sempre ser interpretada do mesmo modo por clientes que fazem o acesso. A especificação formal da *WSDL* pode ser encontrada em ***http://www.w3.org/TR/wsdl***. Um documento *WSDL* define uma ou mais operações e as mensagens que elas recebem e retornam. Estas operações e mensagens são primeiro descritas de forma abstrata e então vinculados a uma ou mais instâncias concretas em termos de protocolo, formato de mensagem e endereço de rede, o que constitui o ***endpoint***. Esta flexibilidade permite a reutilização de definições abstratas.

Um exemplo de documento *WSDL* é mostrado abaixo:

```
<definitions>
 </types>
 <message/><part/></message>
 <portType></operation></portType>
 <binding/>
 <service></port></service>
</definitions>
```

Um documento *WSDL* consiste das seguintes partes:

- **types**: Declaração de tipos de dados usando algum sistema de tipo (normalmente *XML Schema*). Um *XML Schema* externo também pode ser importado.
- **message elements**: Definições abstratas de o que pode ser comunicado, ou seja, mensagens de entrada e saída e seus tipos de dados. Tipos de dados são definidos em elementos filhos denominados *part* e podem se referir aos tipos declarados nos elementos *types* descritos acima.
- **portType**: Um *portType* contém um ou mais elementos *operation*. Cada operação especifica de forma abstrata uma ação suportada pelo serviço e a mensagem que pode ser enviada por/para o serviço. O conjunto de operações em um *PortType* é suportado por um ou mais *endpoints*.
- **binding**: Um *binding* define o formato da mensagem e detalhes de protocolos para operações e mensagens definidas por um *portType*, ou seja, o mapeamento *WSDL-SOAP* do serviço (prevendo que o *binding* utiliza *SOAP*). As regulações de formato de dados e codificações são geralmente ***RPC/encoded***, ***RPC/literal*** ou ***document/literal***. Isto é também conhecido como estilo do *Web Service*.
- **service**: Uma coleção de *endpoints*. Cada *endpoint* é representado por um elemento *port* e cada *port* define um endereço de rede e um nome para um *binding*, determinando assim uma instância de *Web Service*.

4 Desenvolvimento de Web Services

Nesta seção será descrito como serviços funcionam sob a visão do provedor de *Web Services*, ou seja, o que ocorre quando um *Web Service* do lado do provedor de serviços recebe uma mensagem *SOAP* de um *Web Service* cliente.

Como foi visto na seção anterior, um documento *WSDL* é usado para descrever a interface de um *Web Service* para os seus clientes. Um cliente não necessita de qualquer conhecimento sobre o *Web Service* além do que ele vai encontrar na *WSDL*. Do lado do provedor, o *Web Service* é responsável por:

1. Deserializar mensagens *SOAP* que chegam do *Web Service* cliente
2. Implementar um serviço
3. Serializar objetos resultantes em mensagens *SOAP* de reposta
4. Enviar o resultado de volta ao *Web Service* cliente.

Uma engine *SOAP* é responsável pela implementação em baixo nível dos itens 1 e 3 acima descritos. O código do servidor (chamado de *stub* do servidor) interage com a engine *SOAP* de acordo com a funcionalidade do *Web Service*. O *stub* do servidor pode ser gerado automaticamente. É preciso haver uma correspondência completa entre a interface *WSDL* e o *stub* do servidor, uma vez que o *stub* do servidor é a implementação do que é descrito na *WSDL*. O código que implementa a lógica do *Web Service* do lado do servidor é conhecido como código *back-end*. O código *back-end* pode conter qualquer coisa, que vai desde um sistema de software em larga escala com múltiplas interfaces de usuário, até poucas linhas de implementação de código.

Os requisitos descritos anteriormente podem ser aplicados em um motor de processamento de mensagens *SOAP* como o Apache Axis2, ou qualquer outra plataforma de desenvolvimento de *Web Services*. Para construir um serviço, duas alternativas possíveis são mostradas a seguir:

1. Gerar o *stub* do servidor e um documento *WSDL* do código back-end existente
2. Gerar o *stub* do servidor a partir de um arquivo *WSDL* e conectá-lo manualmente com o código *back-end*.

O primeiro método é a mais rápida estratégia e requer pouco esforço do programador. Entretanto, o arquivo *WSDL* e o *stub* presentes no servidor são produtos da ferramenta com os quais eles foram gerados. Neste caso, o desenvolvedor tem pouco controle sobre os detalhes da implementação. Em alguns casos, o *Web Service* gerado pode somente ser entendido por clientes usando a mesma ferramenta para deserialização.

Para permitir que a interface *WSDL* e o serviço suportem rapidamente as normas envolvidas e as melhores práticas, incluindo *WS-I (Web Services Interoperability)*, um controle fino sobre o desenho do serviço é primordial. Com a existência de ferramentas, desenvolvedores são forçados a operar em um nível mais baixo e ter que desenhar o arquivo *WSDL* manualmente para assegurar uma boa interoperabilidade. Devido a este fato, a recomendação é utilizar o segundo método na maioria das situações.

4.1 Estilos de Ligação - Binding

Em um documento *WSDL*, um estilo de ligação (*binding*) descreve o mapeamento *SOAP* do *Web Service*, ou seja, como os tipos de dados recebidos e enviados pelo *Web Service* devem ser traduzidos/encapsulados em um envelope *SOAP*. Este passo é caracterizado no *binding* do documento *WSDL* e implementado pela engine *SOAP* utilizada. Um *binding SOAP-WSDL* pode ou ser no estilo *RPC (Remote Procedure Call)* ou no estilo *Document*. Assim, documentos de entrada e saída podem ser ou *literal* ou codificado (*encoded*).

