**DEFINIÇÃO DE ARQUITETURA ORIENTADA A SERVIÇOS (SOA), UTILIZANDO FERRAMENTAS OPEN SOURCE**

Rodolfo Leopoldo da Cunha Rodrigues Martins

Professor(a) Orientador(a): Rafael Lobato

Arquitetura de Software

**RESUMO**

A grande necessidade das organizações de integrar as informações de várias áreas, ou departamentos, tem aumentado, cada vez mais, a busca de ferramentas SOA que permitem, entre outros, integrar as informações das aplicações de várias áreas de forma eficiente. Atualmente existem diversas ferramentas pagas, já consolidadas no mercado para a aplicação de SOA, porém essas ferramentas possuem custo muito elevado, muitas vezes inviabilizando sua utilização. Além disso, é encontrado em várias literaturas que a utilização de ferramentas Open Source, para se atingir SOA, tem um grau de complexidade maior, com isso, se torna mais difícil criar esse tipo de arquitetura Open Source. Com base nas principais características do padrão SOA, serão apresentadas ferramentas Open Source que podem ser utilizadas para atender SOA e criada uma definição de arquitetura SOA, utilizando apenas ferramentas Open Source.

**Palavras-Chave:** Arquitetura. SOA. Open Source.

**ABSTRACT**

The great need for organizations to integrate information from various areas or departments has increased, more and more, the search for SOA tools that allow, among others, integrate information of applications in different areas efficiently. Currently there are several paid tools, already consolidated in the market for the application of SOA, but these tools have very high cost, often making it impossible to use. Furthermore, it is found in various literatures that the use of open source tools to achieve SOA has a higher degree of complexity, it becomes more difficult to create such an architecture Open Source. Based on the main standard features of SOA will be presented Open Source tools that can be used to meet SOA and created a definition of SOA architecture, using only open source tools.

**Keywords:** Architecture. SOA. Open Source.

1. **INTRODUÇÃO**

Devido a grande complexidade em que os projetos de TI se tornaram, houve a necessidade de se pensar em uma forma de simplificar os projetos, principalmente em relação a integração de aplicações, que no final das contas, estava se tornando um emaranhado de aplicações falando diretamente umas com as outras, de forma caótica, o que tornava o reaproveitamento, manutenção, possibilidade de evoluções tarefas muitos custosas. Com isso, surgiu SOA, que significa Service-Oriented Architecture, ou Arquitetura Orientada a Serviços, com soluções e padrões para resolver esses problemas.

Com a evolução tecnológica, grandes empresas de TI, como IBM, Oracle, Microsoft, Tibco, entre outras criaram ferramentas que possibilitaram a criação de soluções seguindo o padrão SOA. Além das empresas citadas, com ferramentas pagas, também surgiram soluções Open Source, de empresas como, MuleSoft, WSO2, entre outras. O problema de pesquisa deste trabalho busca demonstrar que é possível criar uma arquitetura SOA utilizando apenas ferramentas Open Source.

Para o propósito desse trabalho, foi apresentado um estudo de caso a fim de demonstrar o uso da aplicação de SOA para a solução do problema. Como SOA compreende uma gama muito grande de técnicas, padrões e tecnologias, essa foi a melhor forma encontrada para demonstrar de forma pratica a sua aplicação.

Este trabalho tem como objetivo identificar as principais características, componentes e padrões SOA, para então propor soluções arquiteturais utilizando apenas ferramentas Open Source.

Neste contexto, a escolha do WSO2 foi realizada devido a sua popularidade e por possuir um conjunto muito grande de ferramentas muito bem documentadas, assim como outras ferramentas importantes, o WSO2 Service Bus.

Dessa forma, este trabalho terá como objetivos os itens a seguir:

* Realizar um levantamento dos principais componentes, padrões e características de uma arquitetura SOA;
* Apresentar as ferramentas Open Source que podem ser utilizadas para aplicar as técnicas do item anterior;
* Apresentar um estudo de caso, que seja relevante arquiteturalmente do ponto de vista SOA, a fim de apresentar uma arquitetura para resolver esse problema;
* Utilizar ferramentas Open Source a fim de aplicar os componentes, padrões e características, tendo como objetivo a criação de uma arquitetura a fim de consolidar a aplicação das técnicas e ferramentas propostas para atender os padrões SOA.

Este trabalho apresenta-se dividido em 5 seções, descritas a seguir. Na seção 2 são apresentadas as bases teóricas, na seção 3 são apresentadas as ferramentas utilizadas, na seção 4 é apresentado o estudo de caso, na seção 5 são apresentadas as técnicas e ferramentas para resolver o estudo de caso e na seção 6 são apresentados os resultados e conclusões.

1. **REFERENCIAL TEÓRICO**
   1. SOA

Em abril de 1996 Roy Schulte e Yefim Natis do Garter Group propuseram o conceito de SOA pela primeira vez no artigo Service Oriented Architectures. Eles o apresentaram à partir da análise de experiências de diversos clientes que, na época, utilizavam a tecnologia cliente-servidor (em forte adoção naqueles anos), e que ganhou novamente atenção em virtude das novas possibilidades tecnológicas baseadas em padrões, da demanda crescente por soluções de integração e de relativo insucesso de outras alternativas.

Para o Gartner Group, SOA é uma abordagem arquitetural corporativa que permite a criação de serviços de negócio interoperáveis que podem facilmente ser reutilizados e compartilhados entre aplicações e empresas.

Para a OASIS, SOA é um paradigma para organizar e utilizar capacidades distribuídas que podem estar sob o controle de diferentes domínios de propriedade. Provê uma maneira uniforme de oferecer, descobrir, interagir e utilizar capacidades para produzir efeitos desejados consistentes com pré-condições e expectativas mensuráveis.

De acordo com a Wikipedia, SOA é um estilo de arquitetura de software cujo princípio fundamental prega que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços. Frequentemente estes serviços são conectados através de um "barramento de serviços" que disponibiliza interfaces, ou contratos, acessíveis através de web services ou outra forma de comunicação entre aplicações.

A partir dessas definições podemos chegar a algumas conclusões à respeito de SOA:

* SOA não é uma tecnologia. Há tanto de negócio quanto de tecnologia em SOA. As tecnologias (padrões) que dão suporte a SOA são o que a viabiliza, mas SOA não é uma tecnologia por si só;
* SOA não é uma metodologia. Há várias metodologias (processos, ferramentas, métodos de trabalho) que podem ser usados para implantar SOA com sucesso;
* SOA pode ser considerada uma filosofia arquitetural. SOA é uma linha de pensamento que permeia a implementação de necessidades de negócio, refletida em diretrizes, políticas e metodologias corporativas, não necessariamente restritas à área de TI;
* SOA não é algo que se possa comprar ou instalar;
* SOA não é um webservice;
* SOA não cria nada. Ela apenas sugere, propõe, define;
* SOA baseia-se no conceito do uso de serviços atômicos, independentes e com baixo acoplamento.

Benefícios de SOA:

* Facilidade de Manutenção: mudanças na lógica de negócios (implementação) não afetam aplicações existentes;
* Reuso: novas aplicações e processos (consumidores de serviços) podem reaproveitar mais facilmente as funcionalidades existentes;
* Flexibilidade: sistemas de back-end e infraestrutura podem ser substituídos com menor impacto;
* Resultado: agilidade e redução de custos;
* Qualidade: garantia de homogeneidade de processos;
* Menor tempo: agilidade na análise de impacto e no desenvolvimento evolutivo de seus sistemas;
* Menor custo: redução do custo de manutenção das aplicações;
* Controle: conhecimento dos ativos existentes.
  1. Componentes de SOA

A seguir serão apresentados os principais componentes de uma arquitetura SOA:

* + 1. **ESB (Enterprise Service Bus)**

O Enterprise Service Bus (ESB), também chamado de Barramento Corporativo de Serviços, é considerado o coração de uma infraestrutura SOA. Suas principais responsabilidades são as definições de interface de entrada e saída, transformação de mensagens, tratamento de exceções e monitoramento de mensagens.

