INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO TOCANTINS – CAMPUS PALMAS

JULIANNE PEREIRA LIMA LICÓN

INTEROPERABILIDADE DE SISTEMAS UTILIZANDO ARQUITETURA ORIENTADA A SERVIÇO E WEBSERVICE

Palmas, TO
2011

JULIANNE PEREIRA LIMA LICÓN

INTEROPERABILIDADE DE SISTEMAS UTILIZANDO ARQUITETURA ORIENTADA A SERVIÇO E WEBSERVICE

Trabalho de conclusão de curso apresentado à Coordenação da Área de Informática como requisito parcial para obtenção da conclusão do Curso Superior de Tecnologia em Desenvolvimento de Sistema para Internet no Instituto Federal de Educação, Ciência e Tecnologia do Tocantins – Campos Palmas.

Orientador Prof. Esp. Marinaldo Oliveira Santos

Palmas, TO
2011

JULIANNE PEREIRA LIMA LICÓN

INTEROPERABILIDADE DE SISTEMAS UTILIZANDO ARQUITETURA ORIENTADA A SERVIÇO E WEBSERVICE

Este Trabalho de Conclusão de Curso foi julgado e aprovado como cumprimento às exigências legais do currículo do Curso Superior de Tecnologia em Desenvolvimento de Sistema para Internet pela Coordenação da Área Informática no Instituto Federal de Educação, Ciência e Tecnologia do Tocantins – Campos Palmas.

Palmas, 11 de Fevereiro de 2011.

Banca Examinadora:

_____

Prof. Esp. Marinaldo Oliveira Santos
Orientador

_____

Prof. Msc. Mauro Henrique Lima de Boni
Membro de Banca Examinadora

_____

Prof. MSc. Francisco Willians Makoto Plácido Hirano
Membro de Banca Examinadora

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela minha vida.

Ao meu marido Frederico Licón pela sua compreensão, incentivo e presença em todos os momentos da minha vida.

Aos meus filhos Bruno e Diego pelos belos e inocentes sorrisos de criança, que me confortaram e que tanto me ajudaram nas horas mais difíceis.

A minha família que sempre me incentivou para que eu alcançasse os meus objetivos.

Ao meu professor e orientador Marinaldo Oliveira Santos pela dedicação, paciência e colaboração durante todo o desenvolvimento deste trabalho.

Aos professores do curso de Desenvolvimento de Sistema para Internet pelas trocas de experiências.

A minha amiga Mayra Marques que me ajudou e nunca me deixou desistir nos momentos mais críticos da elaboração deste trabalho.

Aos colegas do curso e do trabalho que sempre me apoiaram e me incentivaram no desenvolvimento deste trabalho.

E finalmente, agradeço a todos que me ajudaram direto ou indiretamente para o desenvolvimento deste projeto.

RESUMO

Sistemas web por si só não é um diferencial competitivo para as organizações, mas sim a maneira como eles são aproveitados. O desenvolvimento de sistemas pensando na interoperabilidade é uma tendência para os dias de hoje e como uma necessidade poderia ser comum a outros sistemas, a idéia hoje é criar serviços para torná-los mais eficientes e utilizáveis. Um conceito que vem sendo adotado atualmente é a arquitetura orientada a serviço (SOA) juntamente com a tecnologia de WebServices. Neste trabalho o objetivo é demonstrar as vantagens obtidas com a utilização de SOA e WebService na Prefeitura Municipal de Palmas, através da interoperabilidade entre o sistema de endereçamento com o sistema FIC (Formulário de Informações Cadastrais) e por fim, demonstrar ainda, como esse serviço criado tornou-se solução para qualquer outro sistema da Prefeitura Municipal de Palmas e futuramente poderá ser útil em qualquer sistema que dele se fizer necessário.

Palavras-chave: Integração de Sistema, WebService, SOA.

ABSTRACT

The development systems thinking on interoperability is a tendency today. The idea is to create services that may be available on the Internet. The concept Service Oriented Architecture (SOA) with the technology of Web Services is usable for create system with interoperability feature. The objective of this work is to demonstrate the benefits obtained with the use of SOA and Web Service helped integration between the addressing system with the system FIC in Prefeitura Municipal de Palmas.

Key-words: System Integration, Web Service, SOA.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Java Application Programming Interface
CEP	Código de Endereçamento Postal
CPF	Cadastro de Pessoas Físicas
COBOL	COmmon Business Oriented Language
DAO	Data Access Object
DNE	Diretório Nacional de Endereços
ESB	Enterprise Service BUS
FIC	Formulário de Informações Cadastrais
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
ISO	International Organization for Standardization
JDBC	Java Database Connectivity
QI	Quadra Interna
SERPRO	Serviço Federal de Processamento de Dados
SIAPE	Sistema de Administração de Pessoal do Governo Federal
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
STF	Supremo Tribunal Federal
TI	Tecnologia da Informação
UDDI	Universal Description Discovery and Integration
URL	Uniform Resource Locator
WSDL	Web Services Description Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Sistemas de Informação tornaram-se uma realidade cada vez mais constante no dia a dia das pessoas. Sistemas, sobretudo web, tem sido uma boa alternativa para quem busca portabilidade, competitividade no mercado, adaptação de novas necessidades e tecnologias. Com o aumento dessa demanda surge também a necessidade de uma maior interação entre esses sistemas, tornando-os mais eficientes e utilizáveis.

A forma mais lógica de pensar é: se um sistema opera bem e é útil, então por que não integrá-lo a outro sistema para utilizar seus serviços em vez de reescrevê-lo? Um exemplo prático seria ter um sistema altamente complexo e eficiente feito em COBOL [1] e um novo em Java que precisa dos dados que o antigo é capaz de fornecer.

Dessa forma, a integração de sistemas passa a ser uma alternativa viável em diversas empresas e órgãos, e na Prefeitura Municipal de Palmas não é diferente, o órgão possui vários sistemas que utilizam diferentes linguagens de programação e que precisam trocar informações constantemente.

Este trabalho pretende demonstrar uma forma de integração entre sistemas utilizando o conceito de arquitetura orientada a serviço (SOA) e a tecnologia WebService em sistemas da Prefeitura Municipal de Palmas.

1.1. TEMA DA PESQUISA

O tema deste trabalho de conclusão de curso é “Interoperabilidade de sistema utilizando arquitetura orientada a serviço e WebService”.

1.1.1 Delimitação do Tema

Existem diversas definições para o termo interoperabilidade. Uma delas citada no site do Governo eletrônico (apud ISO) é:

[1]COBOL – é uma linguagem de programação de Terceira Geração. Este nome é a sigla de COmmon Business Oriented Language (Linguagem Orientada aos Negócios).

> Habilidade de dois ou mais sistemas (computadores, meios de comunicação, redes, software e outros componentes de tecnologia da informação) de interagir e de intercambiar dados de acordo com um método definido, de forma a obter os resultados esperados (ISO).

Em conformidade Rodriguez e Ferrante (2000, pg. 274) define que, “A interoperabilidade é a capacidade de um sistema compartilhar dados e recursos com outros sistemas (*hardware/software*). [...]”.

A Interoperabilidade entre sistemas é um assunto que vem sendo discutido por muitas empresas. Com o surgimento de tantos sistemas e sites na Internet surge também, a necessidade da troca de dados entre eles, ou seja, troca de informações entre sistemas de plataformas distintas.

O presente trabalho visa apresentar a tecnologias WebService como solução para a integração de sistemas.

1.2 PROBLEMA

Como realizar um intercâmbio coerente de informação, entre sistemas que utilizam linguagem de programação e plataformas distintas, evitando sua redundância e facilitando a manutenção de sistemas, sem comprometimento de suas funcionalidades?

1.3 HIPÓTESE

Tal problema se agrava principalmente pela falta de um padrão que permita essa comunicação de forma "direta"; e uma possível solução para o problema apresentado seria utilizar-se de uma tecnologia que disponibilize as informações de um sistema, independentemente de linguagem de programação ou plataforma, para facilitar a reutilização de sistemas legados.

1.4 OBJETIVOS

1.4.1 Objetivo Geral

Apresentar um conjunto de serviços web fundamentados nos princípios da arquitetura orientada a serviço, como uma solução para garantir a interoperabilidade de sistemas.

1.4.2 Objetivos Específicos

- Garantir a interoperabilidade entre sistemas da Prefeitura Municipal de Palmas;
- Aplicar os conceitos SOA para definição dos serviços;
- Definir e criar serviços web de acordo com a arquitetura SOA;
- Garantir a segurança dos dados oferecidos pelo WebService.

1.5 JUSTIFICATIVA

A integração de sistemas é um fator relevante nas empresas e órgãos que possuem sistemas legados. Geralmente esses sistemas funcionam bem e o ideal seria adotar estratégias e conceitos que facilitem a utilização desse sistema já que na maioria das vezes, não é viável para essas empresas e órgãos refazerem os sistemas.

Um cenário atual do problema apresentado neste trabalho ocorre na Prefeitura Municipal de Palmas, onde existem sistemas que precisam trocar informações freqüentemente. Em algumas situações dois ou mais sistemas precisam de uma mesma informação que está contida em outro sistema.

Caso esta troca de informações seja realizada diretamente entre os sistemas, haveria uma repetição dos mesmos códigos em cada um dos sistemas, e ainda, o agravante que, cada vez que houvesse uma alteração no sistema servidor, haveria também a necessidade da alteração dos códigos dos sistemas clientes

Um conceito que vem sendo adotado atualmente por muitas empresas para evitar que ocorra essa redundância de informação é a implementação de SOA.

Segundo Sampaio (2006, pg. 14), "[...] SOA é um novo paradigma de desenvolvimento de aplicações cujo objetivo é criar módulos funcionais chamados de serviços, com baixo acoplamento e permitindo a reutilização de códigos".

Em resumo, SOA orienta os desenvolvedores de *software* na criação de sistemas que não dependam de tecnologia e que sejam flexíveis a mudanças.

Para a implementação de SOA é necessário escolher uma tecnologia que tenha um baixo nível de acoplamento, ou seja, uma tecnologia que disponibilize os serviços independentes de plataforma e que facilite a integração de sistemas distintos.

A tecnologia WebService, por ser uma tecnologia relativamente conhecida, é utilizada para a implementação de SOA (MARZULLO, 2009).

WebService é uma tecnologia que possibilita o desenvolvimento de *softwares* ou componentes de *software* capazes de interagir, seja enviando ou recebendo informações com outros *softwares*, não importando a linguagem de programação em que estes foram desenvolvidos, o sistema operacional em que rodam e o *hardware* que é utilizado (GOMES, 2009).

Isto é possível porque os WebServices utilizam o protocolo padrão de WebService SOAP (Simple Object Access Protocol) para a transmissão de dados. SOAP é um protocolo especificado pelo W3C (World Wide Web Consortium) para troca de informações estruturada em ambientes descentralizados e distribuídos. Ele utiliza a linguagem XML (Extensible Markup Language) para a criação das mensagens e o protocolo HTTP (Hypertext Transfer Protocol) como infraestrutura de transmissão (MARZULLO, 2009). O que facilita a disponibilização de serviços web na internet já que o protocolo HTTP é o protocolo padrão da internet.

1.6 ESTRUTURA DO TRABALHO

Este trabalho está organizado nos seguintes capítulos.

No Capítulo 2 serão apresentados os conceitos de SOA e a infraestrutura necessária para a adoção desta arquitetura.

No Capítulo 3 será apresenta uma visão geral da tecnologia WebService e os componentes essenciais para a utilização da mesma.

No Capítulo 4 serão apresentadas quais empresas já adotaram a tecnologia WebService e obteve êxito.

No Capítulo 5 será apresentado como foi implantando SOA e WebService na Prefeitura Municipal de Palmas.

No Capítulo 6 será apresentada a conclusão deste trabalho.

2 SOA

De acordo com Hurwitz, Bloor, Kaufman e Halper (2009, pg. 78), “[...] SOA é uma arquitetura que convenientemente permite que TI se alinhe com os processos de negócio [...].”

Da mesma forma, Marzullo (2009, pg. 123), “Arquiteturas orientadas a serviços (SOA) representam uma nova abordagem para utilização dos recursos de TI em apoio ao negócio da organização [...]”.

SOA tem a finalidade de unir o setor de tecnologia da informação (TI) com o setor de negócio de uma organização. A idéia é desenvolver serviços com base nas regras de negócio de uma organização utilizando os recursos de TI, ou seja, transformar as funcionalidades de um sistema em serviços que possa ser disponibilizado na internet ou na intranet de uma organização.

SOA também pode ser utilizada como uma estratégia competitiva, que tem como objetivo oferecer visões abstratas de como o arquiteto deve modelar soluções de *software* que possibilite a criação de sistemas desacoplados de tecnologias, sistemas que consigam acompanhar as mudanças que podem ocorrer nas regras de negócio de uma organização (Marzullo, 2009).

Outra vantagem de utilizar SOA é a facilidade de reutilizar códigos existentes, transformando as funcionalidades de um sistema existente em serviços web.

> SOA se trata de reutilização: pegar o que você tem e estruturá-lo para permitir que você não apenas continue a usá-lo, mas que continue a usá-lo seguramente sabendo que alterações no futuro serão simples, diretas, seguras e rápidas. SOA é mesmo uma jornada: não pode ser feita da noite para o dia. Mas empresas podem começar SOA agora e se beneficiar agora. Na verdade, SOA torna um negócio mais flexível - e TI mais confiável, sustentável, extensível, gerenciável e responsável. (HURWITZ, BLOOR, KAUFMAN E HALPER, 2009, pg. 15)

Para que uma organização comece a utilizar os conceitos da arquitetura orientada a serviço, alguns elementos relevantes que fazem parte de SOA devem ser considerados. Marzullo (2009) define esses elementos como Modelo Operacional Triangular (Figura 1) que é composto por três elementos:

- **Provedor:** é considerado o dono do serviço. O provedor é responsável por disponibilizar o serviço com toda infraestrutura necessária para que o cliente que precise do serviço possa acessá-lo;

- **Consumidor:** é o cliente do serviço. Pode ser representado por uma pessoa, uma organização, uma máquina ou um componente de *software*. Em resumo, o consumidor representa aquele ou aquilo que localiza um serviço, entende o seu protocolo de operação e se utiliza desse protocolo para executá-lo;
- **Registro:** é responsável por gerenciar os repositórios que armazenam informações sobre os serviços oferecidos por uma organização. Um registro SOA pode ser entendido com um catálogo eletrônico onde ficam armazenadas as informações que descrevem o que cada componente de *software* faz, para que desenvolvedores de *software* e analistas de negócios consigam localizar os serviços de que precisam (HURWITZ, BLOOR, KAUFMAN, HALPER, 2009).

Figura 1: Modelo operacional triangular SOA
Fonte: Marzullo, 2009.

O Modelo Operacional Triangular definido por Marzullo pode ser entendido da seguinte forma: uma organização (Provedor) cria um serviço conforme sua regra de negócio, depois de concluído o desenvolvimento do serviço, ela faz um registro desse serviço e publica para que um cliente (consumidor) possa localizar esse serviço e utilizá-lo.

Além dos elementos definidos no modelo operacional do Marzullo existe outro componente que pode ou não fazer parte da infraestrutura SOA, que é o barramento de serviço (ESB), responsável por garantir a troca de mensagens entre os componentes de uma implementação SOA.

2.1 BARRAMENTO DE SERVIÇOS

Um ESB representa uma orientação conceitual de como um conjunto de aplicações diferentes e, na maioria das vezes, desenvolvidas em tecnologias diferentes, devem se comunicar (MARZULLO, 2009).

Hurwitz, Bloor, Kaufman, Halper (2009, pg.148) define o ESB da seguinte forma:

> [...] É uma camada de software que permite que programas diferentes conversem uns com os outros em uma maneira padronizada, mesmo se eles estiverem em programas diferentes, rodando em sistemas operacionais diferentes e escritos em linguagens de programação diferentes. [...].

O objetivo de um ESB é garantir a comunicação dos componentes que fazem parte da infraestrutura SOA.

> [...] O barramento de serviços deve garantir que, por exemplo, aplicações desenvolvidas em Java consigam interagir com aplicações construídas em .Net, ao mesmo tempo que aplicações Web Services desenvolvidos em Apache Axis possam comunicar com aplicações desenvolvidas em JEE (MARZULLO, 2009, pg. 134).

No Modelo Operacional Triangular apresentado anteriormente o ESB ficaria entre o provedor e o consumidor gerenciando as mensagens trocadas entre eles.

Quando uma organização decide acrescentar o ESB na sua infraestrutura SOA é necessário que este ESB seja desenvolvido de maneira que o modelo de integração seja único dentro do ambiente de negócio da organização, assim garantindo que todas as aplicações, legadas ou não, estejam interligadas e que possam se comunicar dentro de um protocolo padronizado de troca de mensagens (MARZULLO, 2009).

