UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

UMA ARQUITETURA DISTRIBUÍDA BASEADA EM WEB SERVICES PARA GERÊNCIA E CONSULTA DE INFORMAÇÕES SOBRE TRANSPORTE COLETIVO MUNICIPAL

TRABALHO DE GRADUAÇÃO

Jesse Moura da Silva

Santa Maria, RS, Brasil

2009

UMA ARQUITETURA DISTRIBUÍDA BASEADA EM WEB SERVICES PARA GERÊNCIA E CONSULTA DE INFORMAÇÕES SOBRE TRANSPORTE COLETIVO MUNICIPAL

por

Jesse Moura da Silva

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Profª Andrea Schwertner Charão

Trabalho de Graduação N. 284

Santa Maria, RS, Brasil

2009

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

UMA ARQUITETURA DISTRIBUÍDA BASEADA EM WEB SERVICES PARA GERÊNCIA E CONSULTA DE INFORMAÇÕES SOBRE TRANSPORTE COLETIVO MUNICIPAL

elaborado por
Jesse Moura da Silva

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Profª Andrea Schwertner Charão
(Presidente/Orientador)

Profª Iara Augustin (UFSM)

Profª Marcia Pasin (UFSM)

Santa Maria, 14 de Julho de 2009.

AGRADECIMENTOS

Gostaria de agradecer à toda minha família, pelo grande apoio dado durante todos esses anos. À minha mãe pelas orientações, carinho e por estar sempre apoiando minhas decisões. Agradeço ao meu avô Gugu, Profº João Agostinho, e minha avó Juju, Profª Julieta, que sempre nos ensinaram que a educação é muito importante para nossas vidas.

Agradeço à minha namorada Paula pelo apoio e carinho nas horas boas e ruins.

Aos colegas, agradeço pelo compartilhamento de conhecimento durante o curso e pelas amizades construídas. Ao Henrique pela amizade, companheirismo e pelas horas descontraídas jogando ou batendo um papo.

Aos professores, muito obrigado pelos ensinamentos passados e pelo apoio.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

UMA ARQUITETURA DISTRIBUÍDA BASEADA EM WEB SERVICES PARA GERÊNCIA E CONSULTA DE INFORMAÇÕES SOBRE TRANSPORTE COLETIVO MUNICIPAL

Autor: Jesse Moura da Silva
Orientador: Profª Andrea Schwertner Charão
Local e data da defesa: Santa Maria, 14 de Julho de 2009.

A necessidade de prover informações na Internet tem impulsionado o desenvolvimento de tecnologias que facilitam a troca de dados entre aplicações. Com o intuito de realizar esta comunicação entre aplicativos com diferentes características surgem os Web Services. Neste contexto de provimento de informações, encontramos o Sistema de Transporte Coletivo Urbano Municipal, o qual é um serviço público e necessita de publicação de algumas informações. Estas informações têm o intuito de facilitar a utilização do serviço e são fornecidas pelos responsáveis do serviço, porém, normalmente, de forma não padronizada. Algumas vezes, aplicativos clientes necessitam destas informações para realizar alguma computação e fornecê-las aos usuários, mas a maneira como elas são armazenadas pode tornar difícil o acesso e a manipulação. Desta forma, este trabalho se propõe a construir uma ferramenta que mantenha as informações e forneça acesso aos aplicativos clientes. A criação de um Web Service pode ser uma solução, uma vez que ele pode disponibilizar métodos para gerência e consulta das informações a outros aplicativos. Além disso, pelo fato de o serviço ser bastante utilizado, previu-se que haveria uma demanda grande de acessos e que a tolerância a falhas era inexistente, no caso de um único provedor de serviço. Logo, uma arquitetura distribuída, que dispusesse de técnicas de replicação de dados e dinamicidade na descoberta de provedores de serviço, poderia ampliar o fornecimento dessas informações e oferecer tolerância a falhas.

Palavras-chave: Web Service; Sistemas Distribuídos; Transporte coletivo urbano municipal.

ABSTRACT

Trabalho de Graduação
Undergraduate Program in Computer Science
Universidade Federal de Santa Maria

A DISTRIBUTED ARCHITECTURE BASED ON WEB SERVICES FOR INFORMATION MANAGEMENT AND QUERY ON URBAN PUBLIC TRANSPORT

Author: Jesse Moura da Silva
Advisor: Prof[a] Andrea Schwertner Charão

The need to provide information on the Internet has driven the development of technologies that allow data exchange between applications. In order to perform the communication between applications with different characteristics, Web Services are used. In the context of information provision, there is the urban public transport system, which is a public service and requires publication of information. This information has the aim of facilitating the use of the service and are provided by the responsible department, however, usually, but not standardized. Sometimes, applications customers need this information to perform a computation and provide to users, but the way they are stored can become difficult to access and manipulation. Thus, this work focuses to build a tool to keep information and provide access to client applications. Creating a Web Service can be a solution, since it can provide methods for information management and consultation to other applications. Moreover, because the service is widely used, provided that there would be a great demand for access and that the fault tolerance was absent in the case of a single service provider. Therefore, a distributed architecture, which provided a data replication technique and a discovery dynamic of services, could expand the supply of such information and provide for fault tolerance.

Keywords: Web Services, Distributed Systems.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CORBA	Common Object Request Broker Architecture
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
JAX-WS	Java API for XML Web Services
JVM	Java Virtual Machine
ORB	Object Request Broker
RMI	Remote Method Invocation
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SMTP	Simple Mail Transfer Protocol
W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
WSDL	Web Services Description Language
WS	Web Service
WS-I	Web Services Interoperability Organization
UDDI	Universal Description, Discovery, and Integration
NTP	Network Time Protocol
XML	Extensible Markup Language
XSD	XML Schema Definition

SUMÁRIO

1 CONTEXTUALIZAÇÃO

A necessidade de troca de informações em rede e a interoperabilidade entre sistemas impulsionou o desenvolvimento de diversas tecnologias na área de sistemas distribuídos. Nesta linha, as tecnologias baseadas em serviços Web (Web Services) representaram um grande avanço na realização da interação entre aplicações distintas, devido à simplicidade de implementação e utilização de tecnologias e protocolos abertos.

Um Serviço Web pode ser entendido como um sistema de *software* projetado para suportar a interoperabilidade entre máquinas através de uma rede (David Booth et al., 2004). Utilizando essa tecnologia, as aplicações interagem de forma padronizada, enviando e recebendo dados em um formato pré-estabelecido, mesmo que sejam desenvolvidas em linguagens de programação diferentes.

Este trabalho propõe-se a explorar o uso de serviços Web para a interação entre sistemas de informações no contexto do sistema de transporte coletivo urbano. O objetivo é fornecer uma ferramenta aos desenvolvedores de aplicativos que consultem e gerenciem as informações. Como o transporte coletivo é um serviço público, tornou-se comum algumas empresas prestadoras do serviço publicar as informações relativas a linhas e horários de ônibus. Pela popularidade da Internet, surgiu a alternativa de criar sites na Web para que os usuários realizem as consultas dos horários de ônibus e outras informações relacionadas. Dessa forma, a publicação das informações facilita aos usuários a utilização do serviço. Contudo, apesar das informações estarem chegando aos usuários finais, normalmente cada empresa ou associação responsável pela divulgação das informações a realiza da maneira que lhe convém, de forma centralizada e sem seguir um padrão para publicação destas informações. Como conseqüência disso, torna-se difícil, por exemplo, o aproveitamento dos dados por outro sistema que queira consultar ou realizar operações com estes dados.

Algumas cidades brasileiras possuem um sistema único, na Internet, para prestar informações relativas ao sistema de transporte coletivo municipal. Um exemplo é a cidade de Curitiba, PR, que publica informações de todas as linhas de ônibus do transporte coletivo através de um site na Web (URBS, 2009). Porém, não é o caso, por exemplo, da cidade de Santa Maria, RS, a qual possui somente algumas empresas que publicam seus horários através de sites Web particulares, enquanto outras empresas as publicam por outros meios de comunicação, mas de formas distintas e não padronizadas.

Neste cenário, um serviço Web pode facilitar a gerência e consulta destas informações, oferecendo uma interface para sistemas que as produzem e consomem. Existindo um sistema que provê essas informações de uma forma padronizada, uma aplicação que queira utilizar esse serviço, ou seja, implementar um cliente deste serviço Web, poderá fazê-lo sem a preocupação de ter que manter e atualizar esses dados. Além disso, aplicações clientes deste serviço podem fornecer informações de diversas maneiras aos usuários finais, assim como podem existir uma variedade de mídias que os usuários podem utilizar para receber as informações. A disponibilidade e a autenticidade dos dados são garantidas pelas próprias empresas prestadoras do serviço público, que geralmente são as responsáveis pela gerência dessas informações, já que elas têm acesso restrito às operações que gerenciam os dados armazenados no serviço.

Uma aplicação que motivou o desenvolvimento deste trabalho pode ser vista na figura 1.1. Nesta figura, existe uma aplicação chamada de *próximos ônibus*, desenvolvida por acadêmicos do curso de Ciência da Computação dessa instituição, que tem como objetivo fornecer o tempo para um próximo ônibus sair de um terminal de origem. Esta aplicação pode ser considerada como um cliente em potencial para a ferramenta que pretende-se desenvolver nesse trabalho, uma vez que consome informações de um sistema de transporte coletivo. Para fornecer as informações, esta aplicação necessita buscar dados armazenados em arquivos para então realizar a computação do tempo. Com a criação da ferramenta proposta nesse trabalho, esta aplicação poderia consumir e oferecer as informações de forma mais simples.

Como em todo sistema implantado em uma rede de computadores, existem riscos de o sistema tornar-se indisponível, devido às características estruturais de uma rede. No contexto de um serviço Web, pode-se notar esse problema quando o serviço Web implantado tem sua execução interrompida ou a máquina onde ele se encontra apresenta falhas.

Figura 1.1: Página inicial do curso de Ciência da Computação da UFSM

Como conseqüência, um cliente deste serviço deixa de consumir os dados até então oferecidos. Para abordar esse problema, o trabalho foi direcionado a uma solução que evitasse que o cliente fosse dependente apenas de um serviço Web centralizado, fornecedor de informações. Para isso, foram adicionadas algumas entidades na arquitetura do sistema que auxiliam o cliente a consumir o serviço de forma a tolerar possíveis falhas, além da possibilidade dos serviços possuirem réplicas, o que aumenta a disponibilidade das informações.

Como objetivo na criação do serviço, buscou-se uma modelagem genérica dos dados sobre transporte coletivo municipal, a qual pudesse ser implantada em qualquer cidade. Como parâmetros para criação do modelo do sistema, foram estudadas algumas soluções, encontradas em sites na Web pertencentes a empresas ou prefeituras de algumas capitais do país, que publicam aquelas informações.

Além disso, para validação do serviço, tem-se como outro objetivo deste trabalho a criação de um cliente para o serviço Web. Como os serviços Web caracterizam-se por oferecer interoperabilidade entre aplicações heterogêneas, buscou-se reforçar este argumento através da criação de uma aplicação cliente utilizando uma linguagem de programação diferente da utilizada na implementação do serviço.

O trabalho está estruturado em seis capítulos. No segundo capítulo, apresenta-se uma fundamentação teórica ao realizar uma revisão de conceitos importantes para o contexto deste trabalho. A seguir, no terceiro capítulo, são apresentados os requisitos para criação do sistema, além da arquitetura, da modelagem e da implementação do sistema. No quarto capítulo, é apresentada a aplicação da arquitetura, a qual é dividida na análise da implantação do sistema e na implementação de um cliente do serviço Web desenvolvido, abordando as tecnologias envolvidas. No capítulo cinco, é realizada uma análise das próximas atividades a serem desenvolvidas. No sexto capítulo, as considerações finais do trabalho são apresentadas e ao final são descritas as referências bibliográficas.

1.1 Web Services

Uma necessidade muito comum na área de computação é a integração de sistemas que possuem características diferentes como linguagem de programação e plataforma de desenvolvimento. Um exemplo é a necessidade de provimento de serviços na Internet, que se caracteriza por uma empresa ou entidade fornecer informações através de soluções em *software*, as quais podem ser consumidas por aplicações clientes.

