UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO

n° 0020/2010

Estudos de Enterprise Service Bus e Oracle Service Bus

Henrique Prado Sousa
Leonardo Guerreiro Azevedo
Flávia Santoro
Fernanda Baião

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

Projeto de Pesquisa

Grupo de Pesquisa Participante

Patrocínio

PETROBRAS

Relatórios Técnicos do DIA/UNIRIO, No. 0020/2010
Editor: Prof. Sean W. M. Siqueira Dezembro, 2010

Estudos de Enterprise Service Bus e Oracle Service Bus*

Henrique Prado Sousa, Leonardo Guerreiro Azevedo, Flávia Santoro, Fernanda Baião

Núcleo de Pesquisa e Prática em Tecnologia (NP2Tec)
Departamento de Informática Aplicada (DIA) – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{henrique.sousa, azevedo, flavia.santoro, fernanda.baiao}@uniriotec.br

Abstract. Enterprise Service Bus (ESB) is the main infra-structure in SOA. It is responsible to make available, in a standard way, services from different platforms, implemented in different programming languages, and available from providers in a distributed environment. Despite of several patterns for SOA, there is no pattern for ESB. Vendors agree what are the responsibilities of ESB. However, each vendor has its own implementation of ESB. This work presents the main characteristics of ESB, and analises in practice the ESB implementation of Oracle, called as Oracle Service Bus.

Keywords: SOA, ESB, OSB.

Resumo. Enterprise Service Bus (ESB) corresponde a principal infra-estrutura de SOA. Ela é utilizada para disponibilizar de uma forma padronizada serviços de diferentes plataformas, implementados em diferentes linguagens de programação e disponíveis a partir de provedores em um ambiente distribuído. Apesar de existirem vários padrões para SOA, este não é a regra para ESB. Existe um concenso entre fornecedores quais são as funcionalidades que um ESB deve prover. No entanto, cada fornecedor possui sua implementação de ESB. Este trabalho apresenta as principais características de um ESB, e analisa de forma prática a implementação de ESB da Oracle, chamada de Oracle Service Bus.

Palavras-chave: SOA, ESB, OSB.

---

* Trabalho patrocinado pela Petrobras.

Sumário

1 Introdução 5
1.1 Estrutura do relatório 6
2 Enterprise Service Bus 6
2.1 Responsabilidades do ESB 8
2.2 Benefícios do ESB 11
2.3 JBI 12
2.4 ESBs comerciais e de código aberto 13
3 Arquitetura do Oracle Service Bus 15
3.1 Suíte de produtos do OSB 15
3.2 Componentes da arquitetura do OSB 17
3.3 Fluxo de mensagens via serviços de proxy 20
3.4 WSDL efetivos e WSDL gerado 25
3.5 Web services com atributo *style* igual a SOAP document × RPC 26
3.6 Padrões de troca de mensagens 29
3.7 *Broker* de mensagens 29
3.8 Transformação e processamento de mensagens 30
3.9 Disponibilização de EJB como serviço 30
3.10 Console de teste 31
3.11 Gestão de recursos 31
3.12 OSB e UDDI 32
3.13 Tratamento de erros 32
3.14 Versionamento 32
3.15 Monitoramento 32
3.16 Disponibilização do OSB em servidores 32
3.17 Outros conceitos importantes 34
4 Análise prática do Oracle Service Bus 34
4.1 Publicação do serviço no OSB 34
4.2 Alteração dos parâmetros do serviço sem alterar o cliente 40
4.3 Definição de fluxos em serviços de Proxy 45
4.4 Validação de tipos de dados utilizando serviço de proxy 63
4.5 Versionamento de serviços utilizando o OSB 78
4.6 Monitoramento de serviços 85
4.7 Conclusão 114
5 Conclusão 114
6 Referências 115
Apêndice 1 - Diferenças entre OSB e ALSB 116
Apêndice 2 - Elementos de um WSDL 117

1 Introdução

ESB é a infra-estrutura que permite alta interoperabilidade entre sistemas distribuídos via serviços. Processos disponibilizados via serviços distribuídos em múltiplos sistemas usando diferentes plataformas e tecnologias podem ser executados de uma maneira mais simples utilizando o ESB.

O ESB é um barramento de mensagens, projetado para possibilitar a implementação, desenvolvimento e gerenciamento de soluções baseadas em SOA com o foco no empacotamento, desenvolvimento e gestão de serviços distribuídos [Papazoglou *et al.*, 2007]. É responsabilidade do ESB permitir que consumidores (clientes) invoquem os serviços oferecidos por provedores de serviços, o que envolve várias tarefas, tais como: prover conectividade; transformação de dados; roteamento de mensagens baseado em conteúdo; tratar segurança; tratar confiabilidade; gerência de serviços; monitoramento e *log* de serviços; balanceamento de carga etc. Estas tarefas devem ser consideradas para diferentes plataformas de software e hardware para diferentes *middleware* e protocolos (Figura 1).

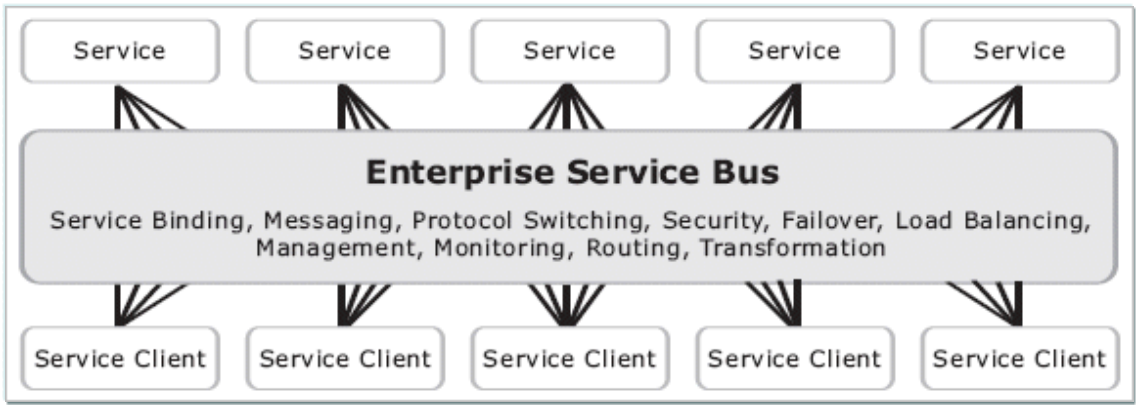

Figura 1 – Enterprise Service Bus [OSB, 2008a]

Uma suíte robusta de um ESB em SOA deve oferecer [OSB, 2008a]:

- Adaptadores: os quais possibilitam conectividade em aplicações empacotadas e sistemas legados;
- Motor de consulta distribuída: que permite a criação de serviços de dados para fontes de dados heterogêneas;
- Motor de orquestração de serviços para tratar tanto processos de longa duração (*stateful*) como processos de curta duração (*stateless*);
- Ferramenta de desenvolvimento de aplicações que permitem a rápida criação de aplicações para o usuário;
- Serviços de apresentação: permitem a criação de portais personalizados que agregam serviços de múltiplas fontes.

Hewitt [2009] apresenta que os padrões (API Java, tecnologias XML, BPEL, WS*) publicados em conjunto por OASIS-Open, W3C, OAGI, Sun permite que as organizações implementem as especificações ou recomendações de forma que muitas partes das tecnologias que seguem estes padrões sejam portáveis através de diferentes im-

plementações. Por exemplo, um web service JAX-WS funcionará praticamente da mesma forma em qualquer container que implementa JAX-WS.

Por outro lado, não existe uma especificação para ESB. Não existe nenhum padrão que suporta ESB e nenhuma definição precisa para ESB. Apesar dos fornecedores concordarem em quais características um ESB deve ter, os produtos variam muito entre si. As funcionalidades existentes em um produto podem não existir em outro ou serem realizadas de maneira completamente diferente. Existem suites, como as propostas pela Oracle, Software AG, IBM e TIBCO, que incluem uma coleção de produtos tais como ESB, motor de orquestração, registro/repositório de serviços, motor de execução e de desenho BPEL, Business Activity Monitoring e outras ferramentas relacionadas. As características presentes nestes produtos não são consistentes. Por exemplo, pode-se ter um ESB de um fornecedor que trata transformações de mensagens, enquanto que o ESB de outro fornecedor delega as transformações para o motor de orquestração.

O objetivo deste relatório é apresentar as principais características que devem estar presentes em um ESB (Enterprise Service Bus) e realizar uma análise do OSB (Oracle Service Bus) no atendimento a estas características.

Este relatório foi produzido pelo Projeto de Pesquisa em SOA como parte das iniciativas dentro do contexto do Projeto de Pesquisa do Termo de Cooperação entre NP2Tec/UNIRIO e a Petrobras/TIC-E&P/GDIEP.

1.1 Estrutura do relatório

Esse relatório está organizado em 5 capítulos, sendo o capítulo 1 a presente introdução.

No capítulo 2, são apresentadas as principais características do Enterprise Service Bus, principal infra-estrutura que apóia uma abordagem SOA.

No capítulo 3, é apresentada, em detalhes, a arquitetura do Oracle Service Bus (OSB), que é o ESB da Oracle.

No capítulo 4, apresentados estudos práticos do OSB.

Nos capítulos 5 e 6, são apresentadas as conclusões do trabalho e referências bibliográficas, respectivamente.

2 Enterprise Service Bus

Enterprise Service Bus (ESB) é a principal infra-estrutura que apóia uma arquitetura orientada a serviços (SOA). Hewitt [2009] aponta que muitos arquitetos vêem o ESB como um conjunto de padrões, e não como um produto. Outros argumentam que não é necessário adquirir um ESB, o qual pode ser substituído pela implementação de padrões básicos de roteamento como os propostos por Hohpe e Woolf [2004]. Os padrões de barramento de mensagens, roteamento baseado em conteúdo, filtros e *pipes*, conexão ponto a ponto, normalizador, modelo de dados canônicos formam a base para os ESB modernos que, além disso, implementam padrões chave de EAI (Enterprise Application Integration). Estes padrões não são simples de implementar, como os padrões de projeto propostos em [Gamma *et al.*, 1994]. Logo, os problemas que existem nesta abordagem são o custo de desenvolvimento e que o resultado poderá ser a repli-

cação de muitas características já disponíveis no mercado, as quais já foram testadas e automatizadas, e podem ser colocadas em execução rapidamente.

Um ESB é capaz de conectar diferentes sistemas que se comunicam via mensagens sem forçar a criação de conexões ponto-a-ponto que são visíveis para aplicações cliente. Clientes se conectam ao barramento que serve como uma camada de mediação que protege os clientes de diferentes formatos e protocolo de mensagens usados em sistemas backend[1].

Um ESB é tipicamente independente de plataforma no sentido de que executa em diferentes sistemas operacionais e é neutro em relação a servidor de aplicação e linguagem de programação. O barramento permite conectar aplicações Java, aplicações .Net, brokers que requisitam objetos, além de sistemas legados. A Figura 2, apresentada em [Papazoglou *et al.*, 2007], descreve uma arquitetura simplificada de um ESB que integra uma aplicação J2EE usando JMS (Java Message Services), uma aplicação .NET em um cliente C#, uma aplicação MQ (Message Queue) que tem interface com aplicações legadas, aplicações externas e fontes de dados sendo acessadas via Serviços Web em um módulo de consultas distribuídas. O ESB provê uma integração mais eficiente entre diferentes componentes de aplicações. Isso é possível, pois o ESB posiciona os componentes através de uma fachada orientada a serviço, utilizando, por exemplo, web services. Na Figura 2, o módulo de consultas distribuídas é usado para abstrair a complexidade de fontes de dados heterogêneas. Todos os pontos finais apresentados na figura provêem abstração de destino físico e informações de conexão. Um portal é um ponto de interface com o usuário que agrega variados recursos representados como serviços, por exemplo, vendas, funcionalidades departamentais, recursos humanos, além de portais de parceiros do negócio.

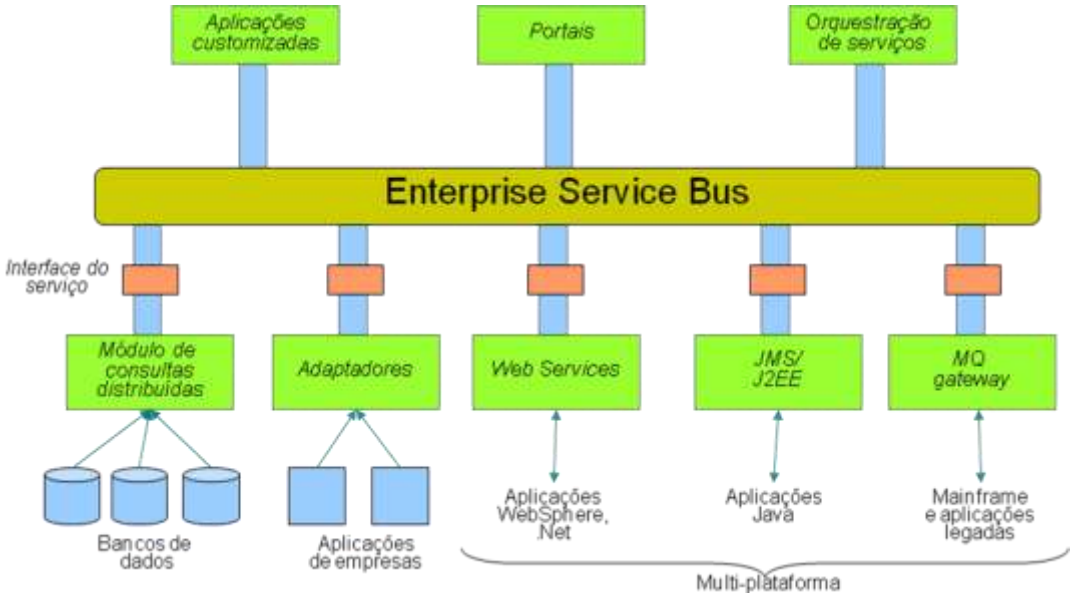

Figura 2 – Exemplo de Enterprise Service Bus [Papazoglou et al., 2007]

Assim como Papazoglou [2007], Hewitt [2009] apresenta o papel do ESB, ilustrado na Figura 3, caracterizando que o barramento coreografa interações via mensagens utilizando diferentes tecnologias, tais como motor de regras (externo ou embutido), mo-

[1] Backend é qualquer sistema responsável por um grupo específico de dados e funcionalidades, por exemplo: um banco de dados em um SGBD, mainframe, SAP, grupo de servidores JEEE, uma conexão com outra empresa. Do ponto de vista do negócio, um backend é um sistema que tem um papel específico e é mantido por um grupo específico. [Josuttis, 2007]

tor de orquestração (utilizando BPEL ou XPDL), adaptadores para sistemas legados ou aplicações empacotadas, e mecanismos de roteamento e transformação internos. Isto permite que todos os nós do barramento interajam sem ter que criar rotas específicas para cada aplicação, dessa forma reduzindo o custo de manutenção. Dado que o barramento abstrai e realiza mediação entre os nós conectados, alguma flexibilidade é alcançada; se for observada a necessidade de substituir uma aplicação por outra, pode-se fazer esta substituição com o mínimo de comprometimento com os clientes desta aplicação [Hewitt, 2009]. Basta manter as interfaces das conexões disponibilizadas no ESB.

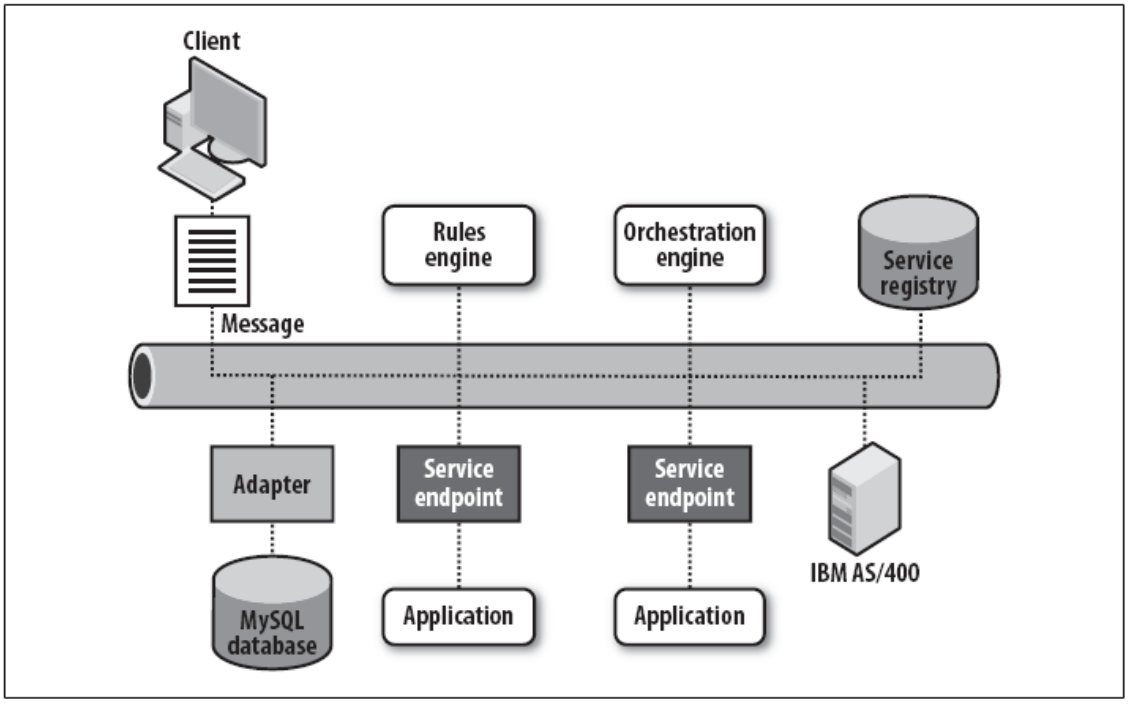

Figura 3 – O papel do ESB em SOA [Hewitt, 2009]

O ESB suporta a invocação de web services e outros nós de aplicações preparadas para se comunicar através da rede. O ESB disponibiliza adaptadores para conectar com aplicações empacotadas ou aplicações legadas, oferecem roteamento robusto de mensagens, permitem orquestração e transformação do conteúdo da mensagem, dentre outras funcionalidades.

2.1 Responsabilidades do ESB

As principais responsabilidades de um ESB apresentadas por Hewitt [2009] são descritas a seguir.

2.1.1 Suporte a web services

O ESB tem a capacidade de invocar web services baseados em WSDL[2] e SOAP[3], bem como serviços POX[4] (Plain Old XML) utilizando o protocolo http. POX correspondem a mensagens empacotadas apenas em XML, sem utilizar o protocolo SOAP.

[2] http://www.w3.org/TR/wsdl

[3] http://www.w3.org/TR/soap/

[4] http://msdn.microsoft.com/en-us/library/aa395208.aspx

Em geral, é criado um *proxy* WSDL para o serviço que se quer expor no ESB. Os clientes, ao invés de se conectarem diretamente ao WSDL do serviço, eles se conectam a um WSDL (*proxy*) exposto no barramento. Esta forma de conexão permite tratar lógica de roteamento e transformação dentro do barramento.

Ferramentas tratam este mecanismo de conexão de forma diferente. GlassfishESB[5] trata este mecanismo diretamente no BPEL, enquanto que Aqualogic Service Bus[6] permite a criação de um serviço de *proxy*.

2.1.2 Adaptadores

Adaptadores são utilizados para conectar aplicações que não suportam a interface SOAP ou XML, tais como, aplicações empacotadas (SAP, PeopleSoft, SAP R/3, Siebel), banco de dados, ferramentas ERP, interfaces via arquivo.

Adaptadores podem ser utilizados tanto no caso da aplicação não fornecer integração via XML ou SOAP, como também nos casos em que se deseja um maior ganho de desempenho evitando o custo em tempo de execução de traduzir para/de XML, se o sistema suporta diretamente serialização de objetos.

Em muitos casos adaptadores são fornecidos como uma funcionalidade que deve ser paga a parte, além do ESB.

2.1.3 Invocação de serviços

Como uma característica padrão, ESB suporta chamadas síncronas e assíncronas de serviços, e algumas vezes callback. Um serviço pode ser mapeado em outro serviço. Alguns ESBs permitem negociação através de WS-MetadataExchange[7], tratando segurança, por exemplo.

A forma que serviços se comunicam é chamada de padrões de troca de mensagens (ou MEP – Message Exchange Pattern). Um MEP define a sequência de mensagens em uma chamada de serviço ou operação do serviço, especificando a ordem, a direção e a cardinalidade das mensagens [Josuttis, 2007]. Existem diferentes tipos de MEP:

- One-way: A operação recebe uma mensagem, mas não irá retornar uma resposta.
- Request-response: A operação recebe uma mensagem e irá retornar uma resposta.
- Solicit-response: A operação envia uma requisição e irá esperar uma resposta.
- Notification: A operação envia uma mensagem, mas não irá esperar uma resposta.

O ESB deve apoiar estes padrões de troca de mensagens. Por exemplo, no padrão Request-Response, apresentado na Figura 4, o consumidor envia mensagem para o ESB que faz o roteamento para o provedor do serviço. Após o processar a requisição, o provedor envia a resposta para o ESB que repassa a resposta para o consumidor do serviço. O consumidor fica parado esperando a resposta, da mesma forma que seria na invocação de um método de um componente de forma síncrona. O ESB deve ter a ca-

[5] https://open-esb.dev.java.net/Downloads.html#

[6] http://download.oracle.com/docs/cd/E13171_01/alsb/docs20/index.html

[7]http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf

pacidade de encontrar o provedor do serviço, bem como fazer o roteamento da resposta para o consumidor que invocou o serviço. Exemplos de MEPs e responsabilidades do ESB são apresentados em [Josuttis, 2007].

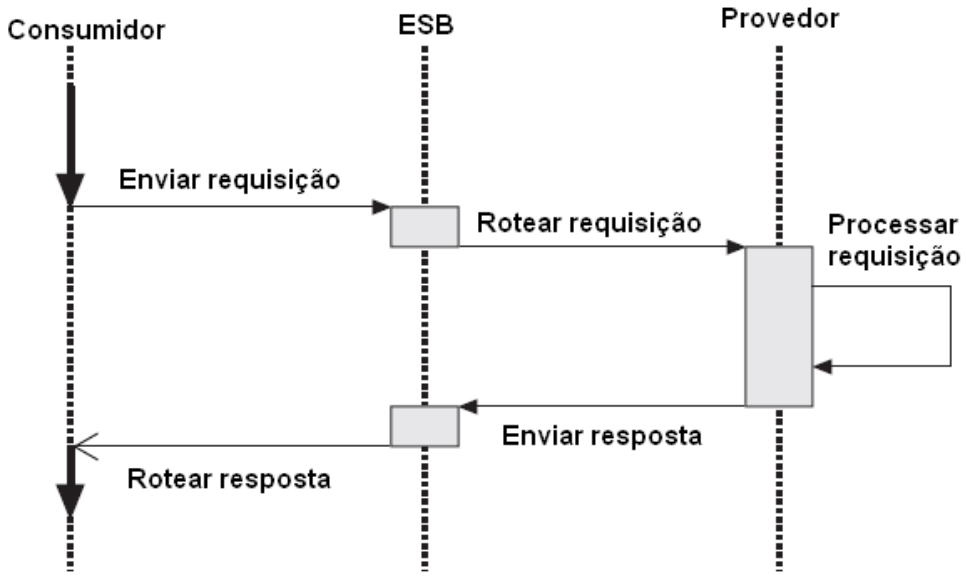

Figura 4 – Exemplo de MEP Request-Response intermediado pelo ESB

2.1.4 Mediação e independência de protocolo

Muitos barramentos permitem que diferentes protocolos de comunicação sejam utilizados durante o caminho de uma mensagem, incluindo não apenas HTTP e JMS, mas também HTTPS, SMTP, XMPP, FTP dentre outros. Muitos barramentos têm a capacidade de conectar diferentes formatos de dados, por exemplo, Eletronic Data Interchange (EDI), JDBC, COBOL *copy books*, e arquivos *flat*.

2.1.5 Roteamento

Barramentos disponibilizam diferentes formas de realizar roteamento de mensagens, por exemplo, a partir do conteúdo da mensagem (usando XPath para navegar na mensagem), roteamento baseado em um serviço de regras e roteamento baseado em políticas.

Alguns barramentos permitem o controle de fila de mensagens a fim de que uma mensagem, quando enviada, seja persistida e não seja perdida. Este é o princípio dos mediadores orientados a mensagens (ou MOM – Message Oriented Middleware), tais como MQ e JMS [Josuttis, 2007].

2.1.6 Transformação

Dados representados em XML podem ser transformados utilizando XSLT e consultados utilizando XQuery e XPath. Estas tecnologias permitem preparar o dado para ser trafegado entre sistemas/serviços. Se um modelo canônico está sendo utilizado, esta é uma característica importante de existir no ESB.

2.1.7 Orquestração

Muitos ESB realizam orquestração através de um serviço de proxy, o qual coordena a execução de múltiplos serviços. Alguns barramentos delegam a orquestração para motores BPEL ou XPDL.

2.1.8 Segurança

O barramento de serviços provê funcionalidades para garantir o uso de políticas de segurança em conjunto com pontos de garantia de políticas, SSL e SAML[8] (Security Assertion Markup Language).

2.1.9 Gerência de serviços

Serviços executando no ESB podem ser monitorados, auditados, mantidos e reconfigurados. No último caso, mudanças no processo podem ser feitas sem necessidade de reescrever serviços ou aplicações subjacentes, dependendo das modificações necessárias e dos serviços existentes [Josuttis, 2007].

2.1.10 Modelo canônico

Dado que o ESB provê uma camada de abstração, clientes podem se comunicar com o barramento, e não saberem a localização do *endpoint* do serviço com o qual realmente desejam se comunicar. Uma vez que a mensagem XML é recebida no barramento, como o ponto de integração central, o barramento pode realizar as transformações necessárias para garantir, por exemplo, que um sistema legado de CRM integre facilmente com um novo sistema de uma empresa que foi adquirida. Esta transformação de dados deve ser feita a partir de um modelo que represente os dados de forma canônica, ou seja, como os dados devem ser estruturados conceitualmente de forma única em relação às diferentes representações existentes na organização. Dessa forma, o barramento pode ser configurado para que sejam feitas transformações das mensagens de solicitação (provenientes dos consumidores) para o modelo canônico, bem como a transformação da mensagem de resposta recebida proveniente dos provedores para o modelo canônico. Portanto, no barramento trafegam apenas informações segundo o modelo canônico. Caso novos consumidores queiram se conectar ao provedor via barramento, basta que seja implementada a transformação da mensagem do consumidor para o modelo canônico que ele conseguirá se comunicar com o provedor. Este é um exemplo de onde um modelo de dados canônico ganha real importância, dado que o sistema cliente pode continuar se comunicando na forma que deseja, e o barramento serve para traduzir a mensagem para o formato esperado.

2.2 Benefícios do ESB

Hewitt [2009] aponta os seguintes benefícios em utilizar um ESB:

- Redução do tempo de integração de aplicações novas e aplicações existentes;
- Aumento de flexibilidade porque a dependência entre serviços é reduzida;
- Gestão simultânea e centralizada do catálogo de serviços, enquanto que os próprios serviços executam de forma distribuída;

[8] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

- A gestão centralizada permite coletar métricas sobre serviços, permitindo monitorar SLA (Service Level Agreements), gerar relatórios para TI e negócio;
- Encoraja o uso de interfaces padrão da indústria;
- Maior agilidade e tempo de resposta para mudanças;
- Obtenção de informação atualizada e precisa sobre recursos da organização via centralização lógica da gestão de dados.

2.3 JBI

Hewitt [2009] apresenta que a especificação JBI (Java Business Integration), lançada pela SUN em 2005, foi uma proposta em direção a um ESB independente de fornecedor. A arquitetura JBI suporta muito baixo acoplamento entre componentes. JBI define um *metacontainer*. Este *metacontainer* não executa nada por ele mesmo. Ele serve como um ponto de gestão de ciclo de vida para uma coleção de *engines* que são capazes de executar um tipo específico de integração, por exemplo, BPEL para orquestração, conectores JDBC, conectores para arquivos etc. A idéia é que alguém implementa uma especificação JBI e então eles (ou outros fornecedores) podem implementar *engines* que são plugáveis dentro do container.

2.3.1 Arquitetura JBI

JBI define o *framework*, as interfaces e o ciclo de vida abstrato que permite que componentes conectados executem juntos e que sejam disponibilizados componentes escritos por desenvolvedores dentro deste ambiente. Para realizar este objetivo, JBI define quatro elementos principais (Figura 5): *engine* de serviços (ou Service Engines – SE), componentes de ligação (ou Biding Component – BC), roteador de mensagens normalizadas (Normalized Message Router – NMR) e ambiente de execução (*runtime evironment*).

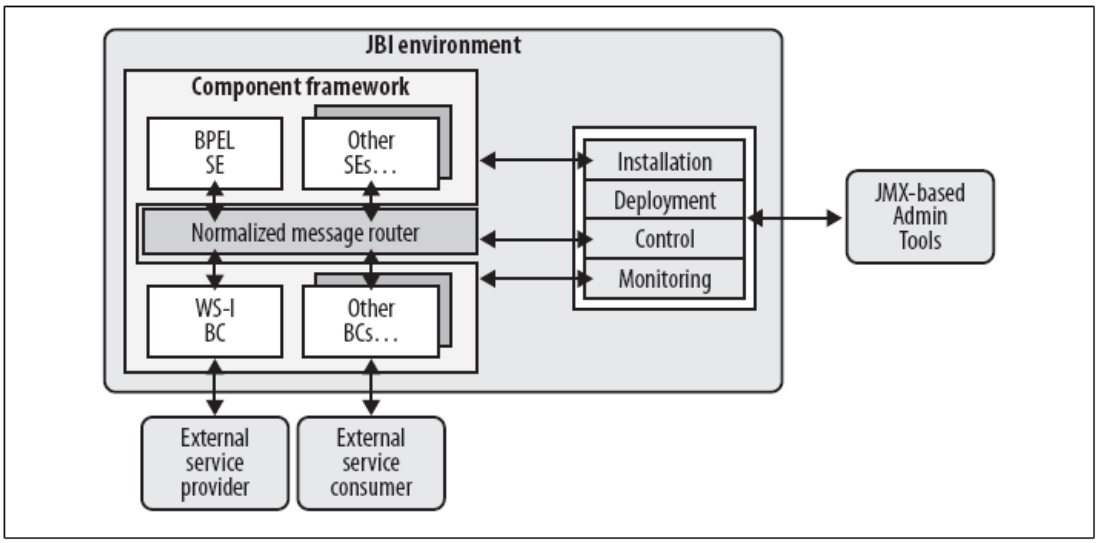

Figura 5 – Arquitetura JBI em alto nível, como demonstrado pela especificação 1.0 [Hewitt, 2009]

Engine de serviços permitem conectar lógica, transformações e processamento do negócio, por exemplo, SE para BPEL, SE para XLT. SE para SQL, agendamento etc.

Componentes de ligação oferecem independência de protocolo. Eles provêem protocolos de transporte (e outro protocolos baseado em comunicação tais como protocolo para se comunicar com arquivos) para serem utilizados por serviços externos. SE e BC criam um canal para o NMR. Eles agem como um *proxy* para serviços publicados no componente de execução JBI que precisam de um protocolo específico. Existem BC para arquivo, LDAP, JMS, HTTP, JDBC, CICS, DCOM, CORBA, XMPP e outros.

O roteador de mensagens normalizadas media a troca de mensagens entre SEs e BCs dentro de um ambiente. O roteamento da mensagem é determinado pelo NMR, e a mensagem normalizada é traduzida no formato para o BC específico que está sendo invocado. SEs e BCs se comunicam com o NMR via um *pipe* chamado de canal de entrega (*delivery channel*). Uma mensagem normalizada é composta de duas partes: a mensagem XML abstrata e os metadados da mensagem, também chamado de dados de contexto da mensagem.

O ambiente de execução JBI engloba SE, BC e NMR, além de um *framework* permitindo disponibilização, gestão, monitoramento e instalação.

2.3.2 Como JBI é tratado na indústria

Vários fornecedores participaram da especificação JBI, tais como, membros da Apache Software Foundation, Borland, Fujitsu, JBoss, IONA, Oracle, SAP AG etc. Isto sugeriria que JBI o principal conceito para a implementação de um Enterprise Service Bus. No entanto, poucos fornecedores suportam JBI, devido a várias razões: muitos fornecedores perceberam que poderiam ter suas ferramentas SOA construídas pelo empacotamento de soluções EAI previamente desenvolvidas; eles tinham suas próprias definições de um ESB; além disso, a idéia de suportar um SE ou um BC como um objeto capaz de conectar de um JBI para outro não estava completamente clara. Além disso, existe uma limitação em JBI de que todos os contratos dos serviços sejam especificados em WSDL. Isto não funciona para contratos WADL, que podem se tornar populares para POX/HTTP, ou Java puro. Outro problema levantado é o container JBI especificar que a mensagem deve ser no formato XML, o que traz carga desnecessária para mensagens que não são baseadas em XML.

Dessa forma, fornecedores trabalharam no desenvolvimento de suas próprias SEs e BCs para HTTP, FPT, JMS, arquivo, BPEL etc. Apesar disto, existem boas implementações de JBI. No entanto, estas implementações não são os ESBs mais populares.

JBI é algo importante de conhecer. Ele pode ser uma boa estratégia para iniciar a construção de um SOA com ferramentas de código fonte aberto durante o período de investigação e então se formar para obter um ESB mais robusto, estável e com maior desempenho.

2.4 ESBs comerciais e de código aberto

Hewitt [2009] apresenta um estudo dos principais ESB dos fornecedores de mercado e aponta como líderes de mercado os seguintes ESB: IBM WebSphere ESB e DataPower; Sonic ESB; TIBCO BusinessWorks e ActiveMatrix Grid; Cape Clear. Já como ESBs de código aberto, ele destaca o OpenESB da Sun, Mule ESB da MuleSource, e Apache ServiceMix. Devido ao interesse do projeto na avaliação das ferramentas da Oracle/BEA, algumas das características do Oracle Service Bus são resumidas a seguir.

No final de 2007, BEA lançou o AquaLogic Service Bus(ALSB). Após a compra da BEA pela Oracle, nos meados de 2008, o barramento foi rebatizado como OSB (Oracle

Service Bus) 10.3. O produto anterior da Oracle, chamado de Oracle Enterprise Service Bus, será mantido, mas não será mais promovido. Dentre as características do OSB, destacam-se:

- Ferramenta para desenvolvimento baseada no Eclipse, chamada Oracle Service Bus WorkSpace Studio;
- Tratamento de falhas na chamada de serviços (tanto roteamento como *pooling* de mensagens);
- Otimização de transporte de mensagens: permite que o container trate EJBs remotos que estão posicionados na mesma JVM com os consumidores invocando os serviços como se fossem EJBs locais. Esta otimização de desempenho evita chamadas RMI, as quais são mais custosas.
- Suporte a WS-ReliableMessaging: permite tanto o reenvio de mensagens das quais não se sabe se a resposta foi enviada, devido a falhas no meio de transporte, como também reenvio de mensagens após falha do cliente ou do servidor.
- Os serviços são divididos em dois tipos: serviços de proxy e serviços de negócio. Um serviço de proxy é exposto ao cliente e serve como um *wrapper* para a implementação do serviço, a qual provê transparência de localização e um oportunidade para injetar várias funcionalidades como segurança, transformação etc. Um serviço de negócio é um conjunto de metadados para um serviço que está externo ao barramento.
- SOAP é usado como formato de mensagem canônica interno ao barramento, logo, mesmo que uma mensagem não SOAP é enviada ao barramento, ela é transformada em uma mensagem SOAP a fim de que seja possível utilizar XQuery e XPath de forma padronizada através da interface.
- OSB permite definir SLA (*Service Level Agreement*) e aplicá-los/monitorá-los em tempo de execução nos fluxos de mensagens que atravessam o barramento. Isto é feito através de regras de alerta que indicam quando o barramento deve sinalizar por uma violação de uma SLA. O barramento já vem com um conjunto de alertas previamente cadastrados. Os alertas podem ser configurados não só para indicar falhas como também para indicar quando o sistema está começando a alcançar limites de desempenho.
- Criação de relatórios a partir da situação das SLAs do barramento permitindo realizar pesquisas sobre o mesmo.
- Permite segurança ao nível de mensagem e de transporte. O barramento é disponibilizado com um conjunto arquivos XML WS-Policy que facilita a aplicação de políticas de segurança nos serviços
- Suporta as especificações: WS-Policy, WS-RealiableMessaging, XACML, WS-Addressing, SCA, XPDL, SAML, PKI.