A Figura 2 representa um modelo de barramento de serviço que possui tecnologias usadas na indústria em projetos reais (MARZULLO, 2009). Este ESB precisa garantir a troca de informações entre aplicações desenvolvidas em tecnologias diferentes. Como pode ser visto este ESB possui aplicações na linguagem Java e .Net, sistemas de legados, WebService entre outros. Para gerenciar a comunicação entre essas aplicações é preciso ter um mecanismo de integração, um mecanismo de conversão e transformação, um roteamento de mensagem e ferramentas de controle e gerência.

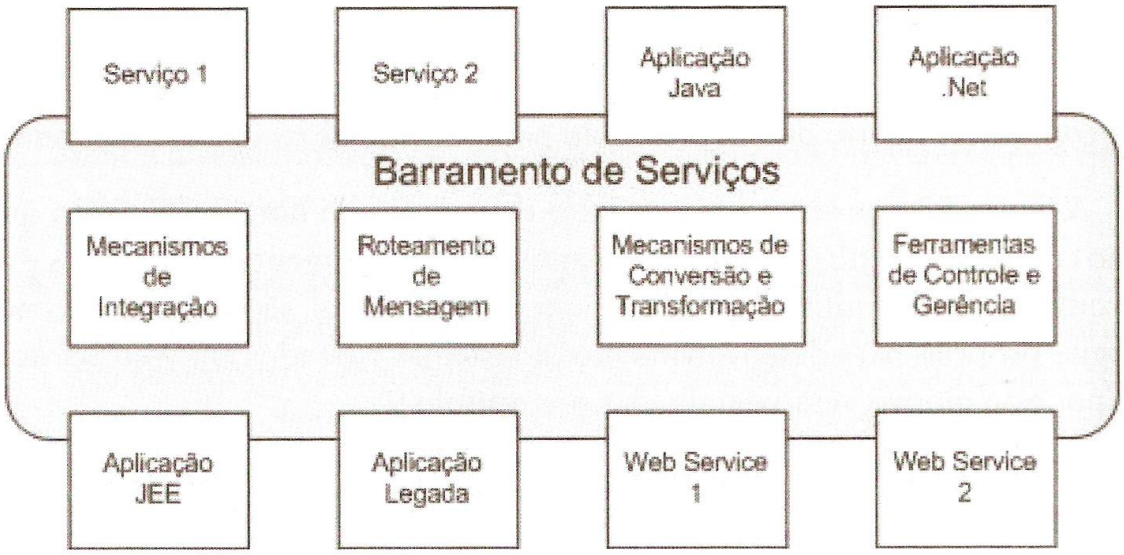

Figura 2: Barramento de serviço.
Fonte: Marzullo, 2009.

3 WEBSERVICE

WebService é uma tecnologia utilizada para a integração de sistemas em ambientes heterogêneos (GOMES, 2009). Com WebService é possível realizar a comunicação de duas aplicações que utilizam linguagem diferente.

Segundo Marzullo (2009, pg. 150), “[...] um Web Service representa a materialização da idéia de um serviço que é disponibilizado na Internet e que pode ser acessado em qualquer lugar do planeta.”

Isto é possível porque os WebServices têm uma infraestrutura leve e desacoplada de plataforma, o que facilita a integração de diferentes tecnologias (MARZULLO, 2009).

A tecnologia WebService foi criada baseada na linguagem XML. Ela segue um padrão estabelecido pelo consorcio W3C o que facilita a integração com várias linguagens de programação que sigam esse padrão (GONÇALVES, 2008). Outra vantagem dessa tecnologia é que ela envia mensagens XML através do protocolo padrão da Internet.

Para implementar um WebService existem alguns componentes que precisam fazer parte da sua arquitetura: O protocolo SOAP que é responsável pela comunicação do cliente com o provedor de serviço web; o arquivo WSDL que descreve o serviço oferecido pelo WebService; o registro UDDI (Universal Description Discovery and Integration) que disponibiliza informações e a localização de um WebService; o cliente que pode ser entendido como o *software* que consumirá o WebService; o provedor de WebServices que é um servidor de aplicações web onde ficará armazenado o WebService.

A Figura 3 representa uma arquitetura para WebServices criada pelo W3C. Esta arquitetura demonstra os componentes que fazem parte dessa estrutura e a seqüência, em que ocorre a chamada de um WebService (GOMES, 2009).

1. **Registra e publica o web service:** quando é desenvolvido um WebService para que o serviço fique disponível para um cliente é necessário registrá-lo e publicá-lo em um provedor de serviço web. Para isso é enviado o arquivo WSDL a um provedor de serviço web conhecido como registro UDDI;
2. **Obtém informações sobre o web service:** quando um desenvolvedor de sistema precisa de um serviço, a primeira etapa para a utilização desse

serviço é localizá-lo, para isso é necessário que ele faça uma pesquisa no registro UDDI;

3. **Efetua *download* do WSDL:** depois que o desenvolvedor localiza o serviço que precisa, para que ele utilize esse serviço é necessário que ele faça o *download* do arquivo WSDL que pode ser baixado tanto do registro UDDI quanto diretamente do provedor onde está o WebService. Após o *download* do arquivo WSDL o WebService pode ser utilizado.
4. **Envia solicitação XML:** o cliente começa a utilizar o serviço fazendo uma chamada ao WebService, utilizando o protocolo SOAP que tem o formato de um arquivo XML;
5. **Recebe resposta XML:** o provedor de serviço web recebe a chamada do cliente, efetua um determinado processamento e envia um resultado utilizando o mesmo protocolo do cliente o protocolo SOAP.

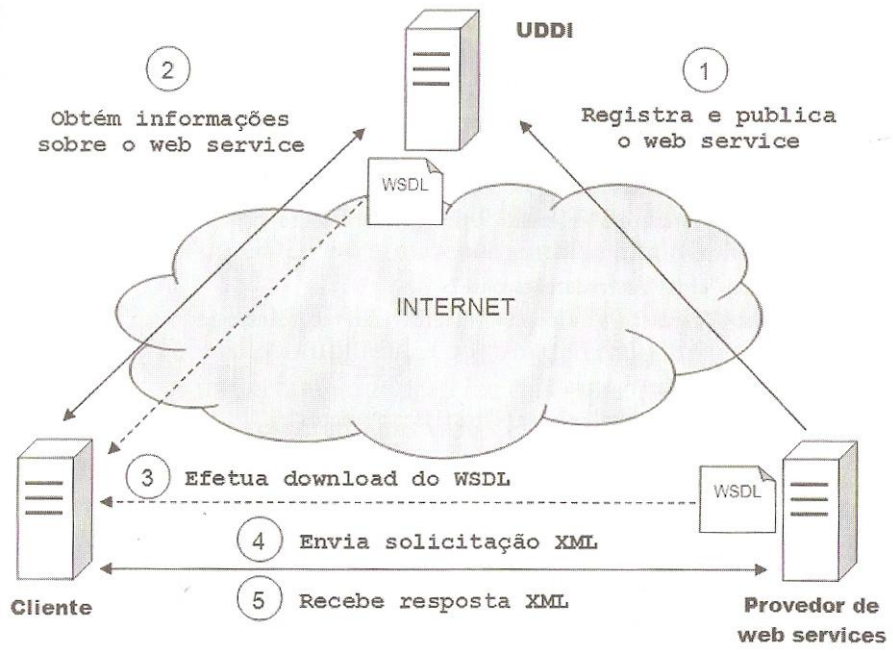

Figura 3: Arquitetura para web services SOAP criada pela W3C.
Fonte: Gomes, 2009.

3.1 SOAP

SOAP é um protocolo utilizado pelos WebServices para enviar e receber mensagens. Gonçalves (2008, pg. 374) define este protocolo da seguinte forma:

> O SOAP (Simple Object Access Protocol) é um protocolo para intercâmbio de mensagens entre programas de computador. Este protocolo surgiu em 1998, apresentado ao W3C pelas empresas DevelopMentor, Microsoft e UserLand Software. Sua especificação ocorreu em dezembro de 1999.

O consórcio W3C define o protocolo SOAP como um protocolo padrão para transmissão de dados da arquitetura WebService. SOAP é um protocolo baseado na linguagem XML é utiliza o protocolo HTTP para transmissão de dados na Internet (GOMES, 2009).

Para transmitir as mensagens de um WebService o protocolo SOAP cria um envelope XML. Este envelope consisti de um corpo e um cabeçalho que pode ser opcional. No corpo contém a aplicação, ou seja, o serviço oferecido pelo WebService e no cabeçalho geralmente contém os dados de segurança (GONÇALVES, 2008).

Marzullo (2009) divide o protocolo SOAP nos seguintes elementos (Figura 4):

- **O Envelope** - é um elemento obrigatório, ele é a raiz da mensagem e define como o documento XML é transformado em uma mensagem SOAP. Nele é determinado o *namespace* da mensagem SOAP que é definido pelo W3C e o tipo de codificação dos dados transmitidos pelo envelope;
- **O Cabeçalho (header) -** é um elemento não obrigatório utilizado geralmente quando necessita de autenticação do usuário para acessar a mensagem enviada;
- **O Corpo (Body)** - é um elemento obrigatório responsável por armazenar o documento que será transmitido e a definição das possíveis falhas que podem ocorrer durante a transmissão;
- **A falha (Fault)** - é um elemento opcional responsável por enviar códigos e mensagens de erro que podem acontecer durante a transmissão do documento e da recepção pelo WebService. Este elemento fica no corpo do envelope e deve ser único.

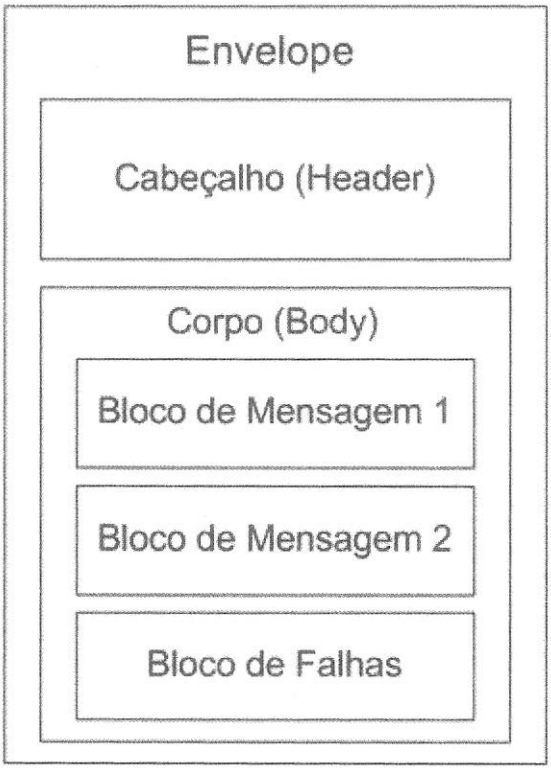

Figura 4: Estrutura de um envelope SOAP.
Fonte: Marzullo, 2009.

Para um melhor entendimento de como funciona a troca de mensagens utilizando o protocolo SOAP. Veja um exemplo de uma mensagem SOAP Resquest (Figura 5) e uma mensagem SOAP Response (Figura 6) de um serviço de login.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
03     <S:Header/>
04     <S:Body>
05        <ns2:login  xmlns:ns2="http://autenticacao.org/">
06          <cpf>…</cpf>
07          <matricula>…</matricula>
08          <senha>…</senha>
09        <ns2:login>
10    </S:Body>
11  </S:Envelope>
```

Figura 5: Mensagem SOAP Request.
Fonte: Autoria própria.

No documento SOAP Request foi definido o elemento Envelope e nele foi especificado o *namespace* “http://schemas.xmlsoap.org/soap/envelope/”, este *namespace* é um padrão definido pelo W3C, dentro do Envelope foram incluídos os elementos Cabeçalho (Header) e Corpo (Body). No elemento Corpo foram declaradas as informações que precisam ser enviada para utilizar o serviço. No caso

desse exemplo que é um serviço de login, é necessário o envio do CPF (Cadastro de Pessoas Físicas), matrícula e senha do usuário para a validação do login.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
03    <S:Header/>
04    <S:Body>
05       <ns2:loginResponse  xmlns:ns2="http://autenticacao.org/">
06         <return>…</return>
07       <ns2:loginResponse>
08    </S:Body>
09 </S:Envelope>
```

Figura 6: Mensagem SOAP Response.
Fonte: Autoria própria.

O documento SOAP Response possui a mesma estrutura do SOAP Request, a única diferença ocorre dentro do elemento Corpo (Body), que agora armazena o resultado que o serviço enviou para o cliente.

3.2 WSDL

WSDL é um arquivo XML que descreve os serviços oferecidos por um WebService. Nele contêm o nome do serviço, os tipos de dados que precisam ser enviados para o serviço e os tipos de dados que será retornado pelo serviço.

Segundo Gomes (2009, pg. 16), “WSDL (Web Services Description Language): é um arquivo do tipo XML, cuja finalidade é descrever detalhadamente um web service. Essa descrição especifica as operações que compõem o web service e define de forma clara como deve ser o formato de entrada e saída de cada operação [...]”.

Em conformidade Gonçalves (2008, pg. 373) define que, “[...] WSDL é uma estrutura XML usada pra descrever um serviço Web (Web Service). Define como o serviço Web é acessado, as operações que executa, como são passada as mensagens, e a estrutura destas mensagens. [...]”.

Quando é desenvolvido um WebService a criação do arquivo WSDL é essencial, pois nele é especificado o serviço oferecido pelo WebService, as informações que precisam ser enviados para o serviço e as informações que serão retornados pelo serviço. Este arquivo pode ser considerado a identidade de um WebService, ou seja, o documento de identificação do serviço.

3.2.1 Estrutura de um arquivo WSDL

Um documento WSDL (Figura 7) é simplesmente um conjunto de definições. Ele inicia com um elemento raiz (definitions) e dentro deste elemento é definido mais seis elementos principais que são (W3C, 2001):

- **type (linha 11 a 16)** - determina os tipos de dados usados para descrever as mensagens transmitidas;
- **message (linha 17 a 19)** - representa uma definição abstrata dos dados a serem transmitidos. Uma mensagem consiste de partes lógicas, cada uma das quais está associada a uma definição dentro de algum tipo de sistema;
- **portType (linha 26 a 35)** - é um conjunto de operações abstratas. Cada operação (linha 27 a 34) refere-se a uma mensagem de entrada e saída de mensagens;
- **binding (linha 36 a 45)** - especifica o protocolo de comunicação e os formatos de dados para as operações e mensagens definidas pelo elemento portType;
- **Port (linha 47 a 49)** - especifica o endereço onde está localizado o serviço;
- **Service (linha 46 a 50)** - é utilizado para agregar um conjunto de portas relacionadas.

```
01  <?xml version='1.0' encoding='UTF-8'?>
02   <definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
03    wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
04   xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
05   xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
06   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
07   xmlns:tns="http://autenticacao.org/"
08   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
09   xmlns="http://schemas.xmlsoap.org/wsdl/"
10   targetNamespace="http://autenticacao.org/" name="AutenticacaoService">
11  <types>
12    <xsd:schema>
13      <xsd:import namespace="http://autenticacao.org/" schemaLocation=
14          "http://localhost:8080/webservice/AutenticacaoService?xsd=1" />
15    </xsd:schema>
16  </types>
17  <message name="login">
18    <part name="parameters" element="tns:login" />
19  </message>
20  <message name="loginResponse">
21    <part name="parameters" element="tns:loginResponse" />
```

```
22    </message>
23    <message name="Exception">
24      <part name="fault" element="tns:Exception" />
25    </message>
26    <portType name="Autenticacao">
27        <operation name="login">
28            <input wsam:Action="http://autenticacao.org/Autenticacao/loginRequest"
29              message="tns:login" />
30          <output wsam:Action="http://autenticacao.org/Autenticacao/loginResponse"
31              message="tns:loginResponse" />
32          <fault message="tns:Exception" name="Exception"
33              wsam:Action="http://autenticacao.org/Autenticacao/login/Fault/Exception" />
34      </operation>
35    </portType>
36    <binding name="AutenticacaoPortBinding" type="tns:Autenticacao">
37        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
38          style="document" />
39        <operation name="login">
40          <soap:operation soapAction="" />
41          <input><soap:body use="literal" /></input>
42          <output><soap:body use="literal" /></output>
43          <fault name="Exception"><soap:fault name="Exception" use="literal" /></fault>
44        </operation>
45     </binding>
46    <service name="AutenticacaoService">
47      <port name="AutenticacaoPort" binding="tns:AutenticacaoPortBinding">
48        <soap:address location="http://localhost:8080/webservice/AutenticacaoService" />
49      </port>
50    </service>
51  </definitions>
```

Figura 7: Estrutura de um arquivo WSDL
Fonte: Autoria própria

3.3 UDDI

O WebService foi criado, agora é necessário disponibilizar esse serviço na Internet, para que o cliente que precisa desse serviço possa encontrá-lo e utilizá-lo. Para isso foi criado o registro UDDI que tem o objetivo de facilitar a localização de um serviço.