Nesse contexto, surgiu o conceito de Web Service no ano 2000, introduzido pela Microsoft Corporation. Web Services referem-se a uma coleção de padrões que provem interoperabilidade e podem ser definidos como uma tecnologia em sistemas distribuídos que tem o intuito de solucionar o problema da comunicação entre aplicações diferentes (Nicloai M. Josuttis, 2007). Ela não é a primeira solução criada para fornecer interoperabilidade entre sistemas através de uma rede de computadores, uma vez que CORBA e RMI são exemplos de outras tecnologias criadas com este objetivo. Porém, devido a algumas limitações das duas tecnologias, os Web Services têm se mantido como melhor opção para prover interoperabilidade, visto que possibilita comunicação entre aplicações que possuem diferente sistema operacional e linguagem de programação.

CORBA é uma arquitetura para comunicação muito utilizada em implementação de sistemas distribuídos heterogêneos, porém possui a desvantagem de, quando usada na Internet, necessitar algumas alterações específicas em firewalls, as quais nem sempre são possíveis. Outra desvantagem é a necessidade de implementação por parte de empresas, de uma camada de *software*, chamada de ORB (Object Request Broker), necessária para implementação de aplicativos. Devido ao fato de diferentes empresas oferecem imple-

mentações diferentes de ORB, algumas vezes torna-se difícil a portabilidade entre plataformas de programação distintas. Já a RMI é uma arquitetura de objetos distribuídos suportada pela Máquina Virtual Java, através de um framework. Embora o Java RMI permita interoperabilidade entre aplicações em ambientes distintos utilizando a JVM, esta dependência da Máquina Virtual Java torna-se uma desvantagem, pois a linguagem de programação utilizada nas aplicações restringi-se a Java. Portanto, o surgimento dos Web Services vem para resolver algumas limitações de outras tecnologias e fornecer uma maior possibilidade de interação entre plataformas de hardware e sistemas operacionais diferentes.

Os Web Services têm sido muito estudados e várias soluções são desenvolvidas, sendo definidos padrões pela World Wide Web Consortium (W3C), OASIS e Web Services Interoperability Organization (WS-I). Os Web Services utilizam protocolos e tecnologias abertas de comunicação e transporte, como HTTP (W3C, 2009b), XML (W3C, 2008a) e SOAP. Dessa forma, essa padronização tem sido a chave para o sucesso dessa tecnologia (Nicloai M. Josuttis, 2007). Além disso, outras tecnologias, assim como CORBA e RMI são consideradas de maior complexidade e mais alto custo do que os Web Services. Segundo (Martin Kalin, 2009), três características distinguem os Web Services de outras tecnologias de *software* de sistemas distribuídos, que são:

- Infraestrutura aberta - Utiliza protocolos abertos de comunicação e transporte.
- Transparência de linguagem - O aplicativo cliente pode ser implementado em uma linguagem de programação diferente da utilizada para implementar o serviço.
- Estrutura modular - Uma arquitetura de Web Services pode facilmente ser ampliada e novas entidades serem adicionadas na arquitetura.

Na arquitetura básica de Web Services pode-se identificar três entidades:

- Provedor de serviços: é o criador do serviço e disponibilizador das informações.
- Registro de serviços: trata-se de um local onde o provedor de serviços publica o Web Service.
- Consumidor: entidade que consome as informações do provedor de serviços.

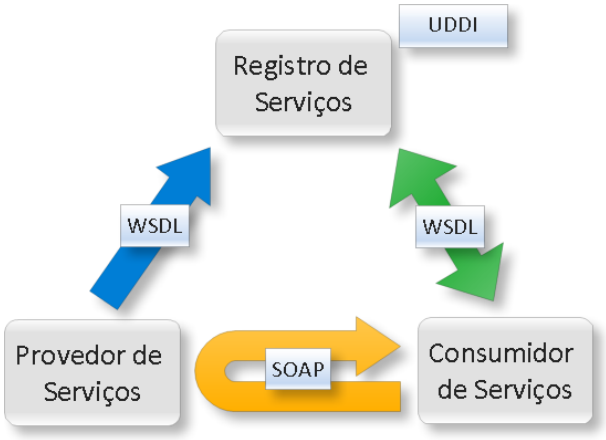

Figura 1.2: Arquitetura básica de Web Services (Nicloai M. Josuttis, 2007)

Analisando a arquitetura do Web Service, figura 1.2, pode-se definí-lo como um *software* que permite trocar informações, o qual é descrito em WSDL, registrado em UDDI (OASIS, 2009) e acessado através de SOAP (Martin Kalin, 2009). A arquitetura de Web Services não tem como objetivo especificar como os Web Services são implementados, nem impor restrições de como eles podem ser combinados, mas tem como intuito identificar os elementos necessários para garantir a interoperabilidade (David Booth et al., 2004).

1.1.1 WSDL

É uma linguagem baseada em XML utilizada para descrever o serviço, especificar o acesso e as operações disponíveis no Web Service (W3C, 2008b). Esta descrição do serviço funciona como um documento, ou seja, um contrato entre o provedor do serviço e o consumidor. Quando se procura um Web Service, o WSDL é a informação que se obtém do serviço e é através dele que se pode implementar o consumidor.

A figura 1.3 apresenta um documento WSDL que descreve um serviço. Neste documento é descrito um serviço, o qual possui um método que fornece uma lista de elementos do tipo "estado". As seções do arquivo WSDL são definidas a seguir, de acordo com (Martin Kalin, 2009):

1. Seção **types** – descreve os tipos de dados do serviço.

2. Seção **message** – define as mensagens de requisição e resposta de cada método do Web Service.

3. Seção **portType** – define um Web Service, as operações que podem ser executadas

e as mensagens utilizadas para executar as operações.

4. Seção **binding** – especifica um *portType* e define as operações. A seção *soap:binding* possui dois atributos importantes: *transport* e *style*. O atributo *transport* especifica o protoloco de transporte para a mensagem SOAP, o qual pode ser, por exemplo, HTTP ou SMTP. Já o atributo *style*, que é o estilo de vinculação do WSDL com a mensagem SOAP, pode ter os valores *rpc* ou *document* e ele define como os tipos de dados são utilizados no arquivo WSDL. Se o atributo tiver o valor *rpc* os tipos de dados são declarados nos elementos *part* da seção *message*, porém se o valor for *document* os tipos de dados são especificados em um documento XSD. Um exemplo de documento XSD pode ser visto na figura 1.4. Na seção *binding*, existe ainda o elemento *operation* que define uma operação do serviço, o qual possui o elemento *input*, que representa a operação de entrada no serviço e o elemento *output* que representa a operação de saída. Nestes elementos existe um elemento filho chamado *soap:body* que possui o atributo *use*. Este atributo pode ter o valor *encoded* ou *literal*. O valor *encoded* especifica que a mensagem SOAP deve incluir os tipos dos dados como atributos dos elementos *part* da seção *message*. Já o valor *literal* não necessita a inclusão daqueles atributos, pois a verificação dos tipos de dados da mensagem SOAP é feita utilizando um XML Schema. Neste trabalho, como pode ser visto no código apresentado anteriormente, o estilo de vinculação utilizado é o *document* e o atributo *use* é o *literal*. Esta opção foi escolhida, pois de acordo com o autor, é a opção que apresenta menos desvantagens que as outras combinações possíveis e é a mais utilizada pelos desenvolvedores.

5. Seção **service** – especifica um ou mais pontos de extremidade. Cada seção *port* especifica um endereço de serviço.

1.1.2 SOAP

É um protocolo de comunicação baseado em XML que define como devem ser estruturadas as mensagens para troca de informações entre o requerente e o provedor do serviço (W3C, 2009a). É padronizado e mantido pelo consórcio W3C.

SOAP foi o primeiro padrão desenvolvido para Web Services. Atualmente, possui a versão 1.1 e 1.2, que se diferenciam no nível dos adaptadores do SOAP e não são visíveis

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://servicos/" name="WebusConsultaService">
  <types>
    <xsd:schema xmlns:tns="http://servicos/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
          version="1.0" targetNamespace="http://servicos/">
1     <xsd:import namespace="http://servicos/"
            schemaLocation="http://localhost:8080/Webus/WebusConsultaService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="getListaEstados">
2   <part name="parameters" element="tns:getListaEstados"/>
  </message>
  <message name="getListaEstadosResponse">
    <part name="parameters" element="tns:getListaEstadosResponse"/>
  </message>
  <portType name="WebusConsulta">
    <operation name="getListaEstados">
3     <input message="tns:getListaEstados"/>
      <output message="tns:getListaEstadosResponse"/>
    </operation>
  </portType>
  <binding name="WebusConsultaPortBinding" type="tns:WebusConsulta">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getListaEstados">
      <soap:operation soapAction=""/>
      <input>
4       <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="WebusConsultaService">
    <port name="WebusConsultaPort" binding="tns:WebusConsultaPortBinding">
5     <soap:address location="http://localhost:8080/Webus/WebusConsultaService"/>
    </port>
  </service>
</definitions>
```

Figura 1.3: Exemplo de documento WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:tns="http://servicos/" targetNamespace="http://servicos/">
   <xs:element name="estado" type="tns:estado"/>
   <xs:element name="getListaEstados" type="tns:getListaEstados"/>
   <xs:element name="getListaEstadosResponse" type="tns:getListaEstadosResponse"/>
   <xs:complexType name="estado">
      <xs:sequence>
         <xs:element name="date" type="xs:dateTime" minOccurs="0"/>
         <xs:element name="estado_id" type="xs:int"/>
         <xs:element name="nome" type="xs:string" minOccurs="0"/>
         <xs:element name="oid" type="xs:int"/>
         <xs:element name="sigla" type="xs:string" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="getListaEstados">
      <xs:sequence/>
   </xs:complexType>
   <xs:complexType name="getListaEstadosResponse">
      <xs:sequence>
         <xs:element name="return" type="tns:estado" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xsd:schema>
```

Figura 1.4: Documento que descreve um XML Schema.

ao programador. Este protocolo utiliza, normalmente, o protocolo de transporte HTTP, mas pode utilizar também SMTP e FTP.

A mensagem SOAP, figura 1.6, é um documento XML dividido em três partes: um envelope obrigatório, um cabeçalho que é opcional e um corpo obrigatório. O envelope é o elemento raiz do documento e nele são descritas informações como: namespaces e atributos adicionais que definem o modo de codificação. O cabeçalho (Header) tem como

objetivo armazenar informações adicionais, como por exemplo, nome de usuário e senha. Já o corpo (Body) é o elemento obrigatório que contém a informação a ser transportada. O corpo pode conter um elemento extra chamado de Fault, o qual tem como objetivo transportar uma mensagem de erro ou estado. Abaixo, na figura 1.5, é apresentado um exemplo de uma mensagem SOAP de requisição enviada ao serviço e outra de resposta a requisição, respectivamente.

```
<?xml version="1.0" encoding="UTF-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   <s:Header/>
   <s:Body xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <getListaEstados xmlns="http://servicos/"/>
   </s:Body>
</s:Envelope>


<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getListaEstadosResponse xmlns:ns2="http://servicos/">
         <return>
            <date>2009-04-20T00:00:00-03:00</date>
            <estado_id>1</estado_id>
            <nome>Rio Grande do Sul</nome>
            <oid>123</oid>
            <sigla>RS</sigla>
         </return>
         <return>
            <date>2009-04-20T00:00:00-03:00</date>
            <estado_id>2</estado_id>
            <nome>Santa Catarina</nome>
            <oid>456</oid>
            <sigla>SC</sigla>
         </return>
      </ns2:getListaEstadosResponse>
   </S:Body>
</S:Envelope>
```

Figura 1.5: Exemplo de mensagem SOAP de requisição e resposta.