O OSB não implementa JBI, BPEL4People ou WS-HumanTask. Se o objetivo for criar processos de negócio que envolva automação de serviços bem como tarefas humanas, então é melhor olhar a suíte BPM.

Um estudo das funcionalidades do ALSB é apresentada em [Souza *et al.*, 2009].

Além do Oracle Service Bus, Hewitt [2009] apresenta as principais características do Software AG/webMethods ESB, TIBCO ActiveMatrix e BusinessWorks.

3 Arquitetura do Oracle Service Bus

O Oracle Service Bus (OSB) [OSB, 2009a] corresponde à nova versão desenvolvida pela Oracle da ferramenta ALSB (AquaLogic Service Bus) que pertencia à BEA e que foi descontinuada após a compra da BEA pela Oracle. Maiores detalhes sobre o ALSB podem ser encontrados em [Souza *et al.*, 2009].

A instalação do OSB pode ser baixada a partir do link http://download.oracle.com/docs/cd/E13196_01/platform/suppconfigs/config_alsb.html, onde se encontram também os links para download das versões anteriores do ALSB.

3.1 Suíte de produtos do OSB

A Oracle possui uma suíte de produtos que pode ser compartilhada via Oracle Service Bus [OSB, 2008a] (Figura 6):

- Oracle User Interaction: permite a criação de soluções contemplando infraestrutura de serviços, incluindo portais e aplicações compostas.
- Oracle Business Process Management: inclui automatização, execução e monitoramento do ciclo de vida de processos de negócio como um todo.
- Oracle Data Service Integrator: permite a criação de serviços de dados que disponibilizam visões unificadas e em tempo real dos dados em diferentes fontes de dados espalhadas pela organização

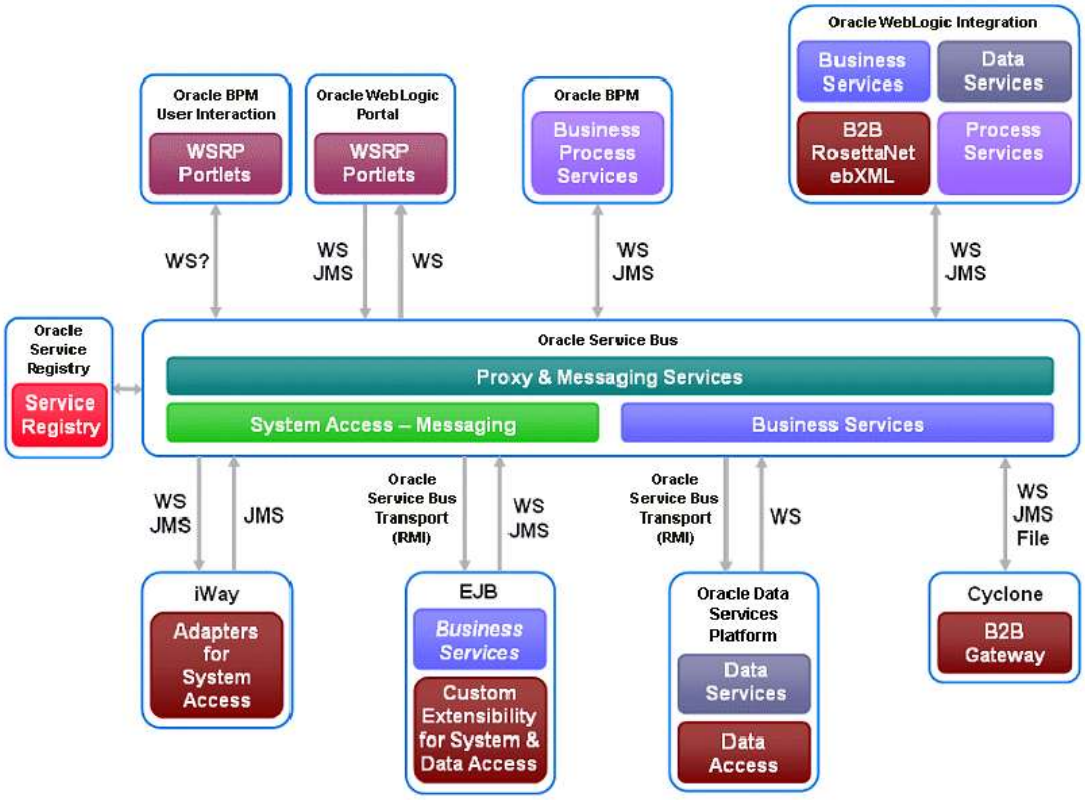

Figura 6 – Arquitetura de produtos compartilhados via OSB [OSB, 2008a]

A Figura 7 apresenta os produtos da Oracle que têm relevância em uma Arquitetura Orientada a Serviços. Dentre estes produtos destacam-se os produtos para desen-

volvimento de portais (Oracle WebLogic Portal e Oracle WebCenter Interaction), para a camada de processos (Oracle WebLogic Integrator e Oracle Business Process Management), serviços de segurança (Oracle Enterprise Security), serviços de dados (Oracle Data Services Platform) e registro de serviços (Oracle Service Registry).

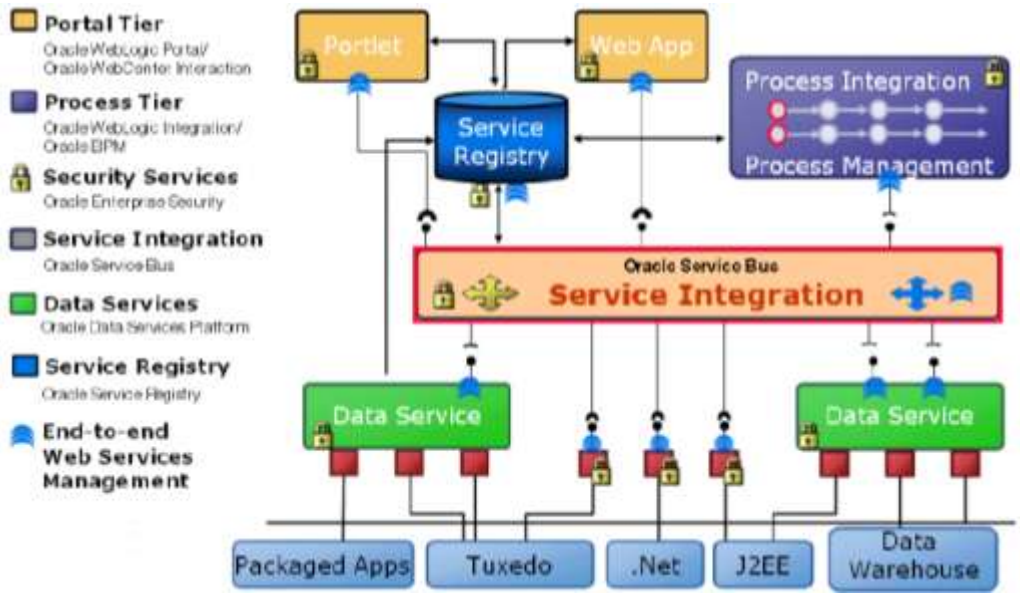

Figura 7 – Produtos da Oracle separados em camadas

Segundo [OSB, 2008a], o Oracle Service Bus atende ao estágio de execução, em um ciclo de vida de desenvolvimento de serviços, tratando as seguintes atividades:

- Publicação e provisão do serviço;
- Gestão e monitoramento do fluxo de mensagens entre consumidores e provedores;
- Separação de usuários e processos das mudanças dos serviços;
- Abstração de serviços e remoção de lógica de integração;
- Transformações, validação e roteamento;
- Visibilidade e gestão operacional do serviço.

Analisando a proposta de ciclo de vida de serviços de Gu e Lago [2007], apresentado na Figura 8, podemos notar a aderência das atividades propostas de serem realizadas no OSB em relação a este ciclo de vida. Além disso, o barramento também pode ser utilizado na fase de projeto em relação aos seguintes aspectos:

- Pesquisa por serviços, utilizando o console do ESB;
- Implementação de serviços de proxy;
- Implementações de orquestrações de serviços utilizando os fluxos definidos nos serviços de proxy;
- Implementações de transformações para tornar a comunicação entre consumidor-provedor menos dependente;
- Definições de roteamento de serviços;
- Realização de testes de serviços e orquestrações através do console de teste.

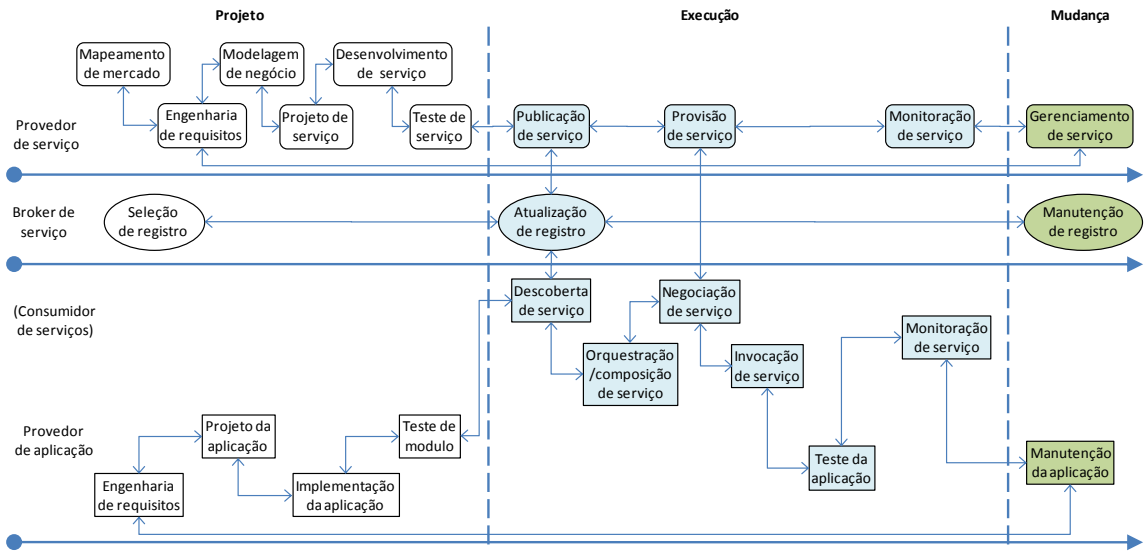

Figura 8 – Ciclo de vida para serviços [Gu e Lago, 2007]

As principais características do OSB são:

- Integração de serviços: integração de *endpoints*, *broker* de mensagens e mediação e exposição de serviços para reuso.
- Segurança de serviços: autenticação e autorização, validação da identidade do usuário.
- Composição de serviços: configuração da lógica de roteamento da mensagem, transformação de mensagens, configuração, validação e registro de serviços.
- Gestão de serviços: monitoramento e gestão das atividades e disponibilidade dos serviços.

3.2 Componentes da arquitetura do OSB

A arquitetura do OSB pode ser dividida em 4 camadas:

- Camada de transporte: permite integrar serviços desenvolvidos utilizando diferentes protocolos. Os protocolos de comunicação que podem ser utilizados no OSB são:
  - HTTP/SOAP
  - WS-I, WS-Security, WS-Policy, WS-Addressing
  - SOAP v1.1 e v1.2: o OSB aceita mensagens de entrada em SOAP v1.1 e ele transforma a mensagem para SOAP v1.2 se for necessário, por exemplo, para se conectar com um serviço que se comunica via SOAP v1.2.
  - EJB/RMI no WebSphere
  - Permite a criação e a customização de protocolos específicos da organização, bem como usar protocolos nativos do Oracle, por exemplo, Oracle Data Service Integrator.
- Camada de segurança (Figura 9): permite os seguintes níveis de segurança:

- Segurança no transporte da mensagem: SSL/Basic Auth
- Segurança na mensagem: WS-Policy/WS-Security, SAML, UserID/Password, X509, Signing & Encryption, e Custom security credentials
- Segurança de console: suporta Web Single-Sign-On e acesso baseado em perfis
- Políticas: WS-Security e WS-Policy

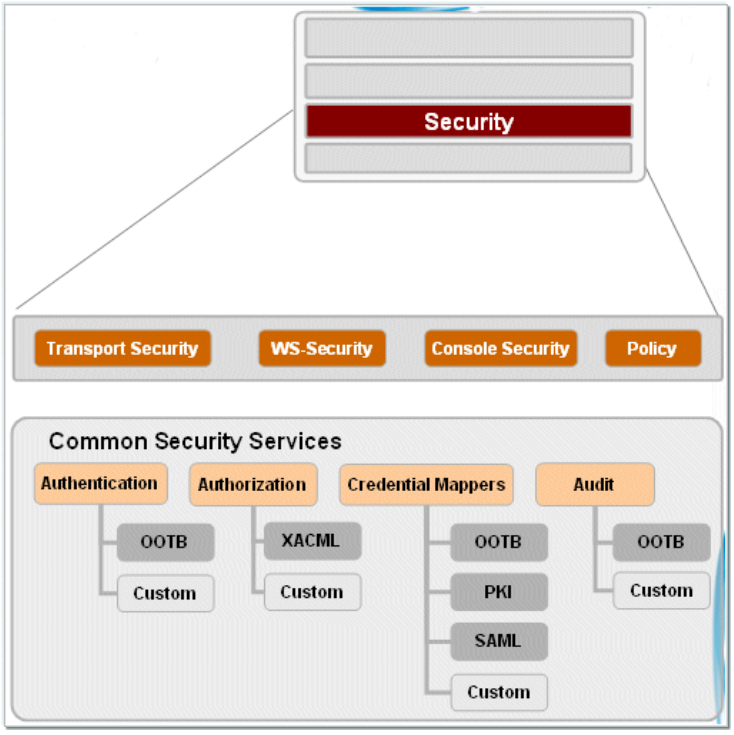

Figura 9 – Camada de segurança

- Camada para composição de serviços (Figura 10): esta camada inclui a modelagem do fluxo de serviços, além de atividades de descoberta de serviços em UDDI, validação, transformação, testes da orquestração, além da especificação de *service callout*. *Service callout* corresponde a uma ação que invoca um determinado serviço de acordo com o conteúdo da mensagem.

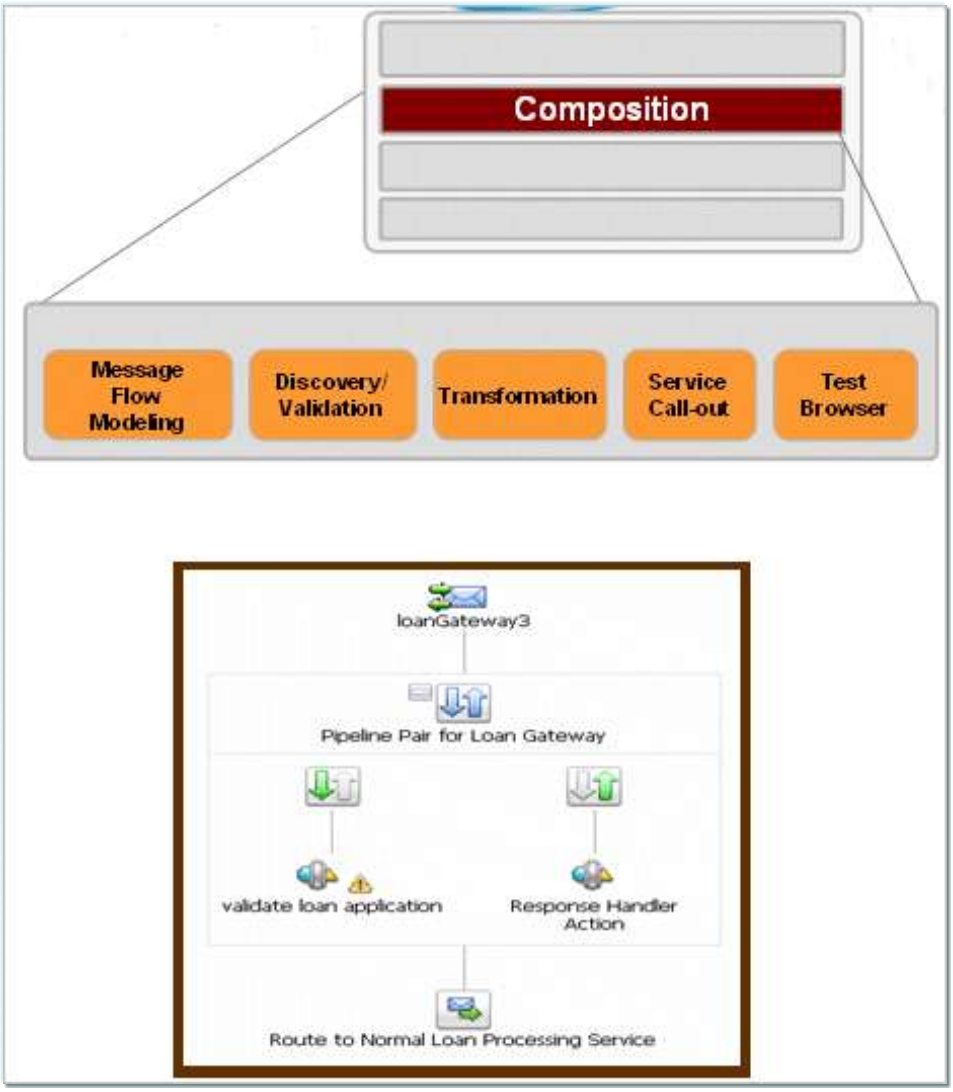

Figura 10 – Camada de composição

- Camada para monitoramento de serviços (Figura 11): permite realizar governança sobre as mensagens que trafegam pelo barramento disponibilizando: dashboard, monitoramento, definição de alertas, relatórios, verificação de SLA etc.

Figura 11 – Camada de gestão

3.3 Fluxo de mensagens via serviços de proxy

O OSB é um intermediário que processa mensagens de entrada para requisição de serviços, determina a lógica de roteamento, e transforma estas mensagens para compatibilidade com outros consumidores de serviços. Ele recebe mensagens através de protocolos de transporte, tais como HTTP(S), JMS, Arquivos e FTP, e envia mensagens através do mesmo protocolo ou de protocolos diferentes. A mensagem de resposta do serviço segue um caminho inverso. O processamento de mensagens pelo OSB é direcionado a metadados, especificados na definição do fluxo de mensagens de um serviço de *proxy*. Serviço de *proxy* é o conceito fundamental na arquitetura do OSB. Eles correspondem à interface que consumidores utilizam para se comunicar com serviços de *back-end* gerenciados.

A Figura 12 ilustra em alto nível o fluxo de mensagens através do OSB, a partir de um *endpoint* de entrada (para um serviço de *proxy*) para um *endpoint* de saída (URL de outro serviço – um serviço de negócio ou um serviço de *proxy*).

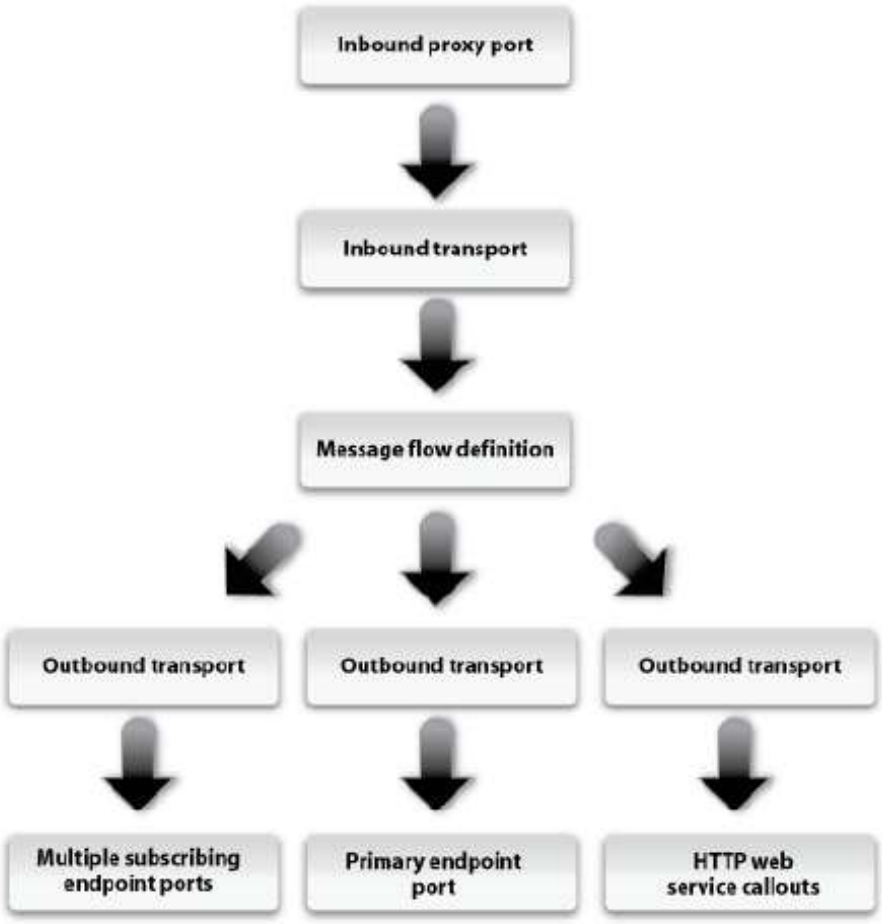

Figura 12 – Processamento de mensagens

Três camadas estão envolvidas no fluxo de mensagens (Figura 13):

- Camada de transporte (*transport layer*):
  - Recebe o stream de dados da requisição do cliente (ou consumidor);
  - Envia o stream de dados da requisição para o provedor;
  - Recebe o stream de dados da resposta do provedor;
  - Envia o stream de dados da resposta para o cliente (ou consumidor).
- Camada de ligação (*binding layer*):
  - Desserializa e serializa mensagens, quando necessário;
  - Trata questões de segurança da mensagem;
  - Manipula mensagens para iniciar fluxo de mensagens (requisição e resposta)
- Serviço de proxy (*proxy service*)
  - Serviços de proxy são definições de web services intermediários que o OSB implementa localmente
  - O OSB permite a definição de serviços de proxy pela definição de sua interface via WSDL (Web Services Description Language) e o tipo de transporte que ele utiliza.

- A lógica de processamento da mensagem é especificada na definição do fluxo da mensagem quando um serviço de proxy é definido.
- O contexto de um serviço de proxy é um conjunto de variáveis XML que são compartilhadas ao longo do fluxo do serviço.
    - Variáveis podem conter informações a respeito de:
        - Mensagem
        - Cabeçalhos de transporte
        - Princípios de segurança
        - Metadados para o serviço de proxy
        - Metadados para roteamento primário
        - Serviços invocados pelo serviço de proxy
    - O contexto pode ser lido ou modificado por expressões XQuery e atualizado por transformações e ações de atualização
    - As variáveis principais do context são: $header (SOAP), $body (SOAP) e $attachments (MIME - Multipurpose Internet Mail Extensions)
    - O contexto dá a impressão de que todas as mensagens são SOAP. Mensagens não-SOAP são traduzidas para este paradigma.
    - Como um serviço de proxy pode rotear mensagens para múltiplos serviços de negócio, um serviço de proxy pode ser configurado com uma interface que é independente do serviço de negócio ao qual ele se comunica.

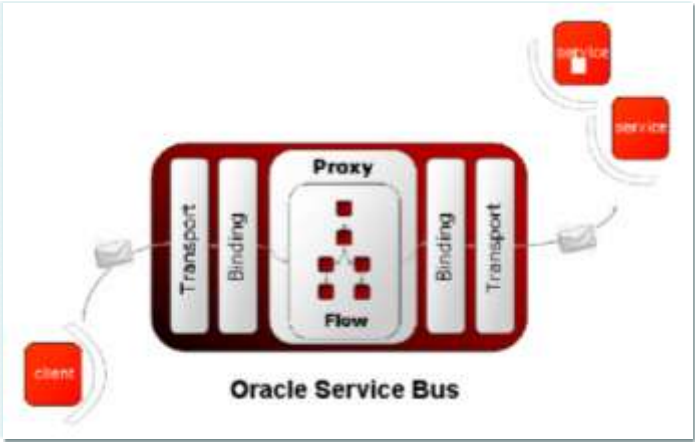

Figura 13 – Camadas do fluxo de mensagens

A maior parte da lógica de processamento em um fluxo de mensagem é tratada em *pipelines*. Um *pipeline* é uma sequência nomeada de estágios representando um caminho de processamento em uma única direção sem ramificações. Um estágio é um passo de processamento configurado pelo usuário. Mensagens de entrada para *pipelines* são acompanhadas de um conjunto de variáveis de contexto da mensagem que contém conteúdos da mensagem. Elas podem ser acessadas ou modificadas por ações dentro do estágio do *pipeline*.

Pipelines pertencem a uma das seguintes categorias:

- *Pipeline* de requisição processa o caminho de requisição do fluxo da mensagem;
- *Pipeline* de resposta processa o caminho de resposta do fluxo de mensagem;
- *Pipeline* de erro trata erros para estágios e nós em um fluxo de mensagem tão bem como erros no nível do fluxo da mensagem (serviço).

Os seguintes elementos são utilizados para construir um fluxo de mensagens:

- Nó inicial: Toda mensagem começa com um nó inicial. Todas as mensagens entram no fluxo de mensagem através do nó incial, e todas as mensagens de respostas são retornada para o cliente através do nó inicial. Não existe nada para ser configurado no nó inicial.
- Par *pipeline*: um nó par *pipeline* combina um *pipeline* de requisição e um *pipeline* de resposa em elemento de alto nível. Um nó par *pipeline* pode ter apenas um descendente direto no fluxo da mensagem. Durante processamento de requisição, apenas o *pipeline* de requisição é executado quando o OSB procesa o nó par *pipeline*. O caminho de execução é o inverso quando o OSB processa o *pipeline* de resposta. Consistem de sequência de estágios que especificam ações a serem realizadas durante o processamento de *request* e *response*.
- Estágios: *pipelines* de requisição, *pipelines* de resposta, e tratadores de erros podem conter estágios, onde podem ser configuradas ações para manipular mensagens passando através do *pipeline*.
- Tratamento de erro: Um tratamento de erro pode ser incluído em qualquer nó ou estágio para tratar erros potenciais naquele local.
- Nó de divisão: permite a divisão do fluxo de acordo com partes da mensagem, contexto da mensagem, ou baseado na operação invocada. Um único caminho é escolhido dentre diferentes caminhos.
- Nó de roteamento: um nó de roteamento realiza comunicação de requisição/resposta com outro serviço. Ele representa o limite entre o processamento da requisição e resposta para o serviço de proxy. Quando o nó de roteamento dispara uma mensagem de requisição, o processamento de requisição é considerado completo. Quando o nó recebe uma mensagem de resposta, o processamento da resposta começa. O nó de roteamento suporta roteamento condicional bem como transformações de requisição e resposta. Dado que o nó de roteamento representa o limite entre o processamento da requisição e resposta, ele não pode ter nenhum descendente no fluxo da mensagem. Logo, ele é usado para definir o destino da mensagem.
- Ações: Ações proveem instruções para tratamento de mensagens em estágios de *pipelines*, estágios de tratamento de erros e nós de roteamento. Ações podem ser dos seguintes tipos. Maiores detalhes são apresentados em [OSB, 2009e].
    - Ações de comunicação: controlam o fluxo de mensagens e podem ser caracterizadas como de publicação dinâmica, de publicação, tabela de publicação, opções de roteamento, *callout* para serviços, cabeçalhos de transporte.
    - Ações de controle de fluxo: implementam roteamento condicional, *looping* condicional e tratamento de erro. Podem ser utilizadas para no-

tificar o requisitante de sucesso ou para pular (*skip*) o resto das ações em um estágio. Ações de controle de fluxo podem ser do tipo *for each*, *if... then...*, levantamento de erro, resposta, *resume*, *skip*.

- Ações de processamento de mensagens: ações nesta categoria processam o fluxo das mensagens, tais como: *assign*, *delete*, *insert, Java callout*, transformação MLF, *rename*, *replace*, *validate*. Em particular, a ação *validate* valida elementos selecionados por uma expressão XPath em relação a um elemento de um XML Schema ou um recurso WSDL. Podem ser validados apenas elementos de variáveis globais; OSB não suporta validação de elementos locais.
- Ações de relatório: são usadas para registrar ou reportar (*log* ou *report*) erros e alertas gerados, se necessário em um fluxo de mensagem dentro de um estágio. Elas podem ser dos tipos: *alert, log* e *report*. Diferente de alertas de SLA, notificações geradas por ações de alerta são ligadas mais ao negócio, ou erros de *report*, e não para monitoramento do sistema. Destinos de alertas devem ser configurados tendo isto em mente.

A Figura 14 apresenta um exemplo de fluxo de mensagem. Nesta figura têm-se um *pipeline* de requisição em um nível de serviço único que ramifica em *pipelines* operacionais (pode-se configurar um *pipeline* padrão para uma operação, mas no máximo um *pipeline*). A determinação da operação é feita via critérios do usuário.

- Processamento da requisição:
  - O processamento da requisição começa no nó inicial.
  - Em seguida, o par *pipeline* executa apenas o processamento da requisição.
  - O nó de decisão (nó *branch*), avalia a tabela de decisão para o fluxo que foi avaliado como verdadeiro.
  - O nó de roteamento realiza o roteamento juntamente com qualquer transformação da mensagem de requisição que tenha sido definida.
    - No fluxo de mensagem, independente se o roteamento exista ou não, o nó de roteamento representa a transição do processamento de uma requisição para o processamento da resposta. No nó de roteamento, a direção do fluxo de mensagem é invertida. Se um caminho de requisição não tem nó de roteamento, o processamento da respostas é iniciado no sentido inverso sem esperar por qualquer resposta.
- Processamento da resposta:
  - O nó de roteamento executa qualquer transformação da resposta que tenha sido definida;
  - O nó de divisão pula qualquer nó de divisão e continua com o nó que precede a divisão;
  - O par *pipeline* executa o *pipeline* da resposta;
  - O nó inicial envia a resposta de volta para o cliente.

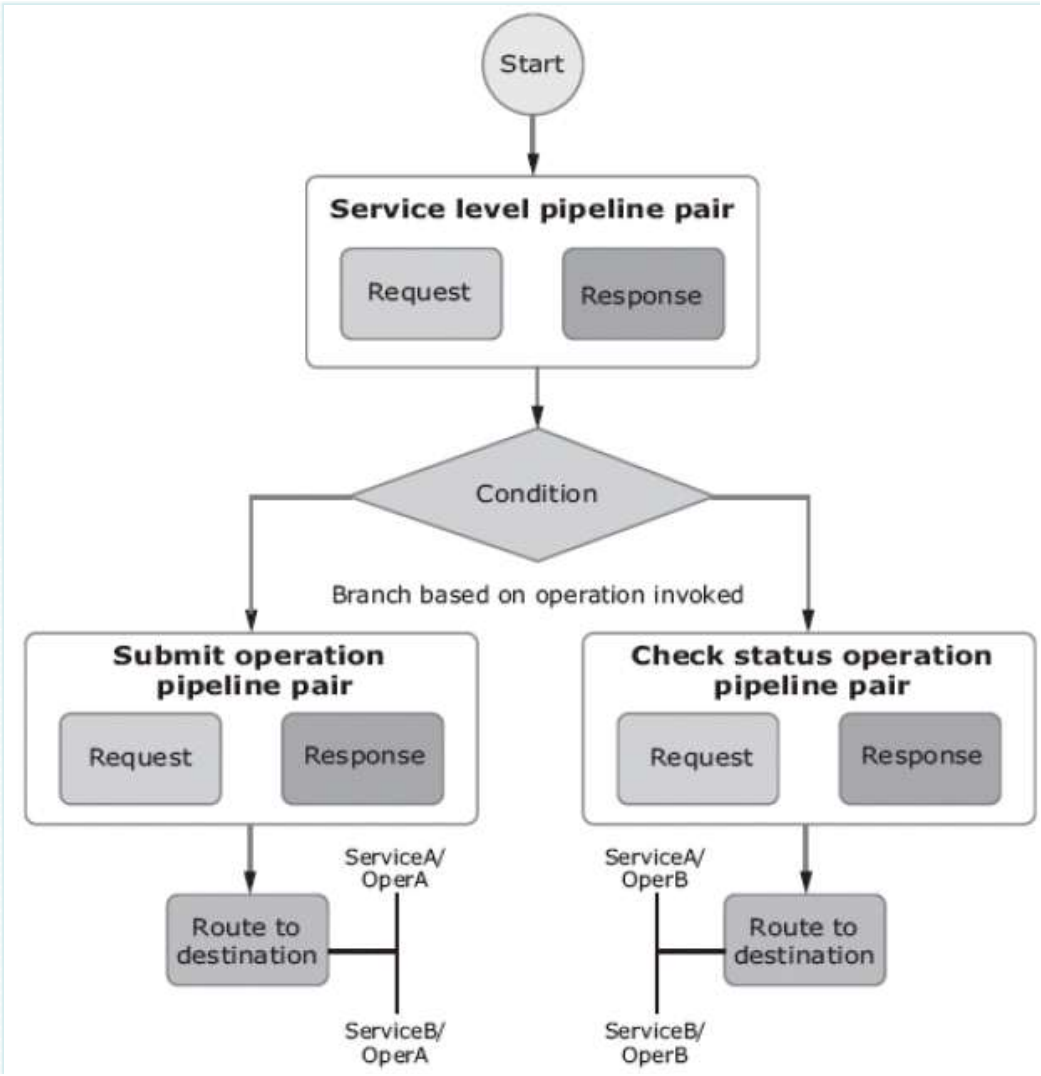

Figura 14 – Exemplo de fluxo de mensagem

Transformações podem ser definidas antes de a mensagem ser enviada para o *end-point* selecionado ou após a resposta ser recebida [OSB, 2009c].

Processamento para tratar questões de segurança (WS-Security) e autorização podem ser utilizados ao invocar serviço com WS-Policy [OSB, 2009d].

3.4 WSDL efetivos e WSDL gerado

Um documento WSDL descreve um serviço, sua localização, suas operações, e o modo no qual clientes podem se comunicar com ele.

No OSB, novos serviços de proxy e novos serviços de negócio podem ser baseados em WSDL (chamados "recursos WSDL") e então sobrescrever ou adicionar propriedades de configuração.

Para serviços baseados em WSDL, OSB usa WSDL efetivos ao invés dos arquivos .wsdl originais. Um WSDL efetivo representa propriedades de um WSDL de serviço considerando as configurações do OSB.

Ao criar um serviço baseado em um WSDL, o OSB cria um WSDL efetivo. WSDL efetivos podem ser criados a partir dos seguintes tipos de serviços de negócio ou de

proxy: serviços SOAP criados a partir de WSDL e serviços XML criados a partir de WSDL.

Outro tipo de WSDL importante a ser considerado é o WSDL gerado, o qual é um WSDL efetivo que o OSB gera para serviços de tipo de transporte que não foram criados a partir de um recurso WSDL, mas que podem ser descritos usando WSDL. Por exemplo, um WSDL gerado a partir de um serviço baseado em EJB é um WSDL gerado.

O OSB permite exportar o WSDL efetivo. A exportação gera um arquivo .zip que contém o WSDL efetivo juntamente com suas dependências, incluindo esquemas e *WS-Policies*. OSB avalia as dependências, e a localização apropriada é adicionada no atributo *location* do elemento *import* do WSDL. Não existe elemento *import* para *WS-Policies*. Neste caso, as referências para as políticas são retidas e o recurso da política é incluído na exportação. WSDL gerado não pode ser exportado.

Maiores detalhes a respeito das características que se aplicam a WSDL efetivos gerados para serviços de proxy, WSDL efetivos gerados para serviços de negócio sem tipo de transporte, WSDL efetivos gerados para serviços de negócio com tipo de transporte e WSDL efetivos gerados em domínios de *cluster*, além de serviços baseados em uma porta, serviços baseados em *binding* são apresentados em [OSB, 2009e].