Entre as suas principais características podem ser destacadas:

* Flexibilidade: É a capacidade de se adequar com facilidade à necessidade do cliente/empresa, com um alto nível de parametrização e conceitos que permitem uma mudança rápida das regras de negócio.
* Escalabilidade: Permite escalar a aplicação de acordo com o volume de informações, através da aplicação de clusters nos servidores e balanceamento de carga para não sobrecarregar somente um servidor.
* Segurança: Garante a autenticidade da informação, pode ser aplicada utilizando SSL nas chamadas de serviços usando o protocolo https.
* Interoperabilidade: Permite que um serviço se comunique com outros serviços de forma transparente. Para que um serviço seja considerado interoperável é importante que ele trabalhe com os padrões XSD, BPEL, WSDL, XML, etc. Dessa forma a empresa fica preparada para se conectar a aplicativos ou serviços já em execução desenvolvida em qualquer plataforma ou linguagem de programação.
* Monitoração: Permite monitorar atividades corporativas no barramento, com o BAM - Business Activity Monitoring pode ser verificado quais serviços mais usados, o tempo de execução, quem esta usando os serviços. Assim pode se detectar gargalos no barramento.
  + 1. **BAM (Business Activity Monitoring)**

Fornece informações e estatísticas de quais processos de negócios estão sendo executados e quais estão parados, permitindo monitorar onde realmente possíveis falhas podem acontecer no processo de negócio.

* + 1. **Web Services**

Tecnologia que permite a criação de serviços interoperáveis, ou seja, são independentes de tecnologia, quem consome não precisa necessariamente conhecer como o serviço implementado, mas sim conhecer o WSDL (Web Service Description Language), que é um arquivo XML que define as interfaces dos serviços, através dele são definidos parâmetros de entrada, saída, tipos de dados, endereço do serviço, etc. Para a definição dos tipos de dados, são usados XSDs(XML Schema Definition), descreve tipos de dados básicos e complexos e regras nas quais um determinado arquivo XML deve estar em conformidade para que possa ser válido. Por fim, podemos definir XML como uma notação que foi padronizada pela W3C de fácil leitura por humanos, amplamente utilizado na descrição e trafego de dados.

* + 1. **SOAP**

É um protocolo básico de Web Service que segue um formato baseado em XML, ele define o formato de cabeçalho e o corpo de uma mensagem, uma das principais vantagens é que ele é transportado pelo protocolo HTTP.

* + 1. **BPEL (Business Process Execution Language)**

É uma linguagem que segue um formato em XML usado para realizar a orquestração de serviços, o resultado dessa orquestração é um novo Web Service.

* + 1. **UDDI (Universal Description Discovery and Integration)**

É uma especificação técnica que tem como objetivo descrever, descobrir e integrar Web Service. É um diretório onde podemos encontrar todos os serviços disponibilizados para consumo.

* + 1. **Orquestração**

É uma maneira de fazer com que os serviços possam se comunicar entre si, o resultado da orquestração é um arquivo BPEL que gera um novo serviço.

* 1. Características de serviços SOA

SOA possui algumas ideias principais que precisam ser atendidas para a adoção pelas organizações:

* Um conjunto de serviços que o negocio deseja prover para seus clientes, parceiros, ou outras áreas da organização.
* Um estilo arquitetural que requer provedor de serviço, mediação, e requisitor de serviços com um descritor de serviços.
* Um conjunto de princípios arquiteturais, padrões e critérios que atendem características como modularidade, encapsulamento, baixo acoplamento, separação de responsabilidades, reuso e composição.
* Modelo de programação com padrões, ferramentas e tecnologias que suportam web services, serviços REST e outros tipos de serviços.
* Middleware otimizado para a construção de serviços, orquestração, monitoramento e gerenciamento.
  1. Service Component Architecture (SCA)

De acordo com a Oasis, SCA é um conjunto de especificações que descrevem o modelo para construção de aplicações e sistemas usando SOA. SCA complementa abordagens anteriores para implementar serviços, e baseia-se em padrões abertos como Web services.

SCA é baseado na ideia que negocio é provido por uma série de serviços, os quais são construídos em conjunto para criar soluções que servem para uma necessidade de negocio em particular. Essa composição de aplicações podem conter novos serviços construídos especificamente para a aplicação e também funções negociais de sistemas existentes, reusados como parte da composição. SCA provê um modelo que permite a criação dessas soluções.

SCA tem como objetivo englobar um grande conjunto de tecnologias para componentes de serviço e para métodos de acesso que são usados para conectá-los. Para componentes isso inclui, diferentes linguagens de programação, frameworks e ambientes usados com essas tecnologias. Para métodos de acesso, SCA compositions permite o uso de várias comunicações e tecnologias de acesso a serviços que são mais comumente usadas, incluindo, por exemplo, Web services, Sistema de mensagens e remote procedure call (RPC).

* 1. Princípios de Design de Serviços

Princípios de design são práticas que definem diretrizes aceitas pelo mercado. Essas diretrizes são comparadas ao que também chamamos de melhores práticas.

Em SOA a construção de soluções orientadas a serviços de negócio que possam ser reutilizados gerando rapidez e redução de custos de TI. É uma tarefa complexa para empresas que ainda não possuem um nível de maturidade adequado. Neste sentido, a utilização dos princípios de design de serviços pode diminuir a complexidade inerente a esse modelo de arquitetura.

Em SOA, são nove os princípios de design que devem ser levados em consideração durante os projetos SOA, conforme ilustrado na figura 1.

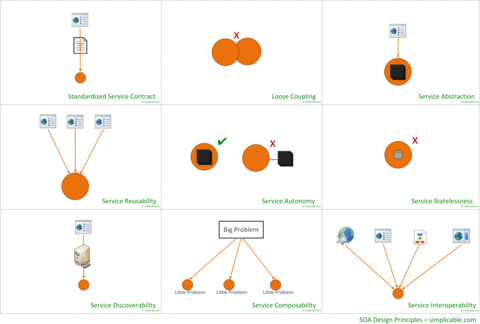


Figura 1: Princípios de Design de Serviços

Fonte: <http://simplicable.com/new/the-9-principles-of-soa-design>

1. **WSO2**

A plataforma WSO2 é 100% Open Source, possui uma gama de ferramentas que permite aplicar o padrão SOA na sua totalidade, a seguir serão destacadas as suas principais ferramentas e características conforme figura 2.



Figura 2: Componentes do WSO2

Fonte: <http://wso2.com/>

A seguir serão apresentadas as funções das principais funcionalidades do WSO2 que serão utilizadas na aplicação da prova de conceito:

* + 1. **WSO2 Enterprise Service Bus**

Entre as principais características do seu ESB podem ser destacadas:

* Conectividade: Suporta várias formas de transporte, protocolos, adaptadores para sistemas legados, adaptadores para serviços na nuvem:
  + Transporte: HTTP, HTTPS, TCP, SFTP, ETC;
  + Formatos e protocolos: JSON, XML, SOAP 1.1, SOAP 1.2, WS-\*, HTML, EDI, CORBA/IIOP, ETC;
  + Adaptadores para sistemas legados: SAP BAPI & IDoc, PeopleSoft, MS Navision, IBM WebSphere MQ, Oracle AQ, MSMQ;
  + Adaptadores para serviços na nuvem: Salesforce, Paypal, LinkedIn, Twitter, JIRA;
* Roteamento, mediação e transformação:
  + Roteamento: Baseado no cabeçalho, baseado no conteúdo, baseado em regras e baseado em prioridade;
  + Mediação: EIPs (filtros de mensagens, garantia de entrega, enriquecimento de mensagens, etc), integração com banco de dados, publicação de eventos, logging e auditoria, validação;
  + Transformação: XSLT 1.0/2.0, XPath, XQuery, Smooks
* Mensagem, Serviços, API e segurança:
  + Permite expor serviços e aplicações existentes através de diferentes protocolos e formatos de mensagens.
  + Virtualização de serviços provendo baixo acoplamento e governança SOA.
  + Balanceamento de carga para escalabilidade e tratamento de falha provendo alta disponibilidade do negocio.
  + Cria fachadas de serviços para legados e serviços não padronizados.
  + Garante e gerencia segurança centralizada, incluindo autorização e autenticação.
  + Expõe serviços e aplicações via RESTful APIs;
  + Canal SSL para suporte de cenários de entrada e saída;
* Alta performance, alta disponibilidade, escalabilidade e estabilidade;
* Desenvolvimento amigável e de fácil deploy;
* Gerenciamento e monitoramento;
  + 1. **WSO2 Application Server**

Servidor de aplicação que provê tecnologias Open Source para Web Applications(Apache Tomcat), Web Services(Apache Axis2), serviços RESTful(JAX-RS).