Segundo Marzullo (2009, pg. 180), “O Universal Description, Discovery, and Integration (UDDI) é um padrão para publicação e localização de Web Services pelo uso de consultas (queries) baseada em mensagens SOAP e documento XML. [...]”.

O registro UDDI tem a finalidade de representar os dados de um serviço web. Um registro pode ser usado tanto em uma rede pública quanto em uma infraestrutura interna de uma organização, ele oferece um mecanismo baseado em padrões para classificar, catalogar e gerenciar um WebService, de modo que possa ser descoberto e consumido por outras aplicações (UDDI).

Para um melhor entendimento de como funciona o registro UDDI, Reckziegel (2006) compara o registro UDDI com uma lista telefônica, da seguinte maneira:

- **Páginas Brancas:** contêm informações sobre nomes, endereços, números de telefone, além de outras informações sobre os fornecedores do serviço;
- **Páginas Amarelas:** contêm listagens comerciais baseadas nos tipos desses negócios, de maneira organizada por categoria específica ou regiões demográficas;
- **Páginas Verdes:** são usadas para indicar os serviços oferecidos por cada negócio, incluindo todas as informações técnicas envolvidas na interação com o serviço. Resumindo, explica como fazer a comunicação com eles.

3.3.1 Estrutura de um registro UDDI

Segundo Marzullo (2009), o registro UDDI possui um modelo de dados composto pelos seguintes elementos (Figura 8):

- **BusinessService** – é um elemento utilizado para descrever os serviços oferecidos pela organização. Este elemento fica dentro do elemento BusinessEntity;
- **BusinessEntity** – possui informações básicas do negócio como: contato, categoria, identificação, descrição e relacionamento com outros negócios;
- **PublisherAssertion** – é utilizado para estabelecer uma relação entre dois elementos BusinessEntity;
- **BindingTemplate** – este elemento fica dentro do elemento BusinessService e nele possui informações do ponto de acesso do serviço e descrições técnicas como interface ou API;
- **tModel** – este elemento possui a especificação do serviço, o arquivo WSDL e determina como deve ser invocado o serviço.

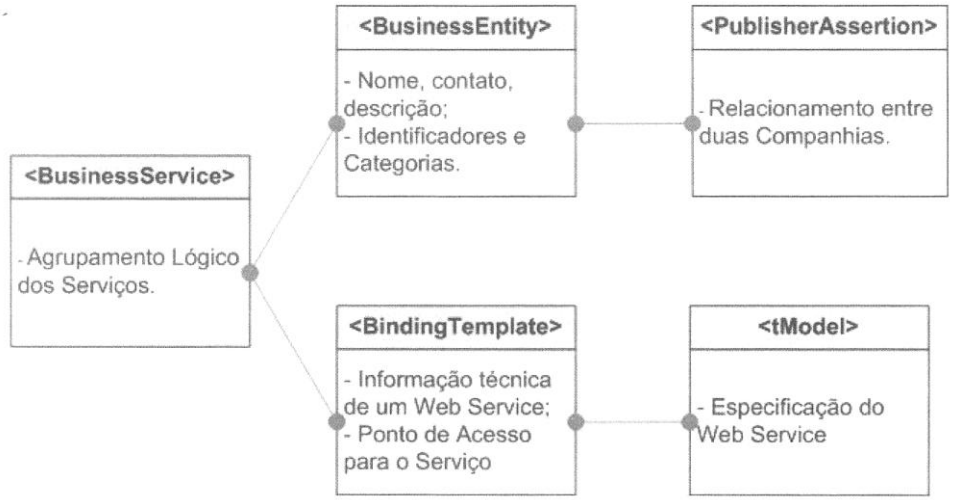

Figura 8: Modelo de dados simplificado de um UDDI.
Fonte: Marzullo, 2009.

3.4 WEBSERVICE SEGURO

A criação de um WebService seguro é um fator relevante para uma organização, pois normalmente um serviço web recebe e envia informações confidenciais.

Além de utilizar a segurança para garantir que os dados de um serviço não sejam acessados com facilidade em uma rede, existe também o serviço privado que só pode ser acessado por um cliente que tenha autorização para usar o serviço.

Quando uma organização decide disponibilizar um serviço web é fundamental que ela decida também a tecnologia que será utilizada para garantir a segurança das informações que serão disponibilizadas.

Para a tecnologia WebService existe o framework WS-Securyt que é utilizado no protocolo SOAP.

3.4.1 WS-Securyt

Segundo Sosnoski (2009), WS-Security é um padrão de segurança que é adicionado em uma mensagem SOAP. Ele insere informações de segurança no cabeçalho de uma mensagem SOAP.

Em conformidade Marzullo (2009) define que, WS-Securyt é um padrão que atua diretamente no protocolo SOAP e possui um conjunto de mecanismo que ajudam a garantir a segurança da comunicação em arquitetura orientada a serviço que utilizam WebService. Além disso, é uma técnica que resolve três questões de segurança essenciais: autenticação e autorização, integridade da mensagem e criptografia de mensagens.

O WS-Securyt é composto por um conjunto de outros padrões que são (MARZULLO, 2009):

- **WS-SecureConversation** – é responsável por criar e controlar contextos de segurança. A idéia é criar zonas de segurança que monitorem a comunicação e o compartilhamento entre serviços alocados nessas zonas;
- **WS-Federation** – é responsável por criar mecanismo que permitem que as zonas de seguranças, especificada pelo WS-SecureConversation, possam negociar níveis e permissões de acesso. Permite a troca ou delegação de autoridades entre serviços;
- **WS-Authorization** – é responsável por definir os perfis de acesso para provedores e consumidores;
- **WS-Policy** – é responsável por representar um conjunto de especificações que definem políticas de segurança para comunicação entre endpoints. Isso inclui a definição dos níveis de segurança para grupos de serviços, algoritmos de criptografia, regras de autenticação e autorização etc;
- **WS-Trust** – é responsável por criar meios para a construção de troca de mensagens SOAP seguras;
- **WS-Privacy** – é responsável por definir como WebService devem implementar políticas de segurança definidas pela organização.

4 CASOS DE SUCESSO

4.1 WEBSERVICES DO CORREIOS

Os Correios desenvolveram um serviço web que faz o Cálculo Remoto de Preços e Prazos de encomendas. Este serviço foi criado para os clientes SEDEX, e-SEDEX E PAC que precisam realizar o cálculo do preço e prazo de entrega de uma encomenda realizada em websites.

Este serviço pode ser utilizado tanto por um cliente que possui contrato com os Correios como por um cliente que não possui, a diferença é que para o cliente que não possui contrato, os preços apresentados serão os mesmo apresentados no balcão de uma agência dos correios (CORREIOS).

4.2 E-STF WEBSERVICES PROCESSO ELETRÔNICO

O e-STF é um serviço desenvolvido pelo Supremo Tribunal Federal (STF) como meio de tramitação de processos judiciais, comunicação de atos e transmissão de peças processuais instituído pela Resolução nº 344, de 25 de maio de 2007, nos termos da Lei nº 11.419, de 19 de dezembro de 2006.

O projeto e-STF criou o serviço web de Processo Eletrônico com o objetivo de inovar e promover a melhoria do processo judiciário ao Supremo Tribunal Federal e aos Tribunais associados. Com esse serviço os tribunais brasileiros podem enviar os processos eletronicamente de forma imediata (E-STF).

4.3 AUTOCEP WEBSERVICE

O WebService AutoCep é um serviço criado para o preenchimento de endereço. Esse serviço disponibiliza o endereço completo de um local, apenas com o preenchimento do CEP (Código de Endereçamento Postal) do local.

A base de dados desse serviço é atualizada freqüentemente com a última versão do DNE[2] disponibilizado pelos Correios.

O AutoCep é um serviço privado e qualquer empresa pode contratar. A cobrança desse serviço é realizada conforme a quantidade de consultas realizadas no servidor do serviço (AUTOCEP).

4.4 WEBSERVICE SIAPE

O WebService SIAPE tem o objetivo de possibilitar o acesso, em tempo real, para os sistemas autorizados, aos dados dos servidores ativos, aposentados e pensionistas, que recebem seus pagamentos pelo SIAPE (Sistema de administração de pessoal do governo federal).

Este serviço foi desenvolvido pelo SERPRO (Serviço Federal de Processamento de Dados) e o Departamento de Administração de Sistemas de Recursos Humanos, para atender as necessidades de outros sistemas do Governo Federal de integração com a base de dados do SIAPE (SIAPE, 2010).

[2] DNE – Diretório Nacional de Endereços é um banco de dados de abrangência nacional constituído de elementos de endereçamento até nível de secção de logradouro e Códigos de Enderçamento Postal (CEP).

5 IMPANTANDO SOA E WEBSERVICE NA PREFEITURA MUNICIPAL DE PALMAS

5.1 TECNOLOGIAS UTILIZADAS NA IMPLANTAÇÃO

As tecnologias utilizadas no desenvolvimento do WebService foram a linguagem de programação Java por ser uma linguagem multiplataforma, o *framework* JAX-WS por simplificar a complexidade do protocolo SOAP e por trabalhar com o protocolo HTTP, o NetBeans que é uma *Integrated Development Environment* (IDE) ou Ambiente Integrado de Desenvolvimento que facilita a criação de aplicações web em Java e o Apache Tomcat que é um servidor Java utilizado para a publicação dos serviços.

Já no desenvolvimento do cliente que está usando esse WebService foi utilizada a linguagem de programação Python e o *framework* Django que são tecnologias usadas no desenvolvimento dos sistemas internos da Prefeitura Municipal de Palmas e a biblioteca SUDS que é um cliente SOAP em Python para consumir WebService.

5.1.1 Java

Java é uma linguagem de programação orientada a objetos multiplataforma que possui recursos que a torna ideal para desenvolvimentos de programas para a Internet. Segundo Deitel (2007, pg. 3), “Java tornou-se a linguagem preferida para implementar aplicativos baseados na Internet e software para dispositivos que se comunicam em uma rede”.

Uma das vantagens dessa linguagem é que ela possui *Java Application Programming Interface* (também conhecida como API do Java ou biblioteca de classes Java) que pode ser utilizada para o desenvolvimento de vários tipos de aplicações em Java assim como para a criação de WebService.

> [...] A API do Java fornece uma rica coleção de classes predefinidas que contém métodos para realizar cálculos matemáticos comuns, manipulações de string, manipulações de caractere, operação de entrada/saída, operação de banco de dados, operações de rede, processamento de arquivo, verificação de erros e muitas outras operações úteis [...] (DEITEL, 2007, pg. 165).

5.1.2 NetBeans

NetBeans é uma IDE para desenvolvedores de *software*. Ela possui ferramentas necessárias para a criação de sistemas desktop, aplicações web e mobile com a plataforma Java, assim como nas linguagens C, C++, PHP, JavaScript, Groovy e Ruby.

> O NetBeans IDE é um ambiente de desenvolvimento - uma ferramenta para programadores, que permite escrever, compilar, depurar e instalar programas. O IDE é completamente escrito em Java, mas pode suportar qualquer linguagem de programação. Existe também um grande número de módulos para estender as funcionalidades do IDE NetBeans. O NetBeans IDE é um produto livre, sem restrições à sua forma de utilização (NETBEANS).

5.1.3 JAX-WS

JAX-WS é uma API do Java para WebService baseada em XML que facilita o desenvolvimento de serviços web utilizando a linguagem Java. Para Smart (2006), “JAX-WS (ex JAX-RPC) é a resposta da Sun para a questão de como desenvolver WebService com facilidade em Java”.

5.1.4 Tomcat

O Apache Tomcat é um servidor web para aplicações Java. É um *software* desenvolvido em código aberto (open source) das tecnologias Java Servlet e JavaServer Pages.

> O Tomcat é uma implementação completamente funcional de servlets e JavaServer Pages (JSP). Ele inclui um servidor Web para que possa ser utilizado como um contêiner teste de terceiros para servlets e JSPs. O Tomcat também pode ser especificado como o handler para solicitações de JSP e servlet recebidas pelos servidores Web populares com o servidor http Apache [...] (DEITEL, 2007, pg. 933 à 934).

O Tomcat pode ser utilizado tanto na implementação como na execução de WebService.

5.1.5 Python

Python é uma linguagem de programação orientada a objetos interpretada e interativa. Python incorpora módulos, exceções, digitações dinâmica, tipos de dados dinâmicos e classes. Possui interfaces para chamadas de sistema e bibliotecas, é extensível em C ou C + ++. Também é usado como uma linguagem de extensão para aplicações que necessitem de uma interface programável. Finalmente, Python é portável: funciona em vários sistemas operacionais como: Linux / Unix, Mac, Windows, Windows NT e OS / 2 (PYTHON).

5.1.6 Django

Django é um *framework* web escrito na linguagem Python que estimula o desenvolvimento de sistemas web rápido.

> O Django é um framework de desenvolvimento web, escrito em Python, que foi criado pelo grupo editorial “The World Company” para a criação da versão web dos seus jornais. Posteriormente, em 2005, foi liberado sob a licença BSD, tornando-se assim um software de código aberto (SANTANA, GALESI, 2010, pg.155).

O *framework* Django foi projetado para lidar com os prazos apertados de uma redação de jornal e ao mesmo tempo atender aos requisitos essenciais para o desenvolvimento de sistemas web. Desta forma, permitindo a construção de aplicações web com rapidez e alto desempenho (DJANGO).

5.1.7 SUDS

SUDS é uma biblioteca baseada em SOAP que permite a execução de chamadas SOAP em Python para realizar a comunicação com um WebService. Suas principais características são: lê o arquivo WSDL em tempo de execução para a codificação e decodificação, não é necessária a criação de classe, fornece um objeto do tipo API, suporta autenticação HTTP e disponibiliza informações básicas do WS-Security (SUDS).

5.2 IDENTIFICANDO OS SERVIÇOS WEB

O conceito de SOA tem como um dos objetivos orientarem os desenvolvedores de *softwares* a transformarem funcionalidades do sistema em serviços.

Utilizando-se desse conceito, foi realizado um estudo na Prefeitura Municipal de Palmas e verificado quais os sistemas que poderiam utilizar-se desse conceito.

Dessa forma, o resultado desse estudo levou à identificação do serviço de endereçamento e do serviço de autenticação.

5.2.1 O Serviço de Endereçamento

A criação do serviço de endereçamento foi identificada devido à dificuldade do cadastro do endereço de Palmas. Em 1997 ocorreu uma mudança na estrutura de endereço de Palmas e as pessoas ainda se confundem no preenchimento do endereço, por vezes colocando o endereço novo e por outras colocando o endereço antigo.

> Lei nº 658, de 19 de junho de 1997: Autoriza o Poder Executivo a alterar as nomenclaturas das quadras e logradouros do Plano Diretor Urbanístico da Capital.

O endereço vigente hoje é o chamado popularmente de novo endereço, no entanto os endereços válidos nos registros de imóveis são os do antigo endereço, mas para alguns, sobretudo os que vieram residir na capital após a mudança do endereçamento, conhecem apenas o novo endereço.

Como existem alguns sistemas na Prefeitura Municipal de Palmas que precisam cadastrar o endereço exato do registro do imóvel foi percebida a necessidade de criar um serviço de endereçamento para acabar com essa confusão de endereços.

Dessa forma, foi utilizado um banco de dados da Secretaria Municipal de Desenvolvimento Urbano e Habitação, que contém os registros dos endereços antigos e seus correspondentes novos, para criação de um serviço de endereçamento, disponibilizando ao usuário os dois endereços para que ele possa consultar o correto e facilitar assim o preenchimento dessa informação nos sistemas.

5.2.2 O Serviço de Autenticação

A Prefeitura Municipal de Palmas desenvolve sistemas internos, em que os usuários são seus funcionários. Cada vez que é desenvolvido um sistema é criado um login e uma senha para este funcionário. Como um usuário pode ter acesso a vários sistemas e pode ter vários logins e senhas diferentes, pode acontecer que surja uma dificuldade na memorização desses dados, principalmente se o usuário não utilizar os sistemas com freqüência ou deixar de utilizar em decorrência de uma ausência temporária do ambiente de trabalho, como por exemplo, em seu período de férias.

Uma solução para evitar a criação de vários usuários e senha para o mesmo funcionário é criar um serviço de autenticação. Este serviço poderá ser utilizado por qualquer sistema e caso o usuário esqueça sua senha terá que modificar uma única vez.

Para a definição do serviço de autenticação, foi estabelecido um padrão de login, utilizando os dados cadastrais do funcionário que é definido no sistema de recursos humanos. Quando um funcionário é cadastrado neste sistema é criado um login e uma senha para ele que será a mesma em todos os sistemas.

5.3 CRIANDO O PROJETO

Embora tenha encontrado dois serviços em que poderia utilizar-se de WebService (endereçamento e autenticação), apenas um WebService, o de endereçamento para exemplificar a implantação de SOA, foi implementado. O modo de desenvolvimento é o mesmo para os dois.