1.2 Trabalhos relacionados

No intuito de oferecer serviços públicos, assim como o serviço de transporte coletivo urbano, e levar as informações para um maior número de usuários, as administrações têm buscado publicar informações em sites na Web. Fornecer horários de ônibus e seus itinerários, no caso do sistema de transporte coletivo, facilita aos usuários a utilização do serviço. Com esse intuito, algumas empresas prestadoras do serviço, associações e administrações públicas têm realizado projetos para construção de sites na Web para publicação das informações.

A prefeitura de São Paulo, SP, que possui um sistema chamado SPTrans (SPTrans, 2009), fornece uma solução bem completa com informações de todas as linhas da capital,

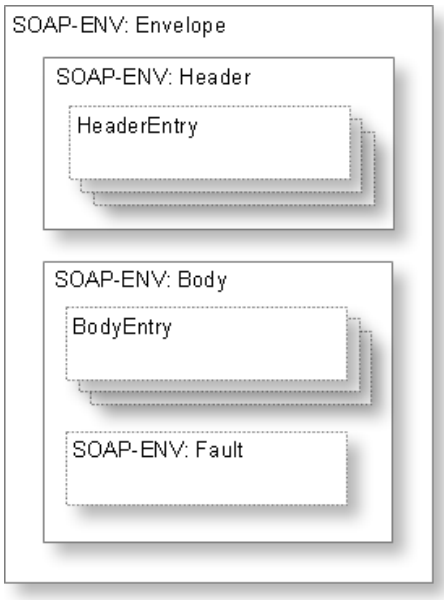

Figura 1.6: Estrutura da mensagem SOAP (W3C, 2009a)

com a possibilidade de consultar roteiro de viagens com percurso e tempo para realizar o deslocamento. A principal vantagem deste sistema é a variedade de consultas que podem ser feitas para descobrir como chegar de um local de origem até um local de destino. Existem várias combinações de consultas com vários parâmetros que podem ser fornecidos.

Outra capital que possui sistema semelhante é Curitiba, no Paraná. Naquela cidade, existe um portal chamado URBS (URBS, 2009), que publica as informações com menos possibilidades de consultas que o sistema SPTrans, mas com as informações básicas, como horários e itinerários das linhas de ônibus da capital.

Além destas duas, a capital do estado de Santa Catarina, Florianópolis, possui um site na Web, da prefeitura municipal, que fornece informações básicas. Porém, vale destacar uma solução encontrada por uma das empresas daquela cidade, chamada de Transol (TRANSOL, 2009). Esta empresa possui um site que disponibiliza as informações relativas às linhas servidas por ela e tem como uma vantagem a presença de um mapa que fornece uma visualização do itinerário das linhas. Além disso, esta solução tem a funcionalidade chamada "Conexão ideal", que é uma consulta que mostra as opções de deslocar-se de um ponto a outro fornecendo uma comparação entre as opções.

Apesar das soluções citadas possuírem vantagens e atingirem o objetivo principal que é fornecer as informações do sistema de transporte coletivo urbano de um município, existe uma limitação referente ao uso das informações. Esta limitação consiste no fato de que é custoso reutilizá-las em outras aplicações, além delas serem publicadas de forma

não padronizada. Seria útil, por exemplo, para um site de uma universidade fornecer os horários de ônibus para os seus usuários, porém na abordagem citada é muito complicado consultar e manter as informações consistentes de forma automática, as quais são publicadas e mantidas pelos responsáveis pelo serviço.

Uma solução que apresenta algumas características semelhantes ao trabalho proposto é encontrada na prefeitura de Uberlândia, MG. O projeto GEOSIT (GEOSIT, 2009), que é mantido pela Secretaria Municipal de Trânsito e Transporte daquele município, tem como objetivo monitorar os veículos que realizam o serviço de transporte público e fornecer informações, como posicionamento do veículo e horários do transporte, aos seus usuários. Como *software* para a implantação do projeto GEOSIT, foi adotado o *software* Arena Control Center de autoria da Maxtrack Industrial (Maxtrack, 2009), o qual utiliza a tecnologia de Web Services. O projeto GEOSIT possui um cliente deste Web Service disponível na Internet, onde se pode consultar os horários de ônibus e visualizar a localização do veículo em tempo real. Portanto, uma vantagem da implantação de um projeto semelhante a este é que as informações podem ser publicadas, por aplicações clientes, de muitas maneiras. Um exemplo, assim como utilizados naquele município, são painéis eletrônicos localizados em terminais de ônibus que mostram os horários dos ônibus e posicionamento em tempo real dos veículos.

2 CONTRIBUIÇÃO

2.1 Sistema Webus

O objetivo deste capítulo é mostrar a arquitetura desenvolvida, a modelagem e a implementação das entidades do sistema Webus.

2.1.1 Requisitos

A prestação de um serviço público, assim como o serviço de transporte coletivo que é amplamente utilizado, deve ter alta disponibilidade. Neste sentido, a construção de um sistema que atenda uma grande quantidade de clientes deve ser formulado de forma que garanta escalabilidade, eficiência e tolerância a falhas.

A criação de um Web Service que ofereça, aos aplicativos clientes, gerência e consulta das informações do tranporte coletivo municipal, necessita estar disponível na maior parte do tempo e ser eficiente no atendimento às requisições dos clientes. Além disso, a existência de procedimentos que garantam tolerância a falhas é muito importante para o funcionamento do sistema e consequente disponibilização das informações.

Para atingir esses objetivos foi proposta a criação de uma arquitetura que oferecesse duas categorias de serviços. Uma categoria de serviços funcionais que contém as funcionalidades que manipulam as informações do sistema de transporte coletivo, ou seja, as funcionalidades da aplicação. Uma segunda categoria, de serviços não funcionais, possui funcionalidades auxiliares com o intuito de garantir tolerância a falhas e eficiência ao sistema.

Com o intuito de aumentar a disponibilidade do serviço foi utilizada a técnica de replicação de dados. Esta técnica consiste em manter cópias dos dados e tem como objetivo fornecê-los em diferentes locais.

Ao aumentar a disponibilidade através da técnica de replicação de dados, uma nova

necessidade surgiu. A descoberta de endereços de serviços disponíveis tornou-se necessária, pois a técnica de replicação aumenta as opções de acessar a informação, ou seja, oferece mais de um endereço de serviço. Portanto, uma funcionalidade auxiliar no sistema, oferecida às aplicações clientes, é a descoberta dinâmica de endereços de serviços disponíveis.

2.1.2 Arquitetura do sistema

A partir da análise dos requisitos, estudou-se a construção de uma arquitetura distribuída baseada em Web Services, como pode ser vista na figura 2.1.

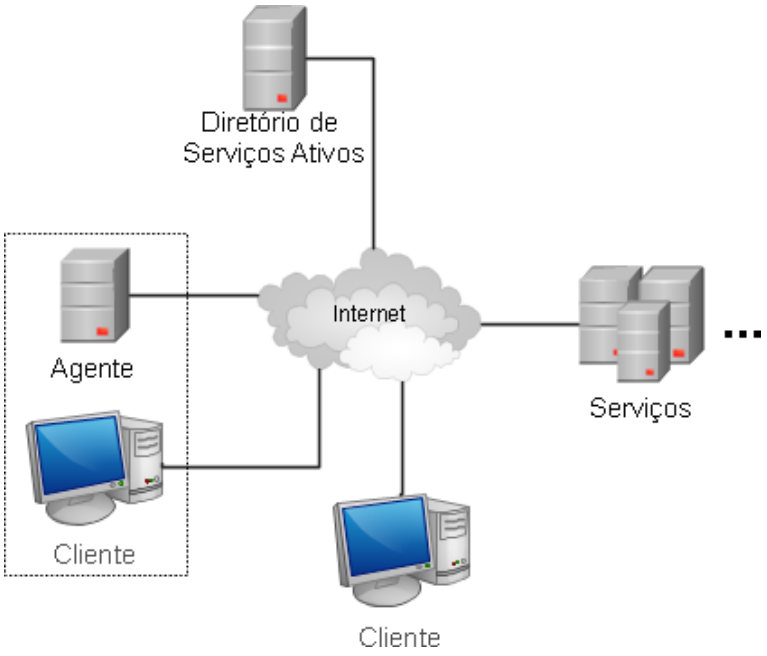

Figura 2.1: Arquitetura do sistema Webus em alto nível

Para solucionar o problema de disponibilidade e eficiência do sistema, desenvolveu-se a idéia de criar réplicas do serviço e da base de dados, o que gera mais opções para os clientes realizarem as suas consultas. Com a inclusão de réplicas do serviço criou-se a vantagem de os métodos oferecidos pelo serviço serem disponibilizados em vários lugares. Neste contexto, são conhecidos alguns desafios como, por exemplo, manter a coerência dos dados em todas as réplicas. Para abordar este problema foram criadas novas unidades no sistema para garantir a integridade das operações realizadas.

Além disso, verificou-se a necessidade de fornecer ao cliente uma interface que disponibilizasse endereços de serviços disponíveis com o intuito de oferecer tolerância a falhas. Esta necessidade resultou na criação de uma unidade no sistema chamada de

Agente. Como consequência foi criada uma unidade na arquitetura, chamada de diretório de serviços ativos (DSA), com a função de manter os endereços de serviços, os quais pudessem ser consultados pelo Agente ou pelo aplicativo cliente.

Nesta abordagem, onde várias unidades do sistema interagem, sendo todas elas implementadas com a tecnologia de Web Services, pode-se classificar a arquitetura do sistema como uma arquitetura orientada a serviços (SOA) (Nicloai M. Josuttis, 2007). SOA pode ser entendido como um conjunto de serviços que comunicam-se entre si, sendo que os serviços podem ser vistos como unidades lógicas que encapsuladas formam um grande serviço (Binildas CA; Malhar Barai; Vincenzo Caselli, 2008). Na figura 2.2, pode ser vista a arquitetura do sistema e suas unidades, as quais são citadas a seguir:

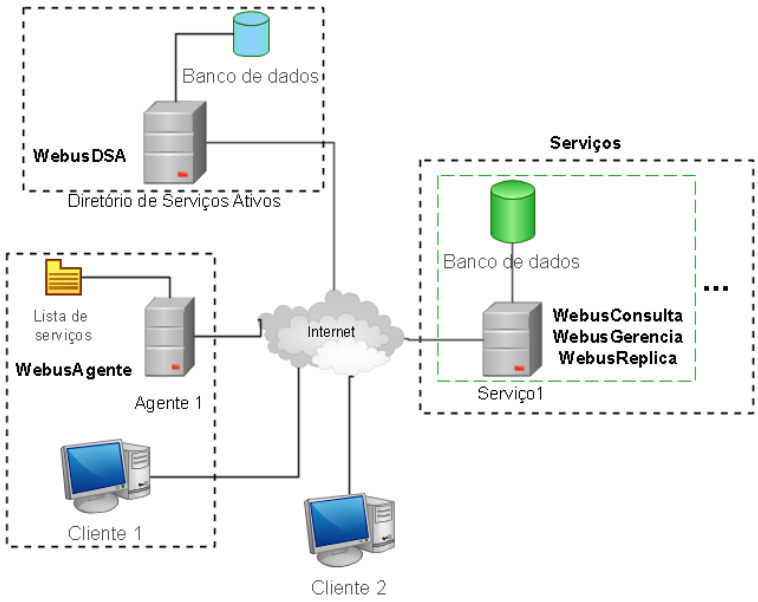

Figura 2.2: Arquitetura do sistema Webus com suas unidades

- **WebusConsulta** – fornece os métodos para consultar as informações do sistema. Esta unidade insere-se na categoria de serviços funcionais.

- **WebusGerencia** – fornece os métodos para gerenciar as informações. Estes métodos necessitam autenticação do cliente para realizar as operações. Pertence à categoria de serviços funcionais.

- **WebusReplica** – Esta unidade é responsável pela replicação dos dados que são mantidos e manipulados na unidade WebusGerencia. Esta unidade fornece funções auxiliares ao sistema, portanto pertence à categoria de serviços não funcionais.

- **WebusDSA** – tem como objetivo manter endereços de serviços ativos para que as outras unidades do sistema descubram estes serviços. Esta unidade insere-se na categoria de serviços não funcionais.