* + 1. **WSO2 Data Services Server**

Ferramenta que provê a integração de data store, criando data views, e hospedando data services. Permite data access e integração com processos de negocio, business intelligence e aplicações mobile. WSO2 Data Server suporta segurança e gerenciamento de acesso, transações, transformação de dados e validação usando lighweight.

1. **Estudo de Caso**

Para a proposta desse trabalho, será realizada a modelagem de um serviço para cadastro e consulta de imóveis para aluguel. Esse serviço será disponibilizado na BUS para consumo de aplicativos em diversas tecnologias e plataformas. Esta escolha foi feita devido a grande quantidade de requisitos arquiteturalmente significativos que esse tipo de serviço exige, demonstrando de forma pratica a aplicação de conceitos SOA. Esse trabalho não tem foco na apresentação dos dados por aplicativos cliente, o consumo do serviço será demonstrado através de ferramentas Open Source. Os demais serviços, como Cliente e Aluguel, podem ser usados como exercício para os leitores do trabalho, para exercitar de forma prática os conceitos apresentados.

A modelagem será baseada em requisitos levantados utilizando o diagrama de caso de uso, baseado na notação UML, apresentado pela figura X .

***\*Adicionar o Diagrama de caso de uso***

Os requisitos arquiteturalmente importantes para o domínio do serviço proposto esta descrito utilizando o modelo FURPS+ na tabela 1.

|  |  |  |
| --- | --- | --- |
| **Código** | **Requisito Arquitetural** | **FURPS+** |
| R01 | O sistema deve ser independente de plataforma | Interoperabilidade |
| R02 | O sistema deve permitir a criação de novos serviços sem afetar os serviços existentes | Extensibilidade |
| R03 | O sistema deve permitir a composição de serviços para a criação de novos serviços | Composição |
| R04 | O sistema deve permitir alterar a implementação de serviços existentes sem afetas os clientes | Gerenciabilidade |
| R05 | O sistema deve suportar aumento de carga de acordo com a necessidade | Escalabilidade |
| R06 | O sistema deve estar disponível 24/7 em 99% do tempo. | Disponibilidade |
| R08 | O sistema deve permitir monitoramento através de ferramentas de análise | Logging |

Tabela 1: Requisitos Arquiteturais

Fonte: Elaborado pelo Autor (2015)

1. **Arquitetura Proposta**

A partir dos requisitos arquiteturais descritos na tabela 1 e o objetivo principal desse trabalho, será criada uma prova de conceito que utilizará como solução principal as ferramentas do WSO2 apresentado nesse artigo. Essa ferramenta foi escolhida, pois possui todas as características para atender os requisitos arquiteturais e atende a proposta desse trabalho.

O projeto será realizado utilizando a ferramenta de modelagem Open Source Astah, amplamente adotada no mercado. Para implementação, será utilizada a IDE Open Source Eclipse para codificação, e para a construção e gerenciamento de dependências será utilizado o Maven. O banco de dados utilizado será o Postgres, e para a sua construção e gerenciamento a IDE PgAdmin. Os fontes do projeto serão disponibilizados pelo servidor de controle de versão github, que também é Open Source para a construção de projetos de código aberto. O endereço com os fontes do projeto se encontram em: https://github.com/rodolfolcrm/soa-opensource.git.

Na figura 1 é apresentado o diagrama de implantação, demonstrando a colaboração entre os componentes da solução.

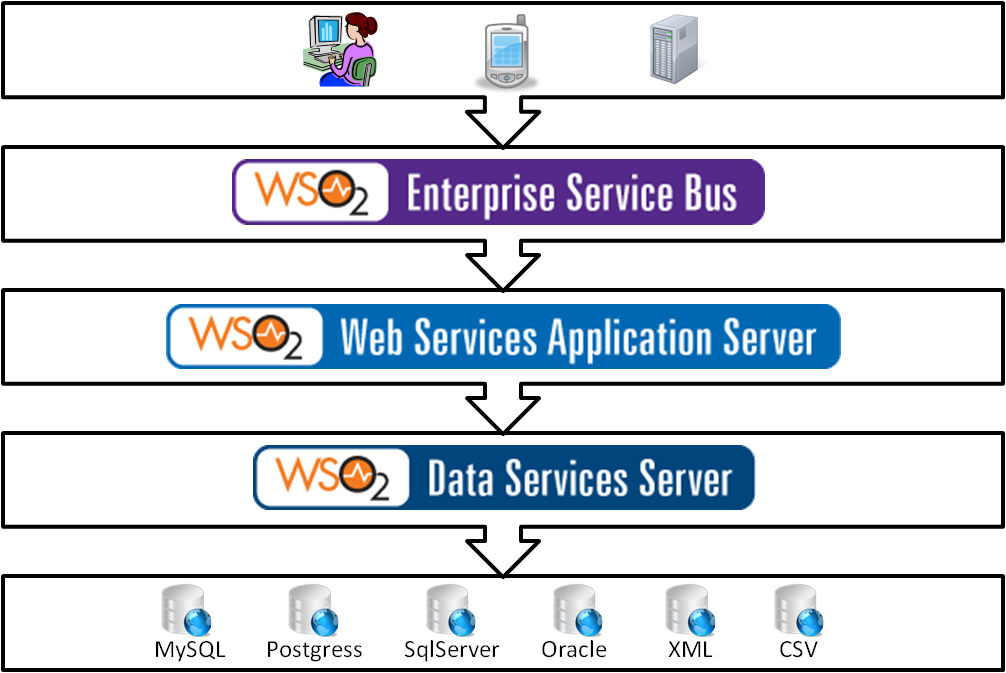


Figura 3: Diagrama de implantação da solução

Fonte: <http://wso2.com/>

A primeira etapa do trabalho é a criação de um diagrama de entidade e relacionamento MER, para representar como o formato dos dados persistidos, esse diagrama é representado pela figura 4.

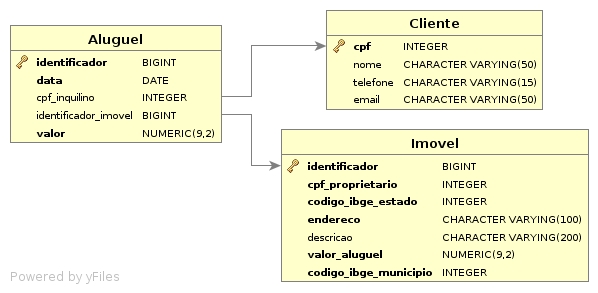


Figura 4: Diagrama de entidade e relacionamento

Fonte: Elaborado pelo Autor (2015)

O SGBD escolhido para a implementação desse trabalho é o Postgres, por ser Open Source e de grande aceitação no mercado. Segue a seguir o Script para a criação do banco de dados.