Nos próximos tópicos será mostrado passo a passo o desenvolvimento do WebService Endereçamento.

5.3.1 Criando projeto no NetBeans

Inicialmente foi criado um projeto Java do tipo aplicação web na IDE NetBeans, através do menu Arquivo → Novo Projeto. O primeiro passo é selecionar

a categoria e o tipo de projeto, foi selecionado Java Web e Aplicação web respectivamente e clicado no botão “Próximo” (Figura 9).

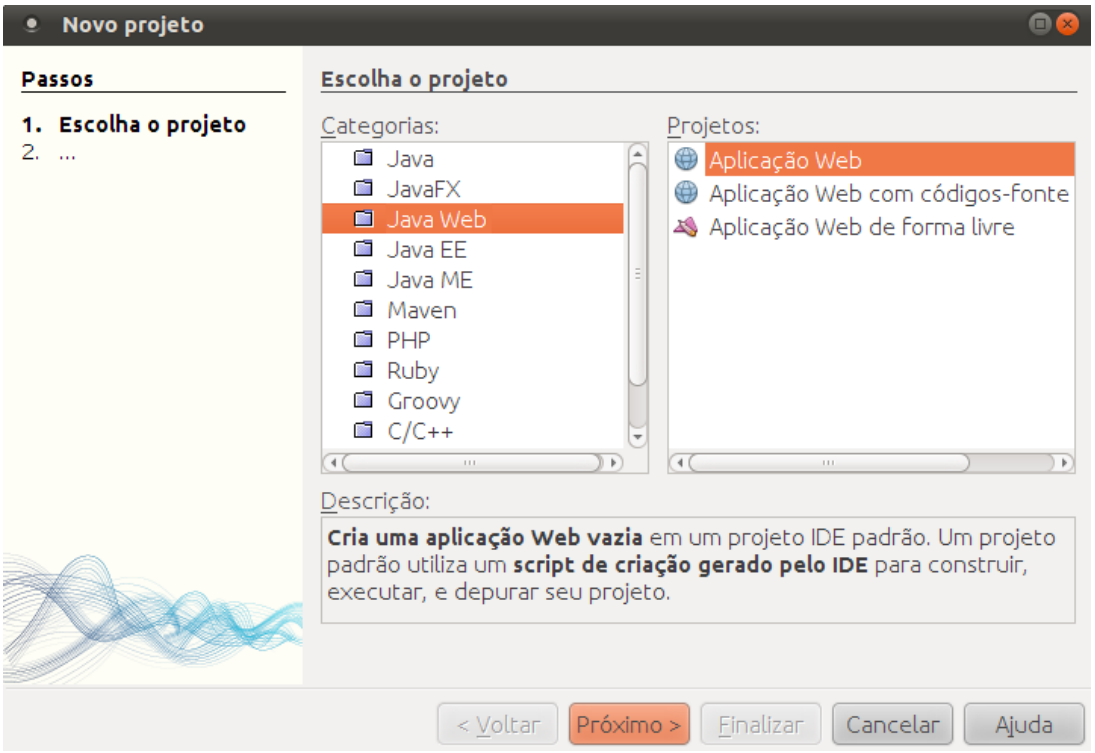

Figura 9: Tela de criação de novo projeto no NetBeans.
Fonte: Autoria própria.

O segundo passo foi definir o nome e o local do projeto (Figura 10), este projeto recebeu o nome de “Servico” e o local ficou definido em “/home/julianne/NetBeanProjects”.

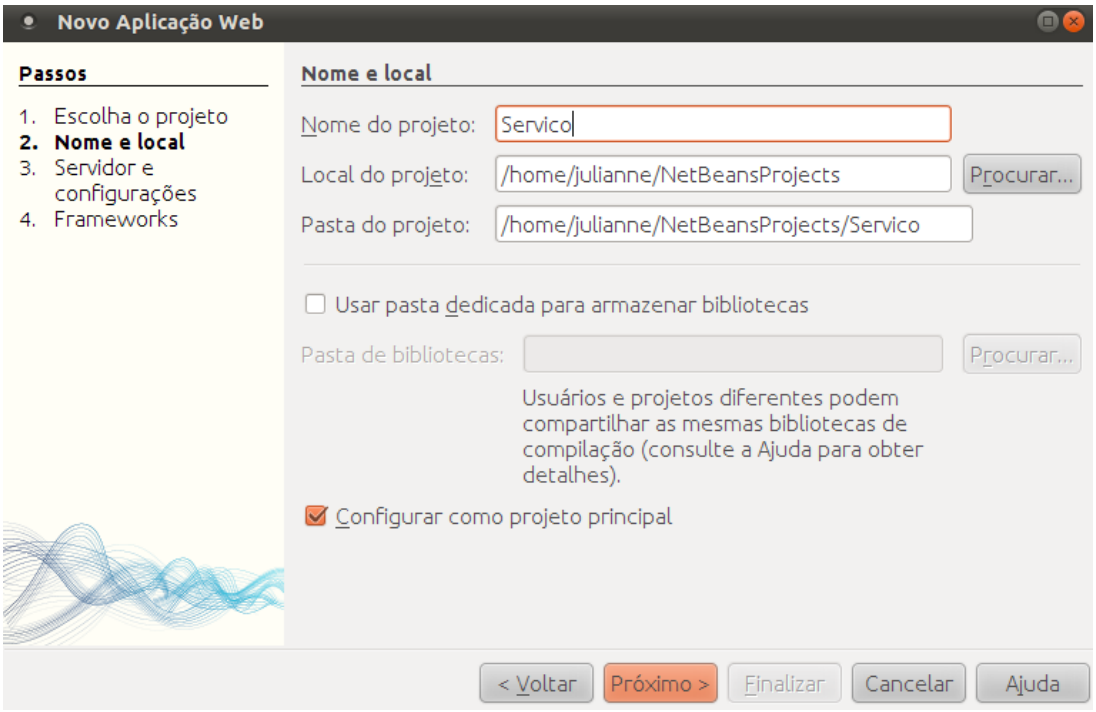

Figura 10: Tela que define o nome da aplicação web no NetBeans.
Fonte: Autoria própria.

O terceiro passo foi definir o servidor web e o caminho do contexto. Para o desenvolvimento deste WebService foi escolhido o servidor web Apache Tomcat por já ser utilizado na Prefeitura Municipal de Palmas. Então as configurações definidas foram: o servidor Apache Tomcat 6.0.26, a versão do Java: Java EE 5 e o caminho do contexto “/ServicoEnderecamento” (Figura 11).

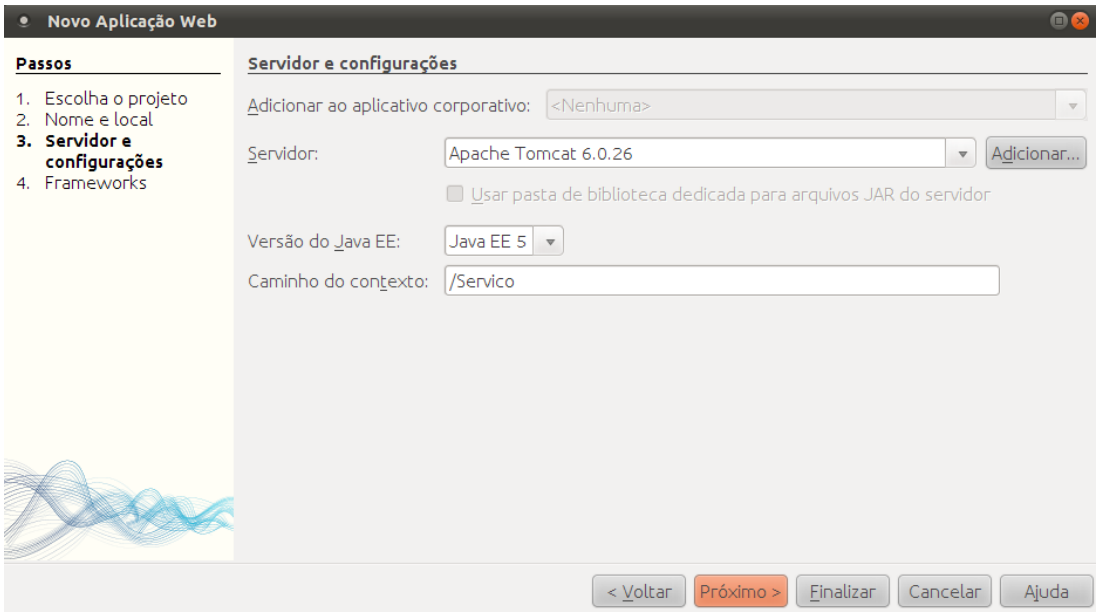

Figura 11: Tela que define o servidor e a versão Java do projeto.
Fonte: Autoria própria.

O quarto passo é a escolha do *framework*, neste projeto não foi necessária a utilização de nenhum *framework*, portando bastou clicar no botão “Finalizar” (Figura 12).

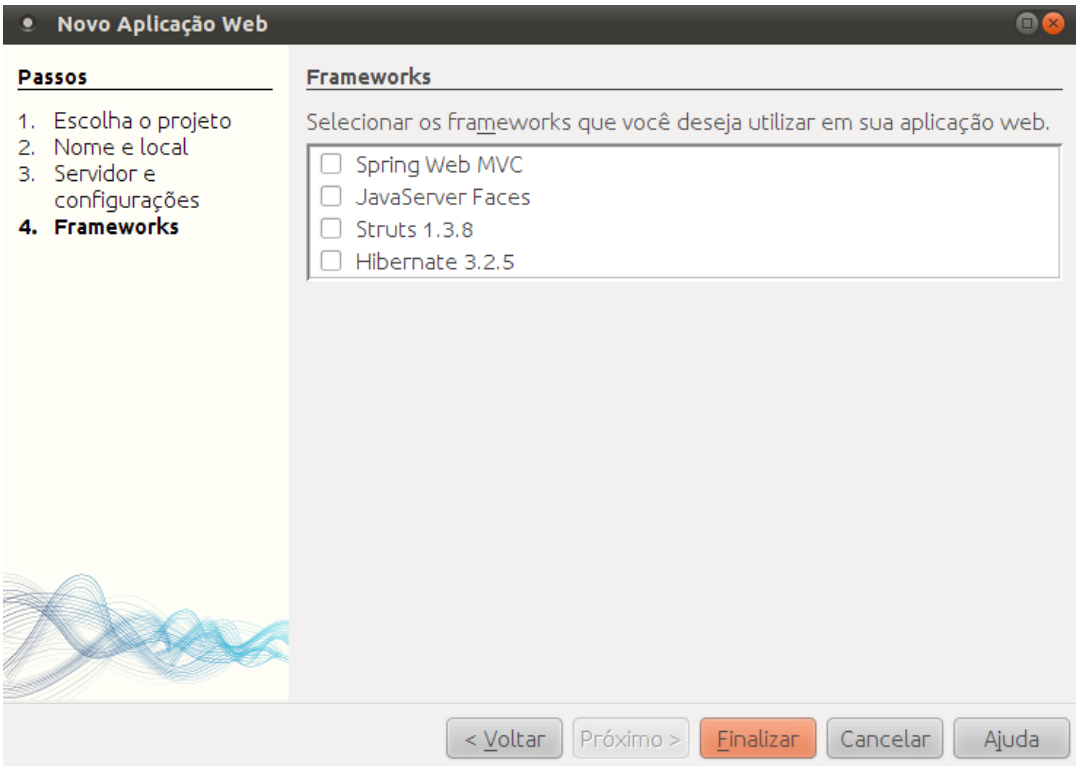

Figura 12: Tela que define qual framework será utilizado no projeto.
Fonte: Autoria própria.

A Figura 13 demonstra como ficou a estrutura inicial do projeto.

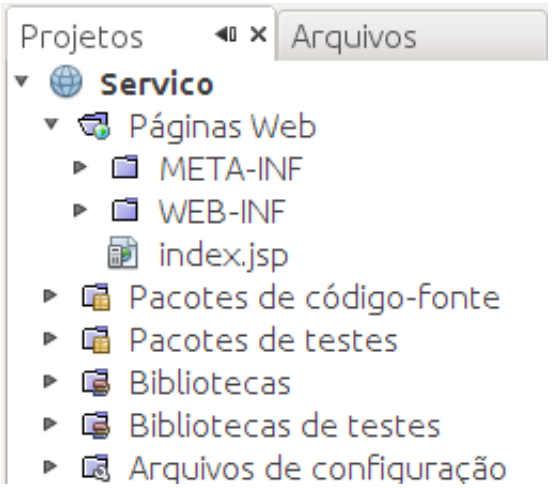

Figura 13: Estrutura do Projeto Servico.
Fonte: Autoria própria.

O próximo passo é a criação dos pacotes que serão utilizados no projeto. Neste projeto foi criado o pacote “conexaodb” clicando com o botão direito do mouse em Pacotes de código-fonte → Novo → Pacote Java em que abriu uma janela (Figura 14) para colocar o nome e local do pacote.

Figura 14: Tela de criação do pacote conexaodb.
Fonte: Autoria própria.

Foi criado também o pacote “enderecamento” em que serão criadas as classes de domínio do serviço de endereçamento (Figura 15).

Figura 15: Tela de criação do pacote enderecamento.
Fonte: Autoria própria.

5.3.2 Criando a conexão com banco de dados

Após a criação do projeto, o próximo passo foi criar a conexão com o banco de dados que será utilizada pelo WebService. A linguagem Java possui APIs JDBC (Java Database Connectivity) para o acesso a banco de dados.

As principais APIs JDBC de conexão com banco na linguagem Java estão no pacote java.sql e javax.sql.

No pacote java.sql, contém as classes core e interfaces que são necessárias para lidar com banco de dados, faz conexão ao banco de dados pelo DriverManager. O DriverManager provê o serviço básico para administrar drives JDBC. A partir do JDBC 2.0, este foi cedido pelo uso de javax.sql.DataSource (GONÇALVES, 2008).

No pacote javax.sql, contém as classes e interfaces que são usadas pelo acesso do lado servidor de fonte de dados. A principal inclusão como parte de javax.sql é o DataSource que possibilita uma alternativa para o DriverManager. Também inclui coisas como pool de conexões, transações distribuídas, e a implementação de RowSet (GONÇALVES, 2008).

Na criação desta conexão foi utilizado pacote javax.sql por permitir a criação do pool de conexão para o acesso a fonte de dados.

Para um melhor entendimento da importância do pool de conexão, veja na Figura 16 como funciona a conexão de um banco com uma aplicação Web sem utilizar pool de conexão.

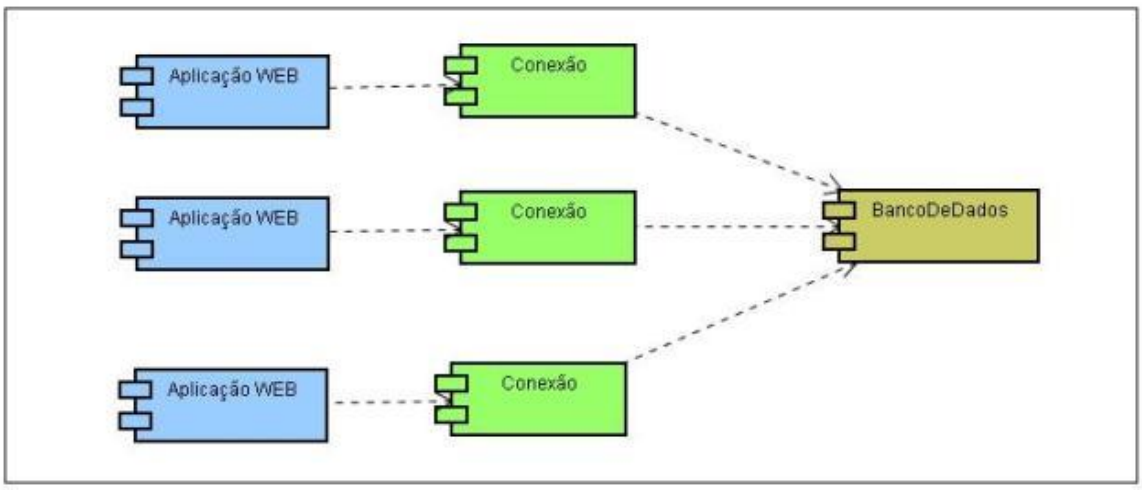

Figura 16: Arquitetura de conexão sem pool.
Fonte: Medeiros.

Cada vez que um usuário inicia uma aplicação web e criada uma conexão dedicada para aquela instância da aplicação web. Caso essa aplicação tenha uma alta demanda de usuários e um grande tráfego de dados corre-se o risco de pedar de desempenho da aplicação e do servidor (MEDEIROS).

Quando é utilizado o pool de conexão (Figura 17) é evitada a criação de uma conexão para cada requisição da aplicação web, pois utilizando essa técnica será criada uma única conexão que será usada por cada instância da aplicação.

Desta forma a conexão será criada somente uma vez pela primeira instância da aplicação web, ou quando o servidor Tomcat for iniciado.