- **WebusAgente** – Localizada próxima ao aplicativo cliente, oferece métodos para fornecer alguns endereços de serviços ativos, os quais são descobertos consultando o WebusDSA. Por fornecer apenas funcionalidades auxiliares na arquitetura, pertence à categoria de serviços não funcionais.

2.1.3 Modelagem do sistema

A modelagem do sistema foi baseada em uma solução que abordasse características dos sistemas distribuídos, as quais fornecessem melhor desempenho e tolerância a falhas ao sistema. A adição da técnica de replicação dos serviços e a dinamicidade na descoberta de endereços de serviços, oferecida ao cliente, trazem benefícios para o funcionamento do sistema, uma vez que oferecem tolerância a falhas.

O sistema de transporte coletivo urbano é um serviço público municipal e a prefeitura do município é responsável pelo controle e gerência do serviço. Este serviço público pode ser realizado por empresas particulares que tem autorização de prestar o serviço em regime de contrato. Haja vista que a prefeitura é responsável pela criação das linhas de ônibus e itinerários, por exemplo, criou-se o papel de administrador de município no sistema. Este administrador pode ser representado por alguém da prefeitura ou pela associação das empresas transportadoras e ele fica responsável pela manipulação de algumas classes do sistema. Foi necessária a criação deste papel no sistema devido ao fato de algumas informações serem compartilhadas entre as empresas que realizam o serviço no município. Além deste papel, existe o administrador da empresa, o qual possui permissões de manipulação restritas à algumas classes do sistema. Logo, os papéis de administração são dois:

- Administrador de município.

- Administrador de empresa.

Na figura 2.3, é apresentado o diagrama de casos de uso do sistema Webus. A seguir são analisadas as unidades do sistema a nível de modelagem e funcionalidades:

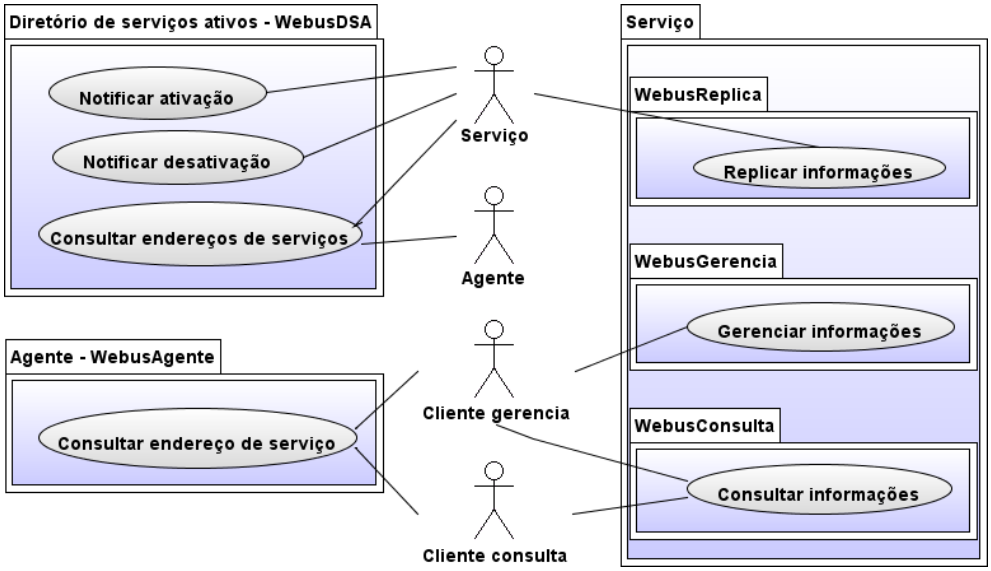

Figura 2.3: Diagrama de casos de uso.

2.1.3.1 WebusConsulta

Esta unidade tem como função fornecer os métodos de consulta do serviço, os quais são utilizados pelos aplicativos clientes. A modelagem dos dados foi baseada em sistemas existentes que manipulam as informações de um sistema de transporte coletivo urbano. Buscou-se modelar os dados de forma que abragesse as informações básicas que este tipo de sistema de transporte necessita, com o intuito de ser utilizado pela maioria das cidades que possuíssem este serviço público. Abaixo, na figura 2.4, pode-se ver um diagrama de classes que apresenta a modelagem dos dados do serviço oferecido pelo sistema Webus.

Os métodos implementados nesta unidade são todos públicos, ou seja, não necessitam autenticação. Devido à existência de dados restritos aos administradores do sistema, algumas classes não possuem métodos de consulta implementados nesta unidade. Os métodos consultam as seguintes classes do sistema, as quais serão descritas mais adiante:

- Categoria
- Cidade
- Empresa
- Estado
- DiaOrigem
- Linha

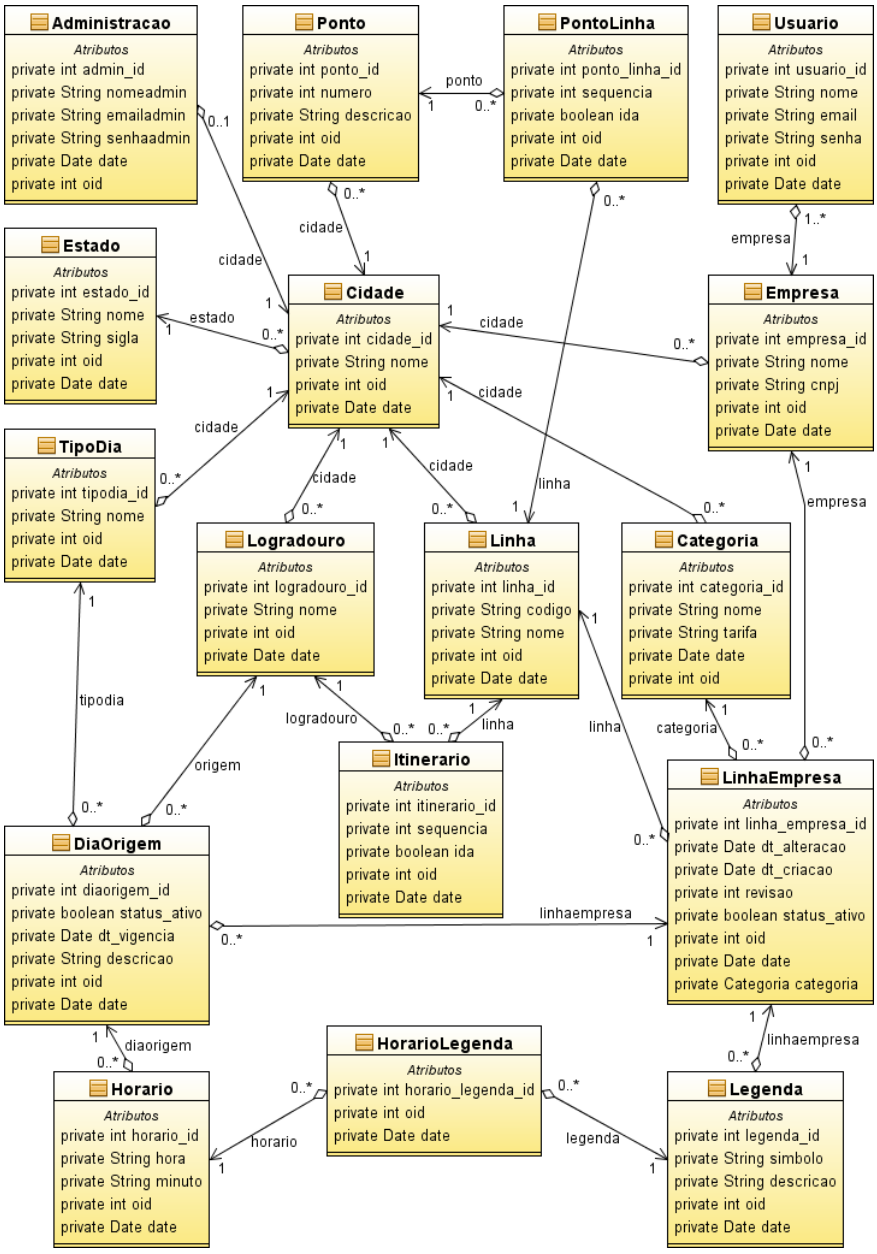

Figura 2.4: Diagrama de classes do serviço

- LinhaEmpresa
- Legenda
- Logradouro
- Horario
- HorarioLegenda
- Itinerario

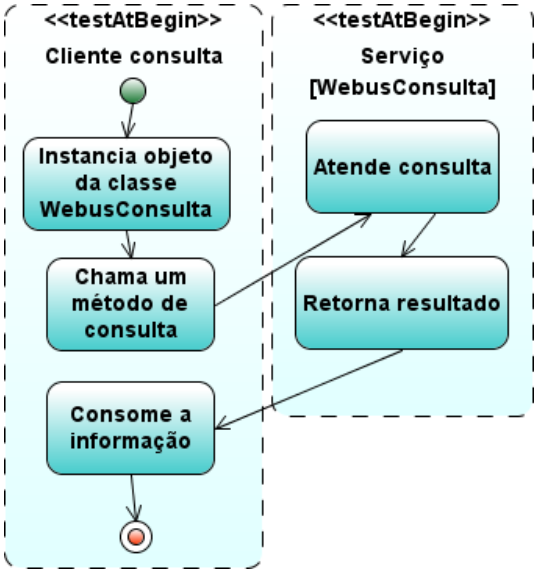

Figura 2.5: Diagrama de atividades do cliente consultando o serviço.

- Ponto
- PontoLinha
- TipoDia

A atividade realizada pela unidade WebusConsulta, a qual é fornecer as informações aos aplicativos clientes através de consultas, pode ser visualizada na figura 2.5.

2.1.3.2 WebusGerencia

A unidade que gerencia as informações possui métodos divididos em dois grupos: métodos para administração do município e para administração de empresas. Este Web Service necessita autenticação a qual é diferenciada para cada papel de administrador. A autenticação é realizada através do envio de informações como, nome de usuário e senha, juntamente com a requisição a uma operação de gerência. No caso da autenticação do administrador do munícipio, é realizada a verificação das informações enviadas comparando-as com as existentes na tabela do banco de dados referente à classe *Administracao*. Já na autenticação do administrador da empresa é realizada uma verificação na tabela referente à classe *Usuario*.

As classes que são gerenciadas pelo papel da administração de município são as seguintes:

- Administracao – mantém o cadastro do administrador da cidade.

- Categoria – armazena as categorias do serviço, ou seja, transporte coletivo ou seletivo, por exemplo.
- Empresa – cadastra as empresas que realizam o serviço na cidade.
- Itinerario – cadastra o relacionamento entre logradouros e linhas.
- Linhas – cadastra as linhas de ônibus da cidade.
- LinhaEmpresa – vincula linhas de ônibus a empresas.
- Logradouro – cadastra os logradouros da cidade.
- Ponto – armazena os pontos de ônibus da cidade.
- PontoLinha – guarda o relacionamento dos pontos de ônibus com as linhas de ônibus.
- TipoDia – guarda os tipos de dia que o transporte é oferecido na cidade.

Além das operações de gerência das informações do município, que são permitidas ao administrador do município, existem outras duas operações, que são: publicar e despublicar o serviço. A operação de publicar, a qual pode ser vista na figura 2.6, tem como objetivo tornar disponível o endereço do serviço, uma vez que ele deve enviar para a unidade WebusDSA o seu endereço, a sua cidade e estado. Já a operação de despublicar, figura 2.7, requisita ao WebusDSA a remoção do registro publicado anteriormente, enviando as mesmas informações enviadas para publicar.

O papel no sistema de administrador de empresa, o qual é tratado como usuário, tem permissão para manipular as seguintes classes:

- DiaOrigem – armazena o relacionamento de um tipo de dia com uma origem para uma determinada linha operada pela empresa.
- Horario – guarda os horários de ônibus de um tipo de dia e origem de uma linha operada pela empresa.
- HorarioLegenda – vincula uma legenda a um horário de ônibus.
- LinhaEmpresa – realiza operação de alterar a data de modificação do vínculo de uma linha e a empresa.