**CREATE** **TABLE** "Cliente"

(

cpf **integer** **NOT** **NULL**,

nome **character** **varying**(50),

telefone **character** **varying**(15),

email **character** **varying**(50),

**CONSTRAINT** "ClientePK" **PRIMARY** **KEY** (cpf)

)

**CREATE** **TABLE** "Imovel"

(

identificador bigint **NOT** **NULL**,

cpf\_proprietario **integer** **NOT** **NULL**,

codigo\_ibge\_estado **integer** **NOT** **NULL**,

endereco **character** **varying**(100) **NOT** **NULL**,

descricao **character** **varying**(200),

valor\_aluguel **numeric**(9,2) **NOT** **NULL**,

codigo\_ibge\_municipio **integer** **NOT** **NULL**,

**CONSTRAINT** "PK\_Imovel" **PRIMARY** **KEY** (identificador),

**CONSTRAINT** "FK\_Imovel\_Cliente" **FOREIGN** **KEY** (cpf\_proprietario)

**REFERENCES** "Cliente" (cpf) **MATCH** SIMPLE

**ON** **UPDATE** **NO** **ACTION** **ON** **DELETE** **NO** **ACTION**

)

**CREATE** **TABLE** "Aluguel"

(

identificador bigint **NOT** **NULL**,

data **date** **NOT** **NULL**,

cpf\_inquilino **integer**,

identificador\_imovel bigint,

valor **numeric**(9,2) **NOT** **NULL**,

**CONSTRAINT** "PK\_Aluguel" **PRIMARY** **KEY** (identificador),

**CONSTRAINT** "FK\_Aluguel\_Cliente" **FOREIGN** **KEY** (cpf\_inquilino)

**REFERENCES** "Cliente" (cpf) **MATCH** SIMPLE

**ON** **UPDATE** **NO** **ACTION** **ON** **DELETE** **NO** **ACTION**,

**CONSTRAINT** "FK\_Aluguel\_Imovel" **FOREIGN** **KEY** (identificador\_imovel)

**REFERENCES** "Imovel" (identificador) **MATCH** SIMPLE

**ON** **UPDATE** **NO** **ACTION** **ON** **DELETE** **NO** **ACTION**

)

A próxima etapa da solução será a criação do DataService utilizando a ferramenta WSO2 Data Services Server. Para a proposta do trabalho, será apresentado a modelagem do DataService mais significativo, que é o ImovelDataService. Esse componente tem a responsabilidade de desacoplar o acesso à origem dos dados da aplicação, através da criação de serviços com essa responsabilidade, dessa forma, o acesso aos dados fica centralizado em serviços, que são utilizados pelas aplicações, evitando que essas, repliquem a logica de acesso aos dados, aumentando a coesão e diminuindo o acoplamento. Dessa forma, serão atendidos os requisitos R01, R02, R03 e R04 na camada de acesso a dados. A ferramenta WSO2 permite que o acesso a dados seja criado sem a necessidade de codificação, tudo pode ser criado dentro da própria ferramenta visual de forma bem pratica e amigável. A ferramenta permite também que o Data Service seja exportado e importado em formato xml. Segue o xml do DataService criado.

<data name=*"ImovelDataService"* serviceNamespace=*"ImovelDataService"* transports=*"http https local"*>

<config id=*"imovelDS"*>

<property name=*"driverClassName"*>org.postgresql.Driver</property>

<property name=*"url"*>jdbc:postgresql://localhost:5432/postgres</property>

<property name=*"username"*>postgres</property>

<property name=*"password"*>postgres</property>

</config>

<query id=*"findAll"* useConfig=*"imovelDS"*>

<sql>SELECT identificador, cpf\_proprietario, codigo\_ibge\_estado, endereco, descricao, valor\_aluguel, codigo\_ibge\_municipio FROM "Imovel"</sql>

<result element=*"Entries"* rowName=*"Entry"*>

<element column=*"identificador"* name=*"identificador"* xsdType=*"long"*/>

<element column=*"cpf\_proprietario"* name=*"cpf\_proprietario"* xsdType=*"long"*/>

<element column=*"codigo\_ibge\_estado"* name=*"codigo\_ibge\_estado"* xsdType=*"integer"*/>

<element column=*"endereco"* name=*"endereco"* xsdType=*"string"*/>

<element column=*"descricao"* name=*"descricao"* xsdType=*"string"*/>

<element column=*"valor\_aluguel"* name=*"valor\_aluguel"* xsdType=*"decimal"*/>

<element column=*"codigo\_ibge\_municipio"* name=*"codigo\_ibge\_municipio"* xsdType=*"integer"*/>

</result>

</query>

<query id=*"**findByDescricao"* useConfig=*"imovelDS"*>

<sql>SELECT identificador, cpf\_proprietario, codigo\_ibge\_estado, endereco, descricao, valor\_aluguel, codigo\_ibge\_municipio FROM "Imovel" where descricao like :descricao</sql>

<result element=*"Entries"* rowName=*"Entry"*>

<element column=*"identificador"* name=*"identificador"* xsdType=*"long"*/>

<element column=*"cpf\_proprietario"* name=*"cpf\_proprietario"* xsdType=*"long"*/>

<element column=*"codigo\_ibge\_estado"* name=*"codigo\_ibge\_estado"* xsdType=*"integer"*/>

<element column=*"endereco"* name=*"endereco"* xsdType=*"string"*/>

<element column=*"descricao"* name=*"descricao"* xsdType=*"string"*/>

<element column=*"valor\_aluguel"* name=*"valor\_aluguel"* xsdType=*"decimal"*/>

<element column=*"codigo\_ibge\_municipio"* name=*"codigo\_ibge\_municipio"* xsdType=*"integer"*/>

</result>

<param name=*"descricao"* ordinal=*"1"* sqlType=*"STRING"*/>

</query>

<query id=*"addImovelQuery"* useConfig=*"imovelDS"*>

<sql>INSERT INTO "Imovel"(identificador, cpf\_proprietario, codigo\_ibge\_estado, endereco, descricao, valor\_aluguel, codigo\_ibge\_municipio) &#13; VALUES (:identificador, :cpf\_proprietario, :codigo\_ibge\_estado, :endereco, :descricao, :valor\_aluguel, :codigo\_ibge\_municipio)</sql>

<param name=*"identificador"* sqlType=*"BIGINT"*/>

<param name=*"cpf\_proprietario"* sqlType=*"BIGINT"*/>

<param name=*"codigo\_ibge\_estado"* sqlType=*"INTEGER"*/>

<param name=*"endereco"* sqlType=*"STRING"*/>

<param name=*"descricao"* sqlType=*"STRING"*/>

<param name=*"valor\_aluguel"* sqlType=*"DOUBLE"*/>

<param name=*"codigo\_ibge\_municipio"* sqlType=*"INTEGER"*/>

</query>

<operation name=*"findAll"*>

<call-query href=*"findAll"*/>

</operation>

<operation name=*"findByDescricao"*>

<call-query href=*"findByDescricao"*>

<with-param name=*"descricao"* query-param=*"descricao"*/>

</call-query>

</operation>

<operation name=*"addImovel"*>

<call-query href=*"addImovelQuery"*>

<with-param name=*"identificador"* query-param=*"identificador"*/>

<with-param name=*"cpf\_proprietario"* query-param=*"cpf\_proprietario"*/>

<with-param name=*"codigo\_ibge\_estado"* query-param=*"codigo\_ibge\_estado"*/>

<with-param name=*"endereco"* query-param=*"endereco"*/>

<with-param name=*"descricao"* query-param=*"descricao"*/>

<with-param name=*"valor\_aluguel"* query-param=*"valor\_aluguel"*/>

<with-param name=*"codigo\_ibge\_municipio"* query-param=*"codigo\_ibge\_municipio"*/>

</call-query>

</operation>

</data>

Como pudemos observar, a criação de um Data Service é muito simples, bastando a definição do data source, da query, da operação que será publicada como serviço, e dos parâmetros de entrada e saída das queries e operações.