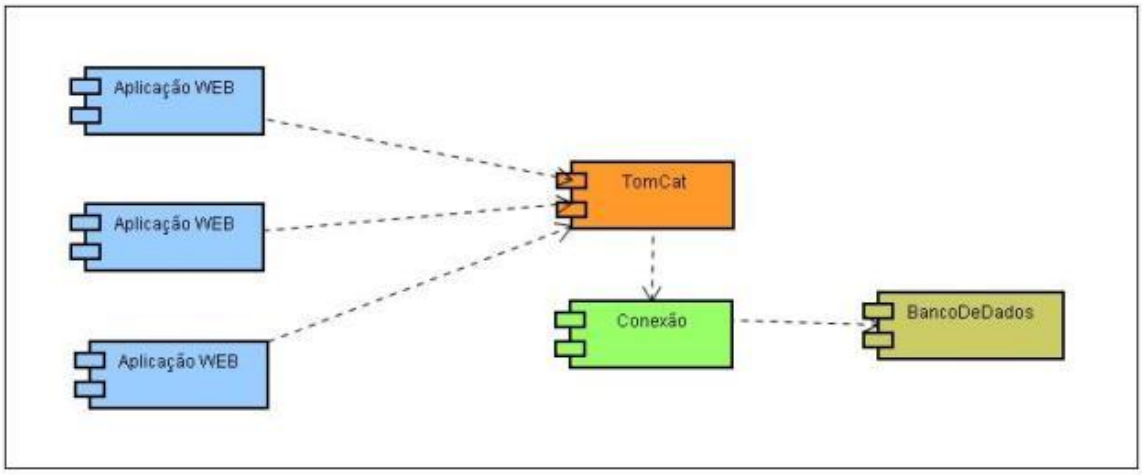

Figura 17: Arquitetura de conexão com pool gerenciado pelo Tomcat.
Fonte: Pimentel.

5.3.2.1 Configurando o pool de conexão

Para configura o pool de conexão é necessário editar o arquivo context.xml localizado no diretório META-INF do projeto (Figura 18).

Figura 18: Localização do arquivo context.xml.
Fonte: Autoria própria.

No arquivo context.xml foi criada a tag Resource, que é responsável por configurar os dados e o nome de um recurso disponível pela aplicação, e definido os seguintes atributos (Figura 19):

- **auth (linha 05)** – atributo que define o Apache Tomcat como responsável por gerenciar a abertura e o fechamento da conexão;
- **driverClassName (linha 06)** – atributo que define nome da classe do driver JDBC do banco de dados. No caso desse projeto para o banco de dados PostgreSQL;
- **maxActive (linha 07)** – atributo que define o número máximo de conexões ativas no pool;
- **maxIdle (linha 08)** – atributo que define o número máximo de inativas no pool;
- **maxWait (linha 09)** – atributo que define o tempo máximo de espera por uma conexão;
- **name (linha 10)** – atributo que define o nome do pool de conexão;
- **type (linha 11)** – atributo que define o tipo como DataSource;
- **url (linha 12)** – atributo que define a localização do banco de dados. A descrição da URL (Uniform Resource Locator) deve seguir o formato URL JDBC;
- **username (linha 13)** – atributo que define o nome do usuário no banco de dados;
- **password (linha 14)** – atributo que define a senha do usuário no banco de dados.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <Context antiJARLocking="true" path="/pool">
03
04      <Resource
05          auth="Container"
06          driverClassName="org.postgresql.Driver"
07          maxActive="20"
08          maxIdle="10"
09          maxWait="-1"
10          name = "jdbc/enderecamento"
11          type="javax.sql.DataSource"
12          url="jdbc:postgresql://localhost:5432/enderecamento"
13         username="postgres"
14         password="123456" />
15
16  </Context>
```

Figura 19: Arquivo context.xml.
Fonte: Autoria própria.

5.3.2.2 Registrando o pool na aplicação web

Para registrar o pool na aplicação é necessário editar o arquivo web.xml localizado no diretório WEB-INF do projeto (Figura 20).

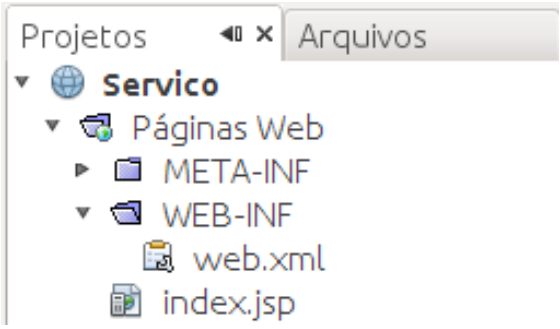

Figura 20: Localização do arquivo web.xml.
Fonte: Autoria própria.

No arquivo web.xml (Figura 21) foi criada uma referência (linhas 17 a 22) para o pool de conexão da aplicação. Na linha 18 é especificado o nome do pool de conexão que deve ser idêntico ao definido no arquivo context.xml.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
03  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
05  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
06
07      <session-config>
08          <session-timeout>
09              30
10          </session-timeout>
11      </session-config>
12
13      <welcome-file-list>
14          <welcome-file>index.jsp</welcome-file>
15      </welcome-file-list>
16
17      <resource-ref>
18          <res-ref-name>jdbc/enderecamento</res-ref-name>
19          <res-type>javax.sql.DataSource</res-type>
20          <res-auth>Container</res-auth>
21          <res-sharing-scope>Shareable</res-sharing-scope>
22      </resource-ref>
23
24  </web-app>
```

Figura 21: Arquivo web.xml.
Fonte: Autoria própria.

5.3.2.3 Criando a classe de conexão

A classe de conexão nomeada de “ConexaoDB” foi criada no pacote conexaodb (Figura 22).

Figura 22: Tela de criação da classe ConexaoDB.
Fonte: Autoria própria.

Na figura 23 mostra como ficou a classe ConexaoDB.

```
01  package conexaodb;
02
03  import java.sql.Connection;
04  import java.sql.SQLException;
05  import javax.naming.InitialContext;
06  import javax.sql.DataSource;
07
08  public class ConexaoDB {
09
10      /*** Conexao com o banco de dados enderecamento */
11      public static Connection abreConexaoPgEnderecamento() throws SQLException {
12          try {
13              InitialContext context = new InitialContext();
14              DataSource ds = (DataSource)
15                                  context.lookup("java:comp/env/jdbc/enderecamento");
16              return ds.getConnection();
17          } catch (ClassNotFoundException e) {
18              throw new SQLException(e.getMessage());
19          }
20      }
21  }
```

Figura 23: Classe ConexaoDB.
Fonte: Autoria própria.

Na classe ConexaoDB foi criado o método “abreConexaoPgEnderecamento” da seguinte forma:

- **Na linha 13** – foi instanciada a classe Initialcontext, esta classe possui o método *lookup* utilizado na busca das informações que foram registradas no arquivo web.xml;
- **Na linha 14 e 15** – foi instanciado DataSource que obtém a origem dos dados pelo método *lookup*;
- **Na linha 16** – foi invocado o método “getConnection” que obtém a conexão.

5.3.3 Criando as classes de domínio

Foram criadas três classes de domínio que fazem referência aos dados utilizados pelo serviço de endereçamento. A primeira classe criada foi a classe de domínio Setor dentro do pacote enderecamento (Figura 24).

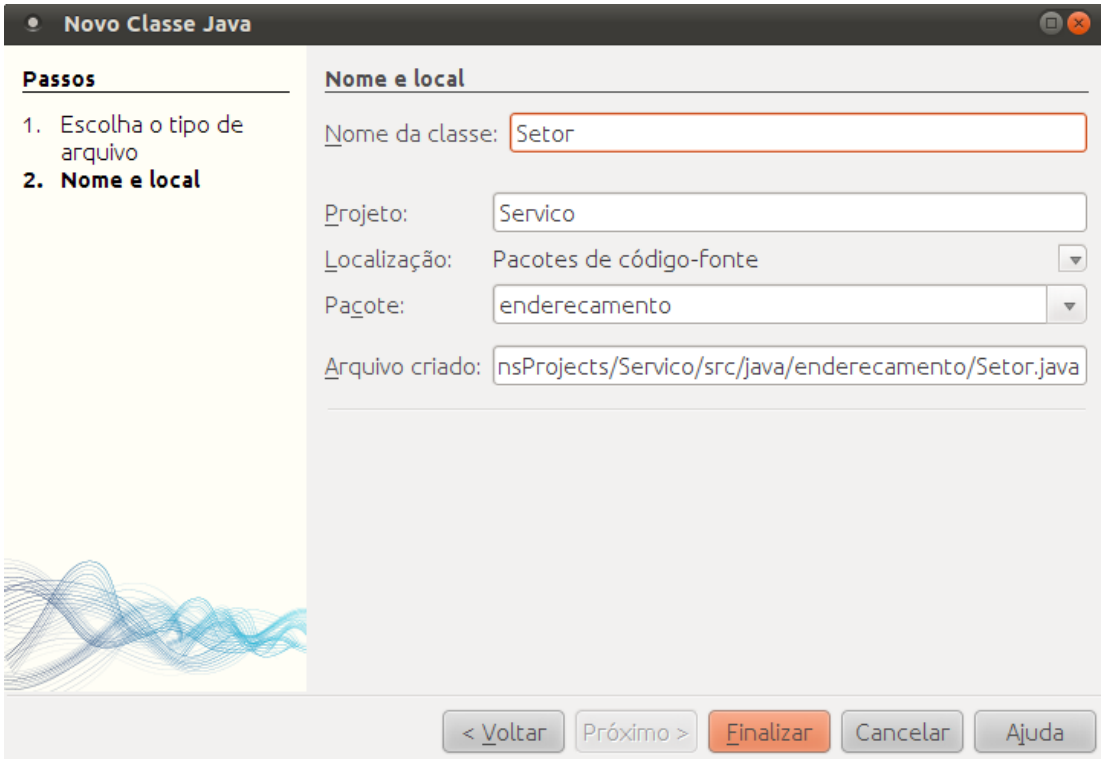

Figura 24: Tela de criação da classe Setor.

Fonte: Autoria própria.

A classe Setor (Figura 25) possui os atributos: codSetor que é o código de identificação do setor, setorNovo que é a nova nomenclatura do setores utilizada no endereço de Palmas e o setorAntigo que é a antiga nomenclatura dos setores no endereço de Palmas.

```
01  package enderecamento;
02
03  public class Setor {
04      private Long codSetor;
05      private String setorNovo;
06      private String setorAntigo;
07
08      public Setor() {    }
09
10      public Setor(Long codSetor, String setorNovo, String setorAntigo) {
11          this.codSetor = codSetor;
12          this.setorNovo = setorNovo;
13          this.setorAntigo = setorAntigo;
14      }
15  // getters e setters omitidos
16  }
```

Figura 25: Classe Setor.
Fonte: Autoria própria.

A segunda classe criada foi a classe de domínio Alameda no pacote enderecamento (Figura 26).

Figura 26: Tela de criação da classe Alameda.

Fonte: Autoria própria.

A classe Alameda (Figura 27) possui os atributos: codAlameda que é o código de identificação da alameda, alamedaNovo que é a nova nomenclatura das alamedas utilizada no endereço de Palmas, alamedaAntigo que é a antiga nomenclatura das alamedas no endereço de Palmas e setor que é a chave estrangeira que referencia de qual setor a alameda faz parte.

```
01  package enderecamento;
02
03  public class Alameda {
04      private Long codAlameda;
05      private String alamedaNovo;
06      private String alamedaAntigo;
07      private Setor setor;
08
09      public Alameda() {    }
10
11      public Alameda(Long codAlameda, String alamedaNovo, String alamedaAntigo) {
12          this.codAlameda = codAlameda;
13          this.alamedaNovo = alamedaNovo;
14          this.alamedaAntigo = alamedaAntigo;
15      }
16   // getters e setters omitidos
17  }
```

Figura 27: Classe Alameda.
Fonte: Autoria própria.

A terceira classe criada foi a classe de domínio Lote no pacote enderecamento (Figura 28).

Figura 28: Tela de criação da classe Lote.
Fonte: Autoria própria.

A classe Lote (Figura 29) possui os atributos: codLote que é o código de identificação do lote; quadraInterna que é a descrição da quadra interna (QI), presente no antigo endereço de Palmas; numeroLoteAlameda que é o numero do lote e alameda do antigo endereço de Palmas; numero que é o numero do lote; setor que é a chave estrangeira que referencia de qual setor o lote faz parte e alameda que é a chave estrangeira que referencia de qual alameda o lote faz parte.

```
01  package enderecamento;
02
03  public class Lote {
04      private String codLote;
05      private String quadraInterna;
06      private String numeroLoteAlameda;
07      private String numero;
08      private Setor setor;
09      private Alameda alameda;
10
11      public Lote() {    }
12
13      public Lote(String codLote, String quadraInterna, String numeroLoteAlameda,
14                      String numero) {
15          this.codLote = codLote;
16          this.quadraInterna = quadraInterna;
17          this.numeroLoteAlameda = numeroLoteAlameda;
18          this.numero = numero;
19      }
20  // getters e setters omitidos
21  }
```

Figura 29: Classe Lote.
Fonte: Autoria própria.

5.3.4 Criando a classe Dao

DAO (Data Access Object) é um padrão utilizado para acessar os dados contidos em um banco de dados.

> [...] O Padrão DAO fornece uma inferface independente, no qual você pode usar para persistir objetos de dados. A idéia é colocar todas as funcionalidades encontradas no desenvolvimento de acesso e trabalho com dados em um só local, tornando simples sua manutenção (GONÇALVES, 2008, pg. 145).

Na classe DAO geralmente e criado os métodos inserir, selecionar, atualizar e excluir objetos de um banco de dados. O padrão pode ser implementado de duas

formas, pode ser criado um DAO para cada classe de objetos de uma aplicação ou pode ter um único DAO responsável por todos os objetos de uma aplicação (GONÇALVES, 2008).

Neste projeto foi criado o DAO EnderecamentoDao (Figura 30) que possuirá os métodos que serão utilizados pelo serviço de endereçamento.

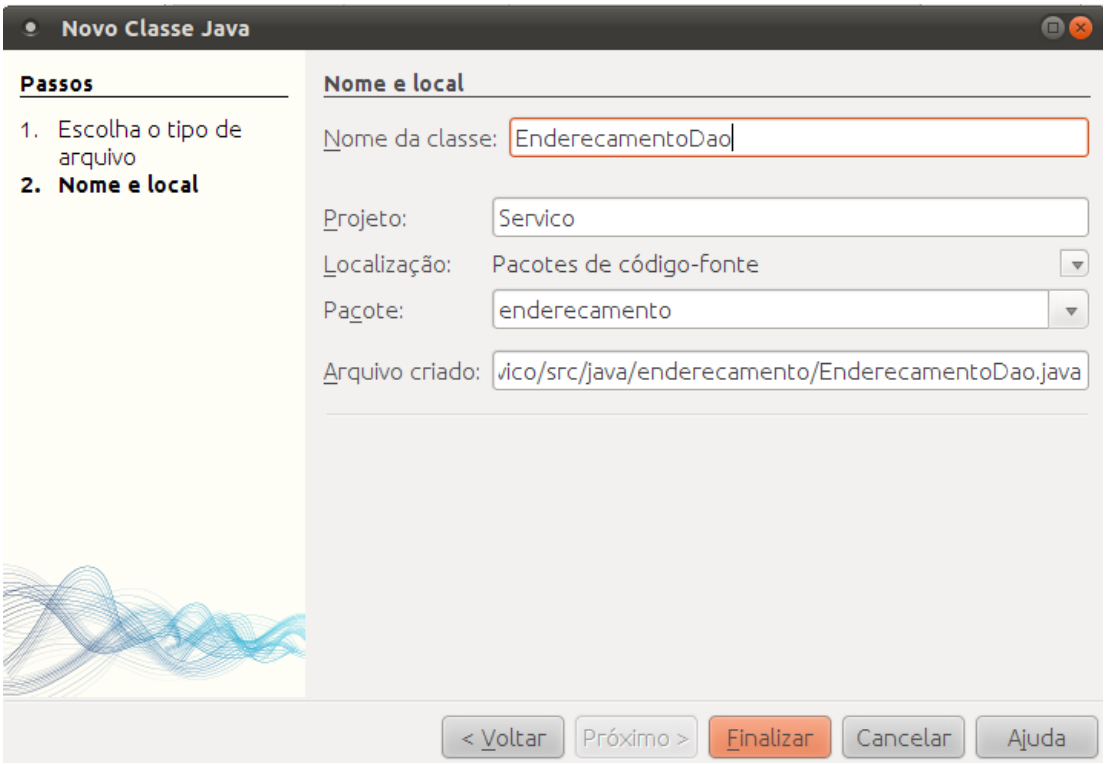

Figura 30: Tela de criação da classe EnderecamentoDao.
Fonte: Autoria própria.