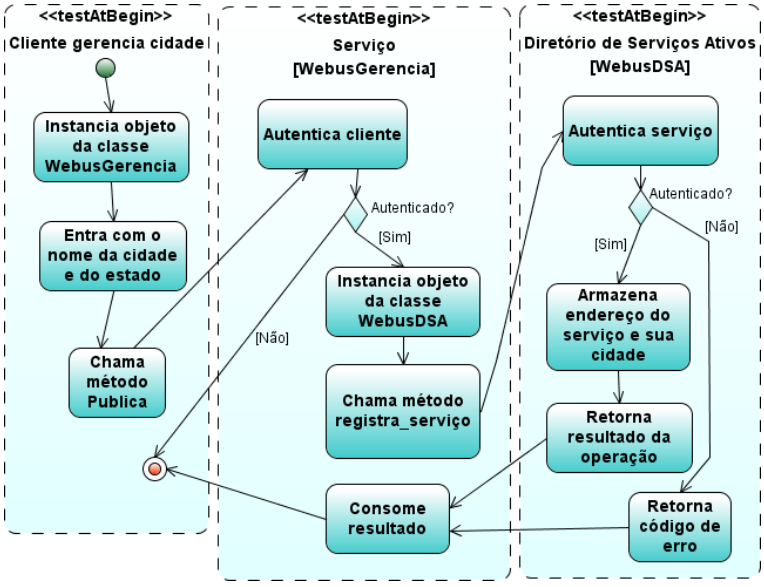

Figura 2.6: Diagrama de atividades do serviço registrando-se no WebusDSA.

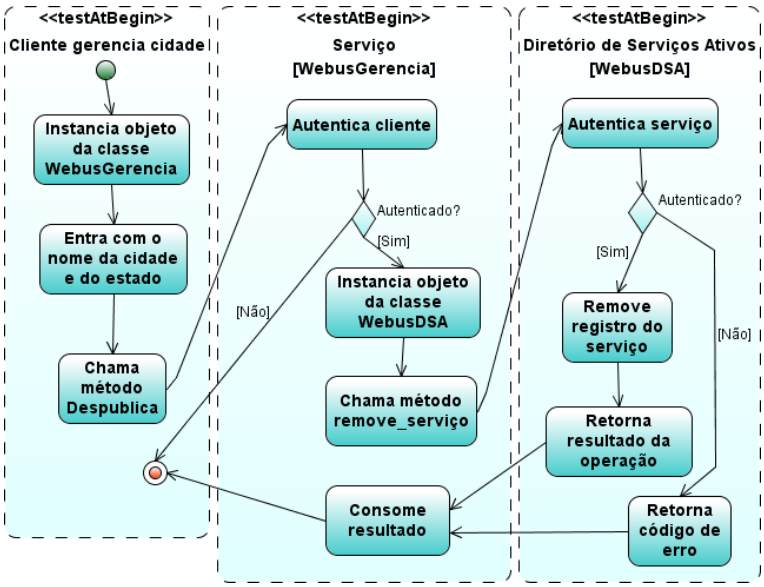

Figura 2.7: Diagrama de atividades do serviço removendo o seu registro do WebusDSA.

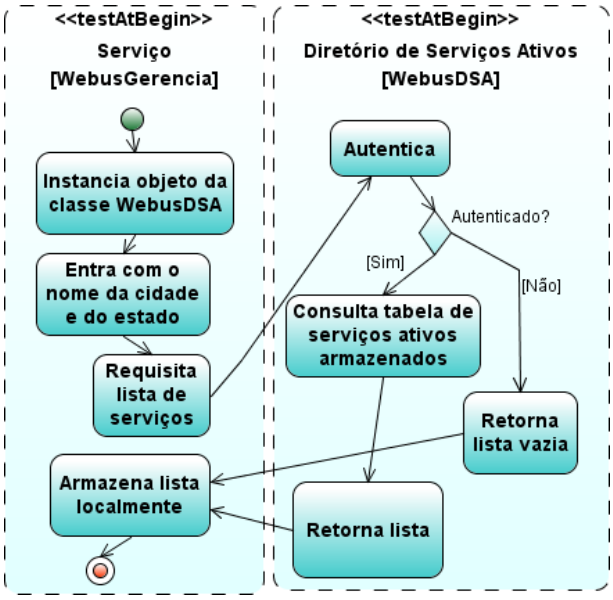

Figura 2.8: Diagrama de atividades do serviço consultando o WebusDSA.

- Legenda – cadastra uma legenda em uma determinada linha operada pela empresa.
- Usuario – armazena os usuários da empresa que podem operar o sistema.

Após uma operação ser realizada nesta unidade, é necessário realizar a operação nas réplicas do serviço, caso existam. Para que seja realizada a replicação, a unidade WebusGerencia, que representa o serviço, deve atuar como cliente da unidade WebusReplica que está localizada nos outros serviços. A unidade WebusGerencia mantém uma lista de endereços de serviços, mas a cada nova operação de replicação iniciada a lista é complementada com endereços retornados de uma consulta ao WebusDSA. Esta atividade pode ser vista na figura 2.8. Para efetuar uma replicação, a unidade WebusGerencia instancia um objeto WebusReplica para cada endereço de serviço da lista e chama o método de salvamento ou de remoção.

Ao utilizar replicação de dados, surgem alguns desafios como, por exemplo, de que forma identificar objetos que estão armazenados em registros de bancos de dados que estão distribuídos. Este problema é verificado quando o banco de dados relacional utiliza chaves primárias que são dependentes apenas localmente, ou seja, o banco de dados não sabe da existência de outros bancos de dados que são suas réplicas.

No sistema desenvolvido, houve uma dificuldade em determinar que tipo de chave primária utilizar nas tabelas do banco de dados e como identificar os objetos. Uma vez que, o sistema é distribuído e utiliza-se uma banco de dados relacional, mas que não é específico

para sistemas distribuídos, deve-se propor um tipo de identificador único para os objetos a nível de sistema. Como solução, foi adicionado um atributo do tipo inteiro chamado de *oid*, que pode ser entendido como o identificador do objeto no sistema distribuído. Este valor é gerado pela classe do objeto, a partir de uma operação realizada com alguns atributos da classe. É importante notar que, a nível do banco de dados, este atributo da classe, ou seja, esta coluna da tabela, não é um atributo chave. As chaves primárias foram mantidas como um atributo inteiro que é incrementado a cada novo registro inserido, ou seja, os registros possuem chaves primárias que são conhecidas apenas localmente. Desta forma, um registro localizado em um banco de dados expressa relação com o seu registro replicado em outro banco de dados através do *oid*, uma vez que ele é gerado a partir dos seus próprios atributos.

Além disso, para manter a coerência das informações mantidas nas réplicas, adicionou-se um atributo do tipo data, chamado de *date*, para realizar um controle de qual registro é mais novo e qual é mais velho. Sabe-se que esta forma de controle de coerência dos dados pode apresentar problemas. Se duas réplicas, por exemplo, estão com datas diferentes, operações terão seu funcionamento alterado, podendo tornar os dados incoerentes. Para abordar este problema e continuar utilizando a solução do atributo tipo data, torna-se necessário um pré-requisito aos servidores que queiram implantar o sistema Webus. Este pré-requisito refere-se à necessidade de existir um mecanismo que mantenha o relógio atualizado e sincronizado a um servidor confiável. Para prover sincronização de relógio entre dispositivos de uma rede, muitos servidores na Internet têm utilizado o protocolo NTP (David L. Mills, 2009). Portanto, para que o controle de coerência dos dados do sistema seja satisfeito, deve-se ter instalado nos servidores um mecanismo que utilize este protocolo.

A partir da atribuição de um identificador único para o objeto, criou-se a necessidade de realizar verificações adicionais nas operações de manipulação dos registros do banco de dados. As operações realizadas são três, as quais são descritas a seguir, sendo que após serem realizadas o processo de replicação é iniciado:

- **Inserção** – Ao inserir um registro no serviço, através de alguns dos métodos da unidade WebusGerencia, criou-se a necessidade de verificar a existência de um registro, de mesmo tipo, com o mesmo *oid*. Se já existir um registro, a operação não pode ser realizada. Do contrário, se a operação for realizada com sucesso, a data

atual é atribuída ao atributo *date*, um fluxo de execução é iniciado e o processo de replicação inicia.

- **Alteração** – a alteração do dado é realizada buscando-o pela sua chave primária. O *oid* é calculado novamente e a data é atualizada. Se a operação é realizada com sucesso também é iniciado o processo de replicação.

- **Remoção** – a remoção do dado é feita buscando-o pela sua chave primária e após a operação ser realizada também é iniciado o processo de replicação.

Além da questão de identificação dos objetos, em determinados momentos uma ou mais réplicas podem estar indisponíveis e uma operação de replicação pode ter sido iniciada no sistema. Quando uma réplica tornar-se novamente disponível, ela deve executar as operações de replicação que foram efetuadas durante o seu período de indisponibilidade. Para solucionar este problema, foi adicionada, no banco de dados do serviço, uma tabela auxiliar que armazena uma referência ao objeto que não teve a operação de replicação realizada em uma ou mais réplicas. Para tratar a nova tentativa de replicar operações pendentes armazenadas na tabela auxiliar, realiza-se uma busca nesta tabela no momento que uma nova operação de replicação é iniciada no serviço. Com o intuito de não interferir no desempenho do Web Service, a criação de operações de replicação, ou seja, chamadas aos métodos da classe WebusReplica, são fluxos de execução diferentes do fluxo normal do Web Service.

A atividade de replicação de uma operação iniciada pelo serviço, neste caso pela unidade WebusGerencia, possui um protocolo que deve ser seguido, o qual pode ser visto na figura 2.9. Este protocolo consiste na seguinte sequência de operações:

1. Serviço instancia um objeto da classe WebusDSA e realiza a chamada de um método que fornecerá uma lista de endereços de serviços que são réplicas.

2. Para cada endereço de serviço da lista é instanciado um objeto da classe WebusReplica. Caso o objeto não seja instanciado para algum elemento da lista, uma referência ao objeto é armazenada em uma tabela auxiliar. Se o objeto for instanciado com sucesso é chamado o método para registrar o objeto, caso seja uma operação de inserção ou alteração, ou para remover um registro, no caso de uma operação de remoção de um dado.

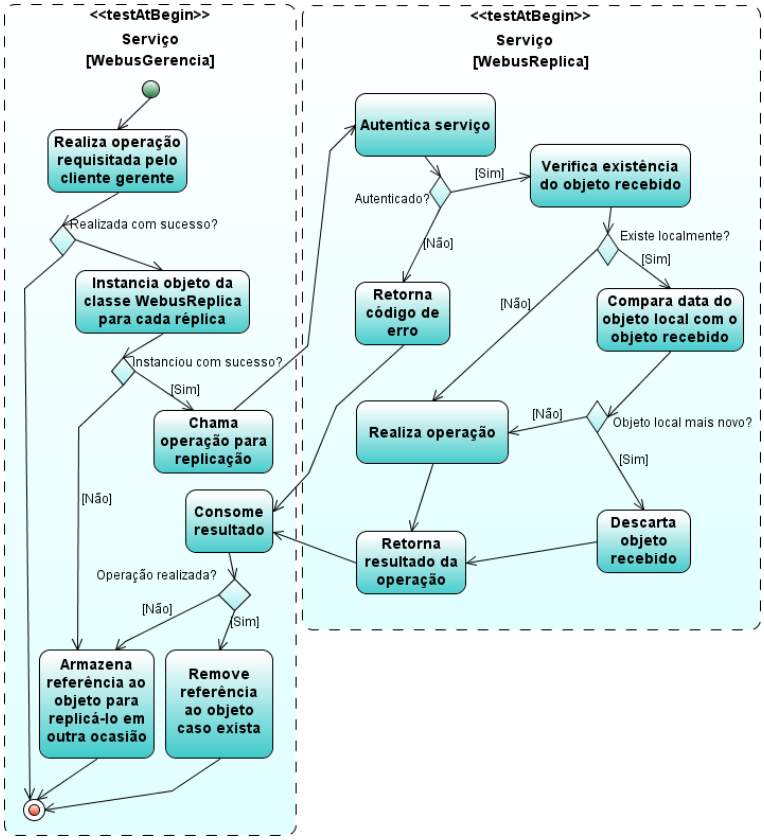

Figura 2.9: Diagrama de atividades do serviço replicando uma operação.