3.5 Web services com atributo *style* igual a SOAP document × RPC

Um web service empacotado como documento é descrito em um arquivo WSDL como um serviço com estilo (*Style*) *Document*. Como padrão, uma operação de um web service orientado a documento pode ter apenas um parâmetro ou tipo de mensagem, tipicamente um documento XML. Isto significa que os métodos que implementam as operações devem ter apenas um único parâmetro. Entretanto, operações de web services empacotados em documento podem ter qualquer número de parâmetros, embora os valores dos parâmetros sejam empacotados em um único tipo de dados complexo na mensagem SOAP. Este tipo de dados complexo empacotado será descrito no WSDL como um único documento para a operação. A Figura 15 apresenta um exemplo de WSDL de serviço orientado a documento, enquanto que a Figura 16 e a Figura 17 apresentam o corpo (*$Body*) das mensagens de requisição e respostas do serviço, respectivamente. Na Figura 16, *soap-env* é namespace SOAP 1.1 préfinido e *req* é o namespace do elemento *PurchaseOrg* (http://example.com/lookup/docs).

```
<definitions name="Lookup"
targetNamespace="http://example.com/lookup/service/defs"
xmlns:tns="http://example.com/lookup/service/defs"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:docs="http://example.com/lookup/docs"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xs:schema    targetNamespace="http://example.com/lookup/docs"
elementFormDefault="qualified">
      <xs:element name="PurchaseOrg" type="xs:string"/>
      <xs:element name="LegacyBoolean" type="xs:boolean"/>
    </xs:schema>
  </types>
  <message name="lookupReq">
    <part name="request" element="docs:purchaseorg"/>
  </message>
  <message name="lookupResp">
    <part name="result" element="docs:legacyboolean"/>
  </message>
  <portType name="LookupPortType">
    <operation name="lookup">
      <input message="tns:lookupReq"/>
      <output message="tns:lookupResp"/>
    </operation>
  </portType>
  <binding name="LookupBinding" type="tns:lookupPortType">
    <soap:binding                                    style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="lookup">
      <soap:operation/>
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
</definitions>
```

Figura 15 – Exemplo de WSDL de serviço orientado a documento

```
<soap-env:body>
  <req:purchaseorg>Oracle</req:purchaseorg>
</soap-env:body>
```

Figura 16 – Exemplo de mensagem SOAP de requisição do serviço orientado a documento

```
<soap-env:body>
  <req:legacyboolean>true</req:legacyboolean>
</soap-env:body>
```

Figura 17 – Exemplo de mensagem SOAP de resposta do serviço orientado a documento

No caso do estilo de *binding* ser RPC, a operação do serviço recebe um conjunto de parâmetros na requisição e retorna como resposta um conjunto de parâmetros. A Figura 18 apresenta um exemplo de WSDL de serviço com estilo RPC, enquanto que a Figura 19 e Figura 20 apresentam o corpo ($Body) das mensagens de requisição e respostas do serviço, respectivamente. O elemento *soap-env* corresponde ao namespace predefinido em SOAP 1.1, *ns* é o namespace da operação (http://example.com/lookup/service) e *req* é o namespace do elemento *PurchaseOrg* (http://example.com/lookup/docs).

```
<definitions name="Lookup"
targetNamespace="http://example.com/lookup/service/defs"
xmlns:tns="http://example.com/lookup/service/defs"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:docs="http://example.com/lookup/docs"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
</xs:sequence>
      </xs:complexType>
    </xs:schema>
  </types>
  <message name="lookupReq">
    <part name="request" type="docs: RequestDoc"/>
  </message>
  <message name="lookupResp">
    <part name="result" type="docs: ResponseDoc"/>
  </message>
  <portType name="LookupPortType">
    <operation name="lookup">
      <input message="tns:lookupReq"/>
      <output message="tns:lookupResp"/>
    </operation>
  </portType>
  <binding name="LookupBinding" type="tns:lookupPortType">
    <soap:binding                                    style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="lookup">
      <soap:operation/>
      <input>
        <soap:body                                  use="literal"
namespace="http://example.com/lookup/service"/>
      </input>
      <output>
        <soap:body                                  use="literal"
namespace="http://example.com/lookup/service"/>
      </output>
    </operation>
  </binding>
</definitions>
```

Figura 18 – Exemplo de WSDL de serviço com estilo RPC

```
<soap-env:body>
  <ns:lookup>
    <request>
      <req:purchaseorg>Oracle</req:purchaseorg>
    </request>
  </ns:lookup>
<soap-env:body>
```

Figura 19 – Exemplo de mensagem SOAP de requisição do serviço com estilo RPC

```
<soap-env:body>
  <ns:lookupResponse>
    <result>
      <req:legacyboolean>true</req:legacyboolean>
    </result>
  </ns:lookupResponse>
<soap-env:body>
```

Figura 20 – Exemplo de mensagem SOAP de resposta do serviço com estilo RPC

3.6 Padrões de troca de mensagens

O OSB permite os seguintes tipos de comunicação de mensagens:

- Requisição/resposta síncrona: cliente envia a requisição e fica esperando a resposta do provedor.
- Publicação um-para-um assíncrona: um cliente se publica como consumidor de um serviço, e o serviço envia a sua resposta para o cliente publicado após a execução da operação para a qual o cliente foi publicado.
- Publicação um-para-muitos assíncrona: vários clientes se publicam como consumidores de um serviço, e o serviço envia a sua resposta para os clientes publicados após a execução da operação para o qual os consumidores foram publicados.
- Requisição/resposta assíncrona (ponte de síncrono para assíncrono).
    - Cliente síncrono envia requisições para provedor assíncrono, e cliente fica aguardando a resposta
    - OSB provê uma fila JMS para a mensagem de requisição e configura outra fila para a mensagem de resposta, com um valor de *timeout* para a resposta. As filas são utilizadas para simular troca de mensagens síncrona.
    - Para o cliente a invocação do serviço é síncrona

3.7 *Broker* de mensagens

O OSB tem como um dos seus componentes principais o *broker* de mensagem. Ele permite o roteamento de mensagens baseado em conteúdo e transformações de dados, incluindo as seguintes características operacionais:

- Políticas baseadas em XQueries ou *callouts* para serviços externos para roteamento de mensagens.
    - A ação de *callout* de serviço é usada dentro de um estágio de roteamento de fluxo de serviço para chamar um serviço de destino para realizar alguma ação na mensagem. A funcionalidade de *Service Callout* do OSB suporta, por exemplo, a codificação RPC e substituição de URL e é extensível pelo uso de Java Callouts e POJOs. Maiores detalhes sobre *Service Callout* podem ser encontrados em [OSB, 2009b].

- Tabela de roteamento fora do serviço de *proxy*, a qual permite a modificação das rotas sem ter que re-configurar definições de serviço de *proxy*;
- Roteamento baseado em identidade: permite classificar clientes em um grupo de usuários definido e aplicar políticas de roteamento para estes grupos.

3.8 Transformação e processamento de mensagens

O Oracle Service Bus suporta as seguintes funcionalidades para transformação ou processamento de mensagens:

- Validação de mensagens de entrada de acordo com *schemas*;
- Seleção de um ou mais serviços, baseado no conteúdo ou no cabeçalho da mensagem;
- Transformação de mensagens baseado no serviço de destino;
- Transformação de mensagens baseado em XQuery ou XSLT;
- Suporta transformações em mensagens nos formatos XML e MFL. MFL (Message Format Language) é um formato de mensagem proprietário da Oracle/BEA. Linguagem usada para descrever como transformar dados binários em XML. A ferramenta Oracle Format Builder é a ferramenta para criar mensagens no formato MFL.
- Suporta chamadas de Web services para obter dados adicionais para transformação (por exemplo, código do país, registros de cliente etc.)

O OSB possui um a funcionalidade chamada de *Database Lookup* através de uma funcionalidade do motor de XQuery do OSB. O *Database Lookup* corresponde à execução de leitura em base de dados a partir de serviços de proxy através do uso da função execute-sql para realizar uma chamada JDBC a um banco de dados para realizar uma leitura simples do banco de dados. Dessa forma, não há necessidade de escrever um EJB customizado ou código Java customizado e sem a necessidade do uso de um componente separado como Oracle Data Service Integrator. O *Database Lookup* deve ser utilizado para inserir dados na mensagem, para ler dados para decisões de roteamento e para customizar comportamento do serviço de proxy.

OSB provê transporte otimizado para comunicação invocando serviços no Oracle Data Service Integrator. Isto permite uma abordagem mais eficiente e flexível para acessar serviços de dados do que expô-los como web services via WebLogic Workshop ou Java Web Services (JWS). Além disso, esta ferramenta suporta segurança e propagação de identidade.

Duas ferramentas para transformação de dados são instaladas com o OSB e Workshop for WebLogic, o Oracle XQuery Mapper como plug-in para o Eclipse 3.1 e Format Builder. O Eclipse 3.1 e o Format Builder são suportados apenas nas plataformas Windows.

3.9 Disponibilização de EJB como serviço

Um EJB pode ser exposto como um web service, sem a necessidade de ferramentas ou modificações de código legado no servidor de aplicação onde o EJB está disponibilizado. O mecanismo de transporte EJB tem as seguintes capacidades:

- Integridade transacional: O transporte EJB pode invocar serviços de negócio no contexto de uma transação global e podem também ser suspensos ou iniciar uma transação global antes de invocar um EJB.
- Propagação de segurança: Uma requisição SOAP sobre HTTP para o OSB é autenticada pelo OSB e o assunto autenticado pode ser propagado para o servidor EJB.
- Provedor JNDI: o transporte EJB disponibiliza um provedor JNDI, o qual pode ser reutilizado por múltiplos serviços de negócio EJB. Isto provê um mecanismo centralizado para administradores gerenciarem configurações do servidor EJB remotamente.
- *Caching* de alto desempenho: o transporte EJB é construído com um cache de alto desempenho e permite o reuso de conexões estabelecidas e minimiza *lookups* de *stubs* EJB.
- Tratamento de falhas e balanceamento de carga: o transporte EJB pode tomar vantagem de cenários onde o mesmo EJB está instalado em múltiplos domínios ou em um *cluster* para balanceamento de carga e/ou tratamento de falhas.
- XML avançado para capacidade de ligação Java: o transporte EJB disponibiliza uma *stack* JAX-RPC para realizar ligação XML para Java;
- *Retries* inteligentes: o transporte EJB toma decisões de *retries* baseado na natureza da falha que pode ocorrer durante a invocação de um EJB.

3.10 Console de teste

O OSB disponibiliza um console de teste que pode ser utilizado para validar recursos e expressões XQuery usadas em fluxos de mensagens. Os objetos a serem testados podem ser: serviço de proxy, serviço de negócio, Xquery, XSLT e recurso MFL. A funcionalidade permite rastrear o fluxo da mensagem quando da execução do teste de um serviço e examinar o estado da mensagem em pontos específicos do teste.

Podem ser testadas partes específicas de um sistema de forma isolada ou pode ser testado o sistema como uma unidade. A forma de invocar o teste pode ser utilizando: Project Explorer, Resource Browser ou XQuery Editor.

3.11 Gestão de recursos

As seguintes funcionalidades são disponibilizadas pelo OSB para gestão de recursos:

- Armazenamento de informações sobre serviços, esquemas, transformações, WSDL e WS-Policies;
- Permite navergar por serviços registrados no OSB;
- Permite propagação de configuração de dados entre ambientes (p.e., ambiente de teste para ambiente de produção).

O OSB provê APIs para customização de: definições de serviço; WSDL; Schemas; XQueries; etc.

Recursos podem ser visualizados através dos seguintes formatos de URL:

- http://host:port/sbresource?WSDL/project/...wsdlname

- http://host:port/sbresource?POLICY/project/...policyname
- http://host:port/sbresource?MFL/project/...mflname
- http://host:port/sbresource?SCHEMA/project/...schemaname
- http://host:port/sbresource?PROXY/project/...proxyname
- http://host:port/sbresource?BIZ/project/...business_service_name

3.12 OSB e UDDI

Serviços de proxy podem ser publicados em qualquer UDDI 3.0 (incluindo Oracle Service Registry). Serviços publicados no OSB podem ser automaticamente publicados no UDDI. Alternativamente, pode-se definir a necessidade de aprovação quando um serviço for alterado no UDDI. Definições de serviços podem ser importadas do UDDI.

3.13 Tratamento de erros

Mensagens de erros podem ser customizadas para serem retornadas para consumidores dos serviços. Tratamentos de erros podem ser configurados para: estágios de pipeline; pipeline inteiro; serviços de proxy. Alertas podem ser enviados de acordo com o conteúdo da mensagem.

3.14 Versionamento

O OSB permite disponibilizar novas versões de serviços. Múltiplas versões de recursos de mensagens (p.e., WSDL, esquemas, parâmetros de segurança) podem coexistir no OSB.

3.15 Monitoramento

O barramento permite configurar alertas para as regras de SLA definidas sobre os serviços, por exemplo: média de tempo de processamento de um serviço; volume processado; número de erros; violações de segurança e erros de validação de esquema.

3.16 Disponibilização do OSB em servidores

OSB pode ser disponibilizado em um servidor único ou em múltiplos servidores (cluster de servidores) (Figura 21).

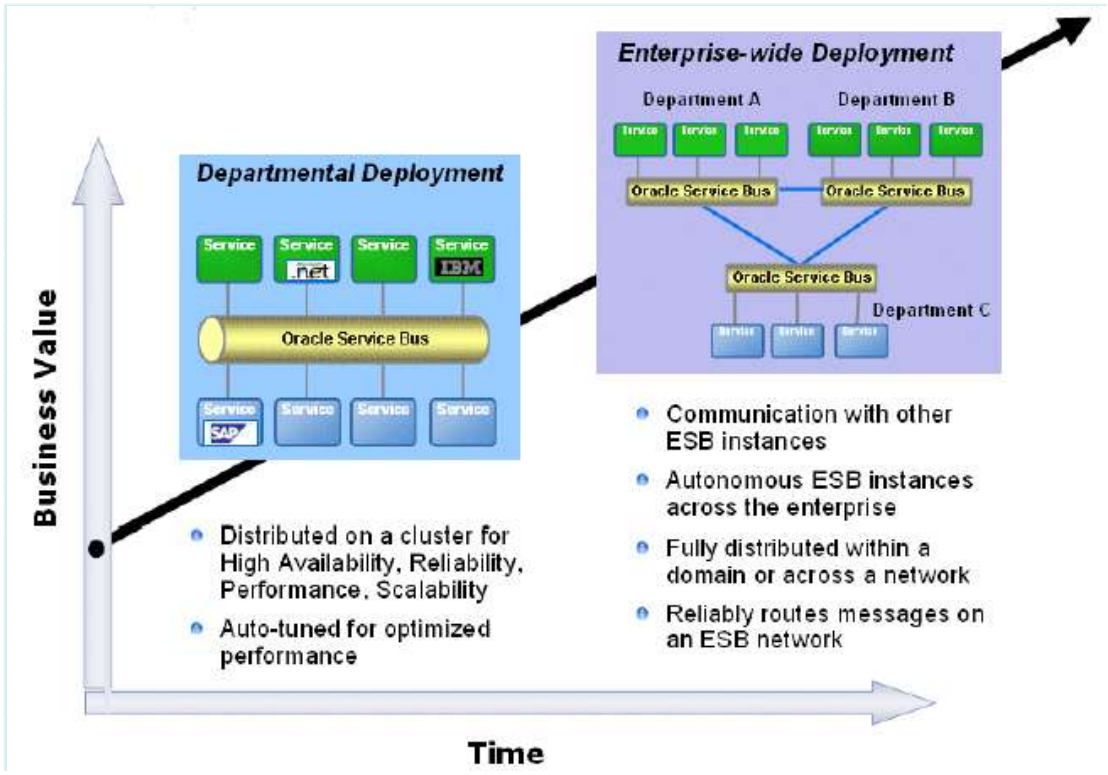

Figura 21 – Formas de implantação do OSB

Quando disponibilizado em um cluster, os outros servidores são gerenciados a partir do servidor central (Figura 22).

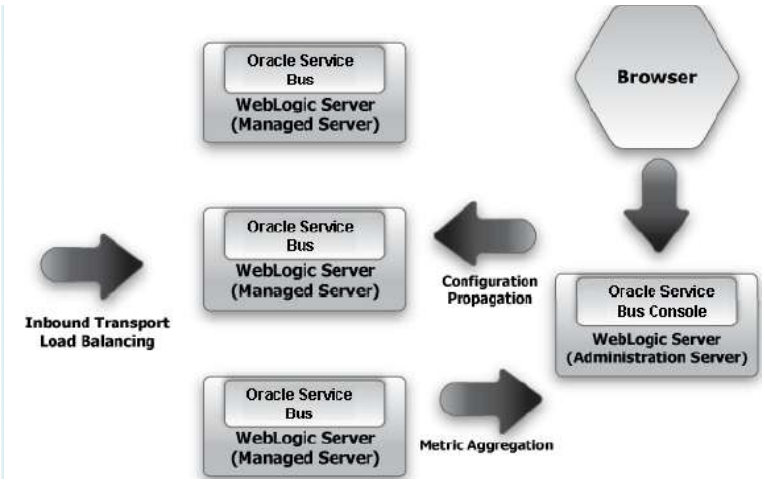

Figura 22 – Gerenciamento do OSB disponibilizado em um cluster de servidores

OSB permite gestão controlada para publicação de serviços em diferentes ambientes (teste, preparação para promoção, produção). O processo pode ser automatizado pela implementação de programas Java ou scripts. As configurações podem ser exportadas em arquivos JAR de um OSB para outro OSB, sendo que a importação pode ser customizada para importar apenas o que se deseja. Tarefas de implantação podem ser customizadas utilizando a ferramenta WebLogic Server Scripting Tool[9].

[9] http://download.oracle.com/docs/cd/E12840_01/wls/docs103/config_scripting/reference.html

3.17 Outros conceitos importantes

Além dos conceitos abordados, outros conceitos importantes em relação ao OSB são listados a seguir, incluindo links para documentação onde maiores detalhes podem ser encontrados.

- Transformação de dados utilizando o XQuery Mapper: http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/usergui de/modelingmessageflow.html
- Segurança (Oracle Service Bus Security Guide): http://download-llnw.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/index.html
- Console de teste: http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/usergui de/testing.html
- MFL (Message Format Language): http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/console help/mfls.html
- Implementações específicas através de uso de APIs: http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/usergui de/appendixAPIs.html
- Definição de papéis para tratamento de monitoramento no OSB: http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/operatio ns/index.html
- Guia e melhores práticas sobre implantação de serviços utilizando o OSB: http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/deploy/ index.html

4 Análise prática do Oracle Service Bus

Esta seção apresenta análises práticas do Oracle Service Bus (OSB).

4.1 Publicação do serviço no OSB

Esta seção descreve os testes de publicação do serviço no OSB e uso do mesmo pelo cliente implementado utilizando o Workshop for WebLogic Platform. Um passo a passo detalhado para publicação de serviços no OSB é apresentado em [Souza *et al.*, 2008, 2009].

A publicação de serviços no barramento compreende duas etapas. Em primeiro lugar, é necessário realizar o *deploy* das aplicações no servidor de aplicação WLS e, em seguida, publicar as interfaces de serviços (WSDL) no OSB. O primeiro passo é necessário, uma vez que o serviço será executado no mesmo WLS que está a sua interface. Para realizar esta etapa, o arquivo ".Jar" do serviço foi gerado e instalado no WLS. A Figura 23 apresenta o resultado da instalação do serviço no WLS.

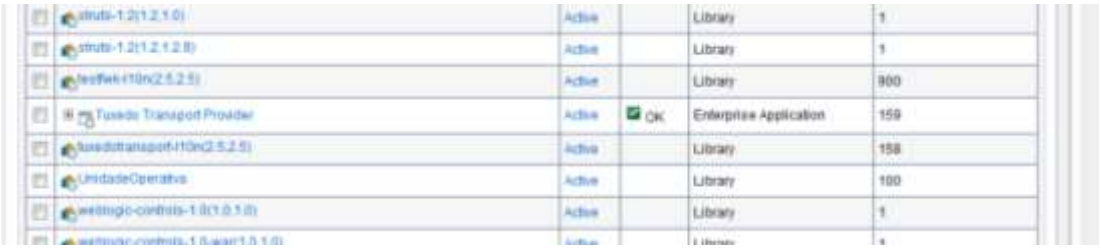

Figura 23 – Arquivo JAR do serviço publicado no WLS

Em seguida, no OSB (Oracle Service Bus), foi criado o projeto UnidadeOperativaPrj e o arquivo WSDL do serviço foi publicado como um recurso deste projeto. A Figura 24 apresenta o resultado da publicação do WSDL do serviço no OSB.

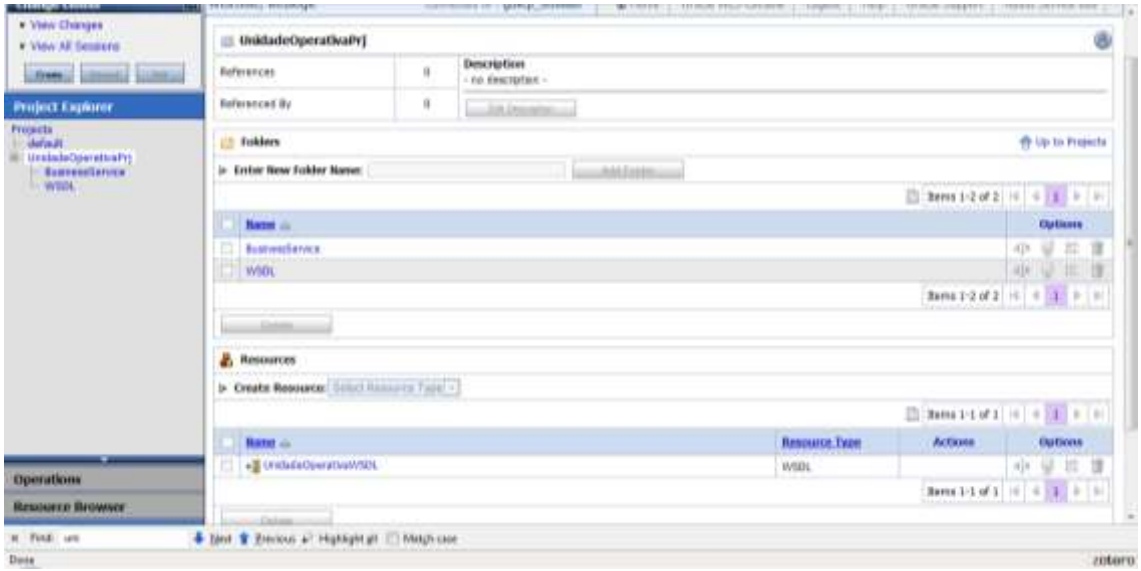

Figura 24 – Publicação do WSDL do serviço no OSB

Após a publicação do arquivo WSDL do serviço, o barramento está parcialmente preparado, pois uma particularidade do OSB é que, ao publicar os WSDL, o serviço não é disponibilizado. Para que o serviço seja disponibilizado realmente no barramento, é necessário criar um recurso chamado Business Service e associar esse recurso ao WSDL que foi adicionado ao projeto. O *Business Service* corresponde à configuração, no barramento, de um serviço cuja execução ocorre fora do barramento. O Business Service foi criado a partir do WSDL publicado anteriormente, como apresentado na Figura 25. O endpoint do serviço é http://localhost:7001/UnidadeOperativa/UnidadeOperativaService , como apresentado na Figura 26.

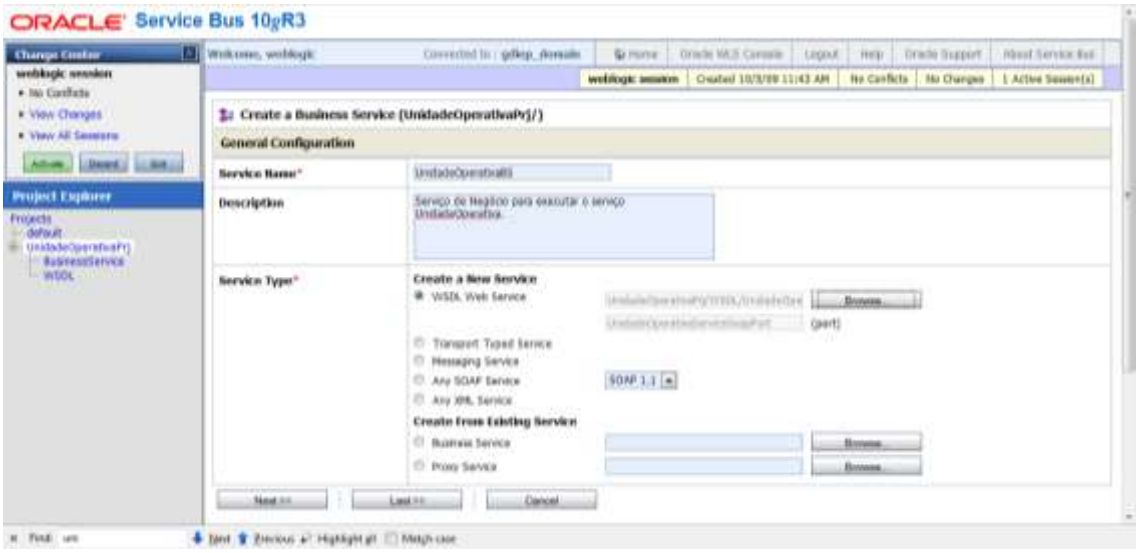

Figura 25 – Criação do *Business Service*

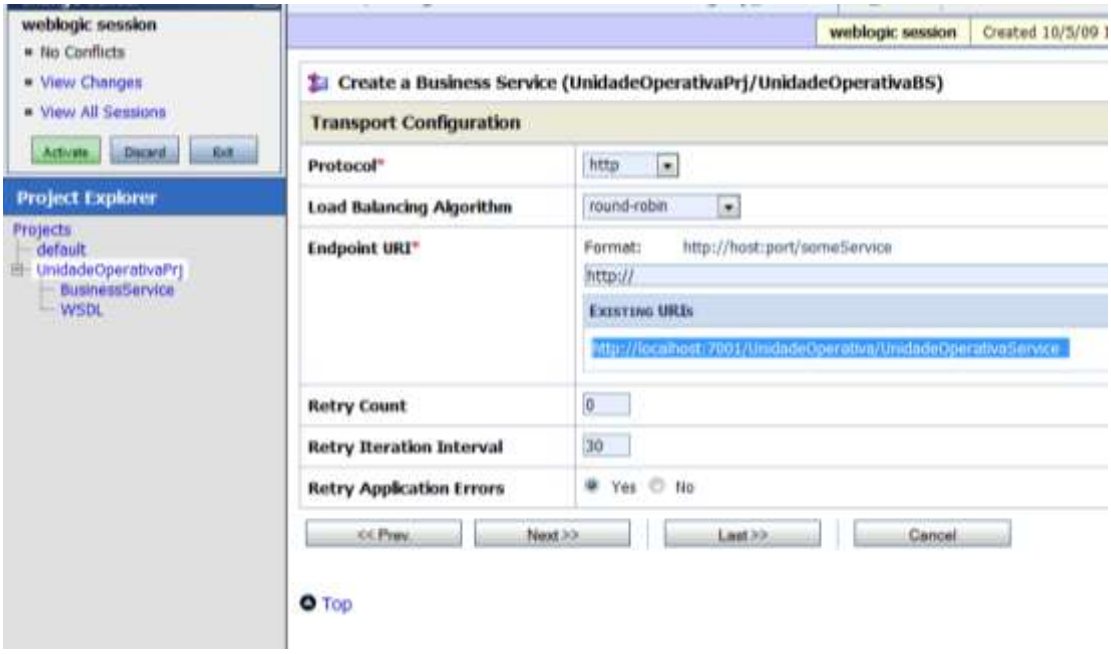

Figura 26 – Endpoint do serviço UnidadeOperativa

Vale ressaltar que o requisito de se criar um *Business Service* para o WSDL aparentemente torna a tarefa de publicação de serviços um tanto redundante. Porém, isso é necessário, pois o barramento permite definir configurações específicas que não são estão no WSDL, tais como: configurações de transporte de mensagens do serviço (método de requisição, se usa autenticação etc.); algoritmo de balanceamento de carga que será usado para esse serviço (*none*, *round-robin*, *random* ou *random-weighted*); número de tentativas a ser executado para re-executar o serviço caso seja necessário, além do intervalo de tempo para as tentativas e se é para tentar novamente se ocorrer erros da aplicação.

A Figura 27 (destacado em verde) apresenta o *Business Service* e o WSDL do serviço criados. Observe que é possível testar a execução do serviço utilizando o ESB. Basta clicar no botão destacado em vermelho na Figura 27. A tela apresentada na Figura 28 é exibida para realizar o teste do serviço.

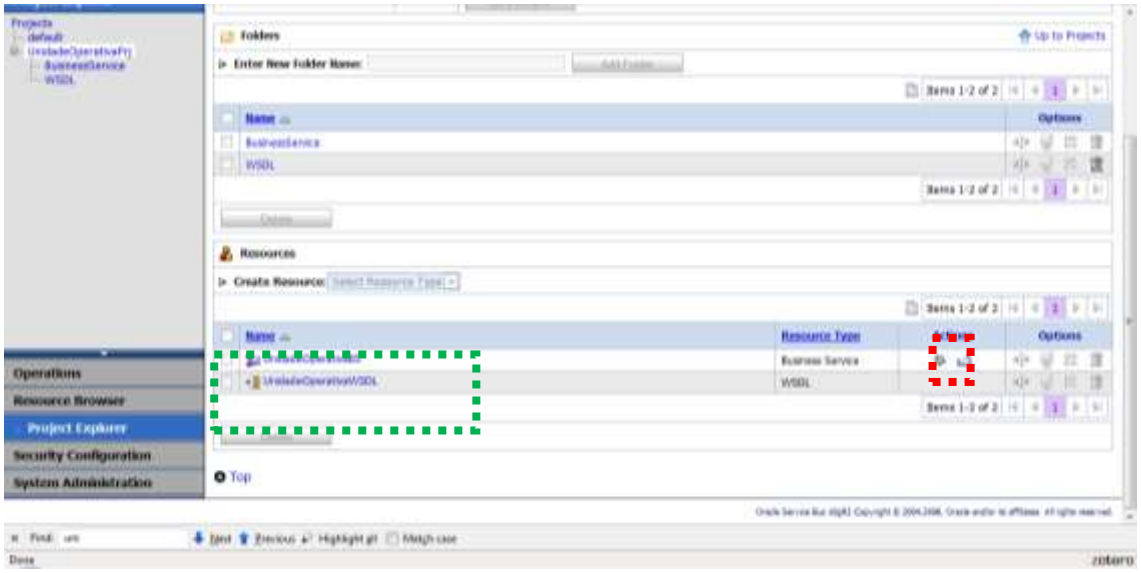

Figura 27 – Business service e WSDL do serviço UnidadeOperativa

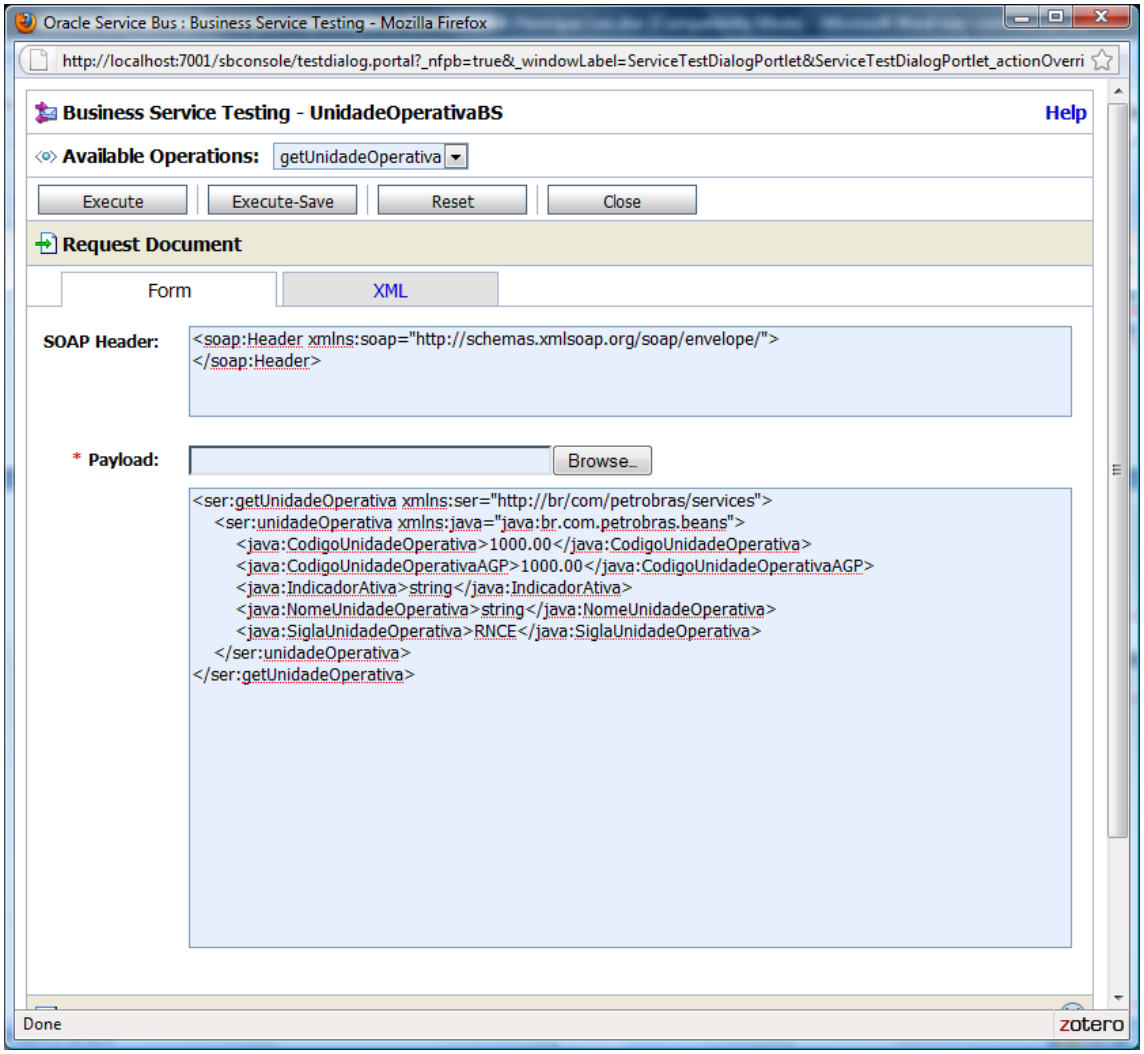

Figura 28 – Interface para teste do serviço

Após a publicação do serviço no barramento, é possível, por exemplo, monitorar o uso do serviço. Para ativar o monitoramento, basta selecionarmos um serviço qualquer

disponível no barramento e clicarmos em Monitoring ➔ Enabled (Figura 29). É possível alterarmos o intervalo de agregação. Esse intervalo corresponde ao intervalo em que as estatísticas do serviço serão agregadas no barramento.

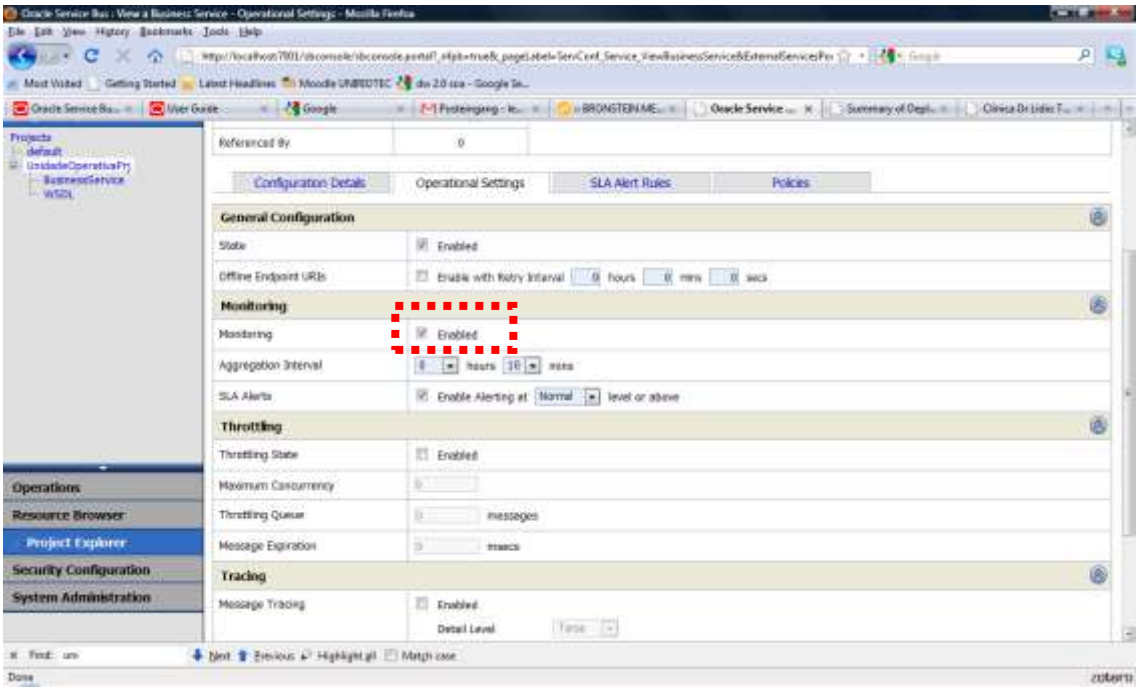

Figura 29 – Habilitação do monitoramento do serviço

Através de alertas, é possível monitorar a execução do serviço. Para este serviço, criamos um alerta para informar quando o serviço for executado em um tempo maior do que 100 milissegundos.