Na classe EnderecamentoDao (Figura 31) foi criado os seguintes métodos:

- **getSetor (linha 18 a 44)** – que retorna uma lista de objetos setor com todos os setores do endereço de Palmas.
- **getAlameda (linha 46 a 75)** – que retorna uma lista de objetos alameda com todas as alamedas de um determinado setor. Para isso é necessário enviar para este método o objeto Setor que será utilizado na consulta do banco de dados.
- **getQuadraQiConjLote (linha 77 a 111)** – que retorna uma lista de objetos Lote com todos os lotes, quadras e QIs de uma determinada alameda e setor. Para isso é necessário enviar para este método o objeto Alameda e Setor que será utilizado na consulta do bando de dados.

```
01  package enderecamento;
02
03  import conexaodb.ConexaoDB;
04  import java.sql.Connection;
05  import java.sql.PreparedStatement;
06  import java.sql.ResultSet;
07  import java.sql.SQLException;
08  import java.util.ArrayList;
09  import java.util.List;
10
11  public class EnderecamentoDao {
12      private Connection conn;
13
14      public EnderecamentoDao() throws SQLException {
15          conn = ConexaoDB.abreConexaoPgEnderecamento();
16      }
17
18      public List getSetor() throws Exception {
19          PreparedStatement ps = null;
20          ResultSet rs = null;
21          List<Setor> lista = new ArrayList<Setor>();
22
23          try {
24              ps = conn.prepareStatement("select tbsetor.codsetor, "
25                      + "tbsetor.novanomenclatura, tbsetor.descricao from tbsetor "
26                      + " group by tbsetor.codsetor, tbsetor.descricao, "
27                      + "tbsetor.novanomenclatura order by tbsetor.novanomenclatura");
28              rs = ps.executeQuery();
29
30              while (rs.next()) {
31                  Long codSetor = rs.getLong(1);
32                  String setorNovo = rs.getString(2);
33                  String setorAntigo = rs.getString(3);
34                  lista.add(new Setor(codSetor, setorNovo, setorAntigo));
35              }
36              rs.close();
37              ps.close();
38              return lista;
39          } catch (SQLException sqle) {
40              throw new Exception("Erro ao inserir dados " + sqle);
41          } finally {
42              conn.close();
43          }
44      }
45
46      public List getAlameda(Setor setor) throws Exception {
47          PreparedStatement ps = null;
48          ResultSet rs = null;
49          List<Alameda> lista = new ArrayList<Alameda>();
50
51          try {
52              ps = conn.prepareStatement("select tblogradouro.codlogradouro,
53  tblogradouro.descricao,"
54                      + "tblogradouro.descricaoantiga as descricao from tblogradouro"
55                      + " where tblogradouro.codsetor='" + setor.getCodSetor() + "' "
56                      + "group by tblogradouro.codlogradouro, tblogradouro.codsetor, "
57                      + "tblogradouro.descricao, tblogradouro.descricaoantiga "
58                      + "order by tblogradouro.descricao");
59              rs = ps.executeQuery();
60
```

```
61              while (rs.next()) {
62                  Long codAlameda = rs.getLong(1);
63                  String alamedaNovo = rs.getString(2);
64                  String alamedaAntigo = rs.getString(3);
65                  lista.add(new Alameda(codAlameda, alamedaNovo, alamedaAntigo));
66              }
67              rs.close();
68              ps.close();
69              return lista;
70          } catch (SQLException sqle) {
71              throw new Exception("Erro ao inserir dados " + sqle);
72          } finally {
73              conn.close();
74          }
75      }
76
77      public List getQuadraQiConjLote(Setor setor, Alameda alameda) throws Exception
78  {
79          PreparedStatement ps = null;
80          ResultSet rs = null;
81          List<Lote> lista = new ArrayList<Lote>();
82
83          try {
84              ps = conn.prepareStatement("select tblote.codlotegis,tblote.quadrainterna, "
85                      + "tblote.numerolotealameda,tblote.numero  "
86                      + "from  tblote inner join tbsetor on tbsetor.codsetor = tblote.codsetor  "
87                      + "inner join tblogradouro on "
88                      + "tblote.codlogradouro = tblogradouro.codlogradouro "
89                      + "where tbsetor.codsetor='"+setor.getCodSetor()+"' and "
90                      + " tblogradouro.codlogradouro ='"+alameda.getCodAlameda()+"' "
91                      + "group by tblote.codlotegis,tblote.quadrainterna,
92                      + "tblote.numerolotealameda, "
93                      + "tblote.numero order by 2");
94              rs = ps.executeQuery();
95
96              while (rs.next()) {
97                  String codlote = rs.getString(1);
98                  String quadraInterna = rs.getString(2);
99                  String numeroLoteAlameda = rs.getString(3);
100                 String numero = rs.getString(4);
101                 lista.add(new Lote(codlote, quadraInterna, numeroLoteAlameda, numero));
102             }
103             rs.close();
104             ps.close();
105             return lista;
106         } catch (SQLException sqle) {
107             throw new Exception("Erro ao inserir dados " + sqle);
108         } finally {
109             conn.close();
110         }
111     }
112 }
```

Figura 31: Classe EnderecamentoDao.
Fonte: Autoria própria.

5.3.5 Criando o Webservice

O WebService Enderecamento foi criado clicando com o botão direito do mouse no projeto e selecione a opção Novo → Serviço Web em que abriu a janela (Figura 32) para colocar o nome do serviço web e o pacote. O nome definido foi Enderecamento e o pacote selecionado foi o pacote enderecamento.

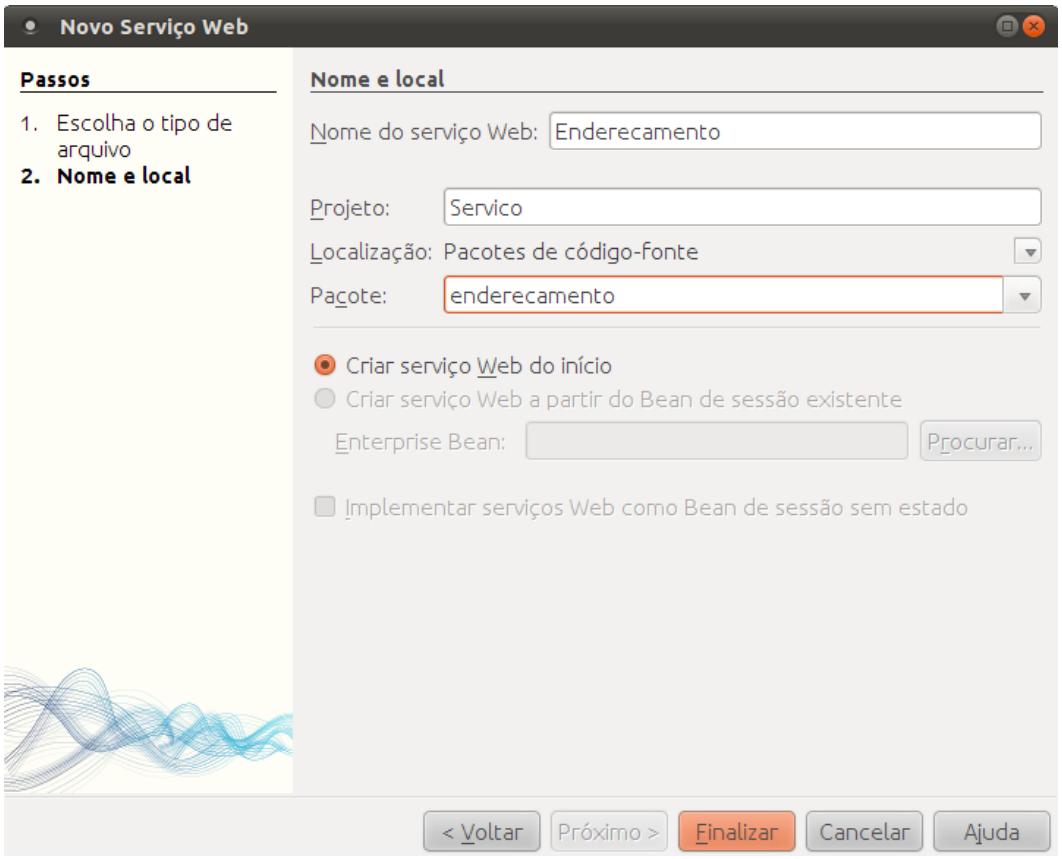

Figura 32: Tela de criação do serviço web.
Fonte: Autoria própria.

Quando é criado um serviço web utilizando o NetBeans, não é necessário importar a biblioteca JAX-WS, porque o mesmo realiza essa importação automaticamente no momento que é criado o serviço.

Depois de finalizada a criação do serviço web, foi adicionada uma nova pasta na estrutura do projeto com o nome Serviços Web. Dentro desta pasta está o serviço Enderecamento (Figura 33).

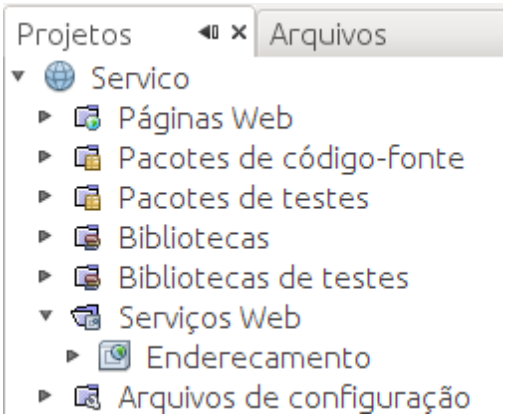

Figura 33: Nova estrutura do projeto.
Fonte: Autoria própria.

Logo após a criação do WebService Enderecamento, o próximo passo foi criar as operações que podem ser chamadas também de web method. Para isso, foi escolhida a visualização “Projeto” na janela da classe Enderecamento.java, a classe Enderecamento.java corresponde o WebService Enderecamento, como pode ser visto na Figura 34.

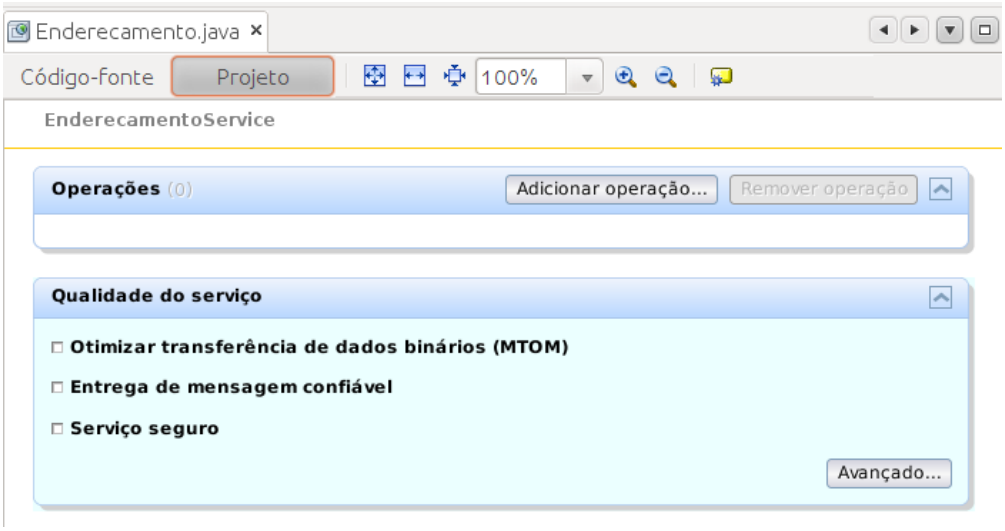

Figura 34: Classe Enderecamento.
Fonte: Autoria própria.

O botão “Adicionar operação” abrirá uma nova janela em que serão inseridas as informações da operação. A primeira operação criada foi a operação getSetor. A operação getSetor não receberá parâmetros e retornará uma lista de objetos Setor (Figura 35).

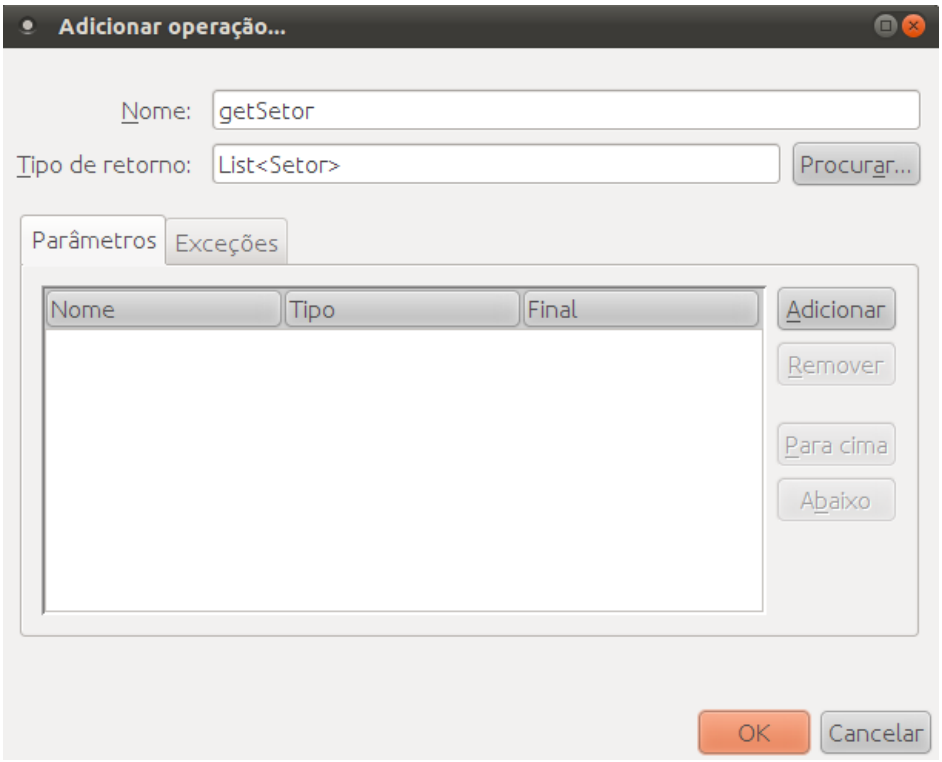

Figura 35: Tela de criação da operação getSetor.
Fonte: Autoria própria.

Na operação getSetor (Figura 36) foi adicionado o código que retornará a lista de setores de Palmas. Para isso foi criada uma instância da classe EnderecamentoDao (linha 04) e uma variável do tipo lista de objeto Setor, nomeada de lista (linha 05), que recebe uma lista retornada pelo método getSetor da classe EnderecamentoDOA.

```
01  ...
02  @WebMethod(operationName = "getSetor")
03      public List<Setor> getSetor() throws Exception {
04          EnderecamentoDao dao = new EnderecamentoDao();
05          List<Setor> lista = dao.getSetor();
06          return lista;
07      }
08   ...
```

Figura 36: Código da operação getSetor.
Fonte: Autoria própria.

A segunda operação criada foi a operação getAlameda. A operação getAlameda receberá como parâmetro um long (código do setor) e retornará uma lista de objetos Alameda (Figura 37).

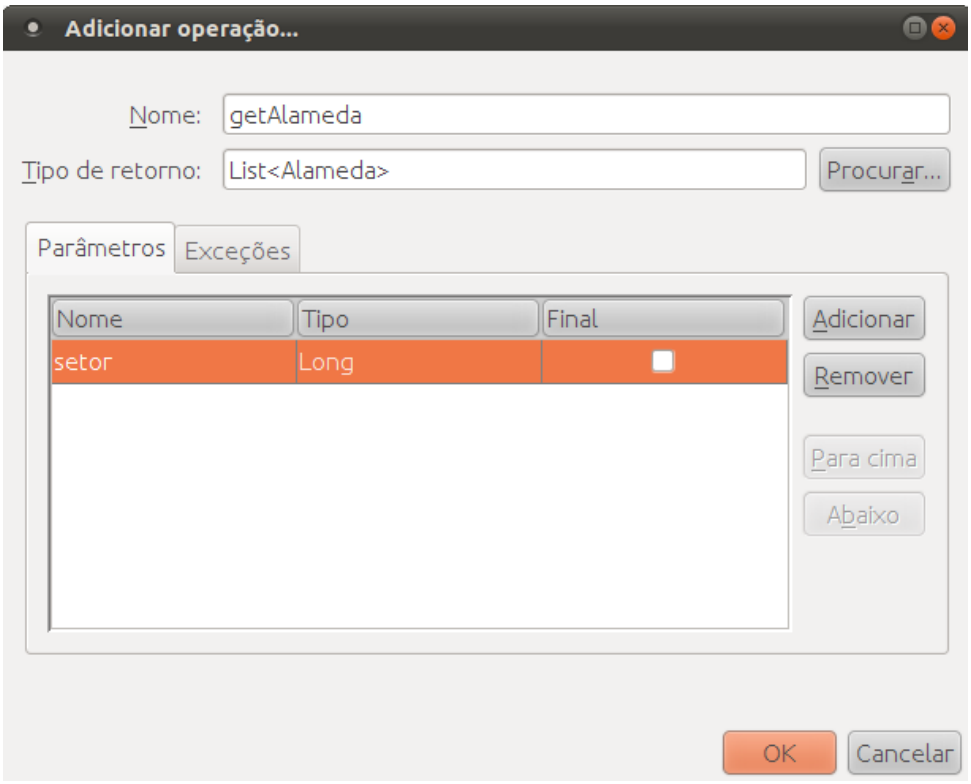

Figura 37: Tela de criação da operação getAlameda.
Fonte: Autoria própria.