Há um estilo denominado ***literal/wrapped*** que segue algumas convenções adicionais de como o serviço e o documento *WSDL* devem ser construídos. Em suma, isto dá origem a cinco diferentes estilos, embora somente quatro sejam utilizados na prática:

1. *RPC/encoded*
2. *RPC/literal*
3. *Document/literal*
4. *Document/literal wrapped*

Esta convenção de nomes *RPC x Document* é talvez infeliz, uma vez que não há erro em trabalhar com um *Web Service* estilo *literal* com operações que se comportem como chamadas remotas de procedimento. O termo *RPC* simplesmente se refere a uma convenção indicando como deserializar objetos para *SOAP*, e nada mais. A recomendação é usar o estilo ***Document/literal wrapped*** na maioria dos casos. Para saber mais sobre outros estilos e como eles se referem, consulte o site:

– ***http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/***.

Este documento abordará somente o estilo ***Document/literal wrapped***.

Estilo Document/literal wrapped

É um caso especial de estilo ***literal/wrapped***. Este estilo define que qualquer declaração de tipos de dados são descritos usando *XML Schema*. As mensagens *SOAP* enviadas de/para um *Web Service*, seguindo este estilo, podem assim serem validadas em relação a este equema, usando um validador *XML*. Isto é uma característica importante e muito útil. Assim, cada parte da mensagem usada para um mensagem de entrada deve se referir a um elemento apresentado na declaração de tipo, que por sua vez possui o mesmo nome da operação. Isto é chamado de *wrapper element*. Sendo assim, o nome da operação aparece na mensagem *SOAP* de chegada/entrada. Para entender melhor esta abstração, a seguir será apresentado um exemplo de um serviço com uma operação denominada ***getAddNumber***. Esta operação aceita dois números e retorna a soma destes como resposta.

```
<types>
 <schema
  <element name="getAddNumber">
   <complexType>
    <numbers>
     <element name="X" type="xsd:int"/>
     <element name="Y" type="xsd:int"/>
    <numbers/>
   <complexType>
  <element/>
```

```
 <element name="getAddNumberResponse">
  <complexType>
   <number>
    <element name="numberResult" type="xsd:int"/>
   <number/>
  <complexType>
 <element/>
<schema

<message name="getAddNumberRequest"/>
 <part name="parameter" element="getAddNumber"/>
<message/>

<message name="getAddNumberResponse"/>
 <part name="parameter" element="getAddNumberResponse"/>'
<message/>

<portType name="XY"/>
 <operation name="getAddNumber"/>
  <input message="getAddNumberRequest"/>
  <outuput message="getAddNumberResponse"/>
 <operation/>
<portType/>

<binding ... />
```

Uma mensagem *SOAP wrapped* para uma requisição de chegada/entrada de acordo com esta operação, deve se parecer como:

```
<soap:envelope>
 <soap:body>
  <getAddNumber>
   <X>1</X>
   <Y>2<Y/>
  <getAddNumber/>
 <soap:body/>
<soap:envelope/>
```

Um estilo de *Web Service* ***literal/wrapped*** possui as seguintes características:

- A mensagem de entrada/chegada na interface *WSDL* possui uma única *part* que se refere a um elemento.
- O *element wrapper* tem o mesmo nome que o da operação.
- Não há qualquer atributo em *element wrapper*

O estilo apresenta algumas vantagens:

– Não há declaração de tipos para variáveis em mensagens *SOAP* (oposto ao estilo *encoded* (codificado)).
– O corpo (*body*) da mensagem *SOAP* é definido por um *XML Schema* e pode ser validado de acordo com este *schema*.
– O método *name* aparece em mensagens *SOAP* de entrada/chegada.

5 Web Services com Apache Axis2

O Axis2 é a evolução natural da mais conhecida *API - Application Programming Interface* para *Web Services* nos dias atuais. Diferentemente do *Axis 1.0*, o *Axis2* apresenta diversas melhorias, principalmente em relação ao desempenho e também em relação à modularidade. A arquitetura do Axis2 é separada em componentes (módulos), e subdivide-se em componentes do núcleo e componentes não pertencentes ao núcleo. Dentre as principais funcionalidades do Axis2 destacam-se:

– **Mecanismo de implantação (deployment) com base na plataforma Java 2 Enterprise Edition (J2EE) - (baseado em arquivo)**: Recursos, arquivos de configuração e binários todos em um único arquivo que descreve o serviço.
– **Hot deployment e hot update**: Possibilidade de implantar atualizar serviços sem reiniciar o servidor de aplicação. No caso da atualização é recomendável somente o uso em ambientes de teste.
– **Presença de um repositório (onde se localizam os serviços e módulos)**: Onde ficam armazenados os serviços que devem ser acessados pelos clientes.
– **Mudanças na implantação de manipuladores (módulos)**: Como a arquitetura do Axis2 é modular, tudo que não é necessário estar no núcleo é disponibilizado como módulo. Esses módulos implementam as especificações de *Web Services* (*WS-Policy, WS-Reliable, WS-Security*).
– **Novos descritores de implantação**:
  • Descritor Global (***axis2.xml***)
  • Descritor do Serviço (***services.xml***)
  • Descritor do Módulo (***module.xml***)

O arquivo ***axis2.xml*** deve apresentar:

– Parâmetros do serviço
– Tipo de Transporte do Rementente
– Tipo de Transporte do Emissor
– Fases
– Módulo Global

O arquivo ***services.xml*** deve ser caracterizado com as opções abaixo:

– Parâmetros de Nível de Serviço
– Descrição do Serviço
– Receptores de Mensagem
– Operação necessária para expor o serviço como uma operação web

– Módulos de Nível de Serviços

O arquivo ***module.xml*** deve apresentar os seguintes parâmetros:

– Manipuladores e suas regras de fases
– Parâmetros do módulo
– Descrição sobre o módulo
– Pontos finais (*Endpoints*) - No caso de utilização de mensagens confiáveis, este parâmetro é fundamental

5.1 Componentes

O Axis2 é um motor de processamento de mensagens *SOAP* bastante modular. Sua divisão é baseada em componentes, como pode ser descrito na figura 2 e destacados a seguir:

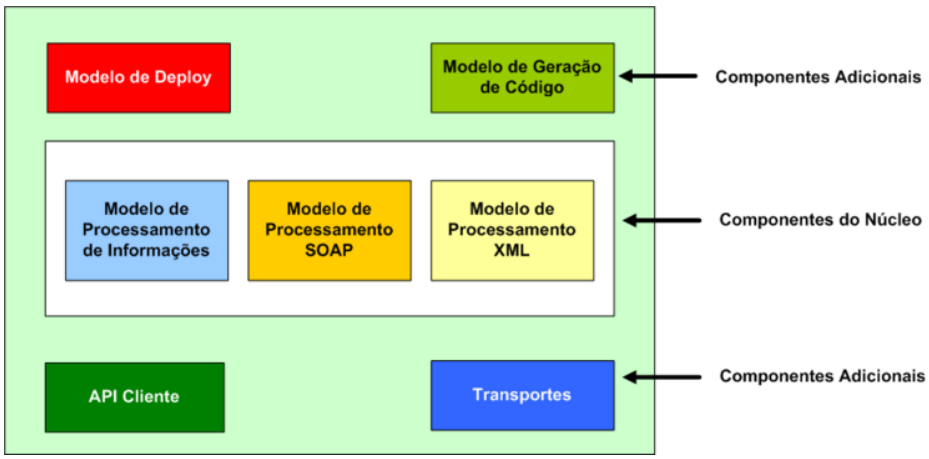

Figura 2. Componentes do Axis2 [4]

– Componentes do núcleo:
  - *AXIOM - Axis Object Model*: Modelo de Objetos em *XML*
  - Módulo de Processamento *SOAP*: Framework Manipulador
  - Modelo de Processamento de Informações: Contextos e Descrições
– Outros componentes:
  - Modelo de *Deploy*
  - Transportes
  - *API* Cliente
  - Modelo de Geração de Código

O Axis2 possui um modelo de processamento *XML* mais adequado que o Axis1, que utilizava *DOM - Document Object Model* como um mecanismo de representação *XML*. O *DOM* apresenta o problema de manter a hierarquia completa dos objetos em memória. No Axis2, qualquer mensagem que chega é representada no modelo *AXIOM*. As diferenças entre o Axis1 e Axis2 é a utilização de duas técnicas: *Pull* e *Push*. Na técnica denominada *Pull* o invocador tem o controle completo sobre o *parser* e pode perguntar sobre o próximo evento. Na técnica *Pull*, o *parser* procede até atingir o final do documento. Na técnica *Pull* o objeto é construído sob demanda, isto é, se for necessária a sua construção ele será instanciado e chamado.

5.2 Arquitetura de Implantação - Deployment Architecture

Um mecanismo de implantação de aplicações (*deployment*) baseada em repositórios é outra importante melhoria presente no Axis2. As principais partes desta arquitetura são evidenciadas na figura 3 e descritas detalhadamente abaixo:

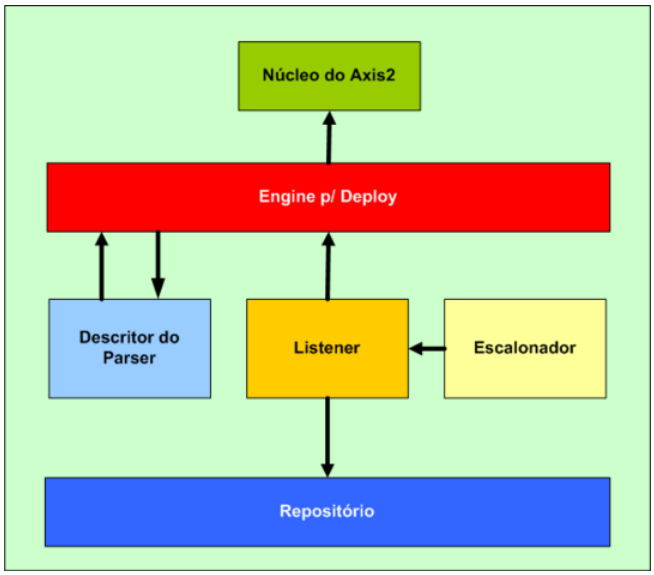

Figura 3. Arquitetura de implantação do Apache Axis2 [4]

- **Escalonador**: Componente que informa ao *listener* para procurar por serviços no repositório *Listener*: Busca por atualizações no repostório de serviços
- **Descritor do Parser**: Um componente para processar os serviços e descritores de módulos para *deploy*: O componente central da implantação (*deployment*) que faz todo o processamento lógico
- **Núcleo do Axis2**: O Axis2 é independente da implantação e vice-versa
- **Repositório**: Um diretório no sistema de arquivos que armazena os serviços implantados (*deployed*)

5.3 Ciclo de vida de um serviço no Apache Axis2

Com a utilização do Axis2 é possível construir serviços que se comunicam através de trocas de mensagens no formato *XML*, utilizando como protocolo de transporte o HTTP. A figura 4 ilustra de modo simplificado como é um ciclo de vida de um serviço.