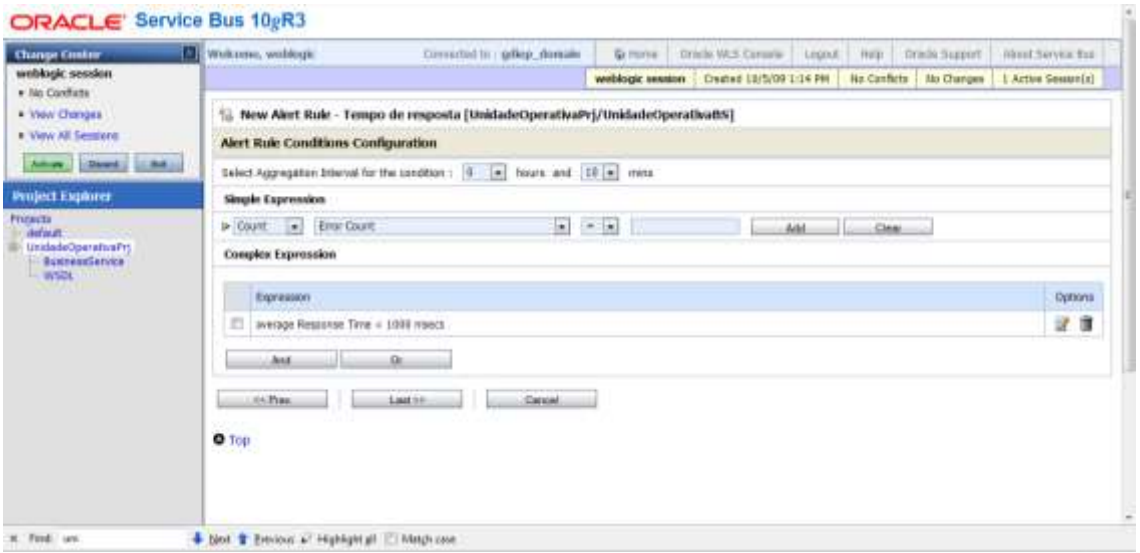

Figura 30 – Configuração do alerta para o serviço

Para que o cliente invoque o serviço utilizando o barramento, ou seja, o barramento servindo como mediador da invocação do serviço real, é necessário criar um serviço de proxy. Maiores detalhes para criação de serviço de proxy são apresentados por Souza *et al.* [2008, 2009]. Neste exemplo, com o objetivo de ilustrar a invocação do serviço de negócio via serviço de proxy, foi criado o serviço UnidadeOperativaPX (Figura 31), a partir do WSDL do serviço (UnidadeOperativa.wsdl). Para este serviço foi adicionada

uma rota (Figura 32) com o objetivo de invocar o método do serviço de negócio (UnidadeOperativaBS), como é ilustrado na Figura 33.

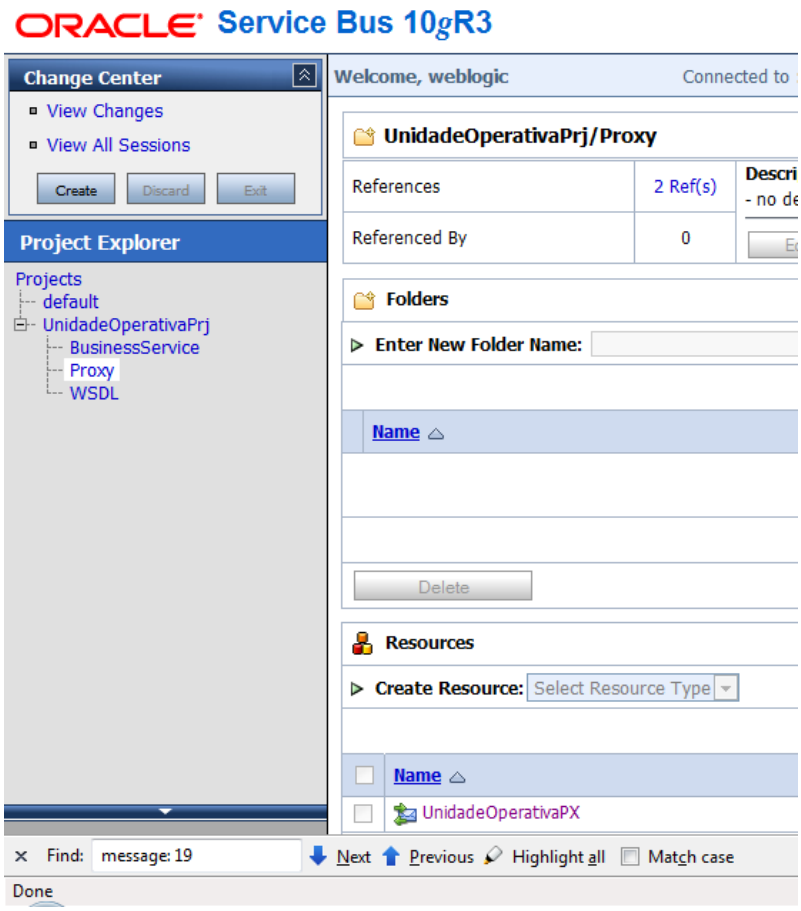

Figura 31 – Serviço de proxy UnidadeOperativaPX do projeto UnidadeOperativaPrj

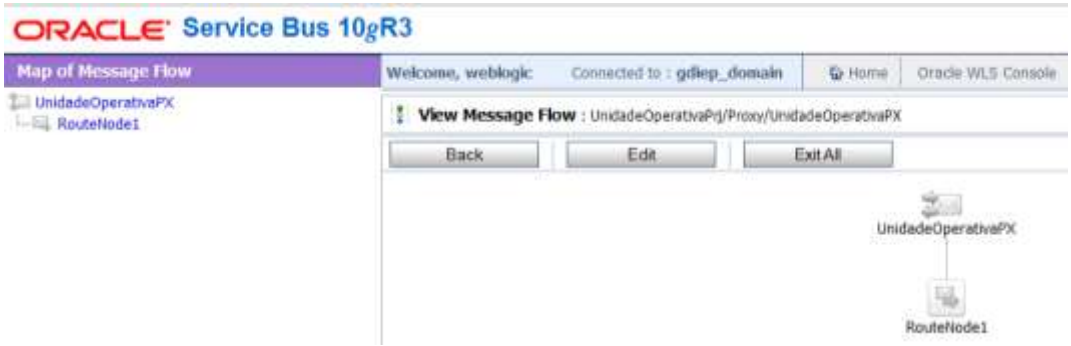

Figura 32 – Rota (RouteNode1) criada para o serviço de proxy invocar o serviço de negócio

Figura 33 – Descrição da rota: invocação do método getUnidadeOperativa do serviço UnidadeOperativaBS

Para o cliente passar a invocar o serviço de proxy ao invés do serviço de negócio, basta substituir a URL de invocação do serviço pela URL do serviço de proxy, como apresentado na Figura 34.

```
*/
public static void main(String[] args) {

    String wsdlurl = "http://localhost:7001/UnidadeOperativaPrj/Proxy/UnidadeOperativaPX?WSDL";

    try{
        locator = new UnidadeOperativaServiceService_Impl(wsdlurl);
        webservice = locator.getUnidadeOperativaServiceSoapPort();

        System.out.println("*****************************************************");


        UnidadeOperativa uo = new UnidadeOperativa();
        uo.setSiglaUnidadeOperativa("BA");
        LstUnidadeOperativa lstUN = webservice.getUnidadeOperativa(uo);

        for (int i = 0; i < lstUN.getUnidadeOperativa().length; i++) {
            Unidadeoperativa un = lstUN.getUnidadeOperativa()[i];
            System.out.println("Codigo: "+un.getCodigoUnidadeOperativa());
            System.out.println("Sigla: "+un.getSiglaUnidadeOperativa());
            System.out.println("Nome: "+un.getNomeUnidadeOperativa());
            System.out.println("Codigo AGP: "+ un.getCodigoUnidadeOperativaAGP());
            System.out.println("*****************************************************");
        }


    } catch (Exception ex){
        ex.printStackTrace();
    }
}
```

Figura 34 – Invocação do serviço de proxy pelo cliente

4.2 Alteração dos parâmetros do serviço sem alterar o cliente

Esta seção descreve os resultados obtidos com alterações no serviço do provedor sem alterar a configuração do serviço no cliente. É importante observar que, utilizando o barramento, o consumidor invoca o serviço de proxy e este invoca o serviço do provedor (serviço de negócio). Os testes apresentados nesta seção tratam da alteração do serviço do provedor sem alterar o serviço de proxy e o consumidor. Os testes realizados seguem os mesmos detalhes de implementação apresentados em [Sousa, 2009], porém o serviço disponibilizado pelo provedor foi implementado no Workshop 10.3, ao invés do Workshop 9.2.

4.2.1 Implementação do serviço antes de realizar o teste

O código referente à implementação do serviço publicado no servidor e configurado no barramento pode ser visto na Figura 35. A classe POJO referente ao objeto retornado pelo serviço e a classe POJO que encapsula os parâmetros para invocação do serviço são apresentadas nas Figura 36 e Figura 37 respectivamente.

```
package br.com.petrobras.services;

import javax.jws.*;

@WebService
public class UnidadeOperativaService {

    @Control
    UnidadeOperativaDbControl unidadeOperativaDbControl;

    @WebMethod
    public UnidadesOperativasListDocument getUnidadeOperativa(ConsultaUnidadeOperativa consulta)
                throws Exception{

        if (consulta != null)
        {
            String where = "";
            UnidadeOperativa[] un = null;

            where="AND UNOP_SG_UNID_OPER = '" + consulta.getSigla() + "' ";
            un = unidadeOperativaDbControl.getUnidadeOperativa(where, "", "", "");

            UnidadesOperativasListDocument xmlUn = parseUnidadeOperativa2XML(un);

            return xmlUn;
        }

        return null;
    }

    private UnidadesOperativasListDocument parseUnidadeOperativa2XML( UnidadeOperativa[] un ) throws Exception {

        UnidadesOperativasListDocument pDoc = UnidadesOperativasListDocument.Factory.newInstance();
            pDoc.addNewUnidadesOperativasList();

            for (int i=0; i < un.length; i++){
                Utils.parsePOJO2XML(un[i], pDoc.getUnidadesOperativasList().addNewUnidadeOperativa());
            }
            return pDoc;
        }
}
```

Figura 35 – Implementação do serviço

```java
package br.com.petrobras.beans;

import java.math.BigDecimal;

public class UnidadeOperativa {
    private BigDecimal codigoUnidadeOperativa;
    private String indicadorAtiva;
    private String siglaUnidadeOperativa;
    private String nomeUnidadeOperativa;
    private BigDecimal codigoUnidadeOperativaAGP;

    public BigDecimal getCodigoUnidadeOperativaAGP() {
        return codigoUnidadeOperativaAGP;
    }
    public void setCodigoUnidadeOperativaAGP(BigDecimal codigoUnidadeOperativaAGP) {
        this.codigoUnidadeOperativaAGP = codigoUnidadeOperativaAGP;
    }
    public BigDecimal getCodigoUnidadeOperativa() {
        return codigoUnidadeOperativa;
    }
    public void setCodigoUnidadeOperativa(BigDecimal codigoUnidadeOperativa) {
        this.codigoUnidadeOperativa = codigoUnidadeOperativa;
    }
    public String getIndicadorAtiva() {
        return indicadorAtiva;
    }
    public void setIndicadorAtiva(String indicadorAtiva) {
        this.indicadorAtiva = indicadorAtiva;
    }
    public String getNomeUnidadeOperativa() {
        return nomeUnidadeOperativa;
    }
    public void setNomeUnidadeOperativa(String nomeUnidadeOperativa) {
        this.nomeUnidadeOperativa = nomeUnidadeOperativa;
    }
    public String getSiglaUnidadeOperativa() {
        return siglaUnidadeOperativa;
    }
    public void setSiglaUnidadeOperativa(String siglaUnidadeOperativa) {
        this.siglaUnidadeOperativa = siglaUnidadeOperativa;
    }
}
```

Figura 36 – Classe POJO

```
package br.com.petrobras.beans;

public class ConsultaUnidadeOperativa implements java.io.Serializable {

  private java.lang.String nome;

  public java.lang.String getNome() {
    return this.nome;
  }

  public void setNome(java.lang.String nome) {
    this.nome = nome;
  }

  private java.lang.String sigla;

  public java.lang.String getSigla() {
    return this.sigla;
  }

  public void setSigla(java.lang.String sigla) {
    this.sigla = sigla;
  }

}
```

Figura 37 – Classe que encapsula os parâmetros da chamada do serviço

4.2.2 Teste de remoção de atributo do objeto retornado pelo serviço

Este teste consiste em remover o atributo codigoAGP da classe POJO retornada pelo serviço, publicar o serviço no servidor com a alteração e realizar a chamada do serviço no cliente, sem alterá-lo, utilizando o barramento como mediador. A Figura 38 apresenta a resposta recebida pelo cliente. Logo, para o atributo removido no serviço, foi atribuído o valor nulo, e não ocorreu erro na invocação do serviço.

```
*******************************************************
Codigo: 1
Nome: AS
Sigla: BRA
Indicador: PT
CódigoAGP: null

*******************************************************
```

Figura 38 – Resultado da invocação do serviço após remoção de atributo retornado pelo serviço

4.2.3 Teste de adição de atributo no objeto retornado pelo serviço

Este teste consiste em adicionar um atributo na classe POJO, publicar o serviço no servidor com a alteração e realizar a chamada do serviço no cliente, sem alterá-lo, utilizando o barramento como mediador. A Figura 39 apresenta a resposta recebida pelo cliente. Observe que todos os atributos do objeto do cliente são preenchidos, dado que nenhum dos mesmos foi removido. Além disso, o cliente não recebe mensagem de erro, mesmo invocando um serviço em uma versão mais nova do que a que o cliente está referenciando.

```
*******************************************************
Codigo: 1
Nome: AS
```

```
Sigla: BRA
Indicador: PT
CódigoAGP: 1
*****************************************************
```

Figura 39 - Resultado da invocação do serviço após adição de atributo retornado pelo serviço

4.2.4 Teste de remoção de atributo no POJO que en capsula os parâmetros do serviço

Este teste consiste em remover um atributo da classe POJO que encapsula os parâmetros do serviço, publicar o serviço no servidor com a alteração e realizar a chamada do serviço no cliente, sem alterá-lo, utilizando o barramento como mediador. O atributo excluído foi "Nome" sendo que não seria possível apagar o atributo "Codigo" porque ele é essencial para a execução do serviço, já que leva o dado que será utilizado na função que obtém e retorna a Unidade Operativa. A Figura 40 apresenta a resposta recebida pelo cliente. Neste caso, o provedor recebeu o valor do atributo "Nome", enviado pelo cliente, mas que foi apagado no servidor. O POJO de retorno do serviço não sofreu alterações, e não ocorreram erros na execução do serviço. Logo, o parâmetro extra na classe POJO que encapsula os parâmetros do serviço, não afeta a sua execução. Além disso, o barramento foi transparente nessa transação.

```
*****************************************************
Codigo: 1
Nome: AS
Sigla: BRA
Indicador: PT
CódigoAGP: 1
*****************************************************
```

Figura 40 – Resultado da invocação do serviço após remoção de atributo da classe que encapsula os parâmetros do serviço

4.2.5 Teste de inclusão de atributo no POJO que encapsula os parâmetros do serviço

Este teste consiste em adicionar um atributo na classe POJO que encapsula os parâmetros do serviço, publicar o serviço no servidor com a alteração e realizar a chamada do serviço no cliente, sem alterá-lo, utilizando o barramento como mediador. O atributo incluído foi chamado de "novoAtributo", do tipo String. A Figura 41 apresenta a resposta recebida pelo cliente. Neste caso, o serviço do provedor não recebeu o valor esperado do atributo que foi adicionado, já que o cliente não está configurado para enviar esse atributo. Esta atualização não provocou erro na execução do serviço, logo, o parâmetro ausente na classe POJO, que encapsula os parâmetros do serviço, não afeta sua execução. Além disso, o barramento foi transparente nessa transação.

```
*****************************************************
Codigo: 1
Nome: AS
Sigla: BRA
Indicador: PT
CódigoAGP: 1
```

```
**********************************************************
```

Figura 41 – Resultado da invocação do serviço após remoção de atributo da classe que encapsula os parâmetros do serviço

4.2.6 Conclusão dos testes

Os mesmos resultados obtidos nos testes realizados para avaliar as abordagens de conexão ponto-a-ponto [Souza *et al.*, 2009] entre cliente e servidor (consumidor e consumidor) foram obtidos nos testes realizados com a utilização do barramento. Portanto, o uso do barramento como simples mediador da chamada do serviço através de um serviço de proxy não interfere na arquitetura em relação à conexão ponto-a-ponto.

4.3 Definição de fluxos em serviços de Proxy

O OSB é um intermediário que processa mensagens de entrada para requisição de serviços, determina a lógica de roteamento, e transforma estas mensagens para compatibilidade com outros consumidores de serviços. No Oracle Service Bus, um fluxo de mensagem corresponde à implementação de um serviço de proxy. Você pode configurar a lógica de manipulação de mensagens usando definições de fluxo de mensagem de serviço de Proxy. A lógica inclui atividades como transformação, publicação e geração de relatório, as quais são implementadas como ações individuais dentro de estágios de pipeline.

A maior parte da lógica de processamento em um fluxo de mensagem é tratada em *pipelines*. Um *pipeline* é uma sequência nomeada de estágios representando um caminho de processamento em uma única direção sem ramificações. Um estágio é um passo de processamento configurado pelo usuário. Mensagens de entrada para pipelines são acompanhadas de um conjunto de variáveis de contexto da mensagem que contém conteúdos da mensagem. Elas podem ser acessadas ou modificadas por ações dentro do estágio do *pipeline*.

Na seção 3.3 foram apresentados os principais conceitos para definição de fluxo de mensagens em um barramento. Nesta seção, são apresentados testes práticos da definição de fluxo de mensagens no OSB.

Para a definição de fluxos entre consumidores e provedores, é necessário que o serviço provido esteja registrado no barramento e que exista um serviço de proxy para fazer o roteamento das mensagens no barramento. A seção 4.1 apresenta o passo-a-passo para publicação de serviço de negócio no barramento, o que envolve a importação do WSDL do serviço e o registro do serviço, além da criação de um serviço de proxy que roteará as mensagens para o serviço de negócio registrado. Estes passos foram seguidos para a execução dos testes práticos apresentados a seguir, os quais incluem:

- Roteamento de mensagens no OSB;
- Transformações de mensagens; e
- Um estudo prático do uso de transformações de mensagens e roteamento das mesmas para auxiliar no versionamento de serviços.

4.3.1 Roteamento de mensagens

Esta seção descreve um exemplo de roteamento de mensagens no OSB.

4.3.1.1 Cenário

Considere o cenário de uma companhia de financiamento apresentado na Figura 42 que realiza roteamento de pedidos de empréstimo para serviços de negócio apropriados baseado na taxa de juros requisitada. Se a solicitação é de uma taxa menor do que 5%, então a solicitação é roteada para um serviço específico. Caso contrário, a mensagem é roteada para um serviço normal.

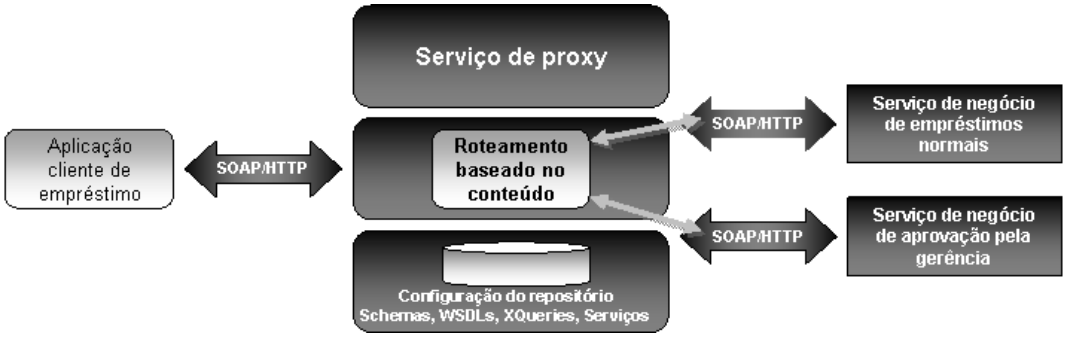

Figura 42 – Cenário de análise de pedidos de empréstimo

Em [Souza *et al.*, 2009] foi apresentado o passo-a-passo de publicação dos serviços de negócio e serviço de proxy para a definição do fluxo para roteamento de mensagens. Neste trabalho, foi utilizado o ALSB (Aqualogic Service Bus), cuja versão foi evoluída pela Oracle após a compra da BEA para o OSB (Oracle Service Bus). Não existem diferenças em relação ao passo-a-passo utilizando o ALSB e o OSB, e maiores detalhes para a realização deste passo-a-passo no OSB são encontrados em [OSB, 2009b]. Nesta seção foi dado foco na apresentação da forma de realizar roteamento de mensagens.

Para os exemplos apresentados a seguir, estamos considerando que os WSDL dos serviços *normalLoan* e *managerApproval* foram importados, os serviços de negócio *ManagerLoanReaview* e *NormalLoan* foram registrados e que o serviço de proxy *LoanGateway* foi criado. O passo-a-passo para realizar estas atividades é apresentado em [Souza *et al.*, 2009] e [OSB, 2009b].

4.3.1.2 Passos para criação do fluxo para roteamento de mensagem

A criação de um fluxo para um serviço de proxy é realizada a partir da edição do fluxo, como apresentado na Figura 43, sendo apresentado o fluxo inicial do serviço de proxy (Figura 44). A partir deste fluxo deve ser adicionada uma rota (*add route*) que inicia a criação de um estágio. Em seguida, devem ser definidas as ações para este estágio.

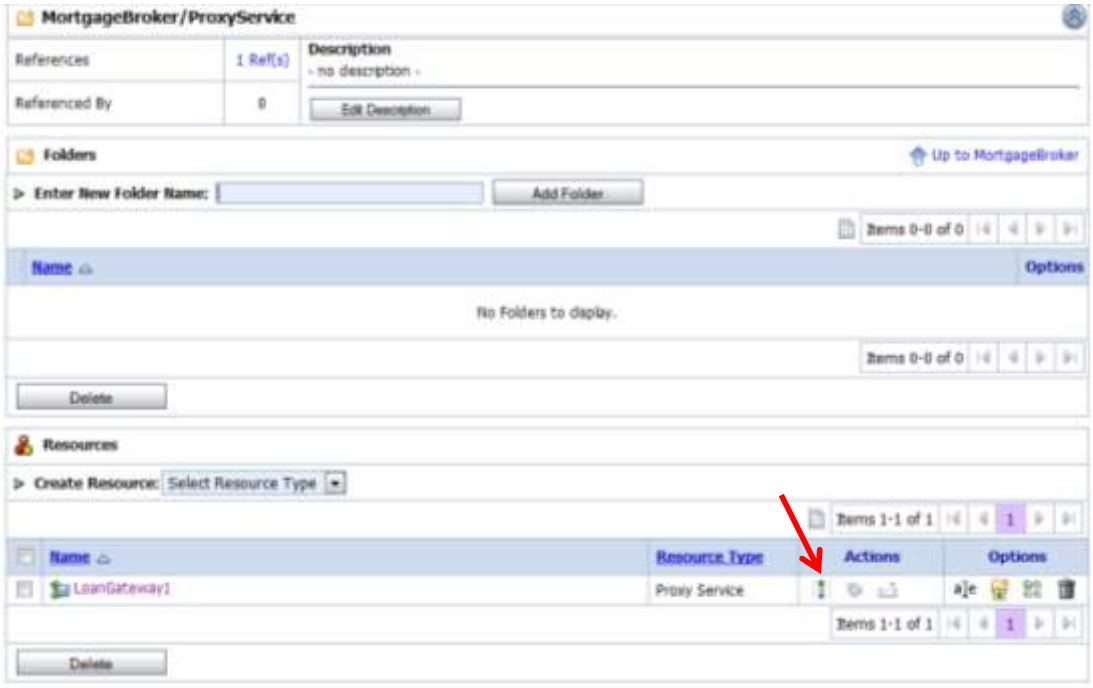

Figura 43 – Criação de fluxo para serviço de proxy

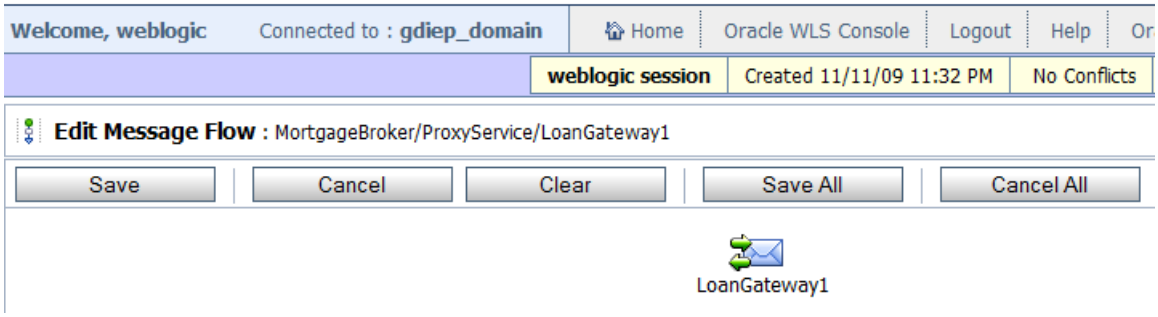

Figura 44 – Fluxo inicial para o serviço de proxy

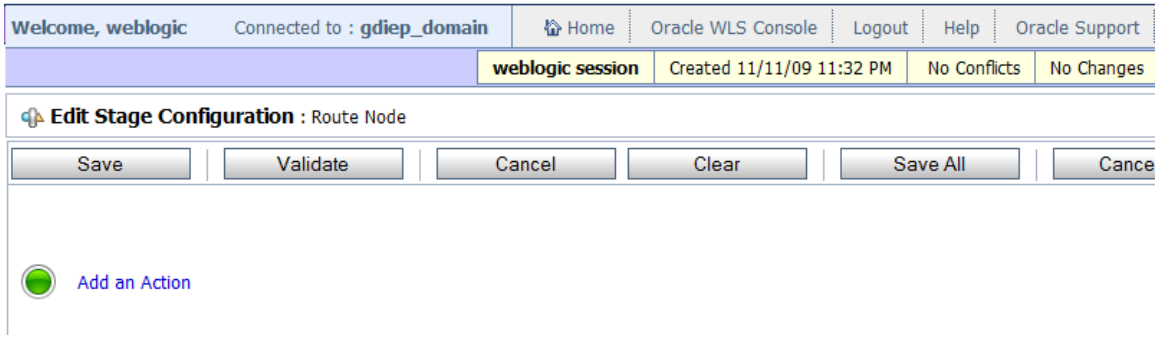

Figura 45 – Adição de ação a um estágio

Para criar ação de roteamento da mensagem, deve-se criar uma tabela de roteamento (“*Communication* > *Routing Table*”), como apresentado na Figura 46. Nesta tela, deve-se definir uma expressão (*Expression*), por exemplo, expressão XQuery para recuperar a taxa de juros no corpo da mensagem (variável de contexto *$body*), como apresentado na Figura 47.

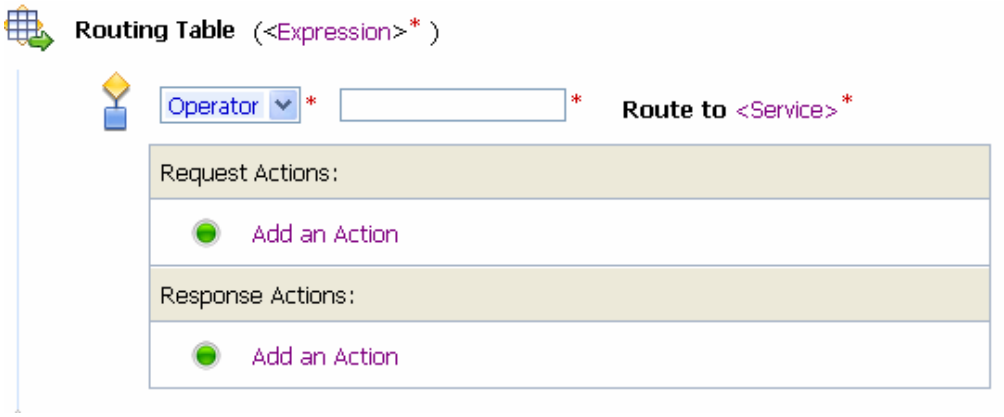

Figura 46 – Criação de tabela de roteamento

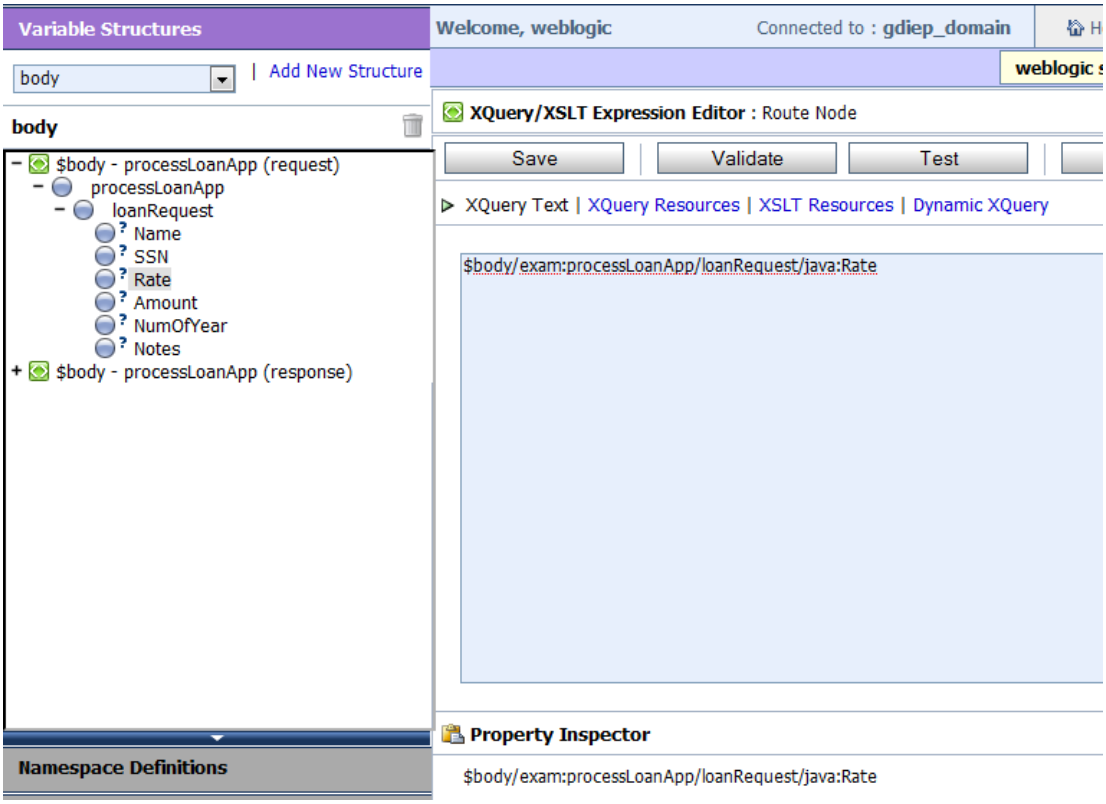

Figura 47 – Definição de expressão para recuperar a taxa de juros solicitada

Após a definição da expressão de roteamento, define-se a regra de roteamento. No exemplo apresentado na Figura 48, a mensagem é roteada para o serviço *ManagerLoanReview*, caso o valor da taxa seja menor do 5. Neste caso, o método *processLoanApp* é invocado. Para o caso da taxa ser maior ou igual a 5%, o serviço *NormalLoan* deve ser invocado (Figura 49). Isto é configurado criando um roteamento default (clicando no botão destacado na Figura 49) e configurando que o método *processLoanApp* do serviço *NormalLoan* deve ser invocado.

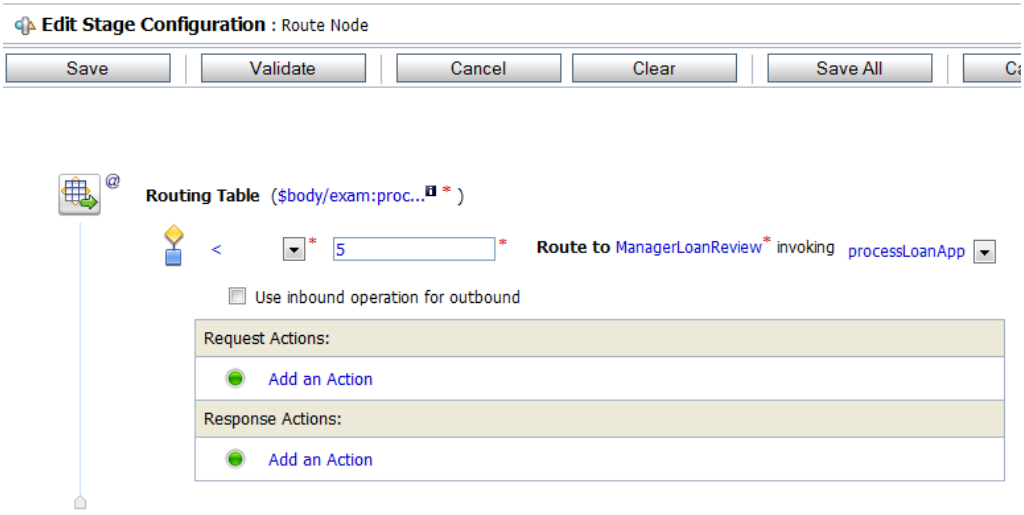

Figura 48 – Roteamento para o serviço que trata solicitação de taxa de juros menor do que 5%

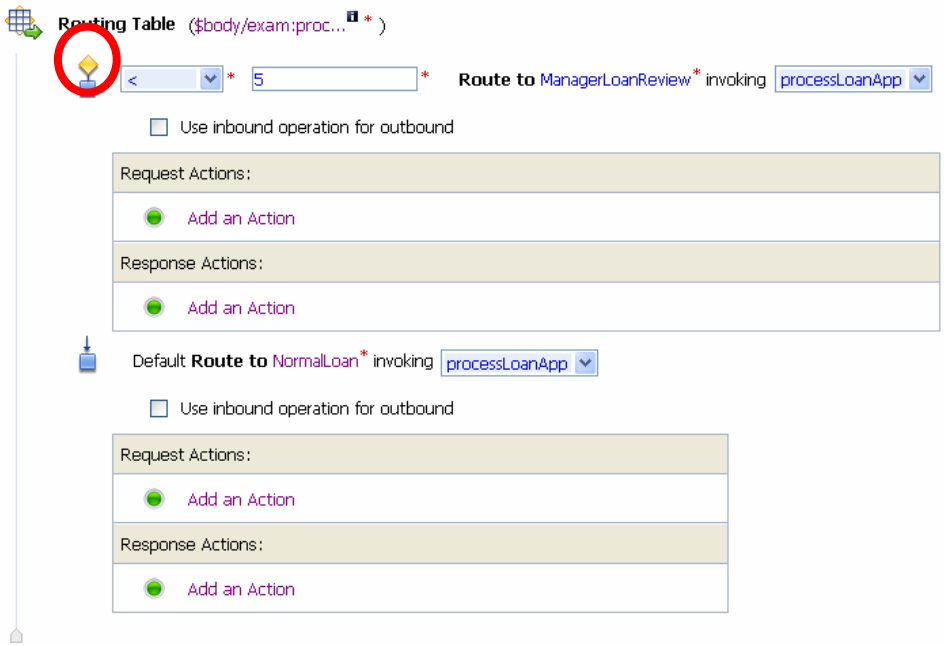

Figura 49 – Definição do roteamento default para invocar o método *processLoanApp* do serviço NormalLoan quando a taxa de juros solicitada for maior ou igual a 5%

4.3.1.3 Testes do serviço de Proxy

Para realizar o teste do serviço de proxy, basta clicar no botão de depuração, como apresentado na Figura 50. Será exibida a tela apresentada na Figura 51. Para realizar o teste, preencher, no *textbox* correspondente a *loanRequest*, o valor da taxa de juros. Se for preenchido um valor menor do que 5%, o serviço de proxy roteia a mensagem para o serviço *ManagerLoanReaview*. Caso contrário, a mensagem será roteada para o serviço *NormalLoan*. A Figura 52 apresenta o resultado da invocação do serviço solicitando taxa menor do que 5%, e a Figura 53 apresenta o *trace* de invocação do serviço. Já a Figura 54 apresenta o resultado da invocação do serviço solicitando taxa maior do que 5%.