Agora que já temos a definição do serviço de acesso aos dados, podemos criar o serviço que vai fazer acesso a esses dados, e aplicar regras negociais sobre os mesmos. Esse serviço será criado em Java e publicado no WSO2 Web Services Application Server. Como existe uma integração forte entre as ferramentas WSO2, é muito simples o consumo do DataService através do serviço escrito em Java. A Seguir é apresentada o código fonte da classe Java ImovelService, fazendo a integração com o Data Service ImovelDataService. Essa implementação atende os requisitos R05 e R06, pois o WSO2 Web Services Application Server permite a criação de cluster.

**public** **class** ImovelService

{

**private** **static** **final** String ***ENDPOINT*** = "http://localhost:9764/services/ImovelDataService?wsdl";

**private** **static** **final** String ***NAMESPACE*** = "ImovelDataService";

**private** **static** **final** String ***NOMEMETODO\_FINDALL*** = "findAll";

**private** **static** **final** String ***NOMEMETODO\_FIND\_BY\_DESCRICAO*** = "findByDescricao";

**private** **static** **final** String ***NOMEMETODO\_SALVA*** = "addImovel";

**private** Conversor<Imovel> conversor = **new** ConversorImovel();

/\*\*

\* Pesquisa todos os imoveis.

\*

\* **@return** lista com todos os imoveis.

\*/

**public** List<Imovel> findAll()

{

IntegradorWebService integrador = IntegradorWebServiceFactory.*cria*(***ENDPOINT***, ***NAMESPACE***, ***NOMEMETODO\_FINDALL***, conversor);

**return** conversor.converteLista(integrador.envia());

}

/\*\*

\* Pesquisa Imovel pela descrição.

\*

\* **@param** nome

\* **@return** lista de imoveis

\*/

**public** List<Imovel> findByDescricao(String descricao)

{

**if** (descricao == **null** || descricao.trim().isEmpty())

{

**throw** **new** RuntimeException("Campo Obrigatório não informado: descricao");

}

Map<String, String> map = **new** HashMap<String, String>();

map.put("descricao", "%" + descricao + "%");

IntegradorWebService integrador = IntegradorWebServiceFactory.*cria*(***ENDPOINT***, ***NAMESPACE***, ***NOMEMETODO\_FIND\_BY\_DESCRICAO***, conversor);

**return** conversor.converteLista(integrador.envia(map));

}

/\*\*

\* Salva um imovel.

\*

\* **@param** imovel

\*/

**public** **void** inclui(Imovel imovel)

{

validaImovel(imovel);

IntegradorWebService integrador = IntegradorWebServiceFactory.*cria*(***ENDPOINT***, ***NAMESPACE***, ***NOMEMETODO\_SALVA***, **null**);

integrador.envia(OMElementHelper.*toOM*(imovel));

}

/\*\*

\* Valida um imovel para salvar.

\*

\* **@param** imovel

\*/

**private** **void** validaImovel(Imovel imovel)

{

**if** (imovel.getCpf\_proprietario() == **null** || imovel.getCpf\_proprietario() == 0)

{

**throw** **new** RuntimeException("Campo Obrigatório não informado: cpfProprietario");

}

**if** (imovel.getCodigo\_ibge\_estado() == **null** || imovel.getCodigo\_ibge\_estado() == 0)

{

**throw** **new** RuntimeException("Campo Obrigatório não informado: codigoIbgeEstado");

}

**if** (imovel.getEndereco() == **null** || imovel.getEndereco().isEmpty())

{

**throw** **new** RuntimeException("Campo Obrigatório não informado: endereco");

}

**if** (imovel.getDescricao() == **null** || imovel.getDescricao().isEmpty())

{

**throw** **new** RuntimeException("Campo Obrigatório não informado: descricao");

}

**if** (imovel.getValor\_aluguel() == **null**)

{

**throw** **new** RuntimeException("Campo Obrigatório não informado: valorAluguel");

}

**if** (imovel.getCodigo\_ibge\_municipio() == **null** || imovel.getCodigo\_ibge\_municipio() == 0)

{

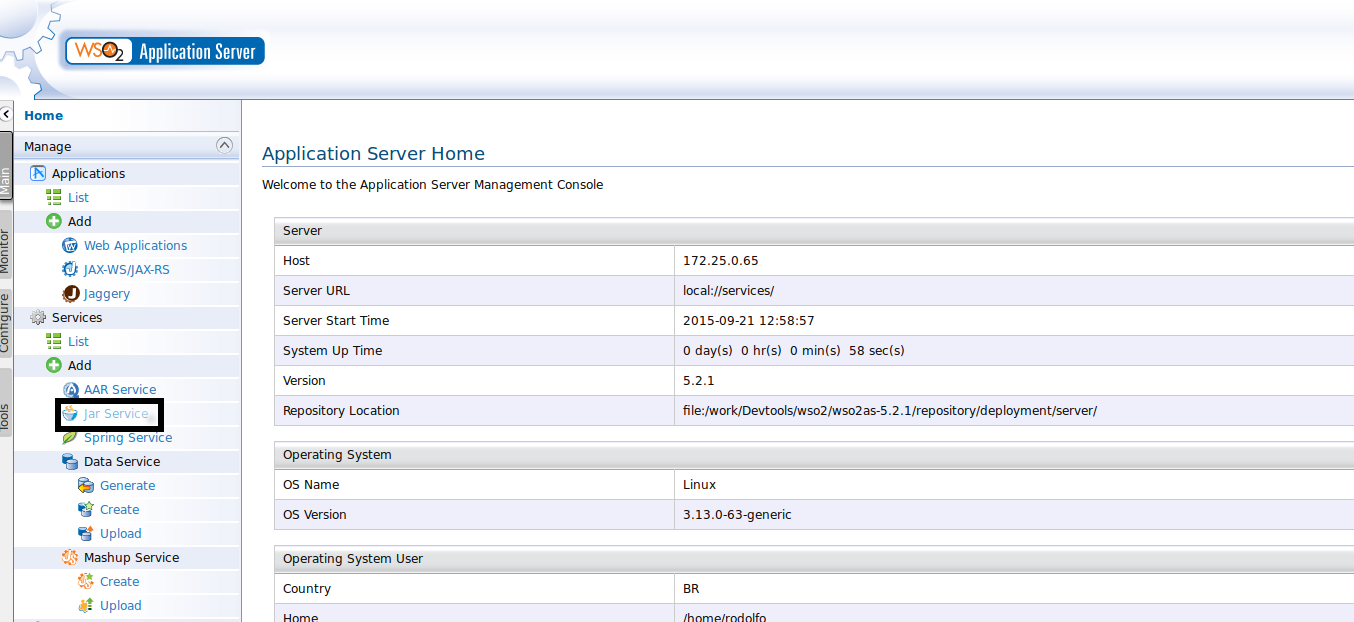
**throw** **new** RuntimeException("Campo Obrigatório não informado: codigoIbgeMunicipio");