Na operação getAlameda (Figura 38) foi adicionado o código que retornará a lista de alamedas de um determinado setor de Palmas. Para isso foi criada uma instância da classe EnderecamentoDao (linha 05), um objeto setor (linha 06 e 07) e uma variável do tipo lista de objeto Alameda, nomeada de lista (linha 08), que recebe uma lista retornada pelo método getAlameda da classe EnderecamentoDOA.

```
01  ...
02  @WebMethod(operationName = "getAlameda")
03      public List<Alameda> getAlameda(
04              @WebParam(name = "setor") Long setor) throws Exception {
05          EnderecamentoDao dao = new EnderecamentoDao();
06          Setor se = new Setor();
07          se.setCodSetor(setor);
08          List<Alameda> lista = dao.getAlameda(se);
09          return lista;
10      }
11  ...
```

Figura 38: Código da operação getAlameda.
Fonte: Autoria própria.

A terceira operação criada foi a operação getQuadraQiConjLote. A operação getQuadraQiConjLote receberá como parâmetros dois long (código do setor e código da alameda) e retornará uma lista de objetos Lote (Figura 39).

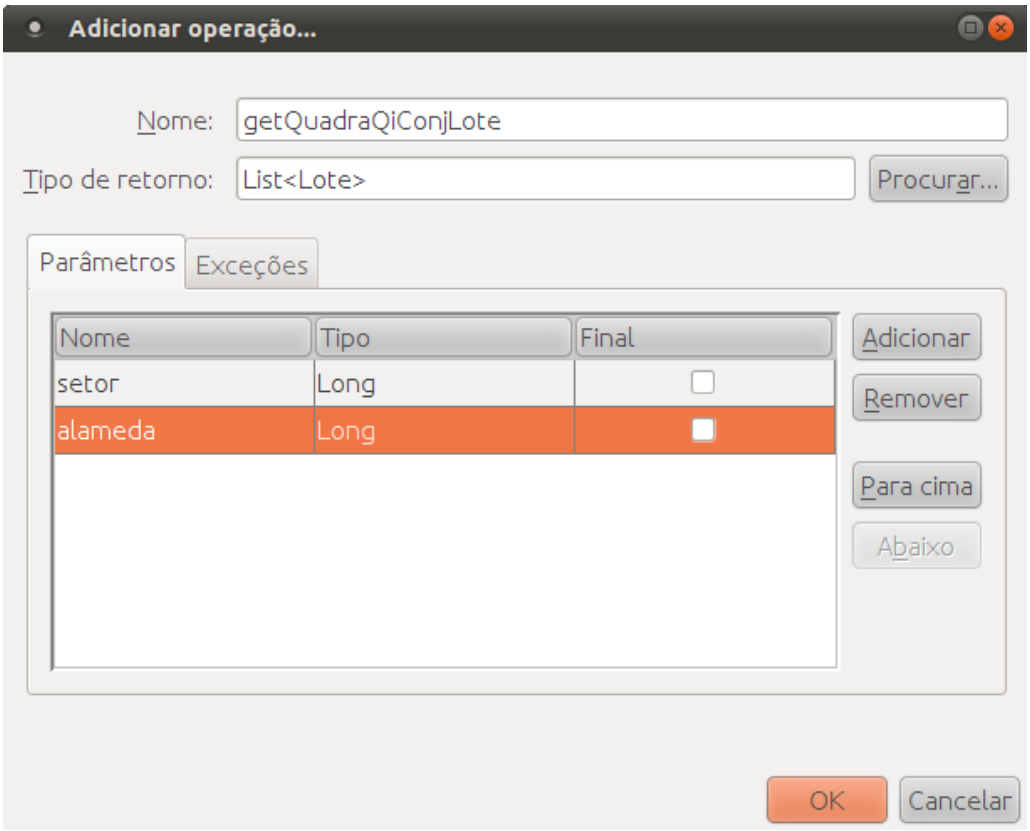

Figura 39: Tela de criação da operação getQuadraQiConjLote.
Fonte: Autoria própria.

Na operação getQuadraQiConjLote foi adicionado o código que retornará uma lista com a alameda, o lote, QI e numero de um determinado setor e alameda de Palmas. Para isso foi criada uma instância da classe EnderecamentoDao (linha 06), um objeto setor (linha 07 e 08), um objeto alameda (linha 09 e 10) e uma variável do tipo lista de objeto Lote, nomeada de lista (linha 11), que recebe uma lista retornada pelo método getQuadraQiConjLote da classe EnderecamentoDOA.

```
01  ...
02  @WebMethod(operationName = "getQuadraQiConjLote")
03   public List<Lote> getQuadraQiConjLote(
04          @WebParam(name = "setor") Long setor,
05          @WebParam(name = "alameda") Long alameda) throws Exception {
06      EnderecamentoDao dao = new EnderecamentoDao();
07      Setor se = new Setor();
08      se.setCodSetor(setor);
09      Alameda al = new Alameda();
10      al.setCodAlameda(alameda);
11      List<Lote> lista = dao.getQuadraQiConjLote(se, al);
12      return lista;
13  }
14  ...
```

Figura 40: Código da operação getQuadraQiConjLote.
Fonte: Autoria própria.

Após finalizar todas as configurações do WebService, o próximo passo foi implantar o projeto. Para isso, foi clicado com o botão direito do mouse no projeto e selecionado a opção “Implantar”.

Depois que o projeto foi implantado foi clicado com o botão direito do mouse no WebService Enderecamento e selecionado a opção Testar serviço Web para verificar se o deploy foi executado com sucesso.

Como não ocorreu erro no deploy uma página foi aberta no navegador com as informações do WebService Enderecamento (Figura 41).

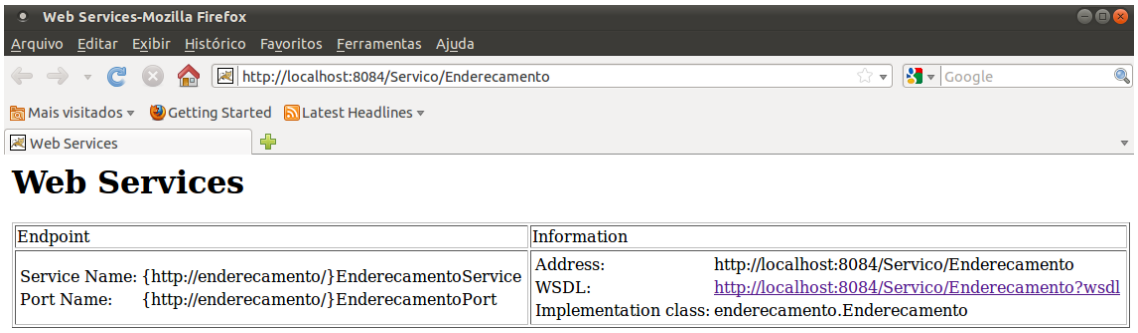

Figura 41: Informações do WebService Enderecamento.
Fonte: Autoria própria.

O arquivo WSDL do WebService Enderecamento ficou com a estrutura como mostra a figura 42.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2-
hudson-752-. -->
<definitions targetNamespace="http://enderecamento/" name="EnderecamentoService"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:tns="http://enderecamento/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://enderecamento/"
schemaLocation="EnderecamentoService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="getSetor">
    <part name="parameters" element="tns:getSetor"/>
  </message>
  <message name="getSetorResponse">
    <part name="parameters" element="tns:getSetorResponse"/>
  </message>
```

```
  <message name="Exception">
    <part name="fault" element="tns:Exception"/>
  </message>
  <message name="getAlameda">
    <part name="parameters" element="tns:getAlameda"/>
  </message>
  <message name="getAlamedaResponse">
    <part name="parameters" element="tns:getAlamedaResponse"/>
  </message>
  <message name="getQuadraQiConjLote">
    <part name="parameters" element="tns:getQuadraQiConjLote"/>
  </message>
  <message name="getQuadraQiConjLoteResponse">
    <part name="parameters" element="tns:getQuadraQiConjLoteResponse"/>
  </message>
  <portType name="Enderecamento">
    <operation name="getSetor">
      <input wsam:Action="http://enderecamento/Enderecamento/getSetorRequest"
message="tns:getSetor"/>
      <output wsam:Action="http://enderecamento/Enderecamento/getSetorResponse"
message="tns:getSetorResponse"/>
      <fault message="tns:Exception" name="Exception"
wsam:Action="http://enderecamento/Enderecamento/getSetor/Fault/Exception"/>
    </operation>
    <operation name="getAlameda">
      <input wsam:Action="http://enderecamento/Enderecamento/getAlamedaRequest"
message="tns:getAlameda"/>
      <output wsam:Action="http://enderecamento/Enderecamento/getAlamedaResponse"
message="tns:getAlamedaResponse"/>
      <fault message="tns:Exception" name="Exception"
wsam:Action="http://enderecamento/Enderecamento/getAlameda/Fault/Exception"/>
    </operation>
    <operation name="getQuadraQiConjLote">
      <input
wsam:Action="http://enderecamento/Enderecamento/getQuadraQiConjLoteRequest"
message="tns:getQuadraQiConjLote"/>
      <output
wsam:Action="http://enderecamento/Enderecamento/getQuadraQiConjLoteResponse"
message="tns:getQuadraQiConjLoteResponse"/>
      <fault message="tns:Exception" name="Exception"
wsam:Action="http://enderecamento/Enderecamento/getQuadraQiConjLote/Fault/Exception"/>
    </operation>
  </portType>
  <binding name="EnderecamentoPortBinding" type="tns:Enderecamento">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getSetor">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="Exception">
        <soap:fault name="Exception" use="literal"/>
      </fault>
    </operation>
    <operation name="getAlameda">
      <soap:operation soapAction=""/>
      <input>
```

```
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
    <fault name="Exception">
      <soap:fault name="Exception" use="literal"/>
    </fault>
  </operation>
  <operation name="getQuadraQiConjLote">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
    <fault name="Exception">
      <soap:fault name="Exception" use="literal"/>
    </fault>
  </operation>
 </binding>
 <service name="EnderecamentoService">
  <port name="EnderecamentoPort" binding="tns:EnderecamentoPortBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
  </port>
 </service>
</definitions>
```

Figura 42: Arquivo WSDL do Webservice Enderecamento.
Fonte: Autoria própria.

5.3.6 Criando um Cliente para o WebService

A Prefeitura Municipal de Palmas possui um Formulário de Informações Cadastrais (FIC) como requisito para ato junto à prefeitura.

Nos formulários da FIC existe a necessidade do preenchimento do endereço de Palmas, como por exemplo, no formulário FIC pessoa jurídica que é preciso cadastrar o endereço da empresa em Palmas.

Para facilitar esse preenchimento foi utilizado o WebService Enderecamento que facilita tanto o preenchimento para as pessoas que só conhece o antigo endereço de Palmas como para as pessoas que só conhece o novo endereço de Palmas.

O sistema FIC foi desenvolvido na linguagem Python utilizando o *framework* Django que possibilita a criação de sistemas com agilidade em curto prazo.

Para utilizar o WebService Enderecamento neste sistema foi necessário o uso da biblioteca SUDS, que é um cliente SOAP utilizado pela linguagem Python para consumir WebService SOAP.

A biblioteca SUDS interpreta o arquivo WSDL de um WebService, com o auxílio de um terminal Python foi executado um teste do funcionamento da biblioteca SUDS como mostra a Figura 43.

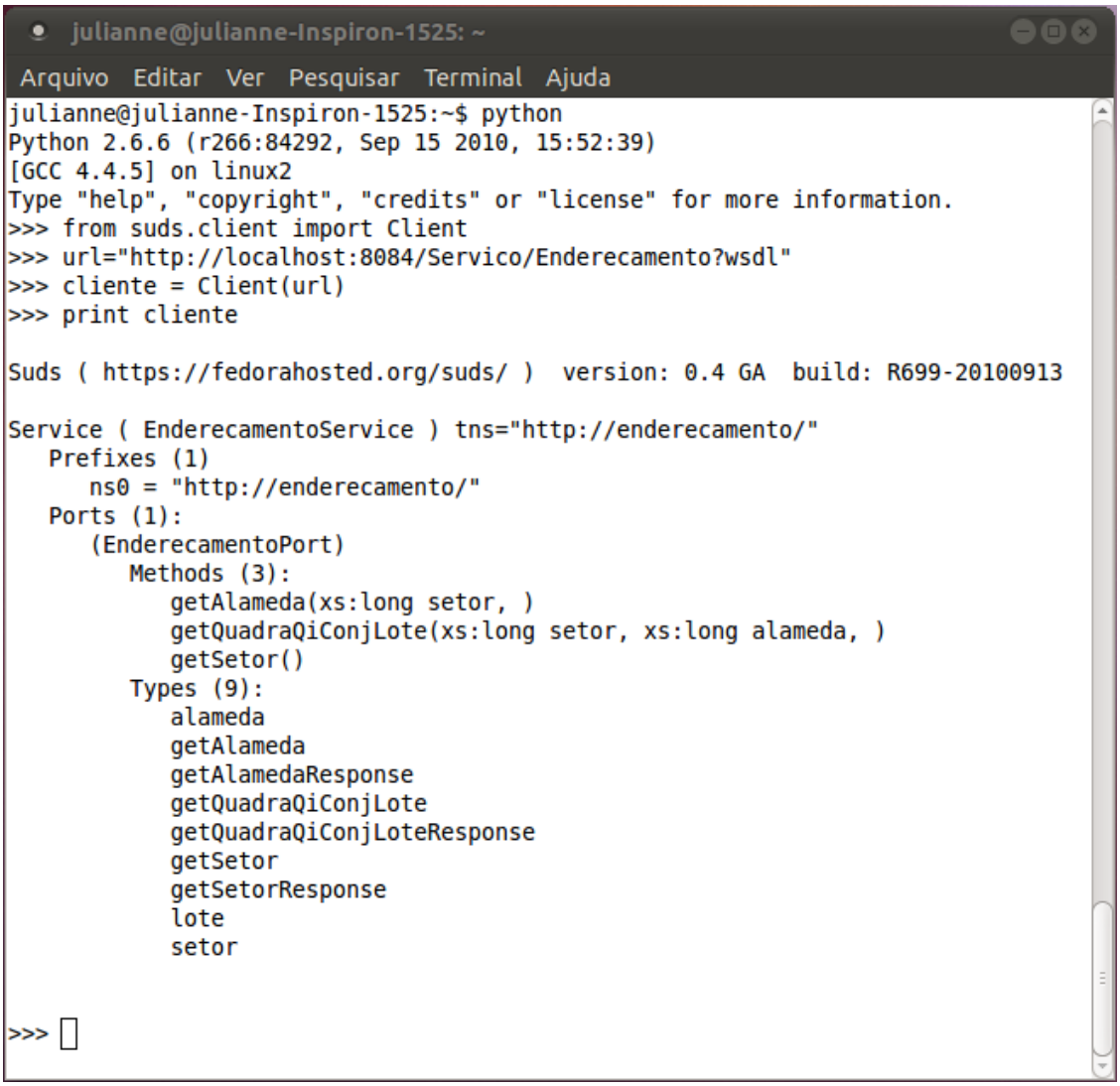

Figura 43: Comando da biblioteca SUDS em um terminal Python.
Fonte: Autoria própria.

Abrindo um terminal Python foi importada a biblioteca SUDS, criada a variável “url” que recebe o endereço do arquivo WSDL do WebService e criada também a variável “cliente” que recebe a interpretação do WebService pelo método Client da biblioteca SUDS e por último foi digitado o comando *print* para imprimir a variável

cliente. Na variável cliente, foi interpretado o nome, os métodos e os tipos de dados do WebService.

Para a utilização do WebService Enderecamento no sistema FIC foi criado um arquivo webserviceEndereco.py e digitado os mesmos comandos testados no terminal Python. No mesmo arquivo foram criadas também as funções que receberão cada método do WebService (Figura 44).