3. Na unidade WebusReplica, chamada pelo serviço que iniciou a replicação, é realizada a sua autenticação. Caso a autenticação seja efetuada o processo continuará, senão um valor de erro é retornado. Se o objeto que está sendo replicado existir na réplica deverão ser comparadas as suas datas. Caso o objeto recebido pela réplica seja mais novo que o objeto local a operação será realizada, senão um valor de erro é retornado. Finalmente, se a operação for realizada um valor informando o sucesso ou não da operação é retornado ao serviço. Na próxima subseção, esta atividade será detalhada.

4. No serviço que iniciou a replicação é verificado o resultado da operação realizada nas réplicas, sendo que quando uma delas falhar uma referência ao objeto é armazenada em uma tabela auxiliar. A tabela auxiliar funciona como uma agenda, que será analisada em uma próxima operação de replicação criada pelo serviço, na qual realizará uma nova tentativa de replicação destes dados.

Como citado anteriormente, existe uma tabela auxiliar no banco de dados que armazena uma referência a um objeto que ainda não foi replicado. Esta tabela é chamada de

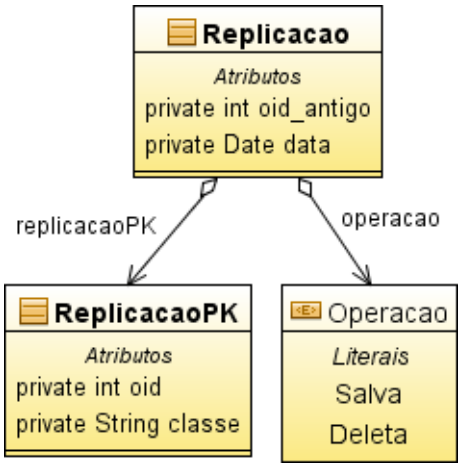

Figura 2.10: Diagrama de classes das entidades referentes ao processo de replicação.

replicacao e pode ser vista no diagrama de classes da figura 2.10. Esta tabela é consultada quando um novo processo de replicação é iniciado, ou seja, todos os registros da tabela, os quais referenciam objetos, são analisados e são novamente processados.

Quando uma operação de inserção falha no processo de replicação, são armazenados na tabela replicação o *oid* do objeto, a classe que ele pertence, o tipo de operação, e a data da operação. Este tipo de registro ao ser consultado, para realizar novamente a replicação, resultará em uma busca na tabela do objeto que ele se refere a partir do seu identificador de objeto, o *oid*. Se este objeto não existir, a operação é cancelada e a referência na tabela replicação é removida. Por outro lado, se ele existir o objeto será enviado, juntamente com a sua classe, aos serviços de replicação das outras réplicas.

Na operação de alteração, caso ocorra falha na replicação, são armazenados o *oid* do objeto, calculado depois da alteração, o *oid* anterior do objeto, a classe que ele pertence, o tipo de operação e a data da operação. É necessário o armazenamento do *oid* anterior do objeto, pois nas réplicas o objeto ainda está sendo identificado com este *oid*. Como na operação anterior, em determinado momento a tabela replicação é consultada e os objetos referenciados são buscados em suas tabelas. Neste processo de alteração, o objeto será enviado aos serviços de replicação, porém com o seu atributo *oid* anterior à atualização.

Assim como, nas duas situações anteriores, também é armazenada uma referência ao objeto na tabela replicação, quando a operação de replicação apresentou falha. Os atributos que serão armazenados são: *oid*, classe, operação e data. Neste caso, quando a tabela replicação for consultada e o registro indicar a operação do tipo remoção, não será realizada busca do objeto, pois ele foi removido. Para que replicação seja realizada

o atributo *oid* do objeto e a sua classe são enviados aos serviços de replicação.

Para que o processo de replicação de uma operação seja iniciado pela unidade WebusGerencia deve ser consultada, primeiramente, a unidade WebusDSA. Pode ocorrer desta última estar indisponível, podendo assim ocasionar inconsistência nos dados em algumas réplicas, uma vez que a unidade WebusGerencia não conseguirá atualizar os seus endereços de serviço para processar a replicação. Talvez a unidade WebusGerencia consiga realizar um processo de replicação para um subconjunto de réplicas, já que possui alguns endereços armazenados localmente, o que causa inconsistência no sistema, caso o endereço de alguma réplica não tenha sido incluído no processo.

Para solucionar este problema, pensou-se em manter um indicador de estado para cada réplica. Este estado pode ter os valores: *seguro* e *inseguro*. O estado seguro é utilizado quando o WebusDSA é instanciado normalmente, porém quando uma falha ocorrer na instanciação da unidade WebusDSA o estado passa a ser *inseguro*. Quando um processo de replicação é iniciado e o estado da réplica tem o valor *inseguro*, todas as operações realizadas são armazenadas na tabela *replicacao*. Quando o WebusDSA tornar-se novamente disponível, o estado volta a ter o valor *seguro* e o processo de replicação volta ao seu modo normal de funcionamento. A unidade WebusGerencia, ao instanciar novamente a unidade WebusDSA e consultar os endereços de serviço, realizará o processo de replicação de todas as operações, armazenadas na tabela replicação, nos endereços até então inacessíveis.

2.1.3.3 WebusReplica

Esta unidade do sistema funciona como um serviço que atende requisições de operações a serem replicadas. Os clientes requisitores são as unidades WebusGerencia, as quais realizam uma operação e enviam informações ao WebusReplica para que ele mantenha a consistência da sua base de dados.

Este serviço possui três métodos, os quais tem como função realizar as operações de inserção, alteração e remoção. A seguir são descritos os três casos:

- Inserção – o método *insere* recebe um objeto e o nome da classe que ele pertence. Antes de realizar a inserção deste objeto recebido, verifica-se pelo seu *oid* a existência deste objeto na tabela de destino. Se o objeto existir e a data do objeto já registrado for mais antiga que a do objeto recebido então atualiza-se o objeto regis-

trado. Caso a data seja mais nova o objeto recebido é descartado. Se na verificação pela existência não for encontrado o registro, o objeto recebido será inserido. Um valor de resultado é enviado para o requisitante para que realize o fechamento do pedido de replicação.

- Alteração – o método *altera* recebe um objeto e o nome da classe que ele pertence. É realizada uma busca pelo objeto usando seu *oid* como chave de busca. Se este objeto existir os campos serão atualizados e o novo *oid* será gerado e atualizado, caso a data do objeto recebido seja mais nova que a data do objeto já cadastrado. Caso o objeto não exista, deve-se verificar a existência de algum registro com o *oid* do objeto na tabela replicação, o qual tenha como tipo de operação a remoção. Se for encontrado, então se deve comparar a data do registro na tabela replicação com a data do objeto recebido, sendo que se a data na tabela replicação for mais nova, o objeto recebido é descartado. Porém, se a data for mais velha, o registro é removido da tabela replicação e o objeto é inserido calculando-se o seu novo oid. Um valor de resultado da operação é enviado ao requisitante.

- Remoção – o método *remove* recebe o *oid* do objeto e a sua classe. Verifica-se a existência do objeto a partir do seu *oid* e então, caso o objeto tenha sido encontrado, é feita a remoção do objeto. É retornado um valor com o resultado da operação ao requisitante.

2.1.3.4 WebusDSA

O WebusDSA é uma unidade do sistema conhecida como Diretório de Serviços Ativos, a qual mantém os endereços dos serviços que estão ativos. Esta unidade funciona como um serviço Web que recebe notificações dos serviços que estão ativos, mantendo informações destes serviços. Para que sejam armazenadas estas informações, ela possui uma base de dados local, na qual podem ser realizadas consultas pelos clientes da unidade WebusDSA.

Três operações são oferecidas por esta unidade. As operações de registro, remoção de um endereço de serviço e consulta. O cliente da unidade WebusDSA, que é o serviço do sistema no papel da unidade WebusGerencia, realiza a operação de registro, tornando-se disponível ao sistema. O endereço deste serviço e a sua cidade são armazenados em uma única tabela da base de dados chamada de *servico*. A unidade WebusGerencia também

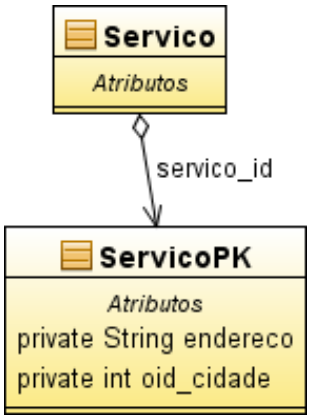

Figura 2.11: Diagrama de classes das entidades do Diretório de Serviços Ativos

pode realizar a operação de remoção de registro. Assim como na operação anterior, o cliente deve fornecer o seu endereço, a sua cidade e seu estado para que o registro seja removido. A terceira operação fornecida é a de consulta. Nesta operação, um cliente deve informar a cidade e o estado de interesse para que a consulta retorne endereços de serviços, ou seja, endereços do conjunto de unidades: WebusConsulta, WebusGerencia e WebusReplica.

O WebusDSA possui três clientes: as unidades WebusGerencia, WebusAgente e WebusReplica. A unidade WebusGerencia pode realizar as operações de registro e de remoção, enquanto as outras duas só podem realizar consultas. Para que estas operações sejam realizadas os Web Services envolvidos necessitam realizar autenticação. Na figura2.11, pode ser visto o diagrama de classe da unidade WebusDSA.

Ao realizarem a operação de consulta, as unidades têm como retorno uma lista com endereços de serviços. No caso da unidade WebusAgente, esta lista é utilizada para fornecer um endereço de serviço ativo a um aplicativo cliente, uma vez que a lista é percorrida e os endereços são testados sequencialmente até que um endereço esteja ativo. Se esta lista é fornecida de maneira igual a várias unidades WebusAgente, pode ocorrer uma sobrecarga em um único endereço, já que ele será identificado como ativo pelas unidades WebusAgente de maneira semelhante. Para solucionar este problema, foi implementado um algoritmo para balanceamento de carga na criação da lista de endereços de serviços, na unidade WebusDSA. Este algoritmo é semelhante ao conhecido *Round Robin*(Andrew S. Tanenbaum, Albert S. Woodhull, 2000) e funciona como uma fila que é incrementada circularmente colocando, em toda nova consulta, um novo endereço como primeiro da lista de endereços de serviços.

Com relação a disponibilidade da unidade WebusDSA, problemas como inconsistência dos dados nas réplicas podem ocorrer.

2.1.3.5 WebusAgente

O Agente também é um serviço Web e fica localizado próximo ao cliente, ou seja, cada cliente pode ter o seu próprio WebusAgente implantado. Ele tem como função manter uma lista de endereços de serviços de uma determinada cidade e fornecer um endereço ativo ao aplicativo cliente. O Agente fornece uma dinamicidade na descoberta de endereços de serviços e portanto tem grande importância no sistema. Esta unidade possui uma única operação, a qual o aplicativo cliente requisita um endereço de serviço informando o nome da cidade e do estado.

Ao ser requisitado por um cliente, o WebusAgente deve fornecer um endereço de serviço ao cliente. O Agente mantém uma lista de endereço de serviços armazenada em um arquivo, o qual a cada requisição do cliente é consultado. Primeiramente, o Agente verifica os endereços armazenados no arquivo e os carrega para uma lista armazenada em memória, porém se o arquivo estiver vazio deve-se consultar o WebusDSA. Quando a lista de endereços estiver carregada os endereços são testados, até que um deles referencie um serviço disponível, e então o endereço é fornecido ao cliente. Caso todos os endereços testados estejam indisponíveis, nenhum endereço será fornecido ao cliente e ele não conseguirá conexão com o serviço. Na figura 2.12, pode-se visualizar a atividade na qual o WebusAgente requisita uma lista de endereços de serviços, de uma determinada cidade, ao WebusDSA.