}

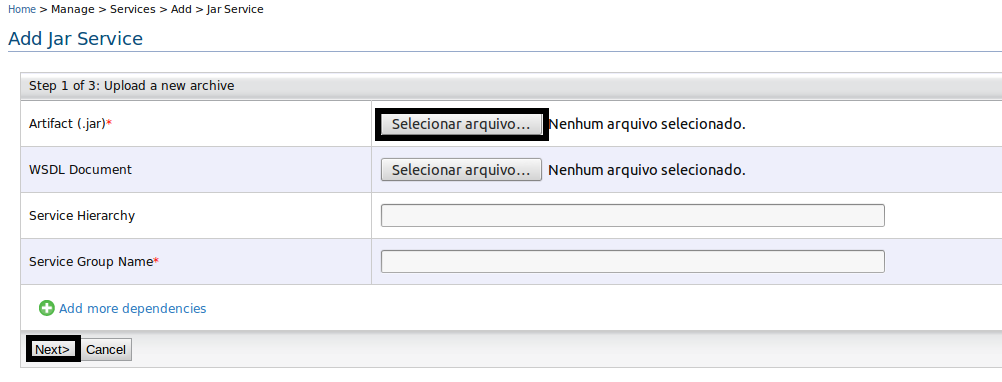
}

}

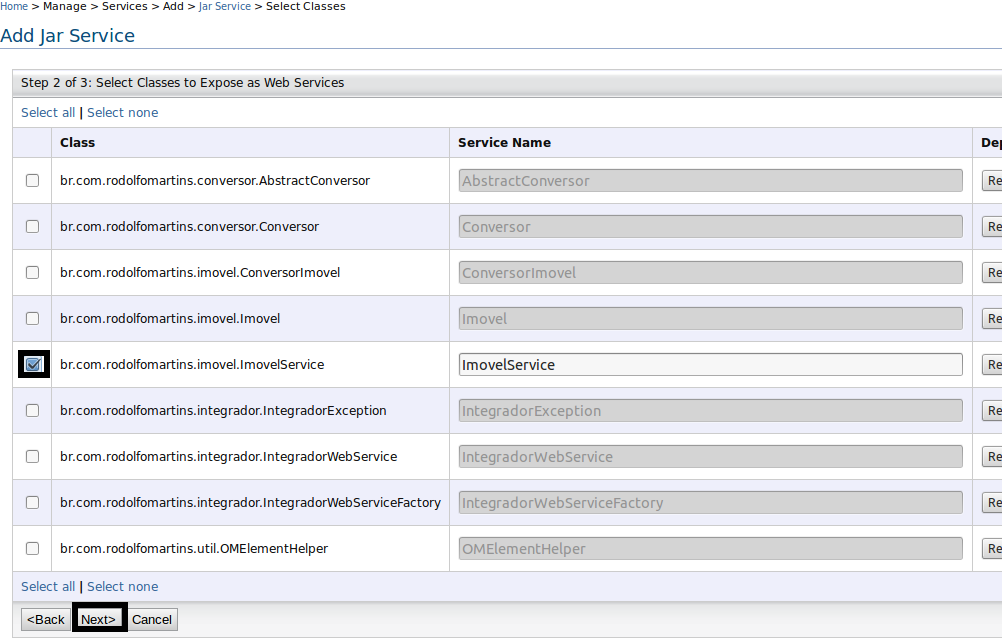
Para a integração com o DataService foi utilizado o framework oasis2 da apache, a sua utilização pode ser visualizada na classe IntegradorWebService, que separa a lógica de integração, aumentando a coesão e diminuindo o acoplamento e consequentemente melhorando o reaproveitamento do código. A classe ImovelService apresentada, possui a responsabilidade de expor os métodos que serão disponibilizados como serviço, e realiza a validação negocial dos dados de entrada. Essa camada negocial é importada dentro do WOS2 application server como um jar, e dessa forma, também é disponibilizado como Web Service. A seguir é demonstrado, os passos para importar um jar, e expor como serviço. Após gerar o jar do projeto no maven através do comando mvn clean install, acessar o console web do application server em jarService, conforme imagem abaixo:



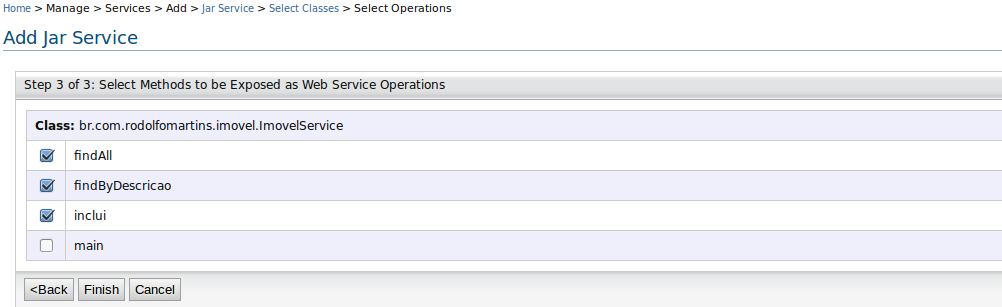
***Selecione o jar, e clique em next, conforme figura abaixo:***



***Selecione o serviço que será exposto como Web Service, no nosso caso, TerminalService e clique em next, conforme figura abaixo:***

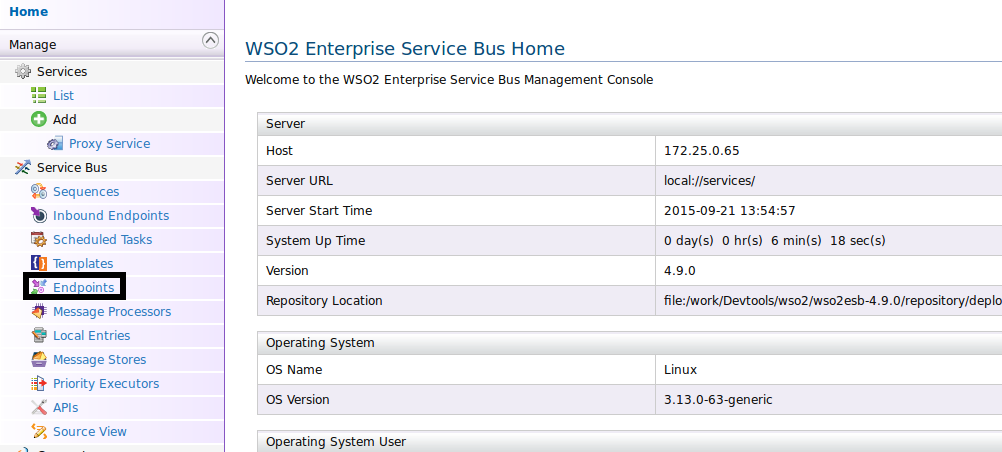


***Selecione o métodos que serão expostos no serviço, e clique em finish, conforme figura abaixo:***

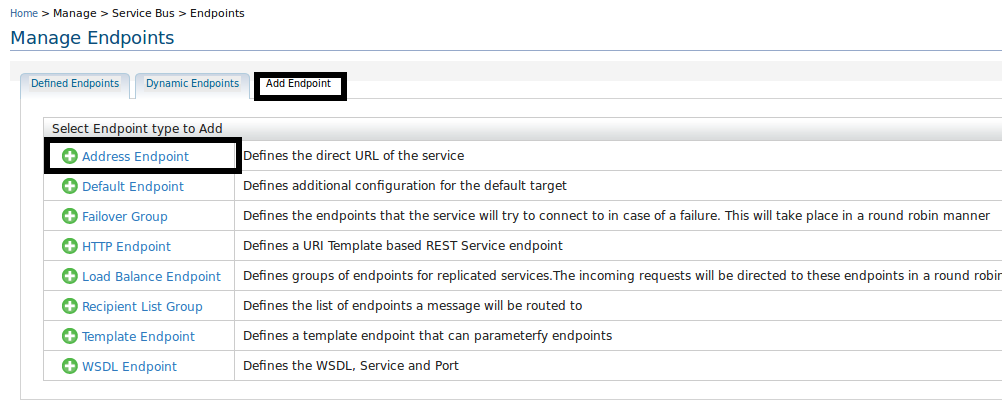


Agora que o serviço foi criado, será publicado para acesso por clientes externos através do Enterprise Service Bus. O barramento permite que o serviço seja desacoplado do cliente, dessa forma a solução ganha todos os benefícios do barramento, atendendo a diversos requisitos de SOA e aos requisitos arquiteturais R01, R02, R03, R04 e R08, segue a seguir um exemplo de cliente consumindo o serviço do barramento.