```
01  # -*- coding: utf-8 -*-
02  from suds.client import Client
03
04  """ endereco do webservice enderecamento """
05  url = 'http:// localhost:8084/Servico/Enderecamento?wsdl'
06  client = Client(url)
07
08  def getSetor():
09      return client.service.getSetor()
10
11  def getAlameda(setor):
12      return client.service.getAlameda(setor)
13
14  def getQuadraQiConjLote(setor, alameda):
15      return client.service.getQuadraQiConjLote(setor, alameda)
```

Figura 44: Arquvio webserviceEndereco.py.
Fonte: Autoria própria.

Depois de criar o arquivo webserviceEndereco.py que consome o WebService Enderecamento, foi acrescentado este arquivo na estrutura do sistema FIC. O sistema FIC possui a estrutura como pode ser visto na Figura 45.

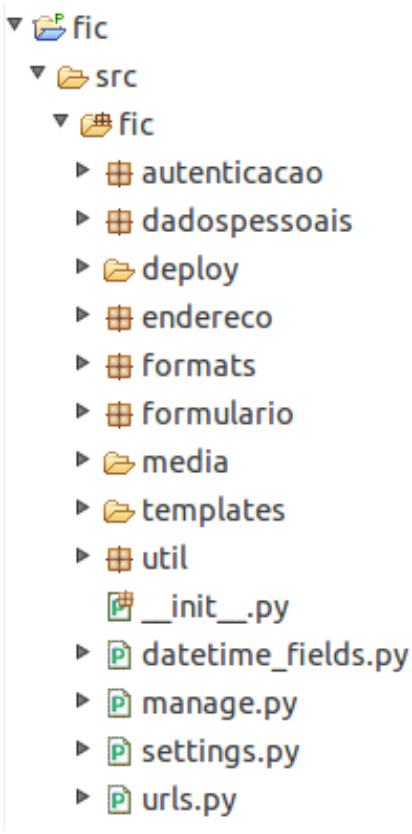

Figura 45: Estrutura do sistema FIC.
Fonte: Autoria própria.

Esta estrutura é um padrão utilizado pelos sistemas desenvolvidos no *framework* Django. Os formulários do sistema são divididos em aplicações como no caso do formulário de endereço que foi criada a aplicação “endereco”.

Como o WebService Enderecamento foi criado justamente para ser utilizado no formulário de endereço. O arquivo webserviceEnderecamento.py foi acrescentado dentro da pasta da aplicação “endereco”. Nesta mesma pasta possui os arquivos models.py e forms.py (Figura 46).

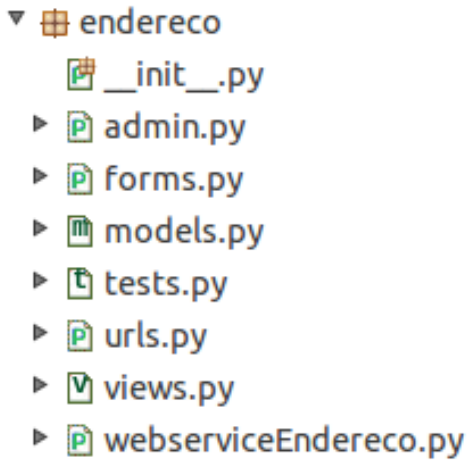

Figura 46: Estrutura da aplicação endereco.
Fonte: Autoria própria.

O arquivo models.py, contém os campos e comportamentos essenciais dos dados que será armazenando no banco de dados (DJANGO). Neste arquivo foi criada a classe de modelo do formulário endereço (Figura 47).

```
01  ...
02  class Enderecamento(models.Model):
03
04      vch_setor = models.CharField(max_length=250)
05      vch_logradouro = models.CharField(max_length=250)
06      vch_quadra_lote = models.CharField(max_length=250)
07      vch_numero = models.CharField(max_length=250)
08      vch_cep = models.CharField("CEP",max_length=250)
09      vch_complemento = models.CharField("Complemento",
10                                  max_length=250, blank=True, null=True)
11      fk_fic = models.ForeignKey(InformacaoGeral)
12      vch_tipo_endereco = models.CharField(max_length=50, default='atividade')
13  ...
```

Figura 47: Arquivo models.py.
Fonte: Autoria própria.

O arquivo forms.py sobrescreve o formulário padrão criado pelo arquivo models. Neste arquivo os campos vch_setor, vch_logradouro e vch_quadra_lote foram transformados em campos do tipo select, no campo vch_setor foi utilizado o método getSetor do WebService Enderecamento para que os campos vch_logradouro e vch_quadra_lote fossem preenchidos de acordo com o código retornado (Figura 48).

```
01  ...
02  def listaSetor():
03      setores = getSetor()
04
05      listaSetor = [('','----------'),]
06      for Setor in setores:
07          listaSetor = listaSetor + [(int(Setor.codSetor),str(Setor.setorNovo)+"
08  ("+str(Setor.setorAntigo)+")")]
09      return listaSetor
10
11
12  class EnderecoForms(forms.ModelForm):
13
14      vch_setor = forms.CharField(label='Quadra', widget=forms.Select(
15                      choices=listaSetor(),  attrs={'onchange':'$ajaxSetor(this.value,this.id)',
16                      'style':'font-size:14px;'}))
17      vch_logradouro = forms.CharField(label='Logradouro',  widget=forms.Select(
18                          choices=[('', '----------')],
19  attrs={'onchange':'$ajaxLote(this.value,this.id)',
```

```
                                'style':'font-size:14px;'}))
    vch_quadra_lote = forms.CharField(label='QI/Lote', widget=forms.Select(
                                choices=[('', '----------')], attrs={'style':'font-size:14px;'}))
    vch_numero = forms.CharField(widget=forms.TextInput(attrs={'size':'10'}),
                            label="Número",initial=0)
    vch_cep = forms.CharField(widget=forms.TextInput(attrs={'size':'10'}), label="CEP")
....
```

Figura 48: Arquivo forms.py.
Fonte: Autoria própria.

O preenchimento desses campos foi realizado via javascript, para isso foram criadas as funções javascript ajaxSetor e ajaxLote (Figura 49) que invoca o arquivo views.py.

Uma view é simplesmente uma função do Python que recebe uma Web Request e retorna uma Web Response. O arquivo views.py também faz parte da estrutura do sistema FIC e é utilizado para criar funções que serão invocadas pelo sistema.

```
(function($){
    $ajaxSetor = function(setor, id){
        var idlogradouro = id.replace('setor', 'logradouro');
        var idquadra = id.replace('setor', 'quadra_lote');
        dados = {
            "setor": setor        };
        $.ajax({
            type: "POST",
            url: '/fic/alameda/',
            dataType: "json",
            data: dados,
            success: function(retorno){
                $("#" + idquadra).empty();
                $("#" + idquadra).append('<option value="">----------</option>');
                $("#" + idlogradouro).empty();
                $("#" + idlogradouro).append('<option value="">----------</option>');

                $.each(retorno, function(i, item){
                    if (item.alamedaNovo != "") {
                        $("#" + idlogradouro).append(
                                '<option value="' + item.codAlameda + '">'+
                                item.alamedaNovo+'   ('+ item.alamedaAntigo +') </option>');
                    }
                    else {
                        $("#" + idlogradouro).append('<span><b>Não Existe</b></span>');
                    }
                });
            },
            error: function(erro){
                alert('Erro.');
            }
        });
    }
})(django.jQuery);

(function($){
```

```
37      $ajaxLote = function(alameda, id){
38        var idsetor = id.replace('logradouro', 'setor');
39         var idquadra = id.replace('logradouro', 'quadra_lote');
40         dados = {
41             "setor": $("#"+idsetor).val(),
42              "alameda":alameda
43         };
44         $.ajax({
45             type: "POST",
46             url: '/fic/quadra/',
47             dataType: "json",
48             data: dados,
49             success: function(retorno){
50                 $("#" + idquadra).empty();
51                 $("#" + idquadra).append('<option value="">----------</option>');
52                 $.each(retorno, function(i, item){
53                     if (item.numero != "") {
54                         $("#" + idquadra).append('<option value="' + item.codLote + '">'+
55                                 item.quadraInterna+' (LOTE:'+ item.numero +' )</option>');
56                     }
57                     else {
58                         $("#" + idquadra).append('<span><b> Não Existe </b></span>');
59                     }
60                 });
61             },
62             error: function(erro){
63                 alert('Erro.');
64             }
65         });
66     }
67 })(django.jQuery);
```

Figura 49: Arquivo javascript endereco.js.
Fonte: Autoria própria.

Na views foi criado o método listarAlameda (Figura 50) que recebe o valor do setor e invoca a função getAlameda do WebService. Essa função retorna uma lista com todas as alamedas do setor selecionado para o javascript, que sobrescreve o campo vch_logradouro com a lista de alamedas.

```
01  ...
02  def listarAlameda(request):
03
04      if request.method == 'POST':
05          setor = int(request.POST.get('setor'))
06          alamedas = getAlameda(setor)
07          listaAlameda = []
08          for alameda in alamedas:
09              listaAlameda = listaAlameda + [{'codAlameda':alameda.codAlameda,
10                                      'alamedaNovo':alameda.alamedaNovo,
11                                      'alamedaAntigo':alameda.alamedaAntigo}]
12          json = simplejson.dumps(listaAlameda)
13          return HttpResponse(json, mimetype="application/json")
14  ...
```

Figura 50: Método listarAlameda.
Fonte: Autoria própria.

Na views foi criado também o método listarQuadraLote (Figura 51) que recebe o valor do setor e do logradouro e invoca a função getQuadraQiConjLote do WebService. Esta função retorna uma lista com todas as quadras, QIs e lotes do setor e da alameda selecionada para o javascript, que sobrescreve o campo vch_quadraLote com esta lista.

```
01  ...
02  def listarQuadraLote(request):
03
04      if request.method == 'POST':
05          setor = int(request.POST.get('setor'))
06          alameda = int(request.POST.get('alameda'))
07          quadras = getQuadraQiConjLote(setor,alameda)
08          listaQuadras = []
09          for quadra in quadras:
10              listaQuadras = listaQuadras + [{'codLote':quadra.codLote,
11                                              'quadraInterna':quadra.quadraInterna,
12                                              'numeroLoteAlameda':quadra.numeroLoteAlameda,
13                                              'numero':quadra.numero}]
14          json = simplejson.dumps(listaQuadras)
15          return HttpResponse(json, mimetype="application/json")
16  ...
```

Figura 51: Método listarQuadraLote.
Fonte: Autoria própria.

O formulário de endereço do sistema FIC ficou como mostra a Figura 52. Entre parênteses ficou a nomenclatura do antigo endereço.

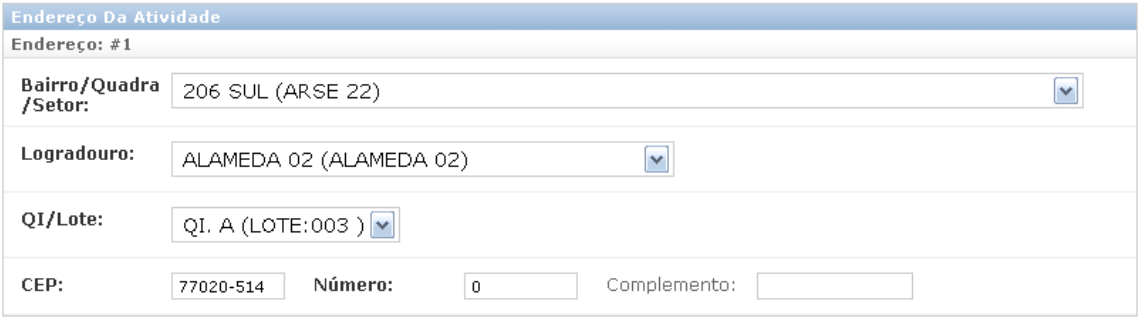

Figura 52: Formulário de endereço.
Fonte: Autoria própria.

6 CONCLUSÃO

A utilização dos conceitos da arquitetura orientada a serviço, junto com a tecnologia WebService mostrou uma solução viável na integração de sistemas.

A utilização de SOA ajudou na identificação do serviço de endereçamento e com a tecnologia WebService foi gerado esse serviço.

Um dos problemas identificados na Prefeitura Municipal de Palmas era a dificuldade de preenchimento do endereço de Palmas, presente em vários sistemas internos. O sistema FIC foi o primeiro sistema a utilizar-se da tecnologia de WebService, a fim de testá-la.

Assim, o sistema FIC e o WebService já foram colocados em produção, e a prefeitura já conta com a vantagem de que agora, seus sistemas internos já podem utilizar-se dessa tecnologia em qualquer outro sistema, além de que, caso a prefeitura deseje, poderá disponibilizá-lo futuramente, para que qualquer outro sistema externo, independente de plataforma ou linguagem o utilize desde que esta linguagem dê suporte à tecnologia de WebService.

Portanto, concluiu-se que a implementação de SOA e Webservice na Prefeitura Municipal de Palmas foram satisfatórios e que o conceito de serviços é uma boa solução para as organizações de modo geral.

6.1 TRABALHOS FUTUROS

Como trabalho futuro será implementado o serviço de autenticação utilizando a mesma técnica e tecnologida utilizada no serviço de endereçamento.

E será configurada uma camada de sergurança no serviço de endereçamento, para que os dados desse serviço sejam disponíveis na internet com segurança.

REFERÊNCIAS

AUTOCEP, Disponível em: <http://www.autocep.com.br/autocep_web_service.html/>. Acesso em: 04/01/2011.

CORREIOS. **WebService correios**. Disponível em: <http://www.correios.com.br/webservices/default.cfm>. Acesso em: 16/12/2010.

DEITEL, H.M. and Deitel, P.J. **Java como programar**. São Paulo: Editora Pearson, 2007.

DJANGO. **Django v1.0 documentation.** Disponível em: <http://docs.djangobrasil.org/faq/index.html>Acesso em: 21/07/2010.

E-STF. **e-STF Processo Eletrônico - WebService 2.1.2**. Disponível em: <http://www.stf.jus.br/portal/cms/verTexto.asp?servico=estf&pagina=WSProcessoEletronico>. Acesso em: 04/01/2011.

GOMES, Daniel Adorno. **Web services SOAP em Java**. São Paulo: Editora Novatec, 2009.

GONÇALVES, Edson. **Desenvolvendo Aplicações Web com NetBeans IDE 6.** Rio de Janeiro: Editora Ciência Moderna Ltda, 2008.

GOVERNO ELETRÔNICO. **O que é Interoperabilidade?**. Disponível em: <http://www.governoeletronico.gov.br/acoes-e-projetos/e-ping-padroes-de-interoperabilidade/o-que-e-interoperabilidade/>. Acesso em: 30/08/2010.

HURWITZ, Judith, BLOOR, Robin, KAUFMAN, Marcia e HALPER, Fern. **Arquitetura Orientada a Serviço - SOA para Leigos**. Rio de Janeiro: Editora Alta Books, 2009.

LUTZ, Mark and ASCHER, David. **Aprendendo Python.** São Paulo: BOOKMAN COMPANHIA ED, 2007.

MARZULLO, Fabio Perez. **SOA na prática: inovando seu negócio por meio de soluções orientadas a serviços**. São Paulo: Editora Novatec, 2009.

MEDEIROS, Manoel Pimentel. **JSTL – Implementando um pool de conexões**. Disponível em: < http://www.devmedia.com.br/articles/viewcomp.asp?comp=4113>. Acesso em: 26/01/2011.

NETBEANS. **O que é o NetBeans?** Disponivel em: < http://netbeans.org/index_pt_PT.html>. Acesso em: 22/05/2010

PYTHON. **Python Documentation**. Disponivel em: < http://docs.python.org/faq/general.html#id1>. Acesso em: 22/05/2010

RECKZIEGEL, Mauricio**. Descrevendo, descobrindo e integrando Web Services - UDDI.** Disponível em: <http://imasters.uol.com.br/artigo/4474/webservices/descrevendo_descobrindo_e_integrando_web_services_-_uddi/>. Acesso em: 30/09/2010.

RODRIGUES, Martius V. e FERRANTE, Agustin Juan. **Tecnologia de Informação e Gestão Empresarial**. Rio de Janeiro: E-Papers, 2000.

SAMPAIO, Cleuton. **SOA e WebServices em Java**. Rio de Janeiro: Editora Brasport, 2006.

SIAPE. **Consulta SIAPE**. Disponível em: <http://catalogo.governoeletronico.gov.br/pasta_servico/consulta-siape/>. Acesso em: 07/01/2011.

SMART**,** John Ferguson. **Web Services Made Easy with JAX-WS 2.0**. Disponível em: <http://java.net/article/2006/06/12/web-services-made-easy-jax-ws-20>. Acesso em: 21/07/2010.

SOSNOSKI, Dennis. **Java Web services: Axis2 WS-Security basics** . Disponível em: <http://www.ibm.com/developerworks/java/library/j-jws4/index.html>. Acesso em: 07/01/2011.

SUDS. Disponível em: <https://fedorahosted.org/suds/wiki/Documentation> Acesso em: 21/07/2010.

UDDI. **Universal Description Discovery and Integration**. Disponível em: <http://uddi.xml.org/uddi-101>. Acesso em: 07/10/2010.

W3C. **World Wide Web Consortium**. Disponível em: <http://www.w3.org/>. Acesso em: 20/07/2010.

XML. **Extensible Markup Language**. Disponível em: <http://www.w3.org/XML/>. Acesso em: 30/08/2010.