2.1.4 Implementação do sistema

As unidades da arquitetura do sistema Webus consistem em Web Services, uma vez que podemos identificar esta arquitetura como uma arquitetura orientada a serviços. A interação entre as unidades é realizada através da comunicação entre elas, sendo que em determinadas situações algumas das unidades apresentam um papel de cliente ou de provedor de serviço.

Para implementação de Web Service foi escolhida a solução padrão atual da Sun Microsystem Inc., a chamada Java API for XML Web Services (JAX-WS) (Sun Microsystems, Inc., 2008a). Esta API está em sua versão 2.1 e é parte integrante do projeto Metro (Sun Microsystems, Inc., 2008b). Basicamente, um Web Service em linguagem Java é

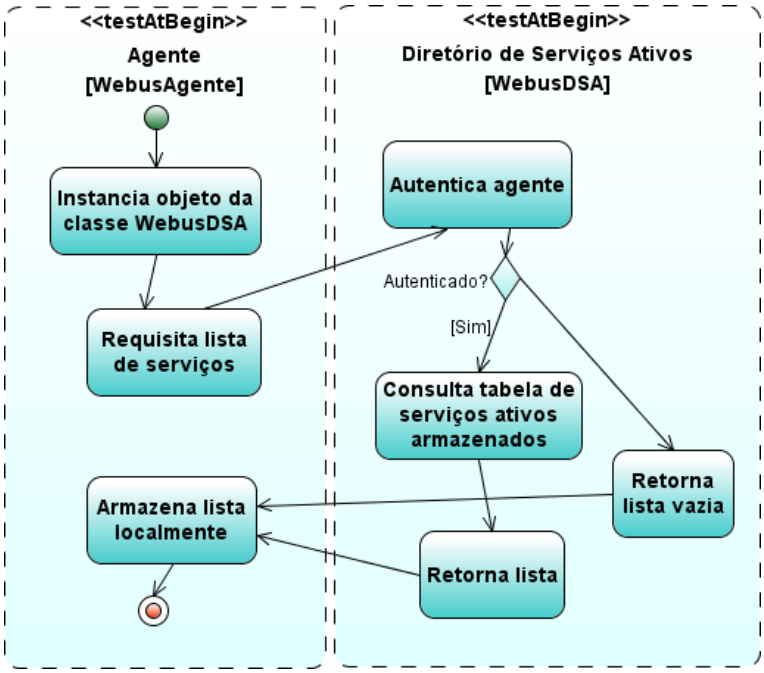

Figura 2.12: Diagrama de atividades do WebusAgente consultando o WebusDSA.

```
@WebService()
public class WebusConsulta{

   @WebMethod(operationName = "getListaEstados")
   public Lista<Estado> getListaEstados(){
      EstadoDAO dao = new EstadoDAO();
      List<Estado> retorno = new ArrayList<Estado>();
      try{
         retorno = dao.recuperaTodas();
      }catch(Exception ex){
         return null;
      }
      return retorno;
   }
}
```

Figura 2.13: Classe em Java que representa um Web Service

uma classe que recebe algumas ”anotações”. Estas anotações são informações adicionadas no código Java que iniciam pelo caractere ”@” e indicam alguma propriedade especial (Sun Microsystems, Inc., 2009a).

Um exemplo de uma classe que é um Web Service pode ser visto na figura 2.13, cuja classe recebe a anotação *@WebService()*, que faz parte da API JAX-WS. Esta classe define a unidade WebusConsulta e implementa um método que retorna uma lista de objetos do tipo *Estado*.

Após a aplicação Web que representa o Web Service ser compilada, ela deve ser implantada, para que o seu documento WSDL torne-se disponível. A implantação pode ser feita de duas formas: sem servidor de aplicação e com servidor de aplicação. Optou-se

implantar os Web Services da arquitetura utilizando-se servidor de aplicação, pois em alguns casos é necessário realização de autenticação, a qual precisa manter dados armazenados em uma sessão do servidor de aplicação. O servidor de aplicação GlassFish (Sun Microsystems, Inc., 2009b) foi escolhido, pois é o servidor padrão desenvolvido pela Sun Microsystem Inc. para a plataforma Java Enterprise Edition (JavaEE), a qual é utilizada para criação de aplicações Web.

Como visto anteriormente, existe a necessidade de autenticação dos aplicativos clientes na unidade WebusGerencia, para que eles possam acessar o serviço. A autenticação pode ser de dois tipos: autenticação do administrador da cidade e do administrador de empresa. A nível de implementação, essa autenticação é realizada através do envio dos dados de autenticação no cabeçalho da mensagem SOAP, juntamente com uma requisição à alguma operação. Estes dados são enviados através da inserção de dois elementos dentro do elemento *Header* da mensagem. No caso do administrador de cidade, os dois elementos são: ”EmailAdminCidade“ e ”SenhaAdminCidade“. Já no caso do administrador de empresa, os elementos são: ”EmailUsuarioEmpresa“ e ”SenhaUsuarioEmpresa“. Dentro destes elementos são enviados os dados para autenticação, os quais serão lidos pelo servidor, verificados na base de dados e assim criada uma sessão de autenticação para aquela operação. Caso a autenticação tenha falhado, a sessão de autenticação não é criada, logo a operação requisitada não é realizada.

Além disso, tornou-se necessário realizar as operações de acesso ao banco de dados de forma que se garantisse as quatro propriedades importantes relacionadas as operações: atomicidade, consistência, isolamento e durabilidade. Para conquistar isto adotou-se o framework Hibernate (Red Hat Middleware, LLC, 2009) para mapeamento da base de dados. Esta ferramenta facilita o mapeamento das tabelas do banco de dados em classes Java utilizando arquivos de configuração do mapeamento em linguagem XML. Além disso, o Hibernate utiliza os conceitos de Transação que garante as quatro propriedades citadas anteriormente.

O banco de dados selecionado para armazenar as informações do serviço prestado pelo sistema Webus foi o PostgreSQL Database Server 8.3 (PostgreSQL Global Development Group, 2009). Este sistema gerenciador de banco de dados objeto relacional é de código aberto e possui muitos recursos que o tornam bastante utilizado.

Para o desenvolvimento das unidades do sistema Webus, foi utilizado o ambiente de

desenvolvimento Netbeans (NetBeans IDE, 2009), que facilita a programação de sistemas. Esta ferramenta tem como vantagem o suporte nativo ao servidor de aplicação GlassFish.

2.2 Aplicação da arquitetura

2.2.1 Implantação do sistema

Como visto anteriormente, o sistema Webus contém cinco unidades: WebusConsulta, WebusGerencia, WebusReplica, WebusDSA e WebusAgente. As unidades de consulta e gerência estão encapsuladas em uma mesma aplicação Web, logo, ao implantar esta aplicação Web os dois Web Services tornam-se disponíveis. As outras unidades são aplicações Web distintas e implantadas separadamente.

A proposta do sistema é fornecer um mecanismo aos aplicativos clientes que queiram consultar e gerenciar informações do sistema de transporte coletivo urbano municipal. Para que este fornecimento se torne possível, é necessário o interesse dos responsáveis pelo serviço do município, sejam eles apenas a prefeitura da cidade ou também as empresas concessionárias do serviço.

Para que uma cidade disponibilize o serviço proposto é necessária a implantação das unidades WebusConsulta, WebusGerencia, WebusReplica e WebusDSA em um servidor de aplicação disponível na Internet. Após esta implantação, o administrador do sistema deve realizar, através de uma aplicação cliente da unidade WebusGerencia, a disponibilização do serviço na unidade WebusDSA, a qual vai armazenar o endereço do serviço e vai aguardar por consultas. Caso o administrador tenha necessidade de maior disponibilidade, uma réplica do serviço pode ser implantada em outro servidor. Esta nova implantação necessita das unidades WebusConsulta, WebusGerencia e WebusReplica, sendo que após a implantação o administrador deve realizar a disponibilização do serviço na unidade WebusDSA, já implantada anteriormente.

As aplicações clientes que queiram consumir este serviço poderão acessar diretamente um dos endereços do serviço implantado, ou utilizar a unidade WebusAgente. Esta unidade fica localizada próxima ao cliente e deve ser implantada em um servidor de aplicação. Ela fornece uma consulta que resulta em endereços de serviços implantados para uma determinada cidade. Desta forma, o aplicativo cliente não fica dependente de apenas um provedor de serviço, uma vez que ele possui uma lista de endereços que podem ser

testados até que um serviço esteja disponível.

2.2.2 Aplicação cliente

Um Web Service tem o intuíto de fornecer um conjunto de métodos para aplicativos clientes, desta forma achou-se necessário implementar um aplicação cliente para exemplificar a interação entre provedor de serviço e consumidor. Como citado anteriormente, os Web Services se caracterizam, principalmente, por prover comunicação entre aplicações implementadas em linguagens de programação diferentes. De acordo com esta característica, buscou-se reforçá-la criando uma aplicação cliente do sistema Webus utilizando um linguagem diferente.

Várias linguagens de programação poderiam ser utilizadas para criação de um cliente, assim como: C/C++, Java, Python, entre outras, porém a linguagem de programação C-Sharp (C#), que pertence a plataforma Microsoft Windows, foi escolhida. A escolha baseou-se na distinção que encontramos entre a plataforma Java e Windows, além da plataforma Windows fornecer uma solução padrão para utilização de Web Services, conhecida como Windows Communication Foundation (Microsoft Corporation, 2008). Esta solução fornece padrões e ferramentas que facilitam o consumo e a criação de Web Services, tornando transparente a programação de uma aplicação cliente que consome um serviço Web. Além disso, foi utilizada a linguagem ASP para construção das páginas Web do aplicativo cliente, sendo que a tela inicial pode ser visualizada na figura 2.14.

Para criação de um cliente de um Web Service, o Windows Communication Foundation oferece uma ferramenta que, a partir do contrato do serviço, ou seja, do arquivo WSDL, gera classes que referenciam as classes implementadas no serviço. Desta forma, a chamada remota dos métodos do serviço torna-se simples, pois uma classe que representa o serviço é instanciada e os métodos são facilmente utilizados.

O sistema Webus oferece aos clientes o serviço de consulta (WebusConsulta) e de gerência (WebusGerencia) das informações, sendo a gerência dividida em gerência das informações da cidade e das informações das empresas. O aplicativo cliente criado utiliza tanto os métodos de consulta quanto os métodos de gerência. A figura 2.17 mostra uma operação de inserção de um logradouro na administração da cidade de Santa Maria. Já a figura 2.18 mostra a tela do para inserção de horários de ônibus, a qual é uma operação para o administrador da empresa. Além disso, o aplicativo cliente pode consultar o Webu-

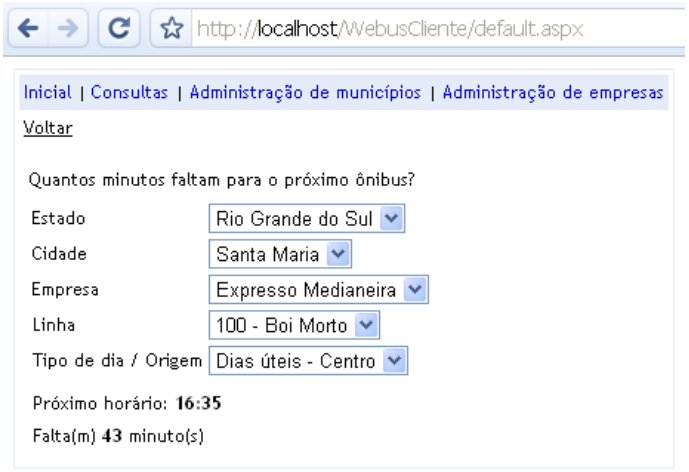

Figura 2.14: Tela inicial do aplicativo cliente

sAgente, o qual fornece endereços do serviço WebusConsulta e WebusGerencia. Portanto, também são geradas classes do Web Service WebusAgente utilizando as ferramentas do WCF. Exemplos de consultas que esse cliente realiza são apresentadas nas figuras 2.15 e 2.16.