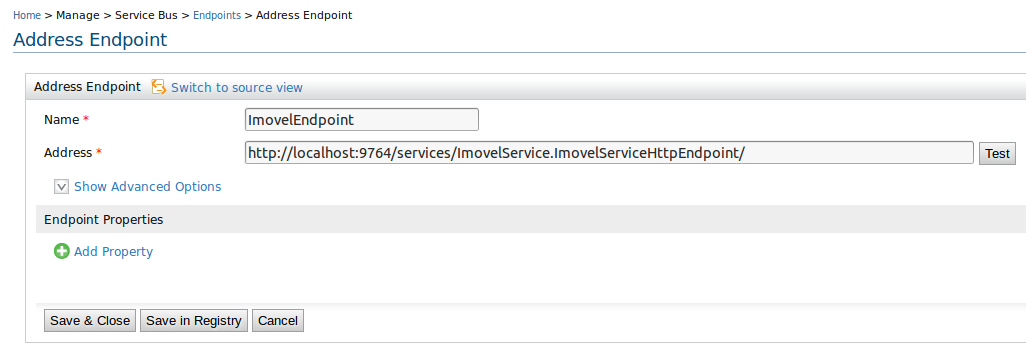
Para criar o Serviço na Bus, acesse o Enterprise Service Bus e clique em Endpoints, conforme figura abaixo:



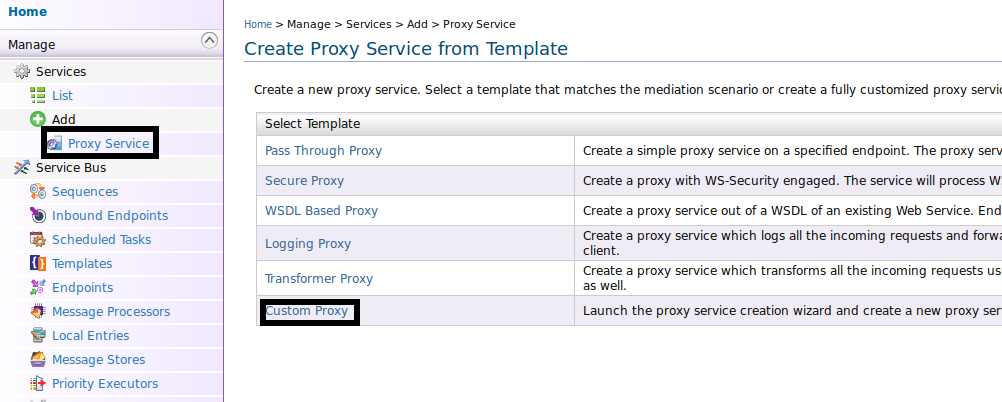
Selecionar a aba Add Endpoint e clicar no botão Addess Endpoint, conforme figura abaixo:

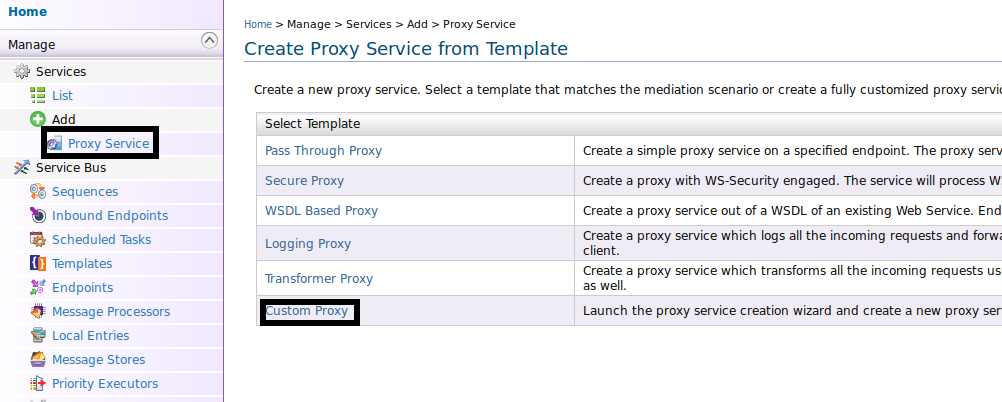


Informar no campo Name o valor ImovelEndpoint, e no campo Address o valor “http://<IP>:<PORTA>/services/ImovelService.ImovelServiceHttpEndpoint/”, substituindo os parâmetros IP e PORTA para apontar para o endereço onde o WSO2 Application Server esta sendo executado, e clique no botão Save & Close.

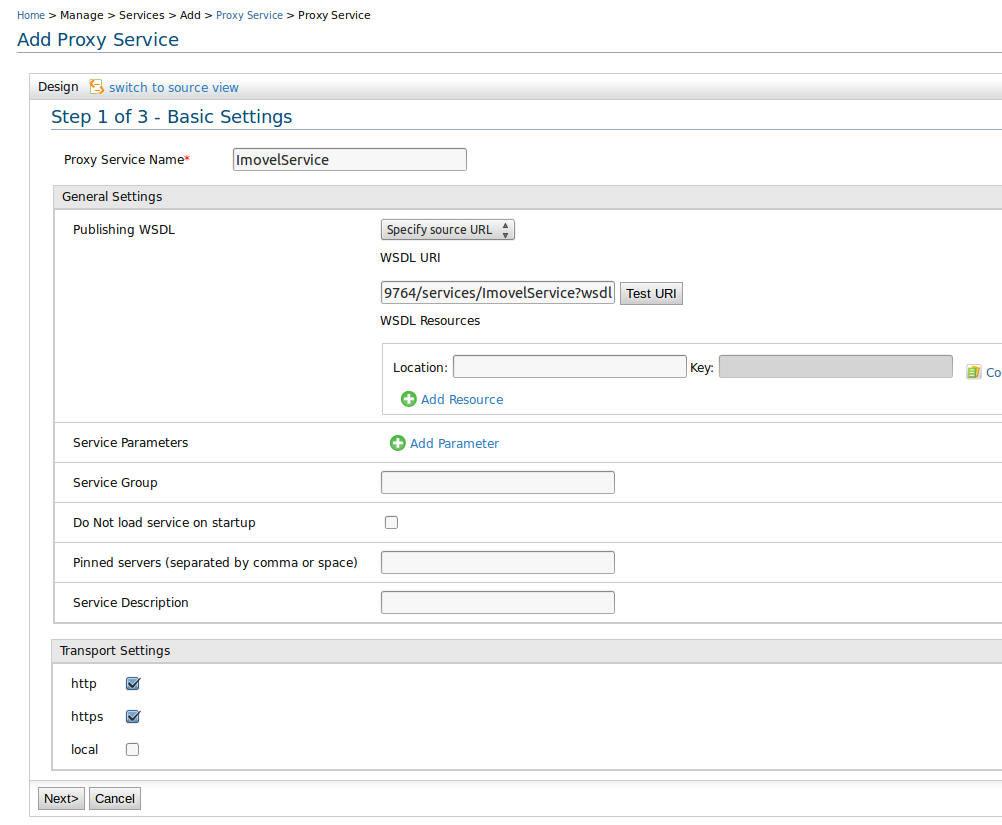


Adicionar o proxy Service, que expoe o serviço no barramento, para isso selecione Add\Proxy Service e clique em Custom Proxy, conforme figura abaixo:

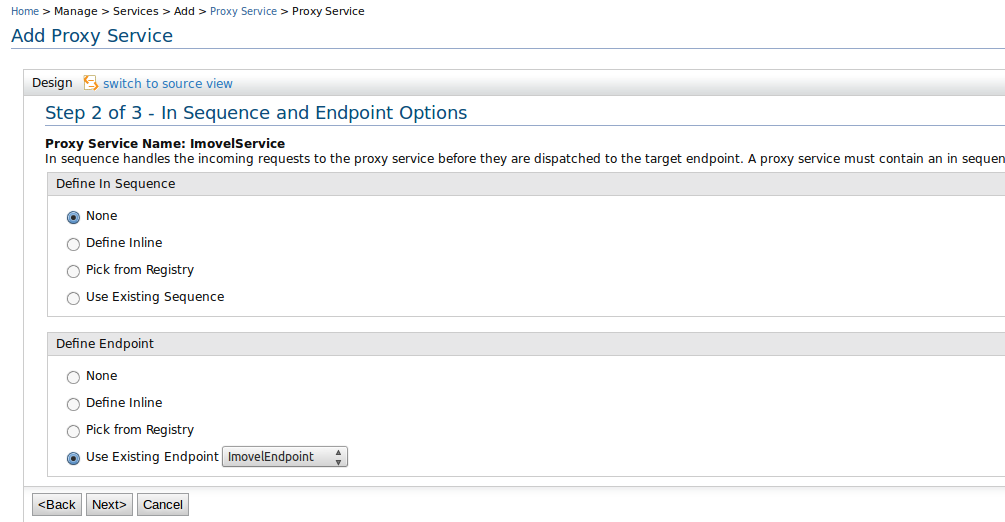




Informar o Proxy Service Name com o valor ImovelProxyService e informar o endereço do wsdl do ImovelService publicado no Application Server, conforme figura abaixo:



Na próxima tela, selecionar o ImovelEndpoint criado, e clicar em Next. Nas próximas telas, clicar em Next e Finish para publicar o serviço.



Na figura 4 é demonstrada através do diagrama de sequência da UML, a colaboração entre os componentes para uma consulta de imóveis através de um filtro. O Ator Consumidor, pode ser qualquer aplicativo que consuma o serviço através de SOAP, Smartphone, Webapps, ou outros serviços que precisem compor esse serviço para prover uma nova funcionalidade, etc. O ImovelServiceProxy é executado na BUS, podendo ser complementado com todas as características do barramento, já explicados nesse trabalho, como adaptadores de dados, roteamento baseado no cabeçalho, filtros, entre outros, nesse trabalho ele tem as características de um serviço SOAP que consome serviços do servidor de aplicação. O ImovelService é executado no servidor de aplicação, e exposto também como um serviço SOAP, contem a regra de negocio da solução e faz uso do serviço ImovelDataService. O imovelDataService é gerenciado e executado pelo WSO2 Data Service Server, e possui a responsabilidade de acesso aos dados.

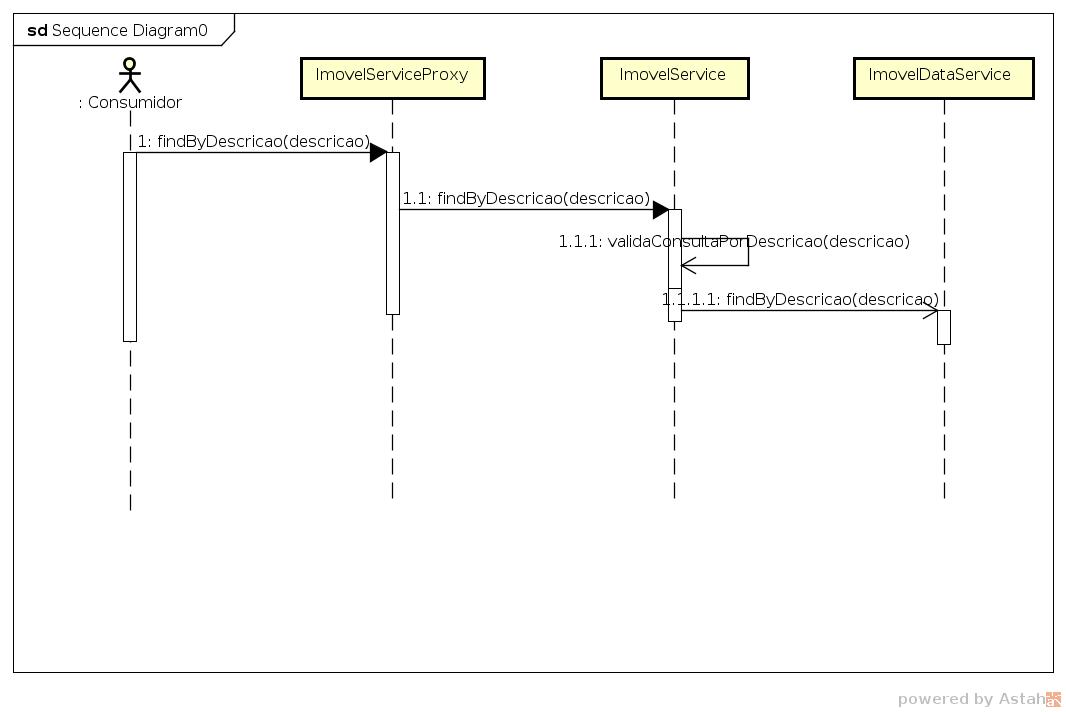


Figura 4: Diagrama de sequência

Fonte: Elaborado pelo Autor (2015)

1. **Resultados**

A fase de projeto e implementação da prova de conceito, relacionada à configuração do ambiente, foi realizada em 16 horas. A criação da modelagem da solução levou 8 horas e a implementação levou mais 8 horas, totalizando 32 horas para a solução completa.

Para atender os requisitos R01, R02, R03 e R04 foram utilizadas as ferramentas WSO2 Data Services Server e WSO2 Enterprise Service Bus.

Para atender os requisitos R05 e R06 foi utilizada a ferramenta WSO2 Web Services Application Server.

Através da prova de conceito foi possível demonstrar que os benefícios de SOA, como facilidade de manutenção, reuso, flexibilidade, controle, foram atingidos. Foram utilizados na aplicação da POC vários componentes apresentados no referencial teórico como: ESB, Web Services, SOAP, UDDI, etc.

Foram atingidas várias características de serviços SOA como: estilo arquitetural, princípios arquiteturais, padrões e critérios que atendem características como modularidade, encapsulamento, baixo acoplamento, separação de responsabilidades, reuso e composição.

Princípios de design foram identificados como, baixo acoplamento, reusabilidade, autonomia, serviços sem estado, composição, interoperabilidade, entre outros.

Apesar do objetivo da prova de conceito ser a criação da solução da camada de serviço, foram utilizadas ferramentas Open Source para a realização de teste de integração simulando a comunicação entre os clientes e o barramento.

1. **Conclusão**

Com os resultados obtidos na prova de conceito é possível concluir que uma arquitetura SOA utilizando apenas ferramentas Open Source pode ser utilizada. Atualmente existem várias ferramentas no mercado, todas muito bem documentadas com esse proposito.

A arquitetura proposta demonstra a simplicidade e produtividade em se criar esse tipo de arquitetura com as ferramentas Open Source. Essas ferramentas podem ser utilizadas tanto para projetos novos, quanto para a migração gradativa de serviços já existentes para o conceito SOA.

Não foi o foco desse trabalho fazer um comparativo com ferramentas pagas, que poderia até servir de proposta para trabalhos futuros, mas demonstrar que as ferramentas Open Source possibilitam a adoção de SOA de forma bem natural.

Todos os requisitos arquiteturais foram atendidos pela solução assim como todos os princípios de design de serviços apresentados, e todas as características que serviços SOA devem prover.

1. **Referências Bibliográficas**

SOA in Practice - The Art of Distributed System Design by [Nicolai M. Josuttis](http://it-ebooks.info/author/1060/)

SOA Principles of Service Design by Thomas Erl

<http://blog.iprocess.com.br/2012/10/soa-arquitetura-orientada-a-servicos/>

<http://www.amazon.com/SOA-Principles-Service-Design-Thomas-ebook/dp/B004UA78E2>

<http://simplicable.com/new/the-9-principles-of-soa-design>

<http://www.leandroprado.com.br/2010/05/ferramentas-soa-open-source/>

http://www.oasis-opencsa.org/sca

<http://www.leandroprado.com.br/2010/05/componentes-de-soa/>