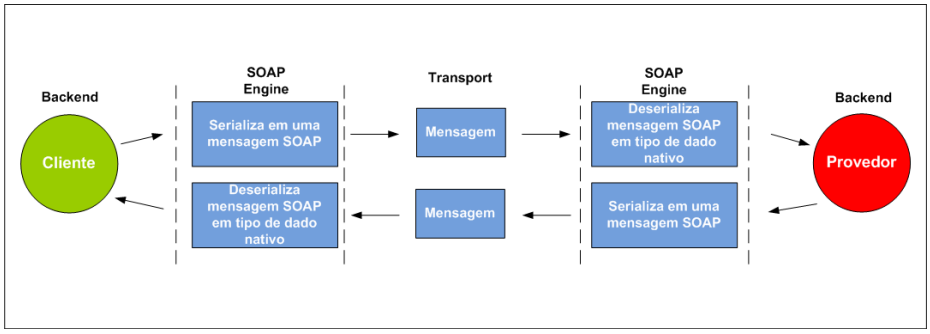

Figura 4. Fluxo de dados entre cliente e provedor de um Web Service [7]

O Axis2 executa as seguintes tarefas:

- Cria mensagens *SOAP*
- Recebe e processa mensagens *SOAP*
- Cria um *Web Service* a partir de uma classe *Java*
- Cria classes de implementação para o servidor e o cliente, usando a *WSDL*
- Recupera facilmente a *WSDL* para um serviço
- Envia e recebe mensagens *SOAP* com anexos
- Cria ou utiliza serviços que tiram a vantagem dos padrões *WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Coordination* e *WS-Atomic Transaction*

Assumindo-se que o Axis2 execute tanto no cliente quanto no provedor de serviços, a interação entre cliente e provedor deve seguir algumas fases:

- O cliente (emissor) envia uma mensagem *SOAP*
- O manipulador do Axis2 realiza ações necessárias de acordo com o que foi definido pelo usuário
- O módulo responsável pelo transporte envia a mensagem
- Do lado do receptor (provedor) a detecção da mensagem é feita pelo módulo responsável por este finalidade
- O responsável pelo trasporte (*transport listener*) passa a mensagem para um dos manipuladores do receptor (provedor)
- Uma vez processada, a mensagem é entregue à aplicação

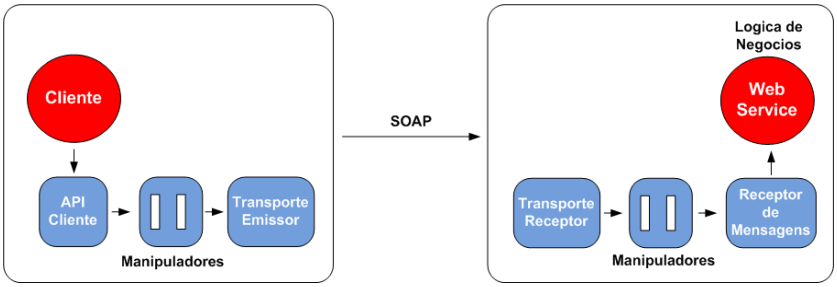

Figura 5. Axis2 e manipulação de mensagens [4]

A manipulação das mensagens *SOAP* pelo Axis2 é mostrada na figura 5 a seguir:

O Axis2 permite quebrar as ações na transmissão de mensagens em várias fases. Essas fases são definidas como:

- ***Fases Pré-Definidas***
  - Pré-Despacho
  - Despacho
  - Processamento de Mensagens
- ***Fases definidas pelo usuário***: As fases pré-definidas são invocadas independente do serviço especificado. As fases definidas pelo usuário são invocadas quando o despachante encontrar uma operação. Cada fase possui uma coleção de manipuladores. O Axis2 permite controlar quais manipuladores estarão em quais fases e a ordem em que tais manipuladores são executados dentro de cada fase. O mais interessante é que pode ser adicionada uma nova fase juntamente com seus respectivos manipuladores. Entenda-se por manipuladores os módulos componentes do Axis2 [4]. Exemplos:
  - WS-Security
  - WS-Reliability

5.4 Outras características do Axis2

Depois de apresentado as funcionalidades do Axis2 bem como sua divisão em componentes e módulos, são mostrados nesta seção outras características importantes do Axis2. Como mencionado anteriormente, o Axis2 é um processador de mensagens *SOAP*. Sendo assim, é preciso destacar que:

- Ele não conhece *WSDL*
- Também não conhece nenhum esquema de transporte e *databinding*. Essas funcionalidades são feitas através de manipuladores/módulos
- Suporta invocação de mensagens assíncronas
- Suporta 8 padrões de troca de mensagens definidas na *WSDL 2.0*
- Apresenta o conceito de Fluxo (Coleção de Fases = Coleção Lógica de Manipuladores)

Quanto à invocação de um *Web Service* por parte de um cliente, o Axis2 apresenta suporte a dois tipos de invocação:

– **Invocação Não-Bloqueante (Assíncrona)**: Neste caso, o cliente invoca o serviço sem bloquear a aplicação
– **Invocação Bloqueante (Síncrona)**: O cliente bloqueia a aplicação até que a resposta seja recebida

A invocação pode ocorrer de dois modos:

– **One-Way**: Em geral utilizado juntamente com o protocolo SMTP
– **Two-Way**: É utilizado com o protocolo HTTP

Para a invocação de clientes *Web Services*, o Axis2 apresenta uma *API* subdivida em duas classes:

– **ServiceClient**: Para usuários que querem somente enviar e receber mensagens *XML* e não se preocupam com atividades mais complexas como manipular cabeçalhos *SOAP*. Esta *API* possui alguns métodos para realizar a invocação de *Web Services*.
  • **sendRobust**: Envia mensagem *XML* e não se preocupa com a resposta
  • **fireAndForget**: Envia mensagem *XML* e não se preocupa com a resposta ou exceção.
  • **sendReceive**: Invoca um *Web Service* que tem um valor de retorno. É o mais comum e pode ser usado para invocar o padrão de troca de mensagem do tipo *in-out*
  • **sendReceiveNonBlocking**: Para invocar um *Web Service* de modo não-bloqueante. Pode ser utilizado quando o *Web Service* tem um valor de retorno.
– **OperationClient**: Para usuários que querem manipular cabeçalhos *SOAP* e outras tarefas avançadas. A *API Operation Client* é indicada a usuários avançados. Ao contrário da *API Service Client* em que o usuário não precisa conhecer detalhes para invocar um *Web Service*, utilizar a *API Operation Client* requer a sequêncua dos passos abaixo:
  1. Criar um *Web Service* cliente
  2. Criar um *Operation Client* usando o *Service Client*
  3. Criar um envelope *SOAP*
  4. Criar o contexto da Mensagem
  5. Adicionar envelope *SOAP* para o contexto da mensagem
  6. Adicionar o contexto da mensagem para o *Operation Client*
  7. Invocar o *Operation Client*
  8. Se houver uma resposta, obtenha como resposta o contexto da mensagem para o *Operation Client*.

6 Instalação e Configuração do Ambiente de Programação com Axis2

Esta seção tem como objetivo conduzir um usuário iniciante ao processo de configuração de um ambiente para a programação de *Web Services* com *Axis2*, *Java*, *Tomcat* e *Ant*.

6.1 Donwload dos Fontes

Crie um diretório chamado *sources*. Faça o download dos pacotes abaixo, e os copie para o diretório criado:

- ***axis2-1.3-bin.zip***
  - *http://mirror.pop-sc.rnp.br/mirror/apache/ws/axis2/1_3/axis2-1.3-bin.zip*
- ***apache-tomcat-6.0.18.zip***
  - *http://ftp.unicamp.br/pub/apache/tomcat/tomcat-6/v6.0.18/bin/apache-tomcat-6.0.18.zip*
- ***apache-ant-1.7.0-bin.zip***
  - *http://linorg.usp.br/apache/ant/binaries/apache-ant-1.7.0-bin.zip*
- ***jdk1.6.0_06***
  - *http://java.sun.com*

6.2 Descompactação dos Fontes

Descompacte cada um dos arquivos (no caso do *JDK*, execute o binário que acabou de fazer o download e quando o script de instalação perguntar em qual diretório deseja instalar a *JVM - Java Virtual Machine*, escolha ***/home/<nome-usuario>/environment/***) e mova-os para o diretório:

```
/home/<nome-usuario>/environment/
```

Depois de descompactados, parte da estrutura do diretório ***/home/<nome-usuario>/environment/*** deve ficar como mostrado a seguir:

- apache-ant-1.7.0
- apache-tomcat-6.0.18
- axis2-1.3
- axis2.war

6.3 Configuração do Classpath

É necessário, antes de iniciar a criação de *Web Services*, setar algumas variáveis de ambiente. Em distribuições Linux isso pode ser feita de vários modos:

- No ***bashrc*** do usuário
- No arquivo ***/etc/bash.bashrc***
- No arquivo ***/etc/profile***

Neste documento optaremos por anexar as informações das variáveis de ambiente no arquivo ***/home/<nome-usuario>/.bashrc***. Para isso, edite este arquivo, e ao final acrescente as informações contidas na listagem 1:

Listagem 1 Arquivo .bashrc

```
#Variavies de ambiente do JAVA
JAVA_HOME=/home/<nome-usuario>/environment/jdk1.6.0_06
JDK_HOME=/home/<nome-usuario>/environment/jdk1.6.0_06/bin
PATH=$JAVA_HOME/bin:$JDK_HOME:$PATH
MAN_PATH=$MANPATH:$JAVA_HOME/man

#Variaveis de ambiente do CATALINA
CATALINA_HOME=/home/<nome-usuario>/environment/apache-tomcat-6.0.18

#Variaveis de ambiente do ANT
ANT_HOME=/home/<nome-usuario>/environment/apache-ant-1.7.0
PATH=$ANT_HOME/bin:$PATH

AXIS2_CLASSPATH="."
for i in `ls $AXIS2_HOME/lib/*.jar`; do
AXIS2_CLASSPATH=$AXIS2_CLASSPATH:$i
done

CLASSPATH=.:$JDK_HOME:$JAVA_HOME/lib/tools.jar:$ANT_HOME:
          $AXIS2_HOME:$AXIS2_BIN:$AXIS2_CLASSPATH

export JDK_HOME JAVA_HOME PATH CLASSPATH MAN_PATH AXIS2_HOME
       AXIS2_LIB AXIS2_BIN CATALINA_HOME
```