Figura 50 – Iniciar depuração do serviço

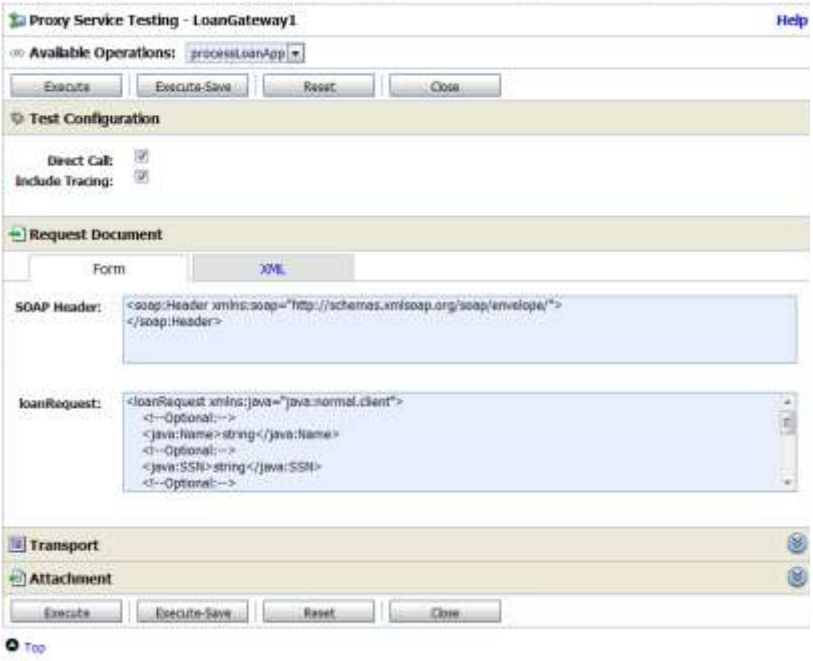

Figura 51 – Tela de depuração de serviço

Response Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header/>
    <soapenv:Body>
        <m:processLoanAppResponse xmlns:m="http://example.org">
            <return>
                <java:Name xmlns:java="java:normal.client">Smith</java:Name>
                <java:SSN xmlns:java="java:normal.client">1234567</java:SSN>
                <java:Rate xmlns:java="java:normal.client">4.1</java:Rate>
                <java:Amount xmlns:java="java:normal.client">9000000</java:Amount>
                <java:NumOfYear xmlns:java="java:normal.client">10</java:NumOfYear>
                <java:Notes xmlns:java="java:normal.client">
                    APPROVED BY THE &lt;b>&lt;i>MANAGER&lt;/i>&lt;/b> LOAN APPLICATION REVIEW SERVICE
                </java:Notes>
            </return>
        </m:processLoanAppResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Figura 52 – Resultado da invocação do serviço solicitando taxa igual a 4,1%

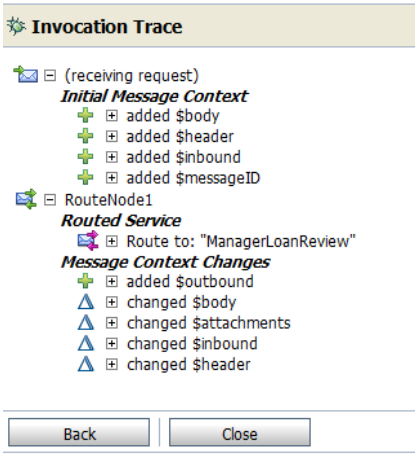

Figura 53 – *Trace* de invocação do serviço

Figura 54 - Resultado da invocação do serviço solicitando taxa igual a 5,3%

4.3.2 Transformação de mensagens

Transformação de dados corresponde a mapeamentos de dados de um formato para outro, por exemplo, com o objetivo de tornar a informação compatível para ambientes de sistemas heterogêneos. O OSB pode ser configurado para rotear e transformar mensagens quando necessário, baseado em configurações de serviço de proxy específicas.

4.3.2.1 Cenário

O exemplo utilizado nesta seção para apresentar a transformação de mensagens é semelhante ao cenário de empresa financeira apresentada na seção anterior. Neste caso, a empresa utiliza o OSB para rotear solicitações de empréstimos que podem ser pagos por uma empresa secundária caso o valor solicitado seja maior do que 25 milhões, ou seja, caso contrário a solicitação é repassada para um serviço de negócio da própria empresa.

A Figura 55 apresenta a arquitetura da aplicação para tratar empréstimos financeiros no OSB. Inicialmente, a empresa financeira principal recebe uma solicitação de empréstimo, pela invocação do método do serviço de proxy (*LoanGateway2*), o qual determina o serviço de negócio que tratará a solicitação. Se a quantia solicitada é maior do que 25 milhões, então a solicitação é roteada para o serviço de negócio *LoanSalePro-*

cessor. Se a quantia é menor ou igual a 25 milhões, a solicitação é roteada para o serviço de negócio *NormalLoan*.

Quando a quantia é maior do que 25 milhões, antes de realizar o roteamento para o serviço da empresa secundária, o *pipeline* de requisição realiza um *callout* para o serviço de negócio *CreditRating* e recebe a taxa de crédito da solicitação usando a variável *$creditRating*. Para preencher os requisitos da interface do serviço da empresa de empréstimo secundária, o corpo da mensagem (*$body*) é transformado pela adição dos detalhes da taxa de crédito. Em seguida, é feito o roteamento da mensagem transformada para o serviço de negócio da empresa secundária. Quando este serviço retorna a resposta para o serviço de proxy, a mensagem é transformada novamente para estar de acordo com o formato da mensagem de retorno do serviço de proxy.

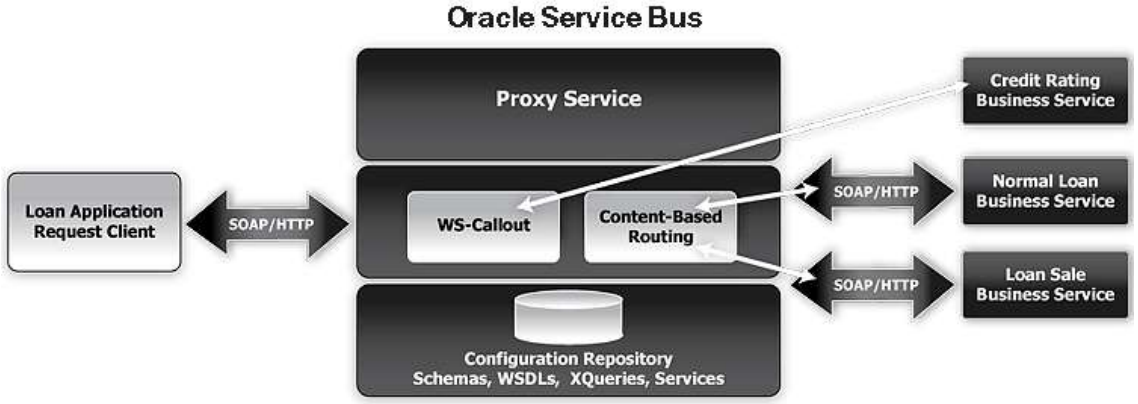

Figura 55 – Exposição da aplicação para tratar empréstimos financeiros no OSB

4.3.2.2 Configuração necessária

Para o exemplo apresentado a seguir, as seguintes configurações são necessárias:

- Os WSDL dos serviços *normalLoan*, *managerApproval*, *loanSale* e *creditRating* serem importados. O resultado da importação destes WSDLs é apresentado na Figura 56.
- O serviço de proxy *LoanGateway2* ser criado a partir do WSDL *normalLoan* e com URI do *endpoint* igual a */loan/gateway2*.
- Os seguintes serviços de negócio estarem registrados:
    - *CreditRatingService*: retorna a taxa de crédito quando a solicitação de empréstimo está de acordo com determinado critério.
        - Este serviço já é instalado no servidor de aplicação (WebLogic Server), logo é necessário apenas registrá-lo a partir do WSDL *creditRating*, escolhendo a porta *helloPort* e endpoint URI igual a http://<host:port>/ cre-jws_basic_ejb/CreditSimpleBean.
    - *NormalLoan*: serviço de negócio da empresa financeira secundária para tratar solicitações menores ou iguais a 25 milhões.
        - Este serviço foi registrado para a execução do exemplo da seção anterior.

- *LoanSaleProcessor*: serviço de negócio da empresa financeira secundária para tratar solicitações maiores do que 25 milhões.
  - Este serviço já é instalado no servidor de aplicação (WebLogic Server), logo é necessário apenas registrá-lo a partir do WSDL *loanSale*, escolhendo a porta *helloPort* e *endpoint* URI igual a http://<host:port>/ljws_basic_ejb/LargeSimpleBean.

O passo-a-passo para realizar importação de WSDL e registro de serviços é apresentado em [Souza *et al.*, 2009] e [OSB, 2009b].

Figura 56 – WSDL necessários para exemplo de transformação de mensagens

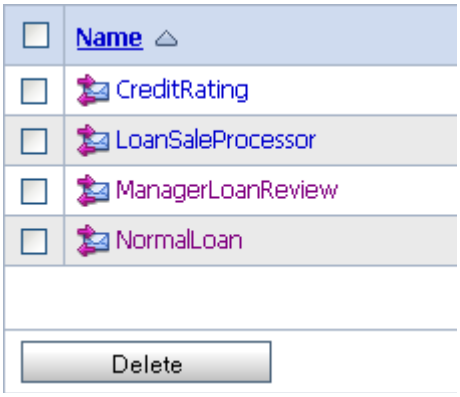

Figura 57 – Serviços de negócio registrados para o exemplo de transformação de mensagens

As mensagens de entrada e saída dos serviços apresentam diferenças as quais fazem necessária a transformação de mensagens para invocá-los:

- A operação invocada do serviço *NormalLoan* é a *processLoanApp* que recebe como entrada uma mensagem *processLoanApp* que tem uma parte (*loanRequest*) que é do tipo *LoanStruct* (Figura 58). Este tipo de dados está definido no *targetNamespace* denominado "java:normal.client".
- O serviço *CreditRatingService* recebe como entrada um objeto *LoanStruct* do namespace "java:credit.client" (semelhante ao da Figura 58 diferindo apenas o namespace) e retorna um objeto *LoanStruct* (semelhante ao da Figura 58 diferindo apenas no namespace).
- Já o serviço *LoanSaleProcessor* recebe como entrada um objeto do tipo *LoanStruct*, mas com um atributo a mais (CreditRating) (Figura 59). O *targetNamespace* da estrutura, neste caso, é "java:large.client".

```
<xs:complexType name="LoanStruct">
```

```
  <xs:sequence>
  <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string"/>
  <xs:element minOccurs="0" name="SSN" nillable="true" type="xs:string"/>
  <xs:element minOccurs="0" name="Rate" nillable="false" type="xs:double"/>
  <xs:element minOccurs="0" name="Amount" nillable="false" type="xs:long"/>
  <xs:element      minOccurs="0"      name="NumOfYear"      nillable="false"
type="xs:int"/>
  <xs:element minOccurs="0" name="Notes" nillable="true" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Figura 58 – Tipo complexo *LoanStruct* do namespace "java:normal.client"

```
<xs:complexType name="LoanStruct">
<xs:sequence>
  <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string"/>
  <xs:element minOccurs="0" name="SSN" nillable="true" type="xs:string"/>
  <xs:element minOccurs="0" name="Rate" nillable="false" type="xs:double"/>
  <xs:element minOccurs="0" name="Amount" nillable="false" type="xs:long"/>
  <xs:element minOccurs="0" name="NumOfYear" nillable="false" type="xs:int"/>
  <xs:element minOccurs="0" name="Notes" nillable="true" type="xs:string"/>
  <xs:element      minOccurs="0"      name="CreditRating"      nillable="true"
type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Figura 59 – Tipo complexo *LoanStruct* do namespace "java:large.client"

Como o serviço de proxy foi criado a partir do WSDL de *NormalLoan*, então não há necessidade de realizar transformação de mensagem para rotear a mensagem para *NormalLoan*.

4.3.2.3 Configuração da transformação da mensagem e roteamento para serviço

Caso o valor do empréstimo seja maior do que 25 milhões, o fluxo de *LoanGateway2* irá invocar o serviço que calcula a taxa de crédito (realizando *callout*) e, em seguida, realizará o roteamento da mensagem para o serviço de empréstimo da empresa secundária. Logo, seguindo os mesmos passos da Figura 43, Figura 44, Figura 45 e Figura 46 criaremos a tabela de roteamento para o serviço *LoanGateway2*. Após escolher a expressão XQuery, no *Namespace Definitions* ir para *Variable Structures* e escolher o elemento *body*. No elemento *$body – processLoanApp (request)* ir para o elemento *loanRequest* e escolher a variável *Amount*. A expressão XQuery será montada: $body/exam:processLoanApp/loanRequest/java:Amount. Esta expressão será utilizada pelo serviço de proxy para recuperar a quantia solicitada para empréstimo.

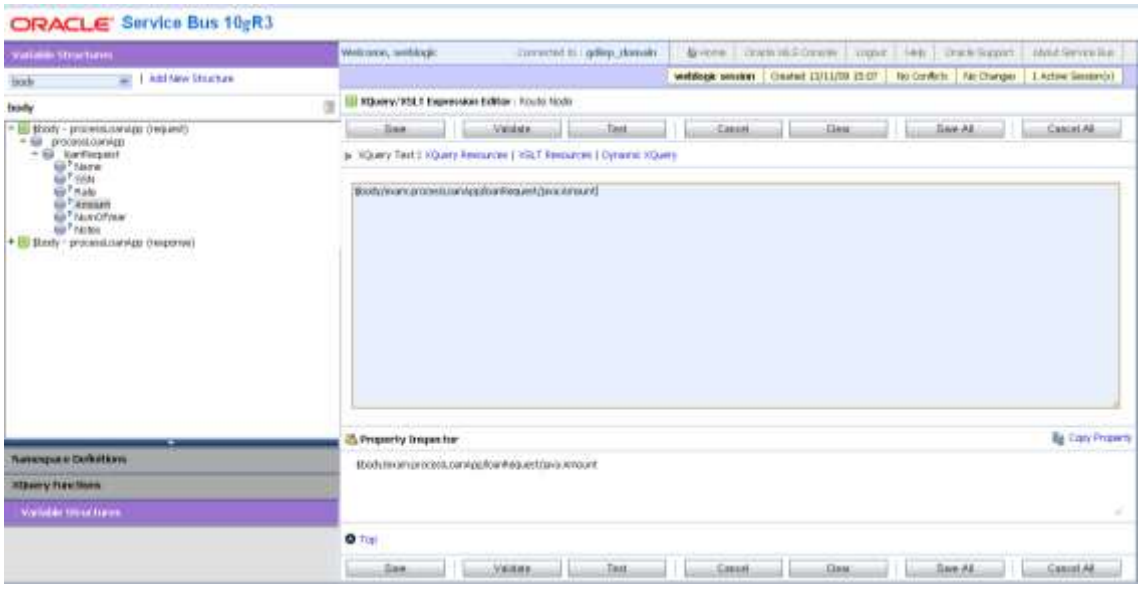

Figura 60 – Definição da variável *Amount* para a expressão XQuery

Em seguida, defina o operador ">", o valor de 25000000, o serviço LoanSaleProcessos e o método *processLoanApp*. Dessa forma, o roteamento para o caso do valor da solicitação ser maior do que 25 milhões foi definido. A Figura 61 apresenta o resultado final desta configuração. Crie um roteamento default para *NormalLoan* da mesma forma que apresentadado na Figura 49.

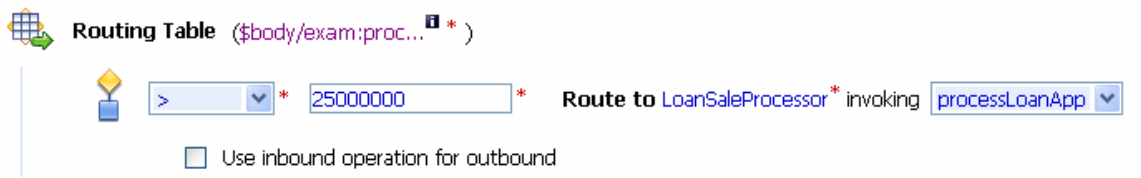

Figura 61 – Definição do roteamento para o serviço que trata solicitações de empréstimo maiores do que 25 milhões

Para obter a taxa de juros, o serviço de negócio *CreditRating* será invocado através da realização de um service *callout*. Para criar a ação de *callout*, é necessário criar o parâmetro de entrada para o serviço realizando as seguintes tarefas:

- Apagar o atributo "xsi:type" da mensagem de entrada do serviço. Quando se tem uma hierarquia de elementos, o atributo "xsi:type" é utilizado para identificar exatamento o tipo derivado de um elemento[10]. A Figura 62 apresenta o envelop SOA de invocação do serviço de proxy e o atributo "xsi:type" destacado. Este atributo deve ser removido para a invocação do serviço *CreditRating*.
- Atribuir em uma variável o parâmetro de entrada para a ação de *callout* do serviço
- Renomear o namespace para o parâmetro de entrada para o *callout* do serviço

Para apagar o atributo "xsi:type" da mensagem é necessário adicionar uma ação de remoção desta *tag* na tabela de roteamento (Figura 63). No campo XPath preencher com a expressão *./exam:processLoanApp/loanRequest/@xsi:type* e preencher *body* no cam-

[10] Explicação sobre o uso do atributo "xsi:type" http://www.w3.org/TR/xmlschema-0/#UseDerivInInstDocs