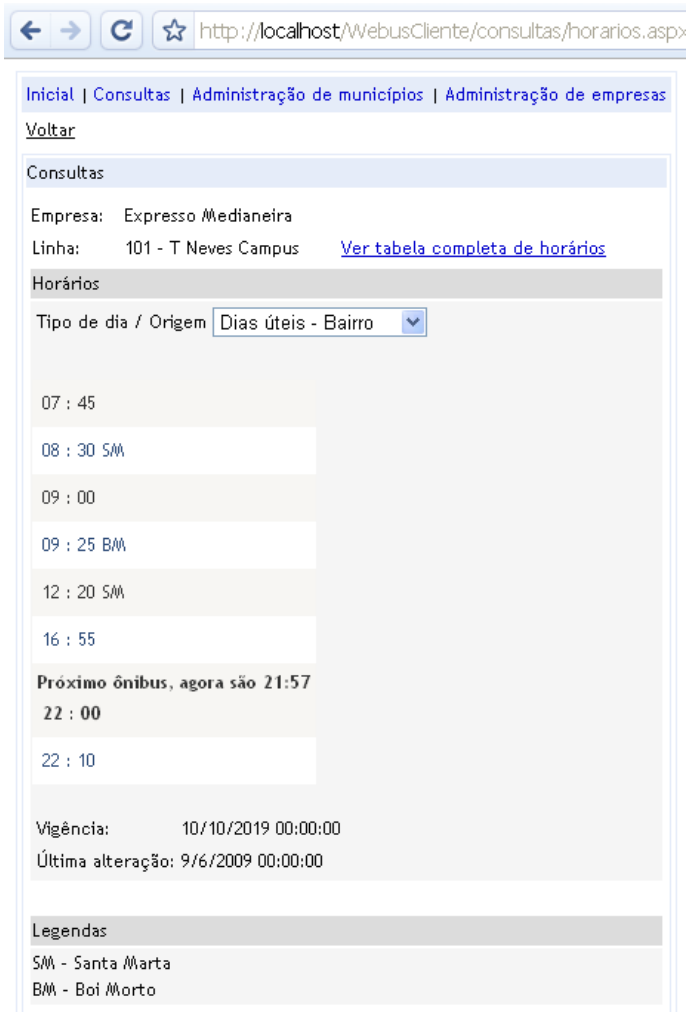

Figura 2.15: Consulta de horários de ônibus de uma linha

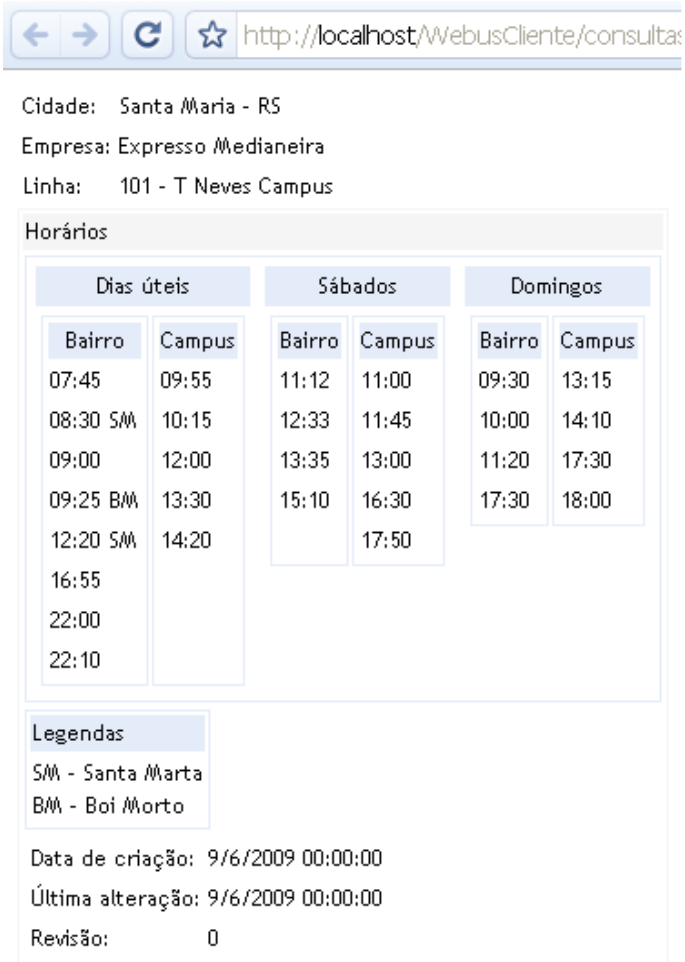

Figura 2.16: Tabela completa de horários de ônibus de uma linha

Figura 2.17: Operação de inserção de logradouro na administração de município

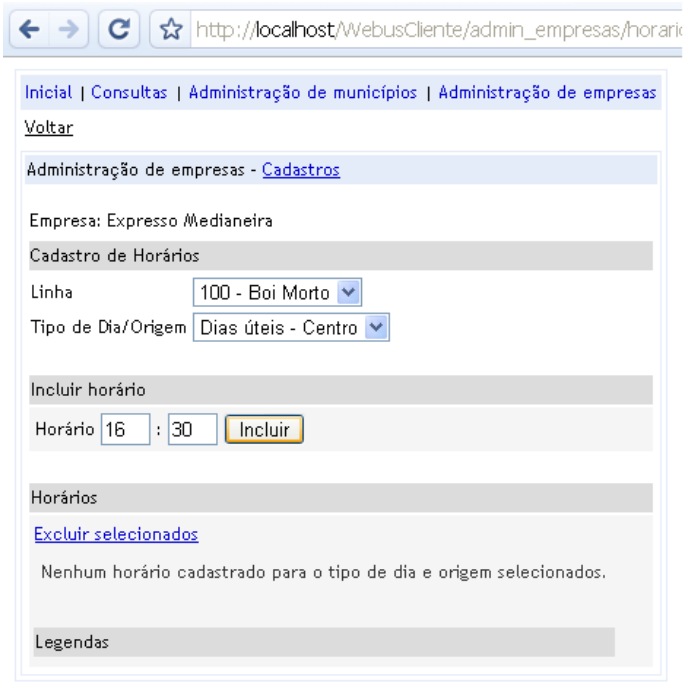

Figura 2.18: Operação de inserção de horários na administração de empresa

3 CONCLUSÕES E CONSIDERAÇÕES FINAIS

A necessidade de prover serviços na Internet e realizar interação entre aplicações com características diferentes, impulsionou nos últimos anos a criação de novas tecnologias em sistemas distribuídos. Web Services surgiram com estes objetivos e têm tido sucesso na área de desenvolvimento de sistemas, devido à utilização de padrões e protocolos abertos.

Neste contexto de prestação de serviços, surge um problema muito comum nos centros urbanos: a disponibilização de informações do sistema de transporte coletivo urbano municipal. Problema este relacionado com as aplicações clientes que necessitam destas informações para realizar alguma operação e então fornecê-las aos usuários, por exemplo. É notável que existe pouca ou nenhuma padronização naquelas informações, tornando assim dificultoso o consumo delas por outras aplicações.

Como solução, foi proposta uma arquitetura distribuída orientada a serviços, que além de fornecer aos aplicativos clientes uma ferramenta para gerência e consulta das informações do sistema de transporte coletivo urbano de uma cidade, tenta aumentar a tolerância a falhas e eficiência. Estas vantagens podem ser notadas, uma vez que buscou-se aplicar técnicas de replicação de dados e descoberta dinâmica de provedores de serviços na arquitetura distribuída.

Com esta solução, pode-se concluir que além do serviço construído atingir seu objetivo, o qual é fornecer facilidades às aplicações clientes, criou-se um leque de novas aplicações que podem ser desenvolvidas. Além de simples programas que mostram alguns horários de ônibus em um página na Web, por exemplo, aplicações como letreiros digitais instalados em pontos de ônibus podem ser úteis aos usuários deste tipo de serviço.

Além disso, foram identificadas algumas propostas de trabalhos futuros, com o intuito de trazer melhoria ao sistema. A arquitetura construída possui unidades que fornecem funcionalidades a outras unidades, ou seja, serviços não funcionais. A troca de infor-

mações entre as unidades WebusDSA, WebusAgente, WebusGerencia e WebusReplica necessita de autenticação entre as partes relacionadas, por isso um método de autenticação poderia ser analisado e implementado. Esta autenticação evitaria que aplicações clientes interfiram nas atividades que aquelas unidades realizam.

Ainda no contexto de segurança, algumas informações transmitidas entre unidades do sistema ou entre unidades e clientes necessitariam de um método de criptografia dos dados. Operações que manipulam as classes que mantém informações dos administradores do sistema são casos que precisariam utilizar criptografia. Portanto, a aplicação de um método de criptografia em operações utilizando Web Services poderia ser estudado futuramente.

REFERÊNCIAS

Sistemas operacionais: projeto e implementação. [S.l.]: Bookman, 2000.

Service Oriented Architecture with Java. [S.l.]: Packt Publishing, 2008.

David Booth et al. **Web Services Architecture**. [S.l.]: W3C, 2004.

David L. Mills. **The Network Time Protocol (NTP) Distribution**. Disponível em: http://www.eecis.udel.edu/ mills/ntp/html/index.html. Acesso em: julho de 2009.

GEOSIT. **Projeto Geosit**. Disponível em: http://www.geosit.com.br/. Acesso em: janeiro de 2009.

Java Web Services: up and running, 1st edition. [S.l.]: O'Reilly Media, 2009.

Maxtrack. **Arena Control Center 13**. Disponível em: http://www.maxtrack.com.br/. Acesso em: janeiro de 2009.

Microsoft Corporation. **Windows Communication Foundation**. Disponível em: http://msdn.microsoft.com/en-us/netframework/aa663324.aspx. Acesso em: dezembro de 2008.

NetBeans IDE. **NetBeans IDE**. Disponível em: http://www.netbeans.org/. Acesso em: maio de 2009.

SOA in Practice. [S.l.]: O'Reilly Media, 2007.

OASIS. **UDDI**. Disponível em: http://uddi.xml.org/. Acesso em: junho de 2009.

PostgreSQL Global Development Group. **PostgreSQL**. Disponível em: http://www.postgresql.org/. Acesso em: maio de 2009.

Red Hat Middleware, LLC. **Hibernate**. Disponível em: https://www.hibernate.org/. Acesso em: maio de 2009.

SPTrans. **SPTrans**. Disponível em: http://www.sptrans.com.br/sptrans08/home/. Acesso em: abril de 2009.

Sun Microsystems, Inc. **Java API for XML Web Services (JAX-WS)**. Disponível em: https://jax-ws.dev.java.net/. Acesso em: dezembro de 2008.

Sun Microsystems, Inc. **Metro Web Services for the Java Platform**. Disponível em: http://java.sun.com/webservices/index.jsp. Acesso em: novembro de 2008.

Sun Microsystems, Inc. **Annotations**. Disponível em: http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html. Acesso em: maio de 2009.

Sun Microsystems, Inc. **GlassFish - Open Source Application Server**. Disponível em: https://glassfish.dev.java.net/. Acesso em: maio de 2009.

TRANSOL. **TRANSOL**. Disponível em: http://www.transoltc.com.br/. Acesso em: abril de 2009.

URBS. **URBS**. Disponível em: http://www.urbs.curitiba.pr.gov.br/PORTAL/tabelahorario/. Acesso em: abril de 2009.

W3C. **Extensible Markup Language (XML)**. Disponível em: http://www.w3.org/XML/. Acesso em: novembro de 2008.

W3C. **Web Services Description Language (WSDL) 1.1**. Disponível em: http://www.w3.org/TR/wsdl. Acesso em: novembro de 2008.

W3C. **SOAP Version 1.2 Part 1: messaging framework (second edition)**. Disponível em: http://www.w3.org/TR/soap12-part1/. Acesso em: abril de 2009.

W3C. **HTTP - Hypertext Transfer Protocol**. Disponível em: http://www.w3.org/Protocols/. Acesso em junho de 2009.