Listagem 2 Comando para a inicialização do Tomcat

```
root@tucunare:~# /home/<nome-usuario>/environment/apache-tomcat-6.0.18/bin/startup.sh

Using CATALINA_BASE:   /home/<nome-usuario>/environment/apache-tomcat-6.0.18
Using CATALINA_HOME:   /home/<nome-usuario>/environment/apache-tomcat-6.0.18
Using CATALINA_TMPDIR: /home/<nome-usuario>/environment/apache-tomcat-6.0.18/temp
Using JRE_HOME:        /home/<nome-usuario>/environment/jdk1.6.0_06
```

6.4 Teste das Configurações

Para o *Ant*, faça o teste a seguir:

```
root@tucunare:~# ant

Buildfile: build.xml does not exist!
Build failed
```

Para o Tomcat, inicie-o com o comando o comando da listagem 2. O resultado do comando é mostrado na mesma listagem.

Abra o navegador *Firefox* e digite:

```
http://localhost:8080
```

Se aparecer uma mensagem de boas-vindas é sinal de que o *Tomcat* foi configurado corretamente. Antes de iniciar o teste do Axis2, é preciso copiar o arquivo ***axis2-war*** para o diretório *webapps* do *Tomcat*. Proceda de acordo com as instruções a seguir:

```
cd /home/<nome-usuario>/environment/
cp axis2.war /home/<nome-usuario>/environment/apache-tomcat-6.0.18/webapps
```

Para confirmar o funcionamento do Axis2 abra o navegador *Firefox* e digite:

```
http://localhost:8080/axis2
```

Se aparecer uma mensagem de boas-vindas é sinal de que o *Axis2* foi configurado corretamente.

7 Exemplo de Aplicação com o Apache Axis2

Este tutorial tem como objetivo criar uma aplicação simples de web services utilizando o Axis2, a terceira geração da implementação de web services. Assume-se que todas as configurações foram efetuadas de acordo com a seção: *Instalação e Configuração do Ambiente de Programação com Axis2* e que ao executar o cliente do web service você também tenha o Axis2 instalado e configurado corretamente.

7.1 Primeira Etapa

A primeira etapa deste tutorial deve envolver primeiramente a criação de um serviço, na forma de uma classe java que será acessada por um cliente do serviço. Em seguida deve ser descrito como este serviço deve ser acessado e qual o tipo de transporte que cada operação do serviço deve suportar.

Criar o Serviço :Antes de criar o serviço que ficará disponível para o acesso de um cliente web, vamos criar a estrutura de diretórios que deve ser usada no momento de empacotar o serviços.

Crie um estrutura de diretórios de acordo com a listagem 3:

Listagem 3 Estrutura inicial do diretório meuservico

```
servicos
 |
  --> meuservico
       |
       --> META-INF
```

Crie um arquivo com o nome de MeuServico.java dentro do diretório (servicos) e adicione o código correpondente à listagem 4:

Descrever o arquivo services.xml :Depois de escrito o serviço, é preciso descrevê-lo para que o Axis2 saiba como manipulá-lo. Crie um arquivo denominado services.xml dentro do diretório META-INF (descrito anteriormente) e adicione o conteúdo abaixo:

No arquivo services.xml são descritos:

Listagem 4 Classe java correspondente ao serviço implementado

```
public class MeuServico {
       public String ecoa(String valor) {
              return valor;
       }
}
```

Listagem 5 Arquivos services.xml

```
<service>
       <parameter name="ServiceClass"
         locked="false">MeuServico</parameter>
       <operation name="ecoa">
              <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
       </operation>
</service>
```

- O nome do serviço;
- O nome da operação do serviço, que será exposta para acesso por um cliente do web services;
- O tipo de receptor de mensagem;

Compilar o Serviço :Entre no diretório servicos e compile o arquivo MeuServico.java.

```
javac MeuServico.java -d meuservico
```

Um arquivo denominado MeuServico.class deve ser gerado como produto da compilação no diretório meuservico.

7.2 Segunda Etapa

A segunda etapa envolve o empacotamento do serviço e o deploy em um repositório de serviços.

Empacotar o Serviço :Antes de empacotar o serviço, certifique-se de que a estrutura de diretórios seja como a mostrada a listagem 6:

Entre no diretório meuservico:

```
cd servicos/meuservico
```

Faça o empacotamento:

```
jar -cvf MeuServico.aar
```

Após o empacotamento a estrutura de diretório deve ficar como mostra a listagem 7:

Listagem 6 Estrutura do diretório servicos

```
<service>
servicos
 |
 |--> MeuServico.java
 |
 ---> meuservico
       |
       |--> MeuServico.class
       |
       ---> META-INF
             |
              --> services.xml
```

Listagem 7 Estrutura do diretório servicos atualizada

```
servicos
 |
 |--> MeuServico.java
 |
 ---> meuservico
       |
       |--> MeuServico.class
       |
       |--> MeuServico.aar
       |
       ---> META-INF
             |
              --> services.xml
```

Fazer o Deploy (Implantação) em um Servidor de Aplicação ou Container de Servlets : Neste momento é preciso copiar o pacote MeuServico.aar para o diretório services do Tomcat. Proceda de acordo com os passos abaixo:

1. cd servicos/meuservico
2. cp MeuServico.aar /usr/local/apache-tomcat-6.0.14/webapps/axis2/WEB-INF/services/

Verificar o Resultado do Deploy :Abra o browser e digite:

`http://localhost:8080/axis2/services/listServices`

Você deve obter como resultado o nome do serviço que acabou de ser implantado. A WSDL do serviço é descrita na figura 6:

7.3 Terceira Etapa

A terceira etapa envolve a criação dos stubs do serviços a partir da WSDL gerada anteriormente e também um cliente que deve fazer o acesso ao serviço através do stubs.