po referente à variável. Validar e salvar, para atualizar as configurações do serviço (Figura 64).

```
<soapenv:Envelope
       xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header       xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 </soap:Header>
 <soapenv:Body>
   <exam:processLoanApp    xmlns:exam="http://example.org">
      <loanRequest         xsi:type="java:LoanStruct"
xmlns:java="java:normal.client"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
       <!--Optional:-->
       <java:Name>string</java:Name>
       <!--Optional:-->
       <java:SSN>string</java:SSN>
       <!--Optional:-->
       <java:Rate>1.051732E7</java:Rate>
       <!--Optional:-->
       <java:Amount>90000000</java:Amount>
       <!--Optional:-->
       <java:NumOfYear>3</java:NumOfYear>
       <!--Optional:-->
       <java:Notes>string</java:Notes>
       </loanRequest>
   </exam:processLoanApp>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 62 – Envelope SOAP com a mensagem para invocação do serviço de proxy

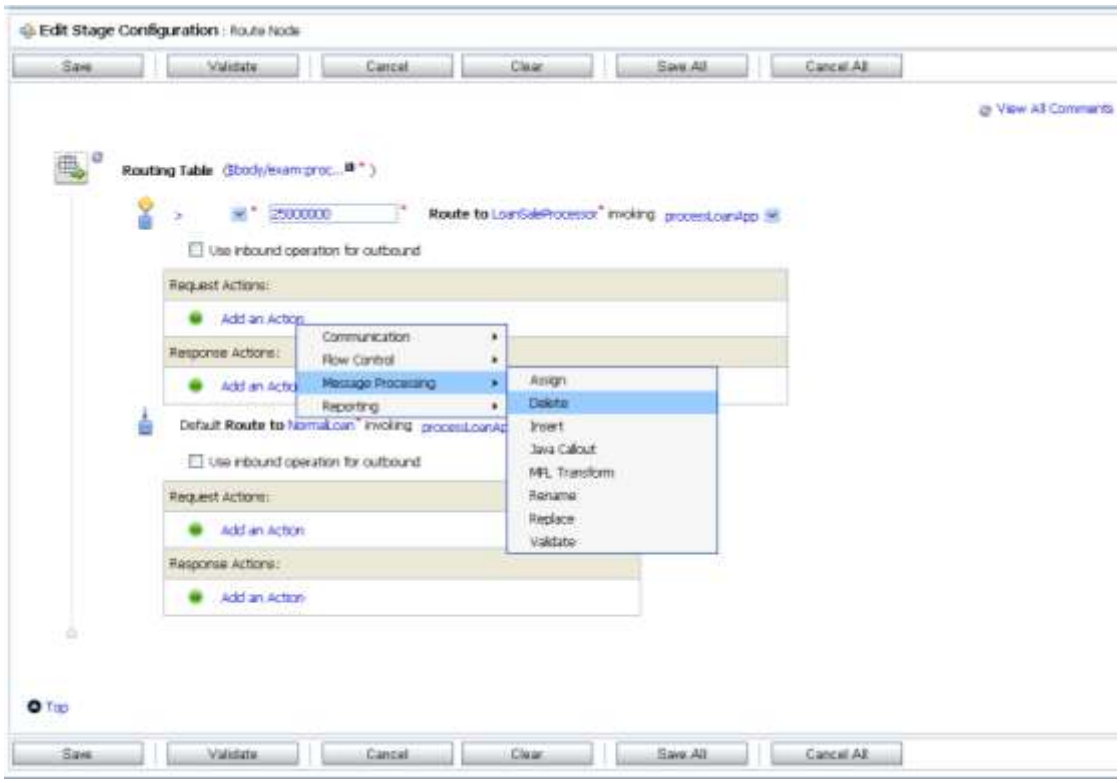

Figura 63 – Adição de ação para remoção

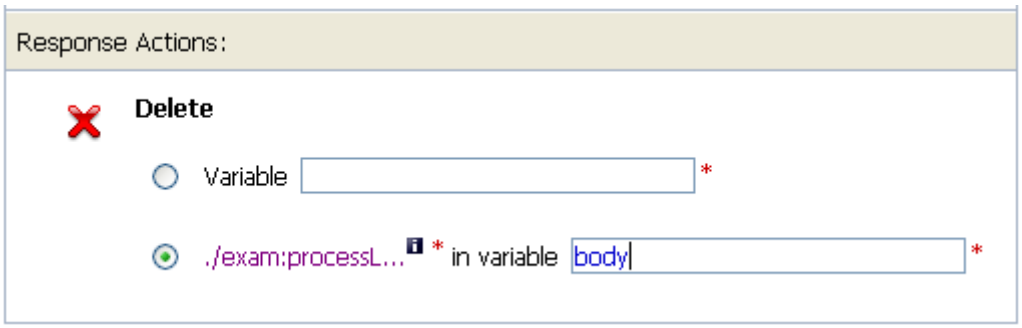

Figura 64 – Definição da expressão e variável a ser realizada a remoção

O próximo passo consiste em atribuir um parâmetro de entrada para o *callout* do serviço. Para isto, a partir do ícone *Delete* adicionar uma ação do tipo *assign* (*Add an Action* > *Message Processing* > *Assign*) e definir a expressão XQuery *$body/exam:processLoanApp/loanRequest* para a estrutura de variável body. O resultado da expressão deve ser atribuído à variável *loanRequestVariable*, resultando na tela apresentada na Figura 65.

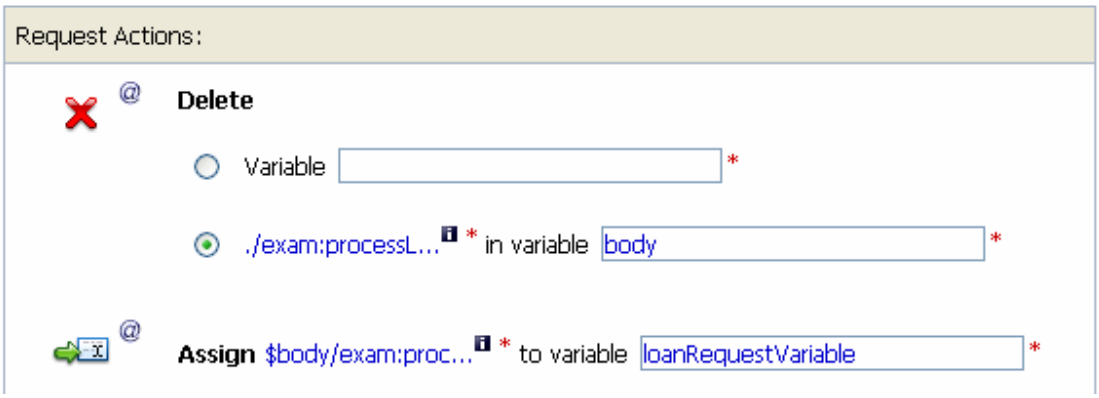

Figura 65 – Definição da ação *assign*

O próximo passo consiste em renomear o namespace "*java:normal.client*" do parâmetro de entrada para compatibilizar com o serviço. A partir do íncone da atividade *assign*, criar uma atividade *rename* (*Add an Action* > *Message Processing* > *Rename*). Na expressão XPath, preencher *.//java:** e definir a variável *loanRequestVariable*. Dessa forma foi definida uma condição para pesquisar por todos os *namespaces* com prefixo Java na variável de contexto *loanRequestVariable*. O próximo passo, define o *namespace* a ser alterado identificado pela expressão XPath. Neste caso escolher namespace e preencher no campo *java:credit.client*. O resultado é apresentado na Figura 66.

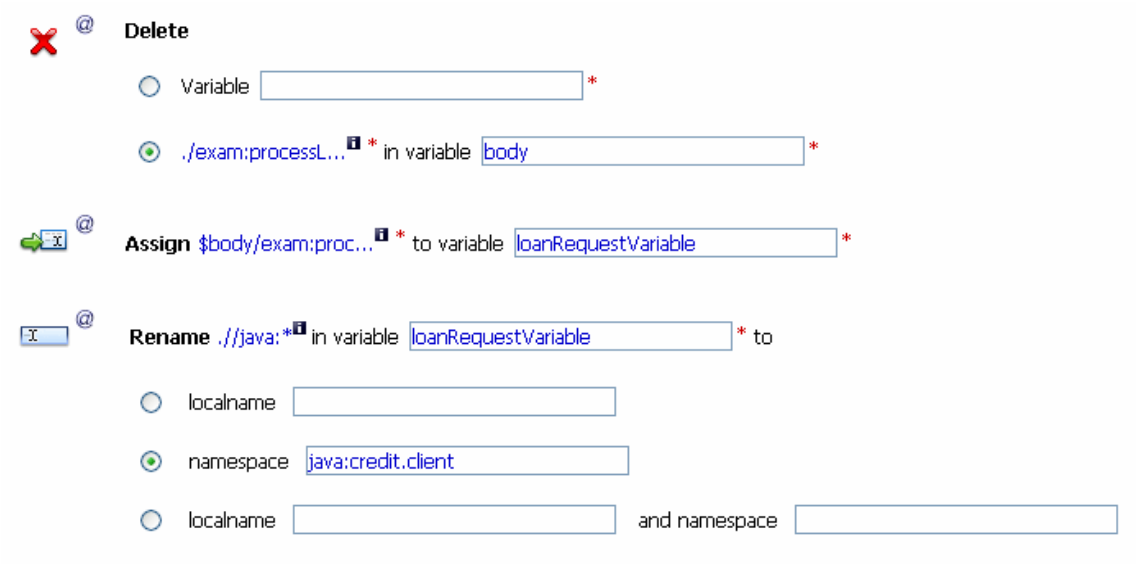

Figura 66 – Criação da atividade de *rename*

O próximo passo consiste em criar a ação de *Callout* para o serviço (*Action* > *Communication* > *Service Callout*). Após escolher esta ação a partir da ação de *rename*, escolher o serviço e o método do serviço. Será exibida a tela apresentada na Figura 67 com campos para configurar a mensagem de requisição e resposta do serviço. Preencher o campo referente à mensagem de requisição com *loanRequestVariable* e o campo de resposta (*return*) com o valor *creditRating*, como apresentado na Figura 68. Dessa forma, após a execução do serviço, o resultado será armazenado na variável de contexto *$creditRating*.

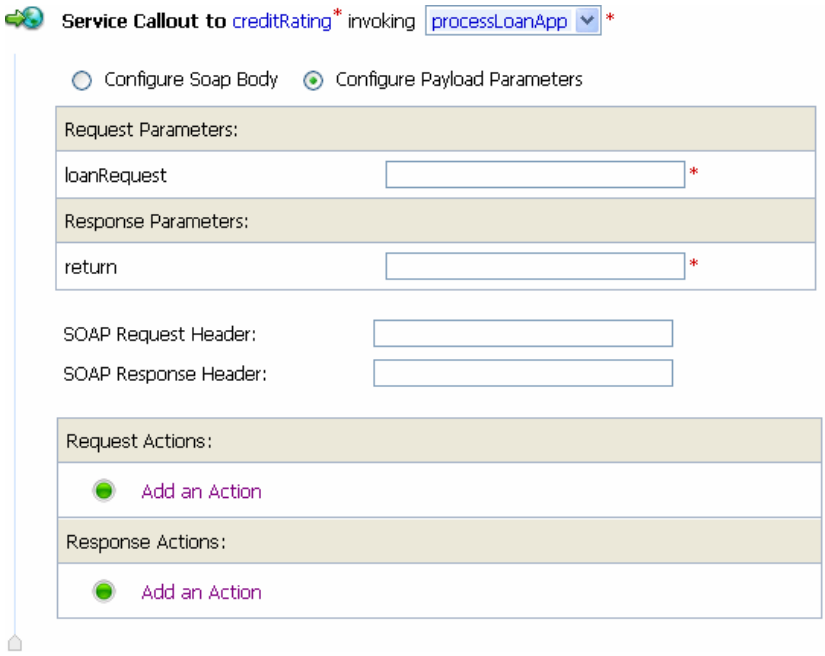

Figura 67 – Campos para configuração da requisição e resposta do serviço de *callout*

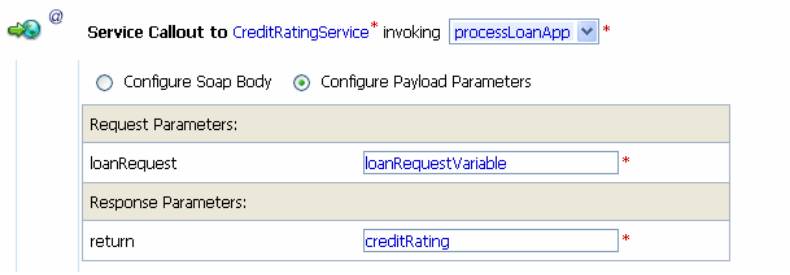

Figura 68 – Preenchimento dos campos de requisição e resposta do serviço de *callout*

Em seguida, deve-se transformar a mensagem resultante da invocação do serviço *CreditRating* para invocar o serviço *LoanSaleProcessor*. Neste caso, é necessário:

- Renomear o namespace da mensagem;
- Inserir um novo elemento na mensagem passada para o serviço *LoanSaleProcessor*.

Para renomear o namespace "*java:credit.client*" para "*java:large.client*". A partir do ícone da atividade *Service Callout*, criar uma atividade *rename* (*Add an Action* > *Message Processing* > *Rename*). Na expressão XPath, preencher *.//java:** e definir a variável *body*, a qual contém o corpo da mensagem. Dessa forma foi definida uma condição para pesquisar por todos os *namespaces* com prefixo Java na variável de contexto *body*. O próximo passo, define o *namespace* a ser alterado identificado pela expressão XPath. Neste caso escolher namespace e preencher no campo *java:large.client*. O resultado é apresentado na Figura 69.

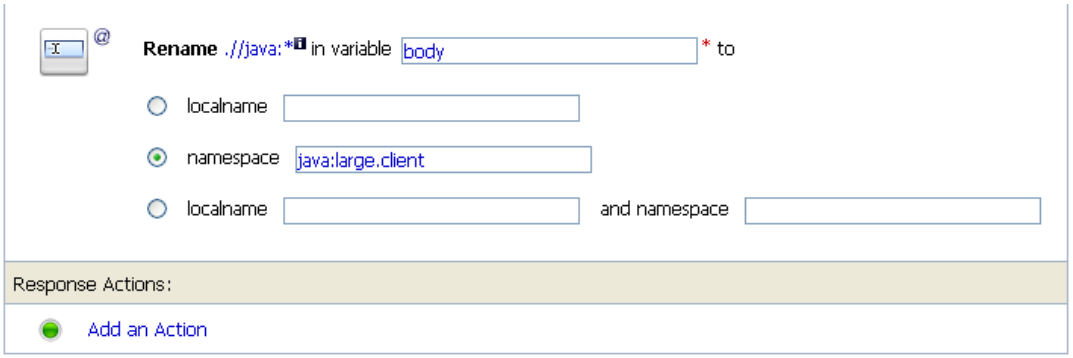

Figura 69 – Renomeação do namespace java:credit.client para java:large.client

O próximo passo é incluir o novo elemento na mensagem. Isto é feito através da ação *Insert*. Crie uma ação *Insert.* Clicando em *expression*, segue-se para o editor de expressão XQuery. Nesta tela, deve adicionar um novo namespace: na paleta *Namespace Definitions*, adicione o namespace (*add namespace*) com prefixo *lg* e URI "*java:large.client*". Na textbox de especificação de expressão XQuery, preencher <*lg:CreditRating*>{*data($creditRating)*}</*lg:CreditRating*>. Os colchetes {} indicam ao motor XQuery que o conteúdo entre {} deve ser considerado como não sendo XML e deve ser interpretado. Dessa forma, será retornado o valor da taxa de crédito atribuída pelo serviço *CreditRating*, montando assim o valor do atributo *CreditRating* no namespace *lg* que se refere à URI "*java:large.client*". Salve o resultado e, na tela de configuração da ação, escolha a opção *after* para indicar a partir de qual elemento será inserido o novo elemento. Clique no link <*XPath*> e, navegando na paleta *Variable Structures* vá até o atributo Notes de *loanRequest*. A expressão construída será *./exam:processLoanApp/loanRequest/java:Notes*. Substitua o namespace *java* por *lg*, resul-

tando em *./exam:processLoanApp/loanRequest/lg:Notes*. No textbox ao lado, preencha *body*. A configuração resultante é apresentada na Figura 70. Clique em salvar para concluir a configuração da transformação para a mensagem de entrada do serviço.

Figura 70 – Resultado da inserção de elemento na mensagem

Em seguida, é necessário configurar a mensagem de retorno do serviço (*Response actions*) a fim de que a mensagem retornada pelo serviço de proxy para o cliente esteja de acordo com o contrato do serviço (WSDL). Logo, é necessário remover o elemento <*CreditRating*> e reverter o namespace para o namespace original "*java:normal.client*".

Logo, na tela de edição da tabela de roteamento adiciona uma ação de delete e, através do link <Xpath>, no editor de expressão XQuery preencha "*./exam:processLoanAppResponse/return/lg:CreditRating*" para indicar que deve ser removido o elemento *CreditRating* da parte com o nome *return* da mensagem com o nome *processLoanAppResponse*. No textbox da variável, preencha *body*, para indicar que o elemento deve ser removido do corpo da mensagem (Figura 71).

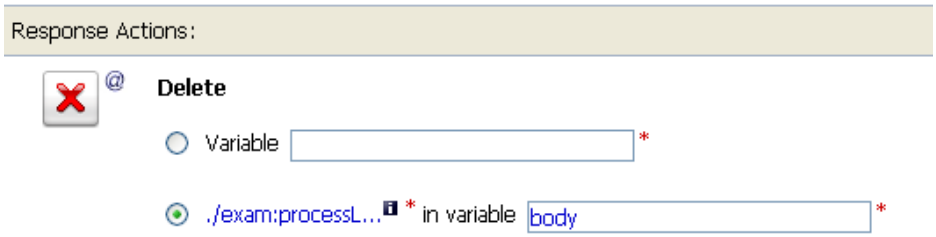

Figura 71 – Remoção do atributo *CreditRating* da mensagem de resposta

Em seguida adicione uma ação para renomear o namespace da mensagem de resposta de *"java:large.client"* para "*java:normal.client*". Na expressão XPath preencha "lg:*", No textbox para a variável preencha *body* e no namespace preencha "*java:normal.client*". Com isso o serviço está configurado (Figura 72).

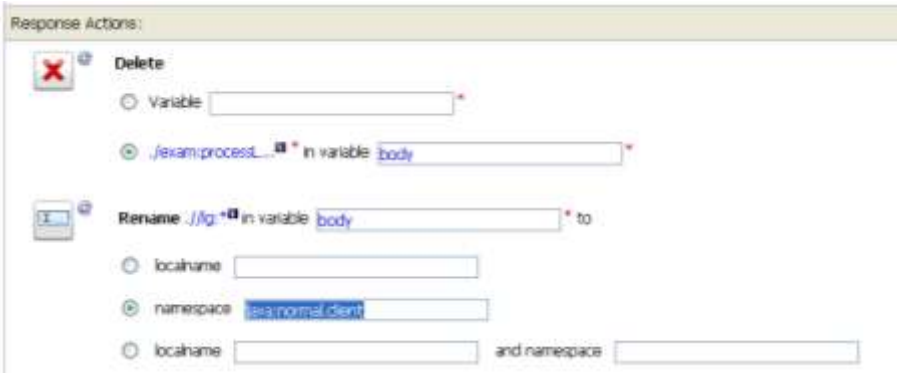

Figura 72 – Renomeação do namespace

4.3.2.4 Testes do serviço de proxy

Os testes do serviço de proxy podem ser realizados da mesma forma que foi apresentado na seção 4.3.1.3 . Após abrir o console de teste, preencher como parâmetro de en-

trada para o serviço, o valor de *amount* maior do que 25 milhões, como apresentado na Figura 73.

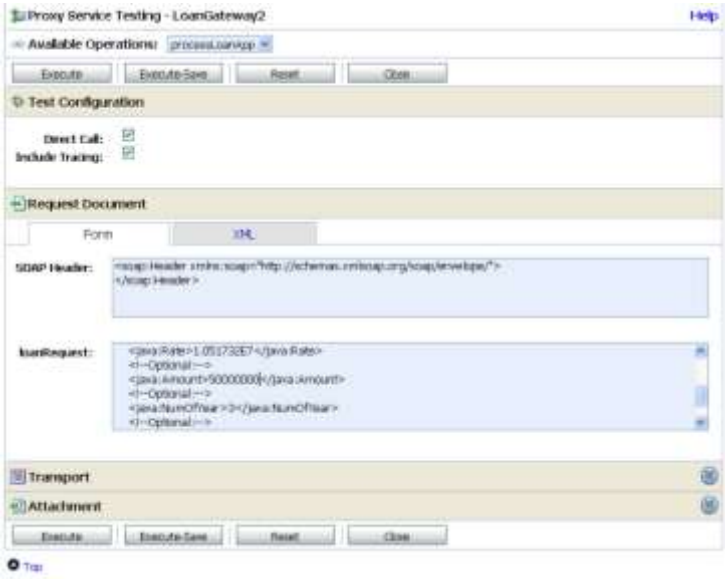

Figura 73 – Invocação de serviço de proxy para realizar roteamento para empresa de financeira secundária

A Figura 74 apresenta o resultado da invocação do serviço de proxy com roteamento para a empresa financeira secundária. Observe que a mensagem de resposta foi "*CREDIT RATING: AA: LOAN PURCHASED BY THE <i>LARGE</i> LOANS SERVICE*" e no trace de invocação de serviço que, na execução de *RouteNode1*, o serviço *CreditRatingService* foi invocado e, em seguida, o serviço *LoanSaleProcessor*. A Figura 75 apresenta a invocação do serviço de proxy realizando roteamento para *NormalLoan*.

Figura 74 – Resultado de invocação do serviço de proxy com roteamento para empresa financeira secundária

Figura 75 - Resultado de invocação do serviço de proxy sem roteamento para empresa financeira secundária

4.3.2.5 Conclusão

Neste teste, foi apresentada a transformação de mensagem para roteá-la para um único serviço. As ações foram definidas para a mensagem de requisição e de resposta da tabela de roteamento. No entanto, se a mensagem transformada fosse enviada para dois ou mais serviços, utilizando esta abordagem seria necessário replicar as ações para cada roteamento. Neste caso, a melhor alternativa é usar um *pipeline pair* para realizar as transformações e, após ter a mensagem transformada, esta ser roteada para os serviços.

4.4 Validação de tipos de dados utilizando serviço de proxy

O objetivo desta seção é demonstrar o uso da funcionalidade correspondente à ação "validate", oferecida pelo OSB para configuração de estágios em fluxos de serviços de proxy. Essa funcionalidade consiste em configurar um estágio que valida alguma estrutura contida na mensagem (por exemplo, um objeto) com uma definição preestabelecida desta estrutura, fornecida através de um arquivo WSDL ou XSD. Caso o conteúdo da mensagem avaliado não esteja de acordo com a definição esperada, o barramento levantará uma exceção e retornará a mensagem de erro para o cliente. Além disso, outras ações podem ser configuradas para serem executadas ao identificar este tipo de problema. Esta seção também descreve como emitir uma mensagem personalizada ao cliente do serviço quando é levantado esse tipo de exceção no barramento.

4.4.1 Validação de estruturas em mensagens

Esta seção apresenta a configuração para validação de estruturas em mensagens.

4.4.1.1 Configuração da validação

Após criar o serviço de Proxy, é possível definir o fluxo de atividades de manipulação das mensagens de requisição e resposta do serviço. Para configurar a validação é necessário ter adicionado o *pipeline pair*, um *stage* e o próprio serviço, dentro do mapa de fluxo de mensagens (*map of message flow*) conforme mostra a Figura 76.

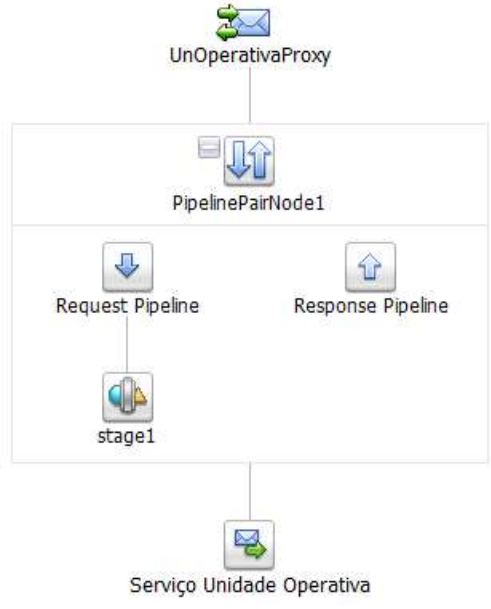

Figura 76 – Mapa de fluxo de mensagens

Ao editar o estágio inserido para realizar a atividade de validação, adicione uma ação de validação indo em "Add an Action/Message Processing/Validate" conforme ilustrado na Figura 77.

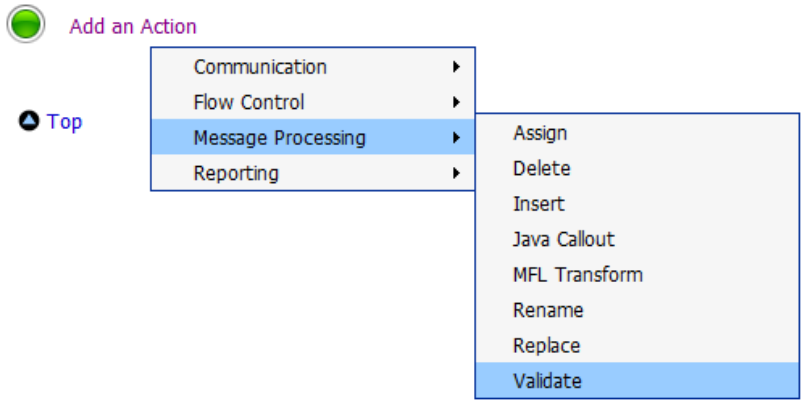

Figura 77 – Inserindo a ação "Validate"

Edite a expressão de validação entrando no link <XPath> (Figura 78).

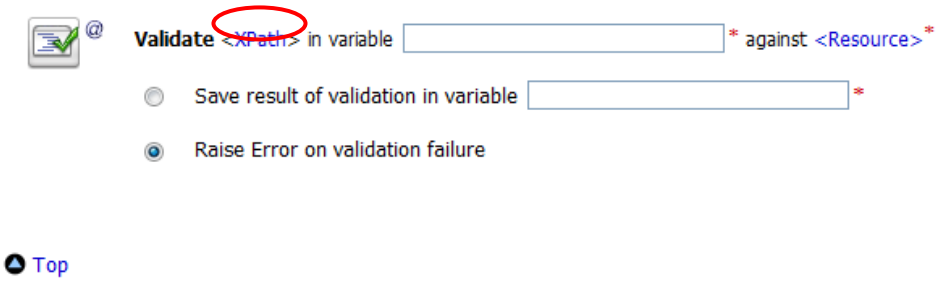

Figura 78 – Editando a expressão Xpath de validação

Na próxima tela deverá ser selecionada a estrutura a qual deseja ser validada. Clique no link "Variable Structures" (Figura 79).

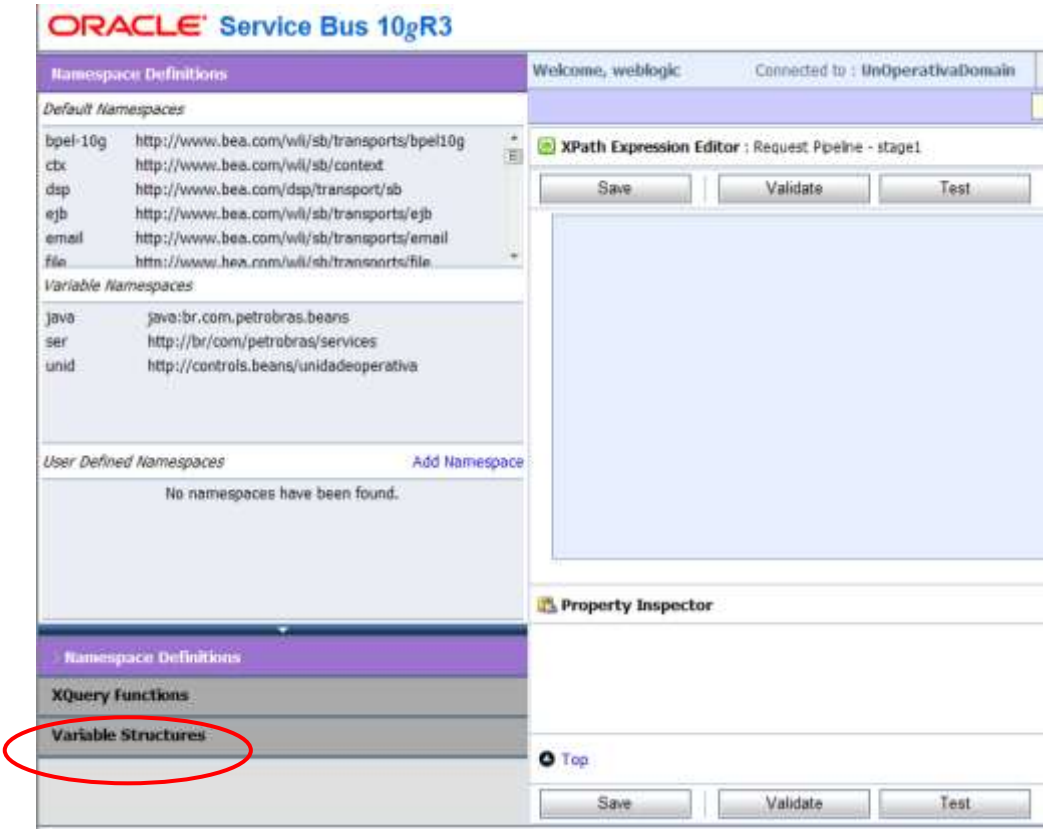

Figura 79 – Link "Variable Structures

Selecione a variável de estrutura "Body", na combobox (Figura 80).

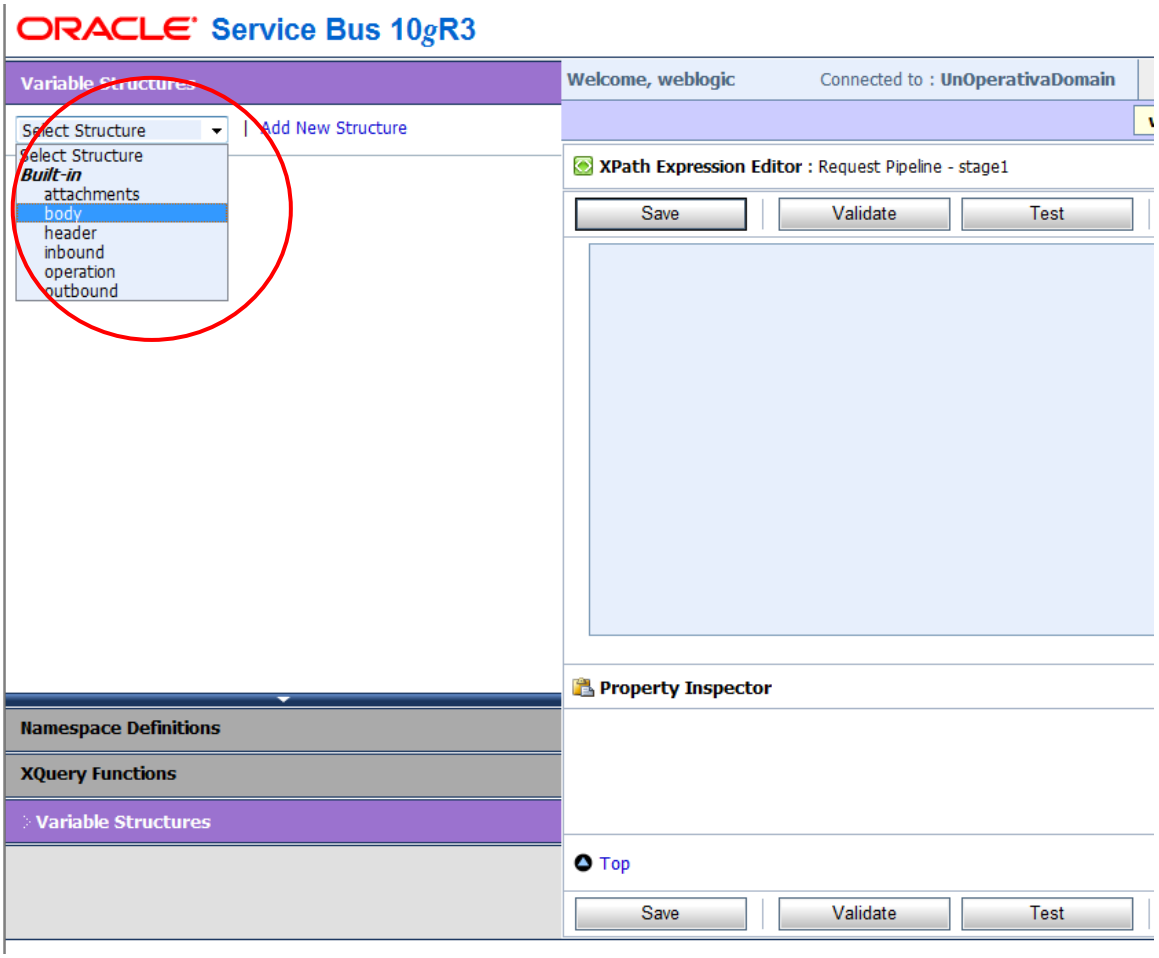

Figura 80 – Selecionando a variável de estrutura "body"

Então, selecione, dentro das operações disponíveis, a estrutura a qual será validada. Ao selecionar a estrutura, no quadro "Property Inspector" aparecerá o comando XPath correspondente a estrutura (Figura 81).

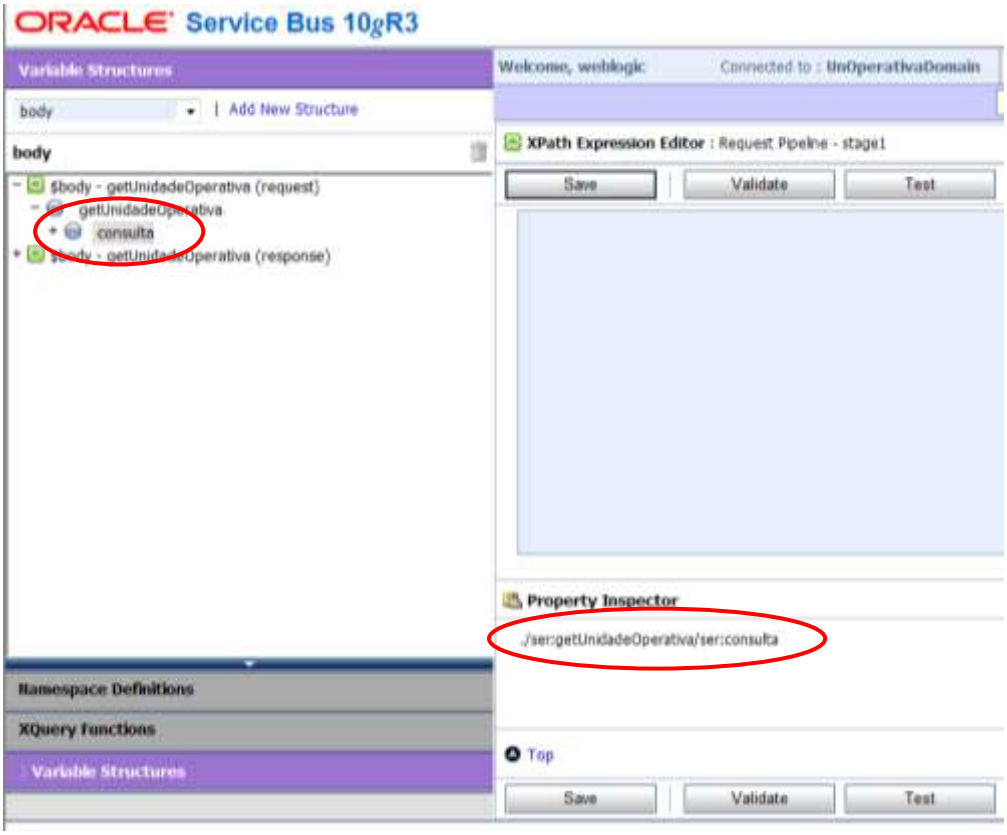

Figura 81 – Selecionando o comando Xpath

Esse comando deverá ser copiado para do XPath Expression Editor. Para fazer isso, caso esteja utilizando o Browser do Internet Explorer, basta arrastar o objeto selecionado para o quadro. Se estiver usando outro Browser, copie e cole o comando. Logo após, clique em "Validade" e depois em "Save" (Figura 82).

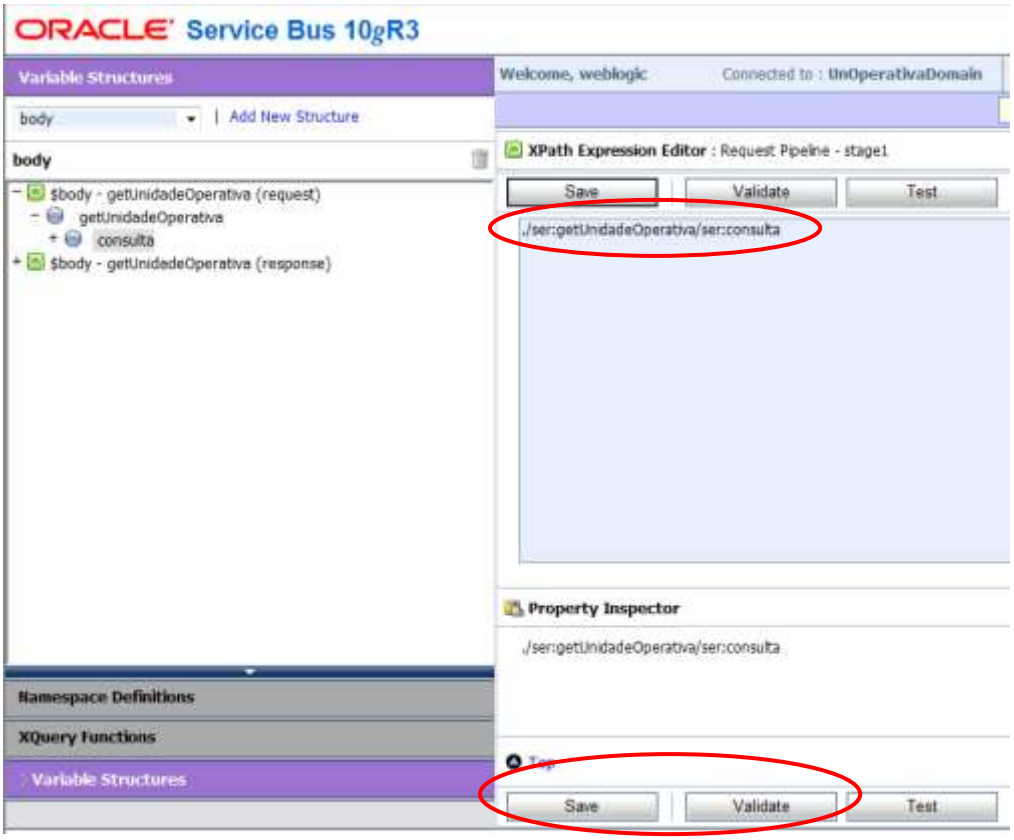

Figura 82 – Validando e salvando o comando Xpath

Ao retornar para a janela de configuração do estágio, insira "body" como a variável da expressão e marque "Raise Error on validation failure", caso não esteja marcado (Figura 83).

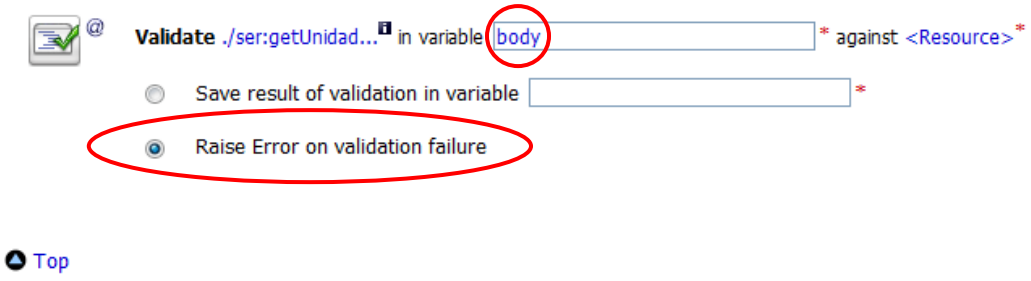

Figura 83 – Configuração da ação de validação

Agora deve-se inserir a configuração da estrutura correta, a qual será comparada com o conteúdo da mensagem. Ela pode ser obtida através de um arquivo WSDL ou XSD. Para isso, clique no link <*Resource*> e escolha o tipo do arquivo (neste exemplo, será um WSDL) conforme ilustra a Figura 84.

Figura 84 – Escolha do tipo de arquivo para seleção da estrutura a ser comparada na

validação

O arquivo, para ser selecionado, já deve ter sido adicionado ao projeto. Caso ainda não tenha adicionado o arquivo WSDL ou XSD como recurso do projeto, faça-o antes de continuar. A próxima janela mostra uma lista de todos os arquivos do tipo que foi selecionado e que estão disponíveis no barramento. Selecione o arquivo correspondente que possui a estrutura a ser configurada (Figura 85).

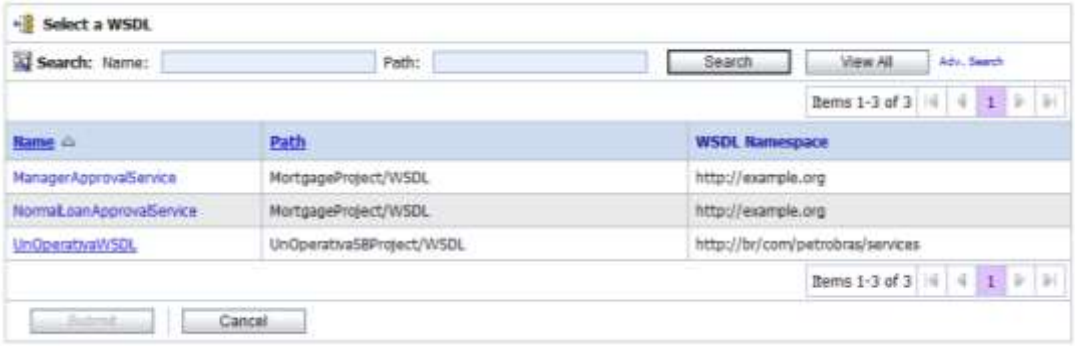

Figura 85 – Arquivos do tipo WSDL disponíveis no barramento para seleção de estruturas

Após selecionar o respectivo arquivo, o OSB apresentará as estruturas disponíveis. Selecione a estrutura desejada e clique em *Submit* (Figura 86).

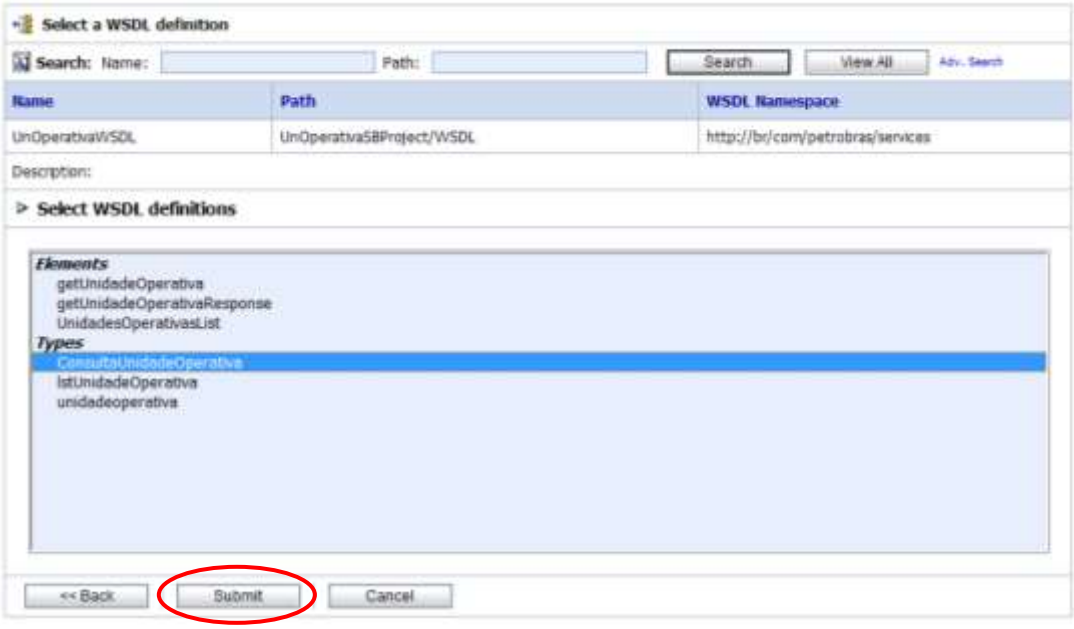

Figura 86 – Estruturas disponíveis dentro do arquivo WSDL selecionado

Ao retornar para a janela anterior, clique em "Validade" e depois em "Save" (Figura 87).

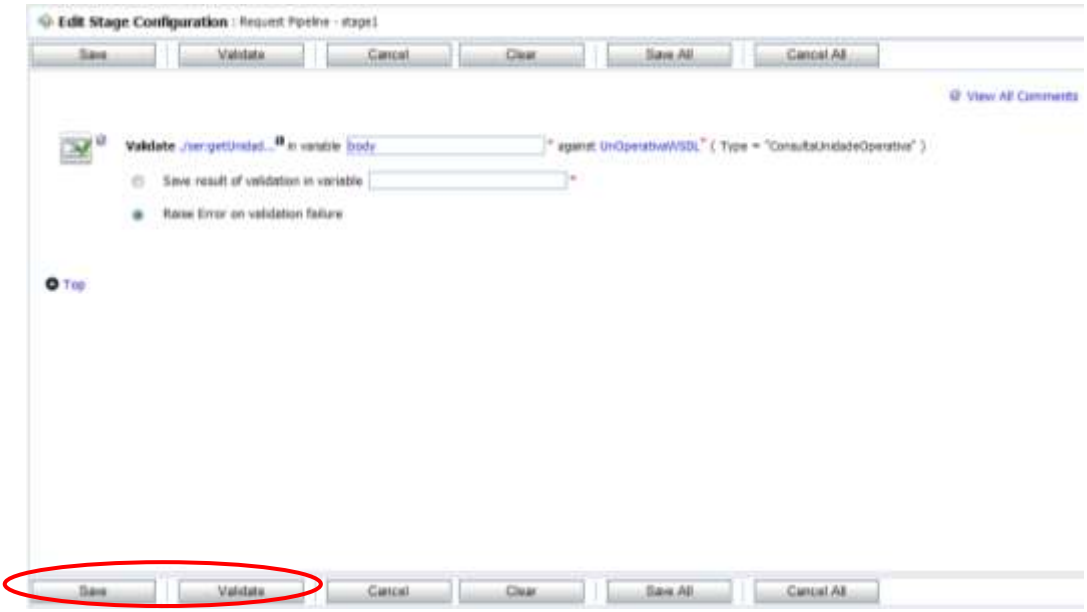

Figura 87 – Validando e salvando as configurações de validação de estrutura no OSB

De volta a janela de edição do fluxo de mensagem, clique em "Save" e ative as novas configurações no OSB (Figura 88).

Figura 88 – Finalizando a configuração da validação de estrutura no OSB

4.4.1.2 Testando a validação da mensagem

Ao executar o serviço onde está configurado o fluxo com a validação foi obtido o resultado esperado, conforme apresentado na Figura 89.

```
**************************************************
Codigo: 1
Nome: AS
Sigla: BRA
Indicador: PT
CódigoAGP: 1
   **************************************************
```

Figura 89 – Resultado da chamada do serviço após configuração da validação no OSB

Em seguida, a fim de testar a validação, o objeto alvo da validação foi modificado. Neste caso, o objeto da classe *ConsultaUnidadeOperativa*, que funciona como parâmetro para chamada do serviço. O atributo "Nome" foi excluído. Ao executar o serviço foi levantada a exceção pelo barramento, uma vez que a estrutura do objeto da mensagem de chamada do serviço não confere com a definição que foi configurada no OSB. A mensagem retornada é apresentada na Figura 90.

```
<WS data binding error>could not find getter for property 'Nome' on
br.com.petrobras.beans.ConsultaUnidadeOperativa
******************************************************
java.rmi.RemoteException: SOAPFaultException - FaultCode
[{http://schemas.xmlsoap.org/soap/envelope/}Server] FaultString [BEA-
382505: OSB Validate action failed validation] FaultActor [null] De-
tail [<detail><con:fault
xmlns:con="http://www.bea.com/wli/sb/context"><con:errorCode>BEA-
382505</con:errorCode><con:reason>OSB Validate action failed valida-
tion</con:reason><con:details><con1:ValidationFailureDetail
xmlns:con1="http://www.bea.com/wli/sb/stages/transform/config"><con1:m
essage>Expected element 'Nome@java:br.com.petrobras.beans' instead of
'Numero@java:br.com.petrobras.beans'
here</con1:message><con1:xmlLocation><java:Numero
xmlns:java="java:br.com.petrobras.beans">0</java:Numero></con1:xmlLoca
tion><con1:message>Expected element 'Nome@java:br.com.petrobras.beans'
instead of 'Sigla@java:br.com.petrobras.beans'
here</con1:message><con1:xmlLocation><java:Sigla
xmlns:java="java:br.com.petrobras.beans">BRA</java:Sigla></con1:xmlLoc
ation><con1:message>Expected element
'Nome@java:br.com.petrobras.beans' before the end of the con-
tent</con1:message><con1:xmlLocation></con1:xmlLocation></con1:Validat
ionFailureDe-
tail></con:details><con:location><con:node>PipelinePairNode1</con:node
><con:pipeline>PipelinePairNode1_request</con:pipeline><con:stage>stag
e1</con:stage><con:path>request-
pipeline</con:path></con:location></con:fault></detail>]; nested ex-
ception is:
      javax.xml.rpc.soap.SOAPFaultException: BEA-382505: OSB Validate
action failed validation
      at
br.com.petrobras.sistema.wsclient.UnidadeOperativaService_Stub.getUnid
adeOperativa(UnidadeOperativaService_Stub.java:37)
      at Main.WSCliente.main(WSCliente.java:29)
Caused by: javax.xml.rpc.soap.SOAPFaultException: BEA-382505: OSB Val-
idate action failed validation
      at web-
logic.wsee.codec.soap11.SoapCodec.decodeFault(SoapCodec.java:265)
      at weblog-
log-
ic.wsee.ws.dispatch.client.CodecHandler.decodeFault(CodecHandler.java:
106)
      at weblog-
log-
ic.wsee.ws.dispatch.client.CodecHandler.decode(CodecHandler.java:91)
      at weblog-
log-
ic.wsee.ws.dispatch.client.CodecHandler.handleFault(CodecHandler.java:
79)
      at weblog-
log-
ic.wsee.handler.HandlerIterator.handleFault(HandlerIterator.java:254)
      at weblog-
log-
ic.wsee.handler.HandlerIterator.handleResponse(HandlerIterator.java:22
4)
```

```
      at weblog-
log-
ic.wsee.ws.dispatch.client.ClientDispatcher.handleResponse(ClientDispa
tcher.java:161)
      at weblog-
log-
ic.wsee.ws.dispatch.client.ClientDispatcher.dispatch(ClientDispatcher.
java:116)
      at weblogic.wsee.ws.WsStub.invoke(WsStub.java:89)
      at weblogic.wsee.jaxrpc.StubImpl._invoke(StubImpl.java:335)
      at
br.com.petrobras.sistema.wsclient.UnidadeOperativaService_Stub.getUnid
adeOperativa(UnidadeOperativaService_Stub.java:32)
      ... 1 more
```

Figura 90 – Exceção levantada pelo OSB

A primeira parte da mensagem <WS data binding error> é a indicação da ausência do atributo que foi excluído, e é emitida pelo Workshop. O resto da mensagem é a exceção gerada pelo OSB. Observe que existe um código que identifica o tipo de erro (BEA-382505) e, logo após, a mensagem "OSB Validate action failed validation".

4.4.2 Configuração para personalização de mensagem de erro

É possível personalizar a mensagem de erro emitida pelo OSB quando é levantada a exceção após a identificação de desvio da estrutura que está sendo validada no barramento.

Para isso, é necessário configurar um estágio para manipulação de erros. Esse estágio é anexado ao estágio de validação, no fluxo de mensagens, para ser executado sempre que ocorrer algum erro. Logo, podem ser configuradas diversas ações ao identificar um erro em um dado estágio. Neste exemplo, será configurada apenas uma ação para enviar uma mensagem personalizada ao cliente após detectar um erro de validação, configurado na seção anterior.

4.4.2.1 Configurando um estágio para manipulação de erros

Na janela de edição do fluxo de mensagem, adicione um estágio para manipulação de erros (Add Stage Error Handler) a partir do estágio onde foi configurada a validação, conforme mostra a Figura 91.

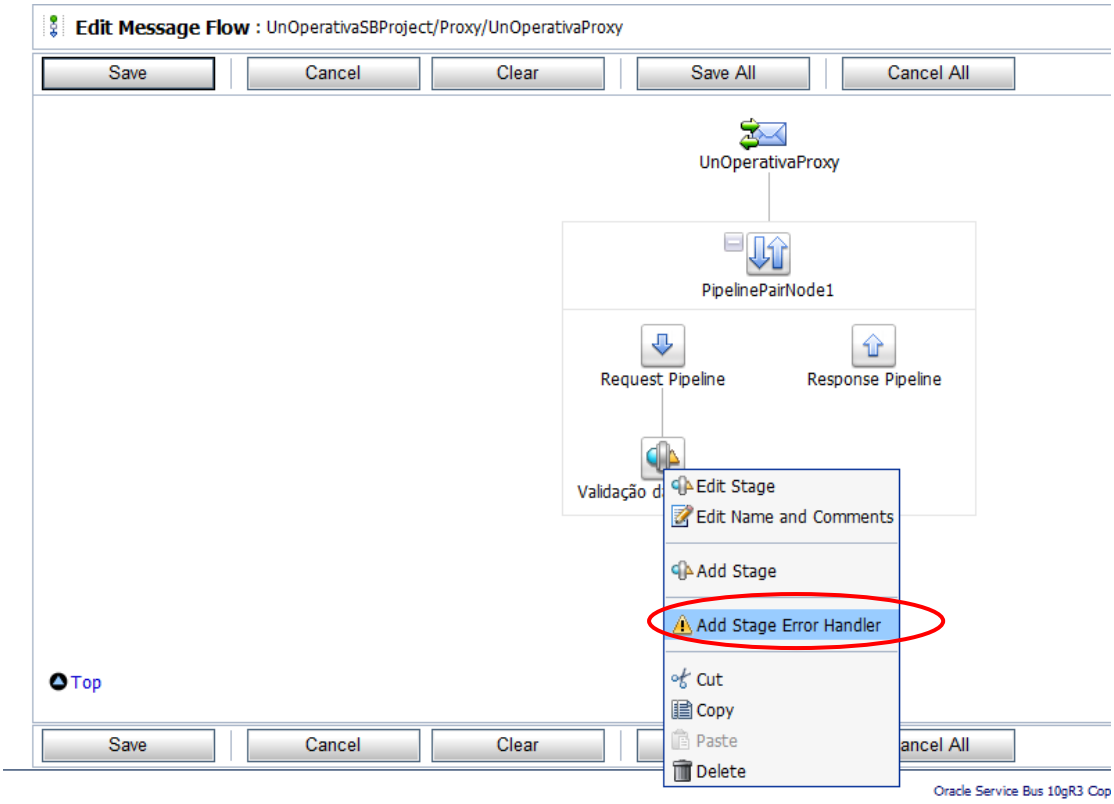

Figura 91 – Adicionando um estágio para manipulação de erro

Clique no ícone "Error Handler" e adicione um estágio (Add Stage) para configurar uma ação ao gerar a exceção pela validação (Figura 92).

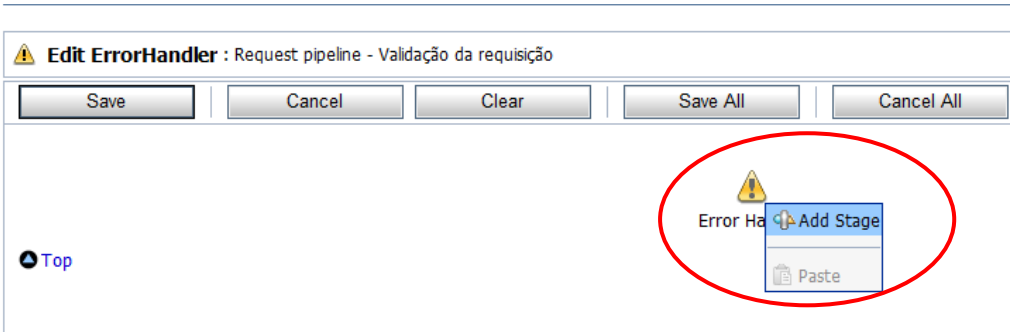

Figura 92 – Adicionando um estágio dentro da edição do manipulador de erro

Edite o estágio e adicione a ação de controle de fluxo "If...Then..." conforme ilustra a figura Figura 93.

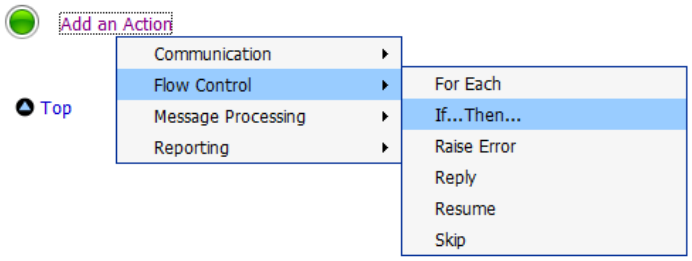

Figura 93 – Adicionando um controle de fluxo

Edite a expressão, clicando no link <Condition> (Figura 94).

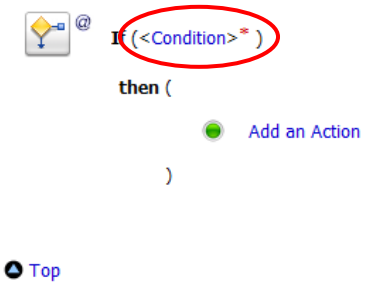

Figura 94 – Link para edição da expressão

Na próxima janela, clique em "Builder", ilustrado na Figura 95.

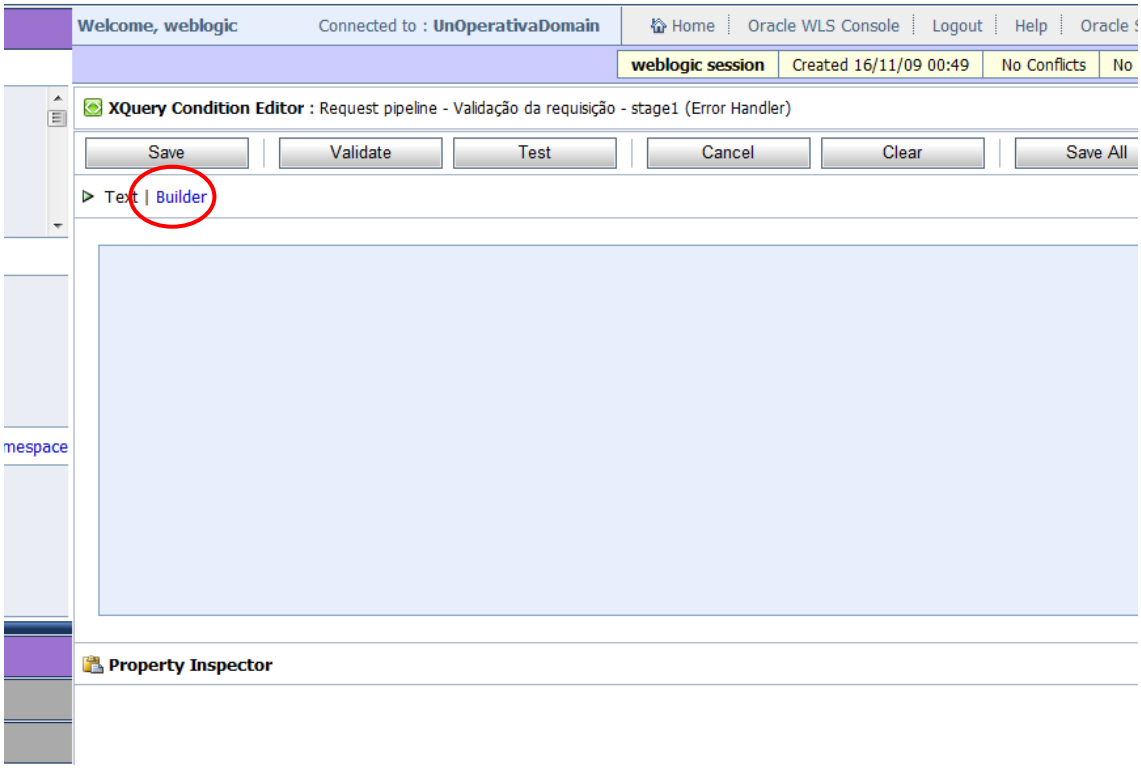

Figura 95 – Link "Builder"

Na janela que é apresentada (Figura 96), configure o operando (*Operand*) com o texto "$fault/ctx:errorCode", sem as aspas. No campo valor (*value*), insira o conteúdo "BEA-382505", inclusive com as aspas. O operador (*operator*) deve ser "=". Por fim, clique em "Add".

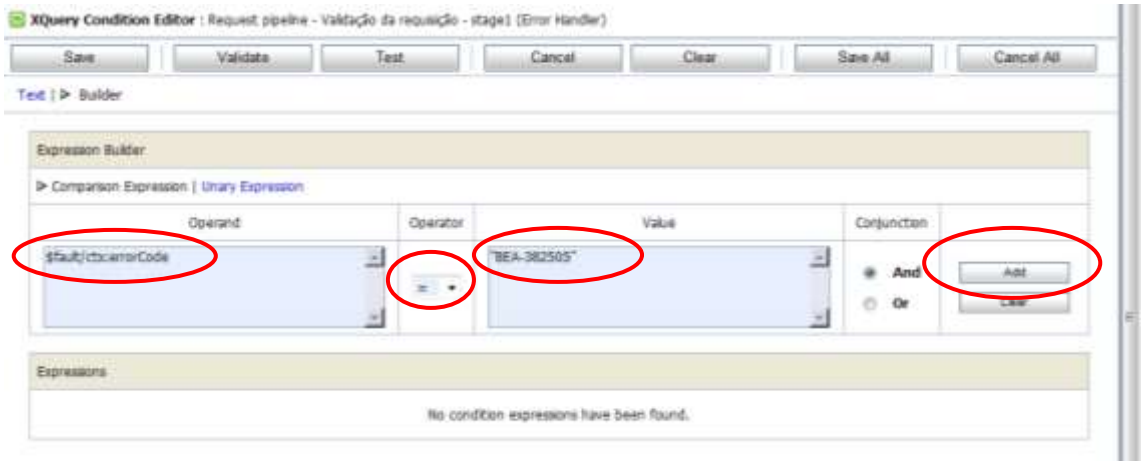

Figura 96 – Configuração da expressão

Nesta configuração é criada a expressão "$fault/ctx:errorCode = "BEA-382505"" que torna-se verdadeira quando ocorre um erro de validação, que é identificado pelo código interno "BEA-382505". Após adicionar a expressão, clique em "Validate" e então em "Save". Caso a expressão não seja validada, redigite manualmente o conteúdo.

Ao retornar a janela anterior, adicione a ação de controle de fluxo "Raise Error", conforme ilustra a Figura 97.

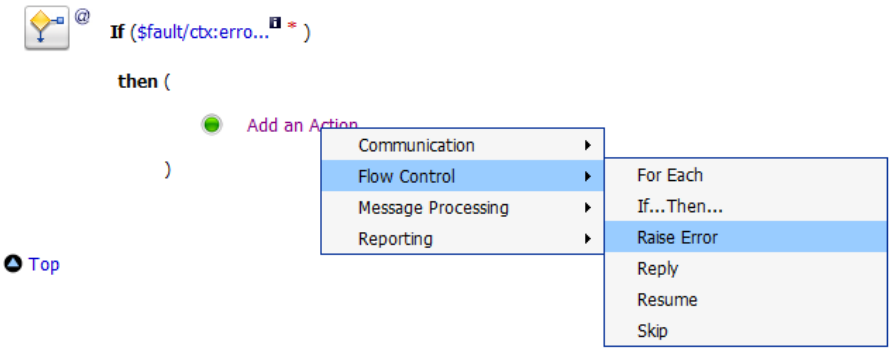

Figura 97 – Adicionando a ação de controle de fluxo "Raise Error"

Na próxima tela, insira a mensagem que gostaria que o cliente recebesse ao levantar a exceção de erro de validação da estrutura de mensagens (Figura 98).

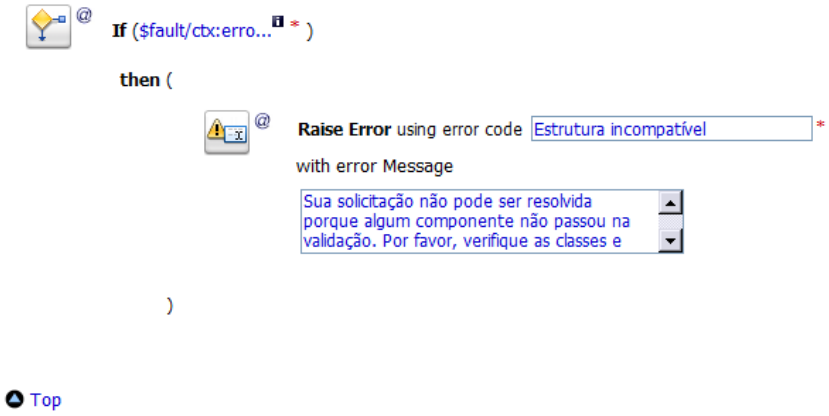

Figura 98 – Configuração da mensagem enviada ao cliente ao detectar erro de validação

Após inserir a mensagem, clique em “Validade” e depois em “Save”. Salve todas as alterações e ative-as no OSB.

4.4.2.2 Teste da configuração da manipulação do erro

Ao executar o cliente novamente com o objeto passado como parâmetro de chamada do serviço faltando o atributo “Nome”, é retornada a mensagem de erro apresentada na Figura 99. Observe que todo o detalhamento da mensagem permanece inalterado exceto a descrição configurada no OSB (destacada na Figura 100).

```
<WS data binding error>could not find getter for property 'Nome' on
br.com.petrobras.beans.ConsultaUnidadeOperativa
******************************************************
java.rmi.RemoteException: SOAPFaultException - FaultCode
[{http://schemas.xmlsoap.org/soap/envelope/}Server] FaultString [Es-
trutura incompatível: Sua solicitação não pode ser resolvida porque
algum componente não passou na validação. Por favor, verifique as
classes e parâmetros de sua mensagem.] FaultActor [null] Detail [<de-
tail><con:fault
xmlns:con="http://www.bea.com/wli/sb/context"><con:errorCode>Estrutura
incompatível</con:errorCode><con:reason>Sua solicitação não pode ser
resolvida porque algum componente não passou na validação. Por favor,
verifique as classes e parâmetros de sua men-
sagem.</con:reason><con:location><con:node>PipelinePairNode1</con:node
><con:pipeline>PipelinePairNode1_request</con:pipeline><con:stage>Vali
dação da requisição</con:stage><con:path>request-
pipeline</con:path><con:error-handler>true</con:error-
handler></con:location></con:fault></detail>]; nested exception is:
      javax.xml.rpc.soap.SOAPFaultException: Estrutura incompatível:
Sua solicitação não pode ser resolvida porque algum componente não
passou na validação. Por favor, verifique as classes e parâmetros de
sua mensagem.
      at
br.com.petrobras.sistema.wsclient.UnidadeOperativaService_Stub.getUnid
adeOperativa(UnidadeOperativaService_Stub.java:37)
      at Main.WSCliente.main(WSCliente.java:29)
Caused by: javax.xml.rpc.soap.SOAPFaultException: Estrutura incompatí-
vel: Sua solicitação não pode ser resolvida porque algum componente
não passou na validação. Por favor, verifique as classes e parâmetros
de sua mensagem.
      at weblo-
gic.wsee.codec.soap11.SoapCodec.decodeFault(SoapCodec.java:265)
      at weblo-
gic.wsee.ws.dispatch.client.CodecHandler.decodeFault(CodecHandler.java
:106)
      at weblo-
gic.wsee.ws.dispatch.client.CodecHandler.decode(CodecHandler.java:91)
      at weblog-
log-
ic.wsee.ws.dispatch.client.CodecHandler.handleFault(CodecHandler.java:
79)
      at weblog-
log-
ic.wsee.handler.HandlerIterator.handleFault(HandlerIterator.java:254)
      at weblog-
log-
ic.wsee.handler.HandlerIterator.handleResponse(HandlerIterator.java:22
4)
      at weblog-
log-
ic.wsee.ws.dispatch.client.ClientDispatcher.handleResponse(ClientDispa
tcher.java:161)
```

```
      at weblog-
log-
ic.wsee.ws.dispatch.client.ClientDispatcher.dispatch(ClientDispatcher.
java:116)
      at weblogic.wsee.ws.WsStub.invoke(WsStub.java:89)
      at weblogic.wsee.jaxrpc.StubImpl._invoke(StubImpl.java:335)
      at
br.com.petrobras.sistema.wsclient.UnidadeOperativaService_Stub.getUnid
adeOperativa(UnidadeOperativaService_Stub.java:32)
      ... 1 more
```

Figura 99 – Mensagem de erro personalizada

```
java.rmi.RemoteException: SOAPFaultException - FaultCode
[{http://schemas.xmlsoap.org/soap/envelope/}Server] FaultString [Es-
trutura incompatível: Sua solicitação não pode ser resolvida porque
algum componente não passou na validação. Por favor, verifique as
classes e parâmetros de sua mensagem.] FaultActor [null] Detail [<de-
tail><con:fault
xmlns:con="http://www.bea.com/wli/sb/context"><con:errorCode>Estrutura
incompatível</con:errorCode><con:reason>Sua solicitação não pode ser
resolvida porque algum componente não passou na validação. Por favor,
verifique as classes e parâmetros de sua men-
sagem.</con:reason><con:location><con:node>PipelinePairNode1</con:node
><con:pipeline>PipelinePairNode1_request</con:pipeline><con:stage>Vali
dação da requisição</con:stage><con:path>request-
pipeline</con:path><con:error-handler>true</con:error-
handler></con:location></con:fault></detail>]; nested exception is:
      javax.xml.rpc.soap.SOAPFaultException: Estrutura incompatível:
Sua solicitação não pode ser resolvida porque algum componente não
passou na validação. Por favor, verifique as classes e parâmetros de
sua mensagem.
```

Figura 100 – Detalhe da mensagem de erro retornada pelo barramento

4.4.3 Conclusão

O ESB pode ser configurado para realizar validações críticas nas estruturas das mensagens e identificar alterações em seus objetos e parâmetros que podem, por exemplo, representar equívocos por parte do código do cliente ou servidor. Como uma exceção permite a execução de outras ações. No OSB, é possível configurar compensações para os erros identificados. Por exemplo, um erro de validação de estrutura poderia iniciar outra ação inserida em um manipulador de erro para rotear essa mensagem para outras versões do mesmo serviço, a fim de verificar se a diferença encontrada na estrutura do serviço é um problema de versionamento ou se realmente trata-se de um erro que deve ser registrado e comunicado ao cliente, seja através de mensagens personalizadas ou até através de outros serviços que poderiam ser invocados a partir do erro de validação.

Portanto, a ferramenta permite a configuração de várias ações a partir de um erro em algum estágio. Sua estrutura é análoga a um processo que possui subprocessos de compensação para possíveis eventos que possam surgir durante a execução do processo principal, por exemplo, um erro levantado por alguma exceção.

4.5 Versionamento de serviços utilizando o OSB

Esta seção apresenta soluções para versionamento de serviços utilizando o OSB, de acordo com exemplos práticos de necessidades de versionamento identificados juntos à GDIEP.

4.5.1 Alteração do parâmetro de entrada de serviço

Nesta seção é apresentada a solução para o problema de versionamento de parâmetro de entrada de serviço utilizando o barramento.

4.5.1.1 Descrição do problema e projeto de solução

A necessidade de versionamento de serviço correspondente ao seguinte cenário:

- O serviço está sendo utilizado por n consumidores e um dos consumidores solicita que um parâmetro p_1 de entrada do serviço seja alterado para um novo parâmetro p_2. Neste caso, existe uma correspondência direta entre os valores de p_1 e p_2.

Este caso corresponde ao seguinte exemplo real:

- O serviço que realiza a consulta às informações de plataforma, na versão v_1, tinha um método que recebia a sigla da Unidade Operativa como parâmetro, e, na versão v_2, foi solicitado que fosse passado o código da Unidade Operativa como parâmetro, ao invés da sigla.
- Uma observação importante é que o código da unidade operativa pode ser obtido a partir da sigla.

Portanto, dado esta modificação no serviço, todos os consumidores da versão v_1 do serviço que passavam a sigla como parâmetro deveriam agora passar o código da unidade operativa como parâmetro, invocando a versão v_2 do serviço. Ou seja, se os consumidores utilizarem a mesma chamada do serviço para invocar a nova versão, ocorrerá erro.

Com intuito de realizar a alteração para atender ao consumidor que solicitou a modificação do serviço sem impactar os demais consumidores, o barramento de serviços foi utilizado para resolver este problema.

A Figura 101 apresenta um esquema de invocação da versão v_1 do serviço PlataformaService a qual estava publicada no barramento. Neste caso, a versão v_1 do serviço foi publicada no barramento e um serviço de Proxy ($Proxy_1$) é responsável por fazer o roteamento das solicitações dos consumidores para o serviço.

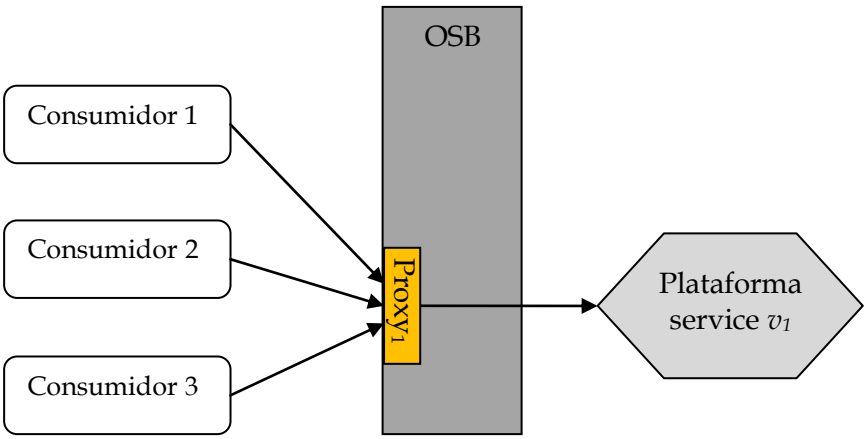

Figura 101 – Esquema de invocação da versão v_1 do serviço PlataformaService publicado no barramento

Para solucionar o problema do versionamento do serviço e dado que existe um mapeamento entre o código da unidade operativa a partir da sigla, decidiu-se então manter uma única versão do serviço (a versão v_2). A Figura 102 apresenta um esquema da solução adotada. Desta forma a versão v_2 do serviço foi publicada no barramento e a versão v_1 foi removida. Um serviço de proxy ($Proxy_2$) foi criado para fazer o roteamento para a versão v_2. O consumidor que solicitou a nova versão (Consumidor 3) passou a invocar o serviço de $proxy_2$. A fim de que os outros consumidores não fossem impactados com a modificação, ou seja, continuassem invocando o mesmo serviço, o serviço de $proxy_1$ foi alterado para passar a invocar a versão v_2 do serviço PlataformaService. Ou seja, os parâmetros de entrado do seviço de $proxy_1$ foram mantidos (recebe a sigla da unidade operativa), no entanto, ao invés do serviço de $proxy_1$ fazer o roteamento para a versão v_1 do serviço PlataformaService ele passou a fazer o roteamento para a versão v_2. Para poder realizar esta invocação, o serviço de $proxy_1'$ faz uma transformação da mensagem recebida que contém a sigla para passar a ter o elemento código. Os seguintes passos são realizados no fluxo do serviço de $proxy_1'$:

- Recebe a mensagem de entrada;
- Realiza um DatabaseLookUp para recuperar o código correspondente à sigla da Unidade Operativa que veio na mensagem;
- Substitui no corpo da mensagem a sigla da Unidade Operativa para ser o código da Unidade Operativa e o nome da função que está sendo chamada no serviço versão v_1 para o nome da função que será invocada no serviço versão v_2;
- Faz o roteamento para a versão v_2 do serviço.

Não foi necessário fazer nenhuma transformação na mensagem de resposta, pois esta não sofreu alteração da versão v_1 para v_2.

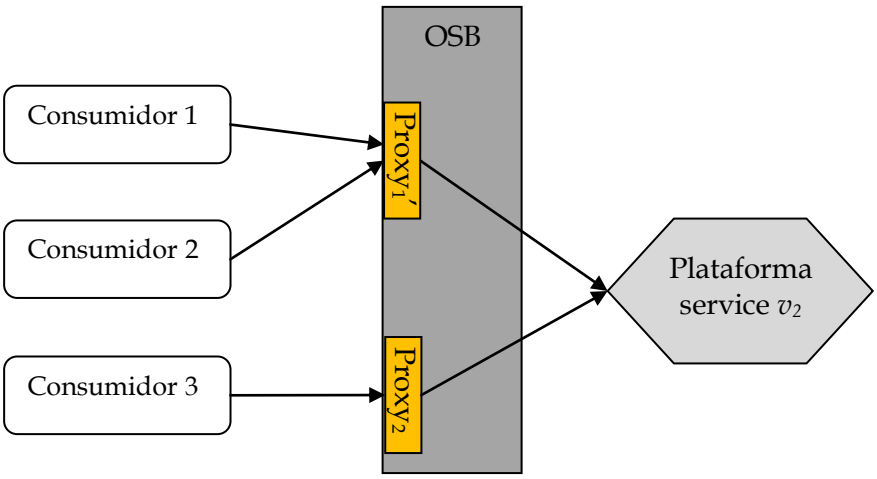

Figura 102 - Esquema de invocação da versão v_2 do serviço PlataformaService publicado no barramento

4.5.1.2 Implementação da solução

O OSB oferece uma funcionalidade em seu fluxo de mensagem que permite a modificação do conteúdo de uma mensagem XML. Para realizar a chamada do serviço versão v_2 é necessário o parâmetro "codigo" que é obtido a partir do parâmetro "sigla" através da execução de uma consulta SQL. Utilizando a função "Replace" para alterar o conteúdo da mensagem em conjunto com a função "fn-bea:execute-sql" para executar o comando SQL, é possível configurar um "*Stage*" no fluxo de mensagem que realiza as alterações necessárias na mensagem de chamada do serviço versão v_1 para a correta invocação do serviço versão v_2.

Para atender à necessidade dos clientes que invocam o serviço passando o parâmetro "sigla" ou "codigo" foram implementados os serviços de proxy UnOpCodigoProxy (Proxy1 da Figura 102) e UnOpSiglaProxy (Proxy 2 da Figura 102). O serviço UnOpCodigoProxy é utilizado pelos clientes que invocam o serviço utilizando o parâmetro "codigo"e ele somente realiza o roteamento da mensagem para o serviço versão v_2, conforme sua configuração de fluxo de mensagem, ilustrada na Figura 103.

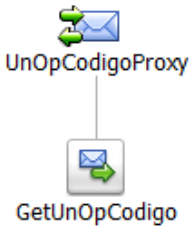

Figura 103 – Configuração do UnOpCodigoProxy

Os clientes configurados para invocar o serviço passando como parâmetro a "sigla" da Unidade Operativa, permanecem invocando o antigo serviço de proxy UnOpSiglaProxy, porém, ele sofrerá modificações em seu fluxo de mensagem para rotear suas mensagens para o novo serviço versão v_2.

Como pode ser visto na Figura 104, o serviço UnOpSiglaProxy está configurado para repassar as mensagens ao serviço versão v_2. Entretanto foi inserido um novo estágio (*stage 1)* no caminho de requisição do serviço (Request Pipeline) que será configurado

para alterar a mensagem original, adaptando-a para invocar corretamente o serviço de destino do proxy (Figura 104).

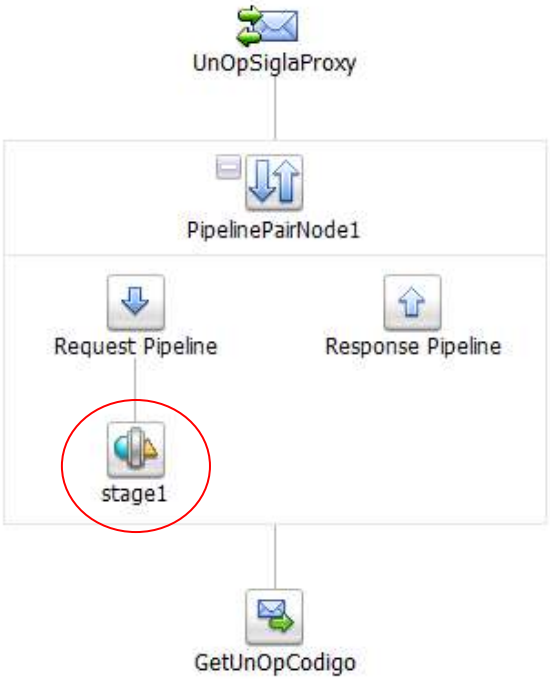

Figura 104 – Configuração do "*Stage 1*", no proxy UnOpSiglaProxy

O passo a passo da configuração do estágio é descrito a seguir:

Ao editar o "*Stage 1*", adicione a ação de processamento da mensagem "*Replace*", conforme ilustra a Figura 105:

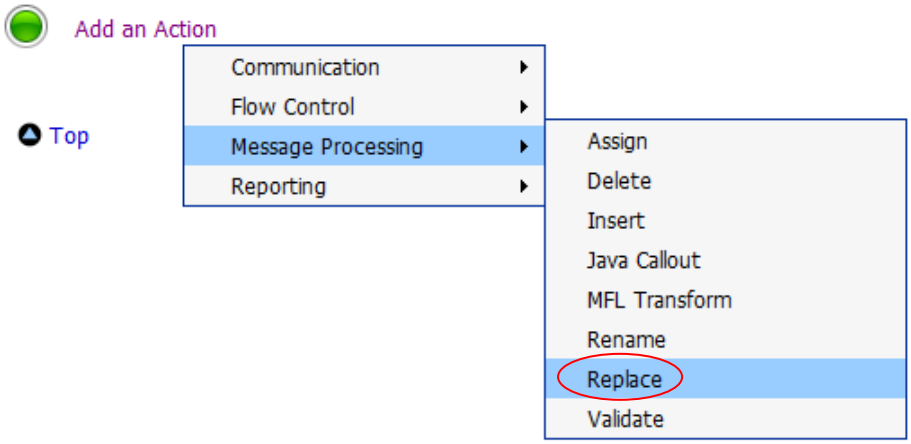

Figura 105 – Adicionando a ação Repleace no proxy UnOpSiglaProxy

A configuração do "*Replace*"consiste em selecionar alguma parte da estrutura da mensagem utilizando o *Xpath"* e alterá-la com um valor definido em "*Expression"*. É possível alterar um nó inteiro na mensagem com o valor definido, caso selecione a opção "*Replace entire node"*, ou substituir somente o valor de variáveis, caso selecione a opção "*Replace node contents"*.

Clique no link <Xpath> para selecionar a estrutura que será alterada (Figura 106):

Figura 106 – Link <Xpath>

Selecione a estrutura a ser alterada e copie seu respectivo comando Xpath para o quadro no centro da janela. Clique em "Save". No exemplo, todo o nó será modificado para permitir a chamada de outro serviço (Figura 107).

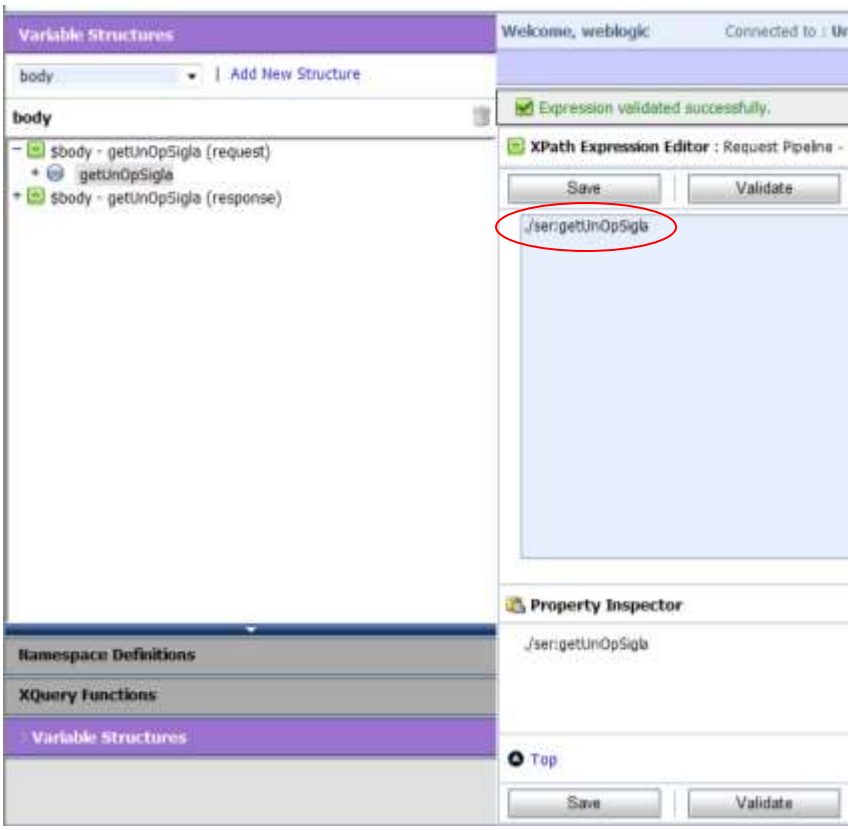

Figura 107 – Seleção da estrutura a ser alterada

Preencha o campo com o nome da "Variable Structure" que é "body"e clique no link <Expression> (Figura 108):

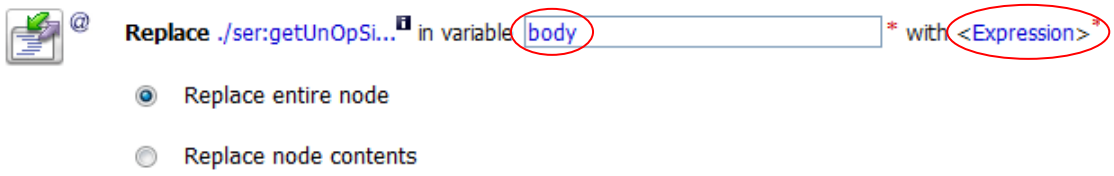

Figura 108 – Configuração da função Repleace

Na próxima janela, deverá ser configurada a expressão que substituirá o nó selecionado anteriormente. O objetivo é reescrever o código da mensagem para invocar corretamente o serviço versão v_2. Para facilitar o entendimento, o código será apresentado gradativamente. No primeiro momento, é construída a estrutura textual com as tags

da mensagem XML (Figura 109) contendo o nome da função que será chamada no serviço e o parâmetro "codigo". Observe que não há conteúdo definido para a tag que representa o código (<ser:codigo>), já que ele será definido pela função "fn-bea:execute-sql".

```
<ser:getUnOpCodigo  xmlns:ser="http://br/com/petrobras/services">
  <ser:codigo>  </ser:codigo>
</ser:getUnOpCodigo>
```

Figura 109 – Código XML que define a estrutura da mensagem para invocar o novo serviço

Para definir o valor do código, será executada uma consulta SQL para obter o valor do banco a partir da sigla. Para isso será utilizada a função "fn-bea:execute-sql", que possui a seguinte sintaxe:

fn-bea:execute-sql($datasource-string, $rowElemName, $sql-string, $params...)

onde:

$*datasource* é o nome do JNDI do *datasource.*

$*rowElemName* é o nome do elemento que define a estrutura de retorno.

$*sql-string* é o comando SQL.

$*params* ... é a enumeração de parâmetros de 1 até k.

Para o exemplo proposto, a linha de comando da função será como a ilustrada na Figura 110:

```
fn-bea:execute-sql(
        'UnOperativaDataSource',
        xs:QName('getUnOpCodigo'),
        'SELECT unop_cd_unid_oper as codigo FROM unid_operativa WHERE unop_sg_unid_oper =?', xs:string($body/ser:getUnOpSigla/ser:sigla)))
```

Figura 110 – Linha de comando da função fn-bea:execute-sql()

O retorno desta função é uma estrutura conforme ilustra a Figura 111:

```
< getUnOpCodigo>
      < CODIGO>99999</CODIGO>
</getUnOpCodigo>
```

Figura 111 – Retorno da função fn-bea:execute-sql()

Para extrair somente o valor do código, que é o valor de interesse, deve-se utilizar a função *fn:string($arg-atomic).* O código pode ser visto na Figura 112:

```
<ser:getUnOpCodigo  xmlns:ser="http://br/com/petrobras/services">

  <ser:codigo>
  {
    fn:string(
      fn-bea:execute-sql(
        'UnOperativaDataSource',
        xs:QName('getUnOpCodigo'),
```

```
        'SELECT unop_cd_unid_oper as codigo FROM unid_operativa WHERE un-
op_sg_unid_oper =?', xs:string($body/ser:getUnOpSigla/ser:sigla)))
  }
  </ser:codigo>

</ser:getUnOpCodigo>
```

Figura 112 – Código que retorna o conteúdo que substituirá a estrutura anterior

Observe que o comando foi inserido entre chaves ({}). Isso é necessário para o OSB interpretar o conteúdo como um comando e não somente um texto. Após finalizar a configuração, clique em "Validate" e depois em "Save".

A configuração final da função "Replace" é ilustrada na Figura 113. A opção "*Replace entire node*" deve permanecer marcada. Clique em "Validate" e depois em "Save".

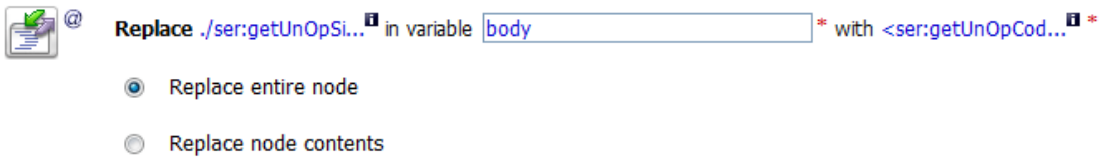

Figura 113 – Configuração final da função "Repleace"

O fluxo de mensagem final pode ser visualizado na Figura 114:

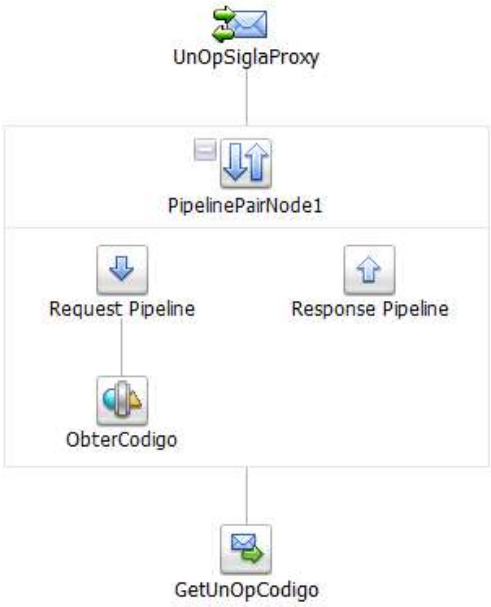

Figura 114 – Fluxo de mensagem do Proxy UnOpSiglaProxy

Ao executar o cliente para o proxy UnOpSiglaProxy, foi obtido o resultado esperado, conforme ilustra a Figura 115:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Header />
  <env:Body>
  <m:getUnOpSiglaResponse xmlns:m="http://br/com/petrobras/services">
   <unid:UnidadesOperativasList xmlns:unid="http://controls.beans/unidadeoperativa">
    <unid:UnidadeOperativa>
     <unid:codigoUnidadeOperativa>1</unid:codigoUnidadeOperativa>
     <unid:indicadorAtiva>PT</unid:indicadorAtiva>
     <unid:siglaUnidadeOperativa>BRA</unid:siglaUnidadeOperativa>
     <unid:nomeUnidadeOperativa>AS</unid:nomeUnidadeOperativa>
     <unid:codigoUnidadeOperativaAGP>1</unid:codigoUnidadeOperativaAGP>
    </unid:UnidadeOperativa>
   </unid:UnidadesOperativasList>
  </m:getUnOpSiglaResponse>
 </env:Body>
 </env:Envelope>
```

Figura 115 – Resposta da chamada do serviço de proxy UnOpSiglaProxy

4.5.1.3 Conclusão

O OSB permite alteração do texto da mensagem XML com flexibilidade, permitindo alterar tanto estrutura ou como também somente os valores das variáveis. No problema real citado nesta seção, houve a necessidade de adaptar uma mensagem endereçada a um serviço para que fosse executado por outro. A função "*Replace*" permitiu a edição da mensagem original e a função "*fn-bea:execute-sql*" permitiu consultar o banco de dados em tempo de execução para obter o parâmetro correto que satisfaz o serviço para o qual a mensagem modificada seria roteada pelo barramento. O resultado foi satisfatório já que todo esse processo se mostrou transparente ao cliente do serviço.