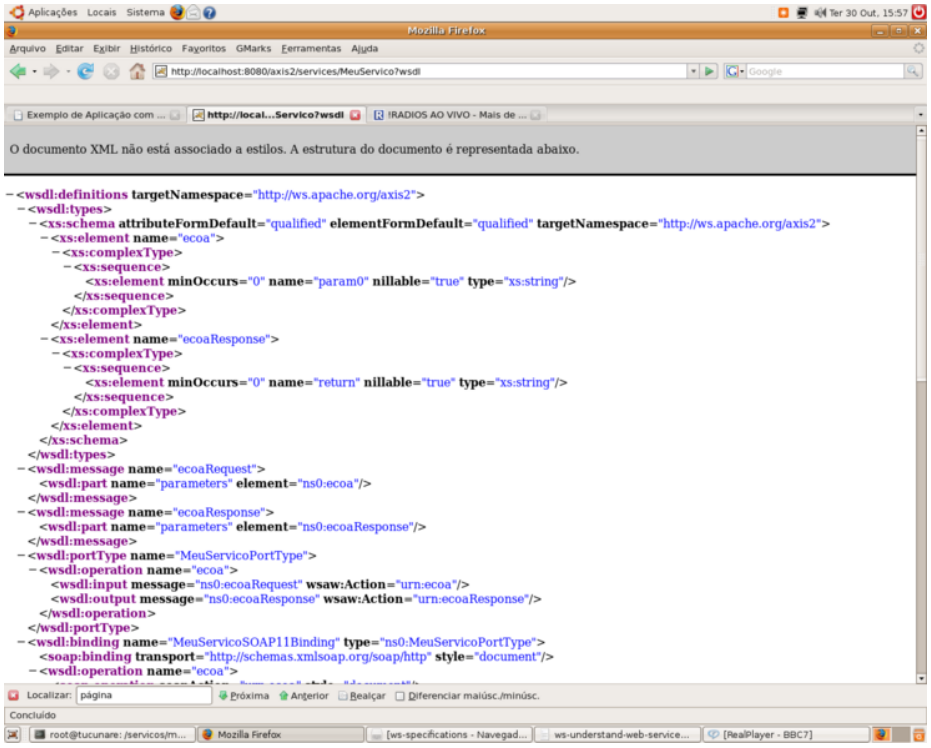

Figura 6. WSDL do serviço

Gerar os Stubs do Serviço A geração dos stubs deve ser feita de acordo com o comando abaixo:

1. cd servicos/meuservico
2. /usr/local/axis2/bin/wsdl2java.sh -uri http://localhost:8080/axis2/services/MeuServico?wsdl -o cliente

onde:

- -uri <caminho> identifica o caminho para a WSDL do serviço;
- -o <diretorio> identifica o diretório onde será armazenado os stubs do cliente.

A partir deste momento a estrutura do diretório meuservico deve ficar como mostra a listagem 8

Foram gerados os seguintes diretórios e arquivos:

- cliente
- src
- build.xml

É a partir dos stubs gerados que o cliente do serviço pode fazer a ligação como serviço.

Listagem 8 Estrutura do diretório meuservico

```
servicos
 |
 |--> MeuServico.java
 |
 ---> meuservico
       |
       |--> MeuServico.class
       |--> MeuServico.aar
       |
       |--> cliente
       |      |---> build.xml
       |      |
       |      |---> src
       |             |---> org
       |                    |---> apache
       |                           |---> ws
       |                                  |---> axis2
       |                                           |---> MeuServicoCallbackHandler.java
       |                                           |---> MeuServicoStub.java
       |
       |
       ---> META-INF
             |
              --> services.xml
```

Criar o Cliente do Serviço :Com os stubs criados, agora é o momento de escrever uma aplicação cliente que acessa o serviço implementado. Assim, crie um arquivo com o nome de Cliente.java e acrescente o código da listagem 10:

Listagem 9 Cliente.java - Código correspondente ao cliente do serviço

```
package org.apache.ws.axis2;

import org.apache.ws.axis2.MeuServicoStub.EcoaResponse;

public class Cliente {

   public static void main(String[] args) throws Exception {

       MeuServicoStub stub = new MeuServicoStub();

       //Cria a requisição para o serviço
       MeuServicoStub.Ecoa request;
       request = new MeuServicoStub.Ecoa();
       request.setParam0("Meu primeiro web service com Axis2");

       //Invoca o serviço
       EcoaResponse response;
       response = stub.ecoa(request);

       System.out.println("Resposta : " + response.get_return());
   }

}
```

Verifique como ficou a estrutura do diretório meuservico na listagem 10, depois da criação do arquivo Cliente.java [1]:

Listagem 10 Cliente.java - Código correspondente ao cliente do serviço

```
servicos
 |
 |--> MeuServico.java
 |
 ---> meuservico
       |
       |--> MeuServico.class
       |--> MeuServico.java
       |--> MeuServico.aar
       |
       |--> cliente
       |      |---> build.xml
       |      |
       |      |---> src
       |             |---> org
       |                    |---> apache
       |                           |---> ws
       |                                  |---> axis2
       |                                           |---> MeuServicoCallbackHandler.java
       |                                           |---> MeuServicoStub.java
       |                                           |---> Cliente.java
       |
       |
       ---> META-INF
             |
              --> services.xml
```

Compilar e Executar o Cliente do Serviço Na opção escolhida para este tutorial, a compilação do cliente deve ser feita sem o auxílio do ant. Compile o arquivo Cliente.java de acordo com os passos abaixo:

1. cd servicos/meuservico/cliente/src
2. javac org/apache/ws/axis2/Cliente.java -d ../../cliente

A estrutura de diretórios depois da geração do cliente pode ser verificada na listagem 11.

Finalmente chegamos ao final deste tutorial, onde agora é possível chamar o cliente do web service e visualizar o resutado. Primeiramente entre no diretório em que se encontra a classe compilada do cliente:

```
cd /servicos/meuservico/cliente
```

Depois, proceda com a execução do código do cliente, cujo resultado é mostrado a seguir:

```
java org.apache.ws.axis2.Cliente
```

[1] O local de criação do arquivo Cliente.java é uma decisão do programador. A decisão de criar no mesmo diretório contendo todos os arquivos .java, é tentar facilitar o entendimento do processo de criação de web services.

Listagem 11 Estrutura de diretórios depois da geração das classes do cliente do serviço

```
servicos
 |
 |--> MeuServico.java
 |
 ---> meuservico
       |
       |--> MeuServico.class
       |--> MeuServico.aar
       |
       |--> cliente
       |      |---> build.xml
       |      |
       |      |---> src
       |      |      |---> org
       |      |             |---> apache
       |      |                    |---> ws
       |      |                           |---> axis2
       |      |                                  |---> MeuServicoCallbackHandler.java
       |      |                                  |---> MeuServicoStub.java
       |      |                                  |---> Cliente.java
       |      |
       |      |---> org
       |             |---> apache
       |                    |---> ws
       |                           |---> axis2
       |                                  |---> Cliente.class
       |                                  |---> MeuServicoStub.class
       |                                  |---> MeuServicoStub$Ecoa$Factory.class
       |                                  |---> MeuServicoStub$EcoaResponse$Factory.class
       |                                  |---> MeuServicoCallbackHandler.class
       |                                  |---> MeuServicoStub$Ecoa$1.class
       |                                  |---> MeuServicoStub$EcoaResponse$1.class
       |                                  |---> MeuServicoStub$ExtensionMapper.class
       |                                  |---> MeuServicoStub$1.class
       |                                  |---> MeuServicoStub$Ecoa.class
       |                                  |---> MeuServicoStub$EcoaResponse.class
       |
       |
       ---> META-INF
             |
              --> services.xml
```

A resposta obtida depois da execução do cliente é mostrada abaixo:

```
Meu primeiro web service com Axis2
```

Referências

1. C. Brooks. A discussion of xml based middleware for web services, 2002. Disponivel em: http://www.cs.usask.ca/ cab938/papers/898finalpaper.doc - Ultimo acesso: Jan/2009.
2. D. Ehnebuske, D. Rogers, and C. V. Riegen. Uddi version 2.0 data structure reference, 2001. Disponivel em: http://www.uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm - Ultimo acesso: Jan/2009.
3. A. Erradi and P. Maheshwari. A broker-based approach for improving web services reliability. In *Proceedings of IEEE International Conference on Web Services (ICWS-05)*. IEEE CS Press, 2005.
4. A. S. Foundation. Apache axis2, 2009. Disponivel em: http://ws.apache.org/axis2 - Ultimo acesso: Jan/2009.
5. M. Gudgin, M. Hadley, J. Moreau, and H. Nielsen. Soap version 1.2 part 1: Adjuncts, 2007. Disponivel em: http://www.w3.org/TR/2007/REC-soap12-part2-20070427/ - Ultimo acesso: Jan/2009.
6. M. Gudgin, M. Hadley, J. Moreau, and H. Nielsen. Soap version 1.2 part 1: Messaging framework, 2007. Disponivel em: http://www.w3.org/TR/2007/REC-soap12-part1-20070427 - Ultimo acesso: Jan/2009.
7. K. Pani. Securing web services using soap extensions, 2004. Disponivel em: http://www.codeproject.com/KB/webservices/Securing_web_services.aspx - Ultimo acesso: Jan/2009.
8. M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE-03)*. IEEE CS Press, 2003.
9. W. School. Xml tutorial, 2009. Disponivel em: http://www.w3schools.com/xml/ - Ultimo acesso: Jan/2009.
10. J. P. Thomas, M. Thomas, and G. Ghinea. Modeling of web services flow. In *IEEE International Conference on E-Commerce, Newport Beach, California, USA, June 24 - 27, 2003*, pages 391–398, 2003.
11. P. Walmsley. *Definitive XML Schema*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
12. Wikipedia. Soap - simple object access protocol, 2009. Disponivel em: http://en.wikipedia.org/wiki/SOAP - Ultimo acesso: Jan/2009.
13. Wikipedia. Web services, 2009. Disponivel em: http://en.wikipedia.org/wiki/Web_service - Ultimo acesso: Jan/2009.
14. Wikipedia. Xml namespace, 2009. Disponivel em: http://en.wikipedia.org/wiki/XML_Namespace - Ultimo acesso: Jan/2009.