4.6 Monitoramento de serviços

O OSB gerencia as transformações e roteamento de mensagens no barramento e oferece opções administrativas e de monitoramento. Por exemplo, o OSB permite coletar informações em tempo real sobre o fluxo de mensagens que percorrem o barramento em um determinado período.

O painel de informações do OSB (*dashboard*) permite monitorar o estado do sistema e visualizar as notificações emitidas pelo OSB quando ocorrem alertas emitidos pelos serviços. Com estas informações o responsável pela gerência do ambiente de serviços pode rapidamente isolar e diagnosticar os problemas que ocorrem.

O *framework* de monitoramento do OSB provê várias funções de acesso a informações do que está ocorrendo no barramento. Por exemplo, informações sobre o número de mensagens que foram processadas com sucesso ou falha, o projeto que o serviço pertence, a média de tempo de execução do processamento de mensagens e o número de alertas associados a um serviço.

4.6.1 Configuração do tempo de coleta de dados

O OSB realiza a coleta de dados em um intervalo fixo de tempo, chamado de "intervalo de agregação" (*aggregation interval*). Este intervalo pode ser configurado pelo usuário do OSB através de uma lista com valores pré-definidos oferecida pela ferramenta.

Para um dado intervalo de agregação, é automaticamente configurado um subintervalo chamado de "intervalo de amostra" (*sample interval),* onde serão computadas as estatísticas em tempo real. No intervalo de amostra são coletadas, a cada período, as informações relativas à função configurada no OSB. Essas informações são armazenadas a cada ciclo do intervalo de amostra até o momento em que o intervalo de agregação também finaliza seu ciclo, e então todas as amostras coletadas são publicadas no console do OSB. A relação entre o intervalo de agregação e o intervalo de amostra pode ser vista na Tabela 1. Por exemplo, no primeiro caso, de 1 em 1 minuto serão gravadas as amostras (vide coluna intervalo de amostra) e, de acordo com o intervalo de agregação escolhido (1 ou 2 ou 3 ou 4 ou 5 minutos), as informações serão publicadas no barramento.

Tabela 1 – Relação entre intervalo de agregação e intervalo de amostra

Intervalo de agregação	Intervalo de amostra
1, 2, 3, 4 e 5 minutos	1 minuto
10, 15, 20, 25 e 30 minutos	5 minutos
40, 50 e 60 minutos	10 minutos
90 e 120 minutos	30 minutos
3, 4, 5 e 6 horas	1 hora
8, 10 e 12 horas	2 horas
16, 20 e 24 horas	4 horas
2, 3, 4, 5, 6 e 7 dias	1 dia

O intervalo de agregação no OSB possui as seguintes propriedades:

- Só é possível configurar e obter estatísticas de um serviço em relação a somente um intervalo de agregação.
- Não é possível configurar valores arbitrários para o intervalo de agregação. Somente os valores oferecidos pela ferramenta podem ser selecionados.
- O intervalo de agregação somente pode ser configurado para serviços de proxy ou de negócio (*business service*).
- Uma regra de alerta (*alert rule*) deve ser configurada através da definição de uma condição e do respectivo valor do intervalo de agregação.

É importante observar que, se um serviço for renomeado, todas as estatísticas que foram coletadas sobre ele serão perdidas, todas as configurações serão reiniciadas e o monitoramento do serviço iniciará do começo. Caso seja modificado somente o intervalo de agregação de um serviço, as estatísticas desse serviço serão reiniciadas, porém as métricas de contagem configuradas e registradas não serão modificadas.

Para configurar o intervalo de agregação no OSB, siga os passos apresentados a seguir.

1. Clique no serviço (*business service* ou *Proxy service*) para o qual deseja configurar o intervalo de agregação (Figura 116). É importante observar que, para realizar esta configuração, é necessário criar uma seção.

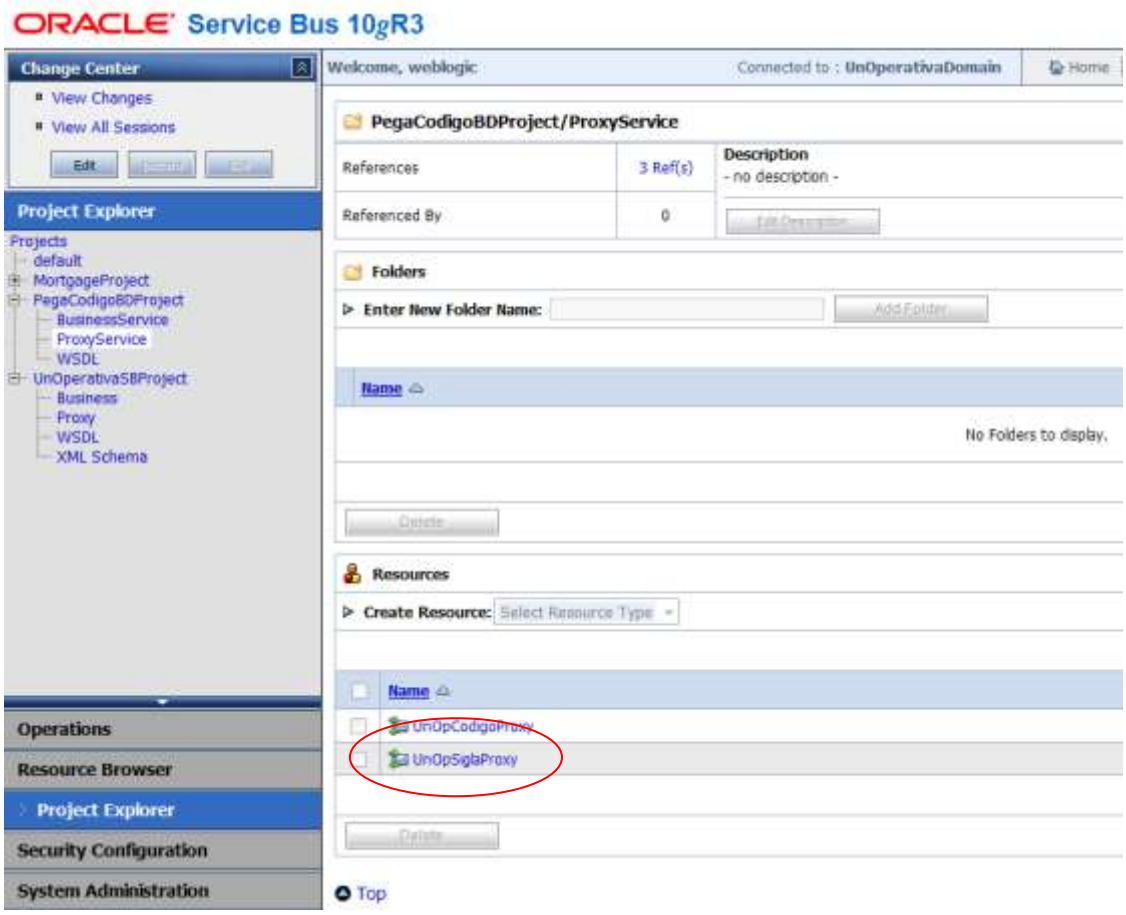

Figura 116 – Seleção do serviço para configuração do intervalo de agregação

2. Clique no link "Operational Settings", conforme mostra a Figura 117:

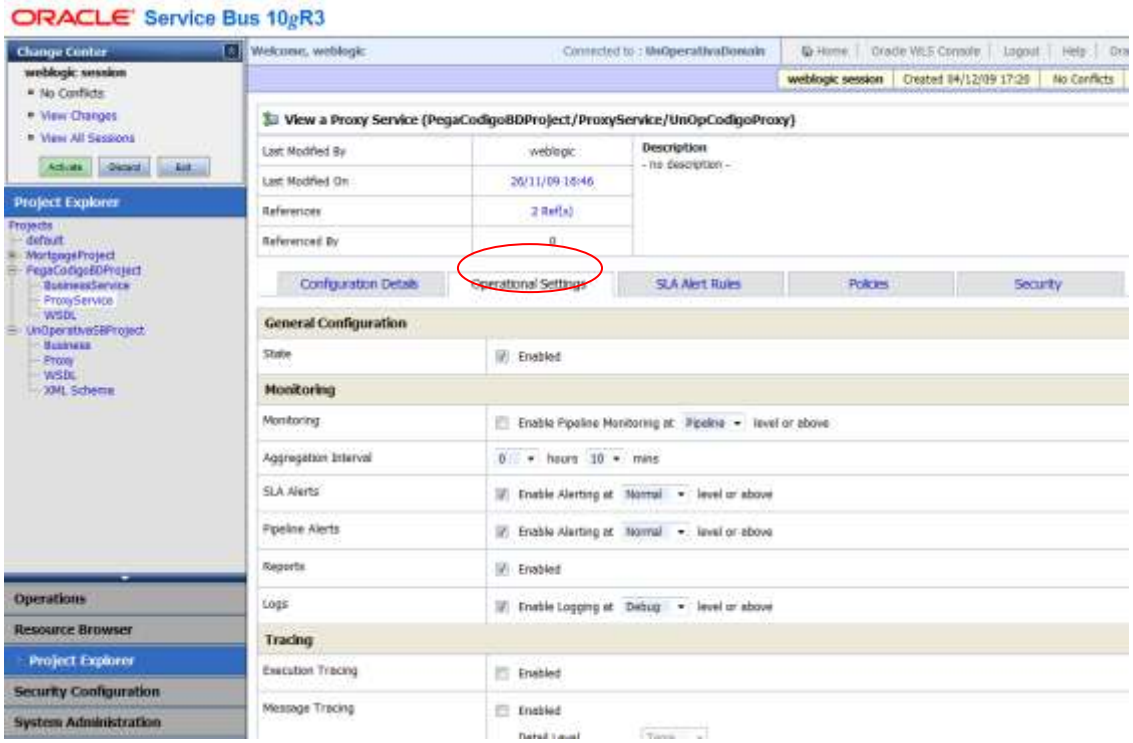

Figura 117 – Link "Operational Settings"

Nesta tela estão as opções que habilitam os diferentes tipos de mensagens que podem ser registradas pelo monitoramento do OSB. Em *"Aggregation Interval"* é possível configurar o intervalo de agregação utilizando os valores que estão pré-definidos entre 0 minutos e 7 dias. Outros alertas podem ser habilitados devendo ainda configurar o nível de gravidade mínima da mensagem que deverá ser considerada pelo monitor. Ao selecionar um nível de gravidade, somente as mensagens deste nível e de níveis mais críticos serão alertadas pelo OSB. A Tabela 2 detalha as opções desta tela, as quais são descritas em mais detalhes nas próximas seções.

Tabela 2 – Opções do Oracle Service Bus Console

Funcionalidade	Descrição	Padrão
State	Esta opção habilita ou desabilita o serviço	Habilitado
Monitoring	Esta opção habilita o serviço de monitoramento a partir de um nível específico ou superior. Ainda é possível desabilitar o monitoramento do serviço para um serviço de Proxy.	Desabilitado
Aggregation Interval	Esta opção configura o intervalo de agregação do serviço.	10 minutos
SLA Alerts	Esta opção habilita o alerta de SLA para os serviços com um nível de gravidade específico ou maior. Ainda é possível desabilitar o alerta de SLA para o serviço.	Habilitado
Pipeline Alerts	Esta opção habilita o alerta de *pipeline* para serviços de Proxy com um nível de gravidade específico ou maior. Ainda é possível desabilitar o alerta de *pipelining* para o serviço de Proxy.	Habilitado no nível Normal ou maior
Reports	Esta opção habilita ou desabilita a mensagem de relatório para o serviço de Proxy.	Habilitado nível Normal ou maior
Logs	Esta opção habilita o Log para mensagens com um nível de gravidade específico ou maior. Ainda é possível desabilitar o Log para serviços de Proxy.	Habilitado no nível de Debug ou maior
Execution Tracing	Esta opção habilita ou desabilita o "*Execution Tracing"* para serviços de Proxy.	Desabilitado
Message Tracing	Esta opção habilita ou desabilita o "*Message tracing"* para um serviço de Proxy com um nível de detalhe específico ou maior. Ainda é possível configurar o limite da carga (em KB) e o codificador padrão.	Desabilitado
Offline Endpoint URIs	Esta opção habilita ou desabilita *endpoints* de serviços de negócio que não respondem. Ainda é possível especificar um intervalo de tempo para espera, antes de tentar novamente uma chamada ao *endpoint* que se encontra *offline*. Somente é possível habilitar e desabilitar *endpoints offline* pra serviços de negócio.	Desabilitado
Throttling State	Esta opção habilita ou desabilita *throttling* para um serviço de negócio.	Desabilitado
Maximum Concurrency	Esta opção restringe o número de mensagens que podem ser processadas concorrentemente por um serviço de negócio.	0

Throttling Queue	Esta opção restringe o número máximo de mensagens na fila do throttling.	0
Message Expiration	O intervalo máximo de tempo (em milissegundos) no qual uma mensagem pode ser instalada em uma fila de throttling.	0

4.6.2 Mensagens de monitoramento no OSB

Esta seção apresenta as funcionalidades apresentada na Tabela 2, explicando a configuração prática destas funções.

4.6.2.1 Alert-Destination

Os destinos de alerta (*Alert-Destination)* são recursos inseridos no OSB que permitem configurar o destino para a emissão das mensagens dos alertas levantados pelo barramento. Por padrão, o Oracle Service Bus Console é um destino para as notificações de qualquer alerta, independente da configuração de outros destinos de alerta. No OSB é possível configurar um e-mail, SNMP Traps, relatório e JMS como destinos de alerta.

O SNMP (Simple Network Management Protocol) *traps* oferece uma interface a outros softwares para monitorarem os níveis de acordo de serviço configurados no OSB. Habilitando notificações de alerta do SNMP Trap, as ferramentas *Web Services Management* (WSM) e o *Enterprise Service Management* (ESM) podem monitorar violações de SLA e alertas de *pipeline* através de suas notificações de alerta.

Para maiores informações sobre a configuração de destinos de alerta no OSB, consulte o guia de operações (Operations Guide) do Oracle Service Bus [OSB, 2008b].

4.6.2.2 Configuração de alertas de SLA (SLA Alerts)

Os alertas de SLA[11] são mensagens emitidas automaticamente quando é identificada a violação de algum contrato de nível de serviço ou condição pré-definida configurada no OSB. Os alertas de SLA podem ser utilizados para:

- Monitoramento e geração de notificação por e-mail de erros de segurança (WS-Security).
- Monitoramento do número de mensagens que trafegam em um dado *pipeline*.
- Detecção da violação do contrato de nível de serviço (SLA) com produtos de terceiros (SNMP *Trap*).
- Detecção de *enpoints* que não respondem.

Para configurar um alerta de SLA no OSB, execute os seguintes passos:

1. Clique no serviço (*Proxy service* ou *Business service*) o qual deseja configurar o alerta de SLA, conforme mostra a Figura 118.

11 http://pt.wikipedia.org/wiki/Acordo_de_N%C3%ADvel_de_Servi%C3%A7o

Figura 118 – Selecionando o serviço para a configuração do alerta SLA

2. Na próxima janela, clique em "SLA Alert Rules", conforme ilustra a Figura 119:

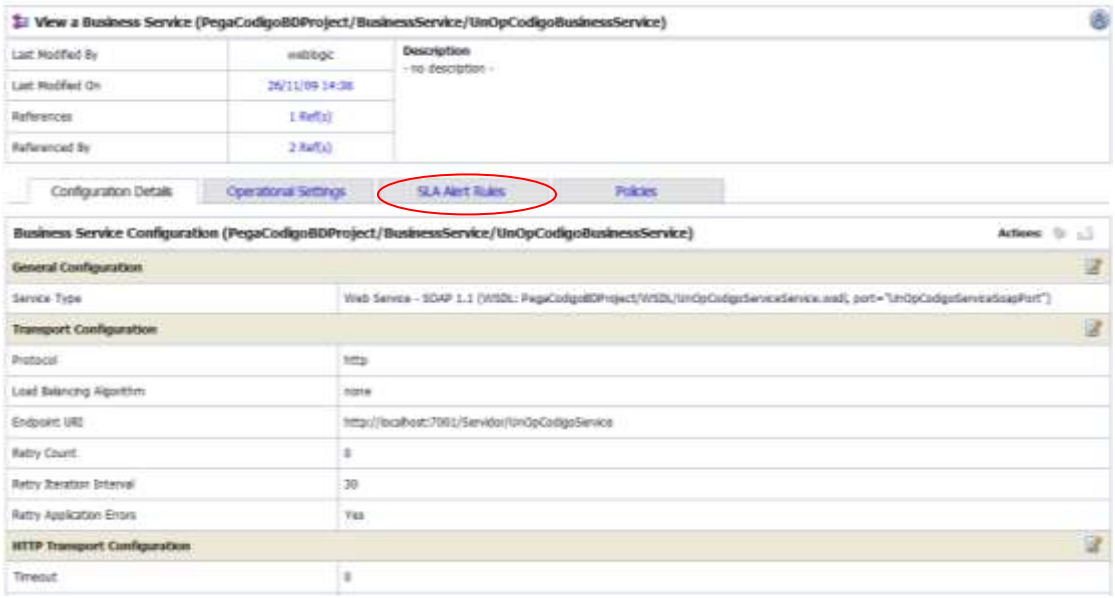

Figura 119 – Janela de configuração do serviço de negócio (Business service)

3. Na próxima janela, clique em "*Add new*" (Figura 120).

Figura 120 – Adicionando uma regra para o alerta SLA

4. Na janela de configuração do alerta insira os parâmetros necessários e clique em "Next"(Figura 121).

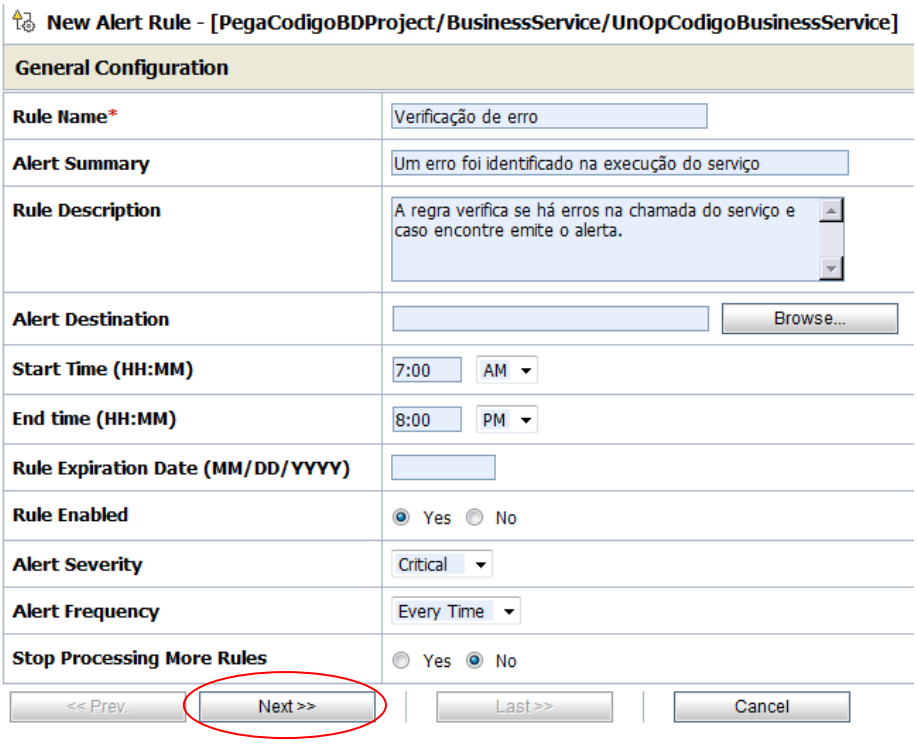

Figura 121 – Configuração de nova regra de alerta

5. Na janela de configuração de condições de regra de alerta, configure o intervalo de agregação (*Aggregation Interval*) para esta regra e a expressão condicional para disparar a mensagem. Caso seja necessário utilizar expressões compostas utilizando os operadores "*And*" e/ou "*Or*", insira primeiro todas as expressões condicionais, depois selecione as expressões que farão parte da composição e clique no botão "*And*" ou "*Or*"(Figura 122).

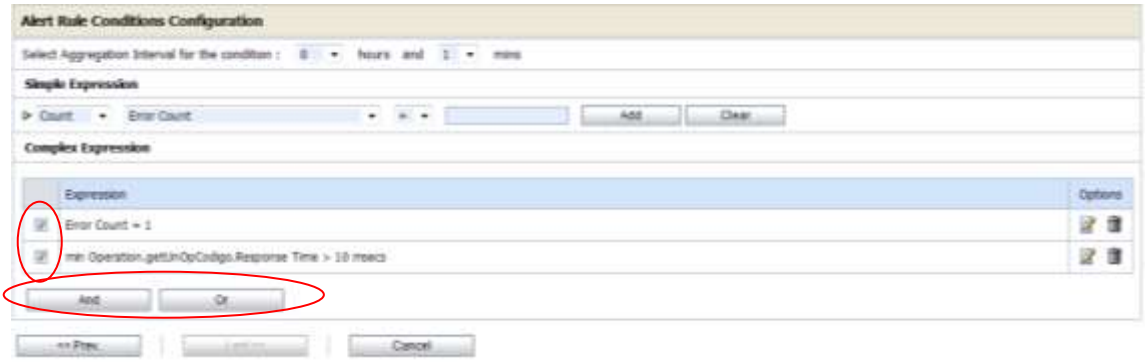

Figura 122 – Configuração de expressão composta para alerta de SLA

6. É possível realizar composições com mais de duas expressões e/ou com expressões que já foram compostas. Para isso, selecione as expressões desejadas e clique no respectivo operador lógico (Figura 123).

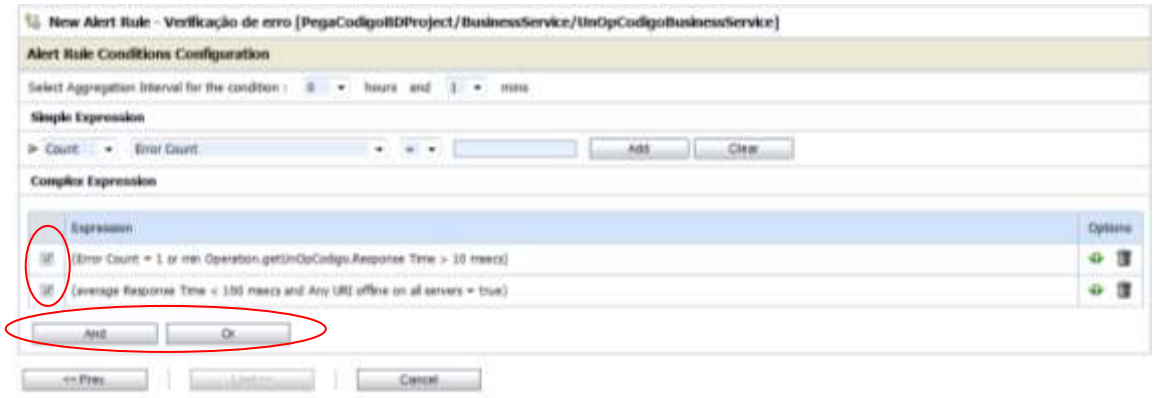

Figura 123 – Composição de expressão utilizando expressões já compostas

7. Ao finalizar a configuração da expressão, clique em "*Last*" (Figura 124).

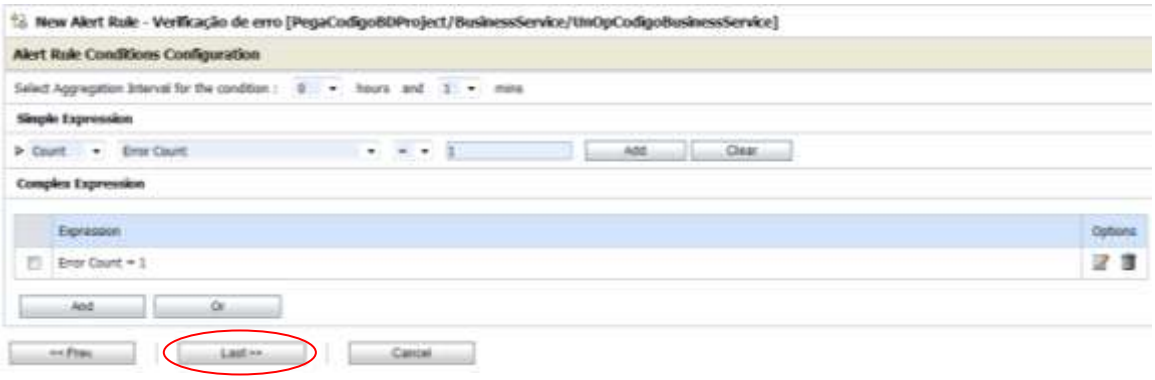

Figura 124 – Janela de configuração da condição de alerta configurada

8. Na próxima janela, que mostra o resumo das configurações realizadas, clique em "*Save*".

Figura 125 – Resumo da configuração do alerta SLA

Ao executar o serviço que obedece a condição para emissão da mensagem de alerta SLA, o Oracle Service Bus Console registra a ocorrência do alerta conforme mostra a (Figura 126), respeitando o tempo de agregação configurado para o aparecimento do registro.

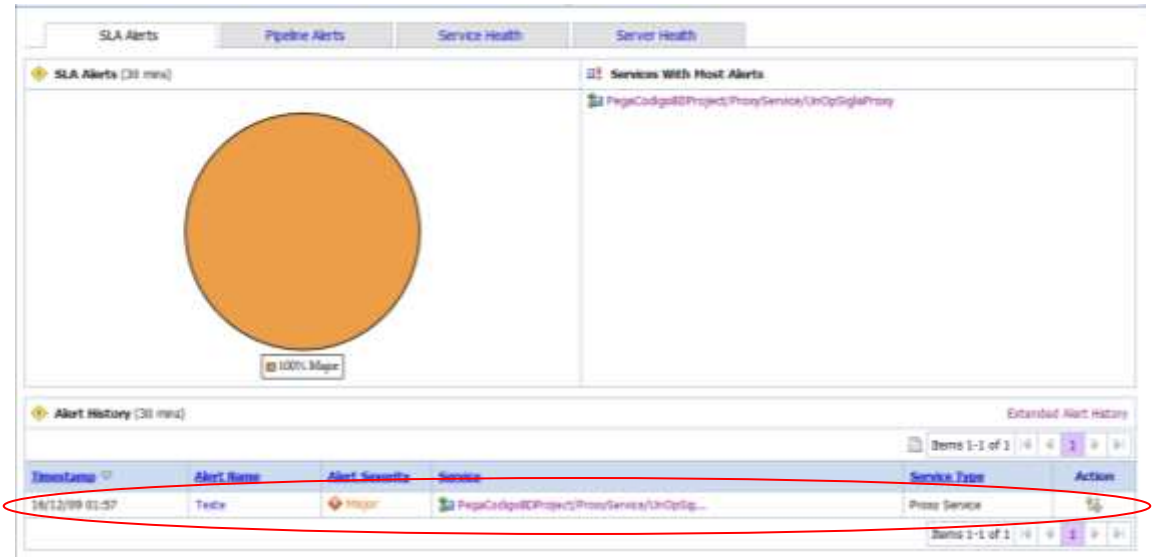

Figura 126 – Registro do alerta de SLA no Oracle Service Console

Observe o gráfico em "pizza" demonstrando a estatística de 100% de mensagens do tipo "*Major*", considerando que só existe 1 mensagem registrada.

Para obter mais informações sobre o registro, clique no link respectivo ao "*Alert Name*". A Figura 127 mostra um exemplo de detalhamento do alerta.

Figura 127 – Detalhamento da mensagem de alerta SLA

4.6.2.3 Configuração de alertas de Pipeline (*Pipeline Alerts*)

O alerta de pipeline pode ser gerado ao configurar uma ação de alerta no fluxo de mensagens de um serviço de Proxy. A mensagem de alerta pode ser usada para detectar erros no fluxo de mensagem e/ou indicar ocorrências do negócio, sendo emitida caso a sua condição de disparo seja satisfeita. Para programar a condição no fluxo, poderá ser usada a linguagem XQuery para acessar valores nas mensagens e os construtores do fluxo de mensagem "if-then-else" para programar as condições. Ainda é possível configurar destinos adicionais (o destino padrão é o OSB) para a emissão de mensagens de alerta de pipeline. O texto de mensagem do alerta é editável, podendo incluir texto e valores de variáveis de contexto. Os alertas gerados pelo serviço e registados pelo OSB podem ser visualizados na página do painel principal no Oracle Service Bus Console.

Para configurar um alerta de pipeline dentro do fluxo de mensagem do OSB, siga os passos a seguir:

1. Ao editar o fluxo de mensagem no serviço de Proxy alvo no OSB, selecione o estágio onde deseja realizar a configuração de um alerta, e então clique em "Edit Stage" (Figura 128).

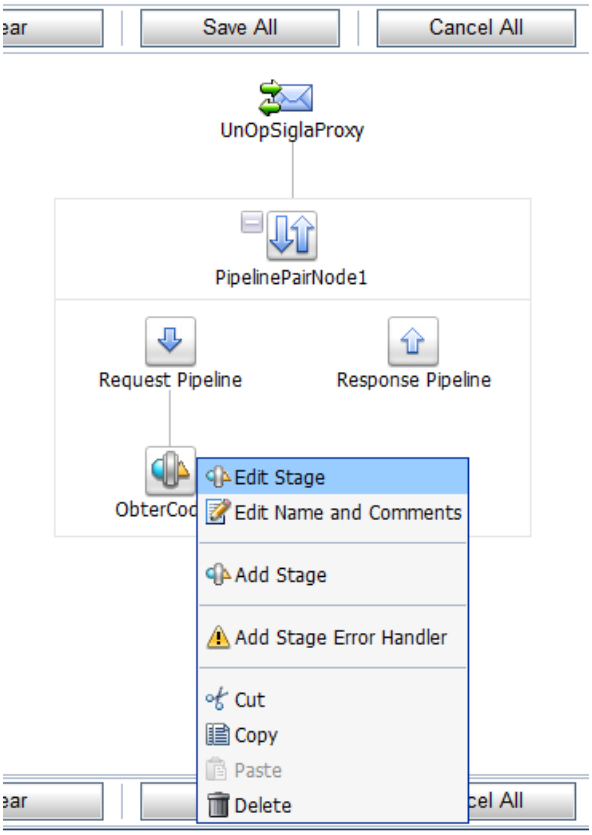

Figura 128 – Editando um estágio no fluxo de mensagem

2. Neste exemplo será mostrada a utilização do alerta utilizando uma condição (se o código da Unidade Operativa for igual a 1). Portanto, na janela de edição do estágio, insira a condição de "IF" e adicione a ação de alerta clicando no dado estágio ou em "Add an Action" e indo em "Add an Action/Reporting/Alert" conforme mostra a Figura 129. Caso não deseje utilizar uma condição, mas apenas configurar a ação de alerta, simplesmente adicione a ação correspondente no local mais adequado dentro do fluxo de mensagem.

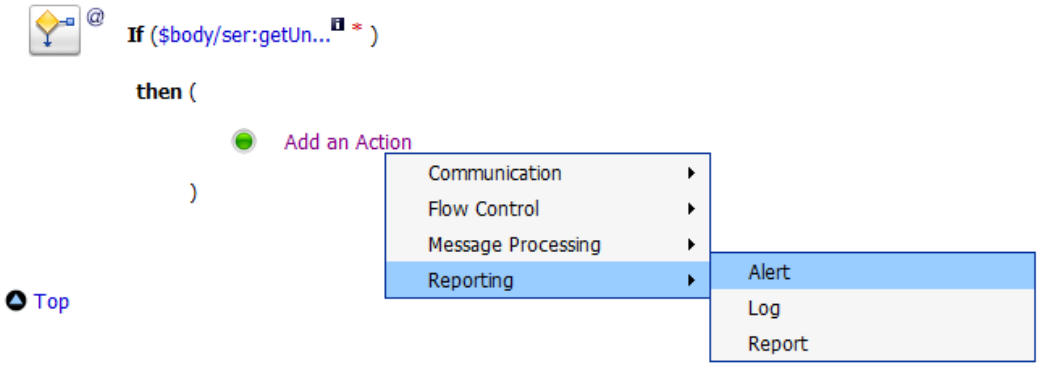

Figura 129 – Adicionando uma ação de "*Alert*"

3. Insira um parâmetro para o "*Alert*" clicando em "*Expression*" e inserindo o respectivo "*Xquery*". Preencha o campo "*Alert-Summary*" com um texto que descreva os motivos do alerta e selecione o nível de gravidade do alerta na caixa de rolagem (Figura 130). É possível configurar múltiplos destinos para a mensagem de alerta. Para isso, é necessário configurar recursos de destino (*Destination).* As informações de como realizar esses procedimentos podem ser encontradas no guia de operações (Operations Guide) do Oracle Service Bus [OSB, 2008b].

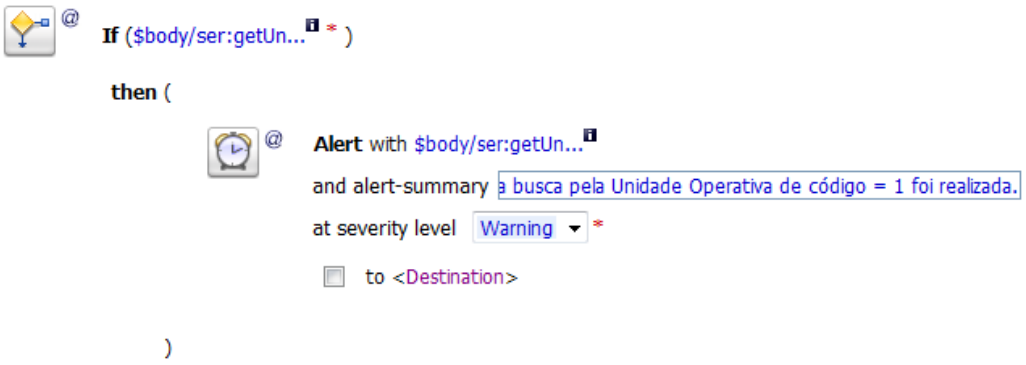

Figura 130 – Configurando uma mensagem de Alerta

Ao executar o serviço que obedece a condição para emissão da mensagem de Alerta (código = 1), o Oracle Service Bus Console registra a ocorrência do alerta conforme mostra a (Figura 131), respeitando o tempo de agregação configurado para o aparecimento do registro.

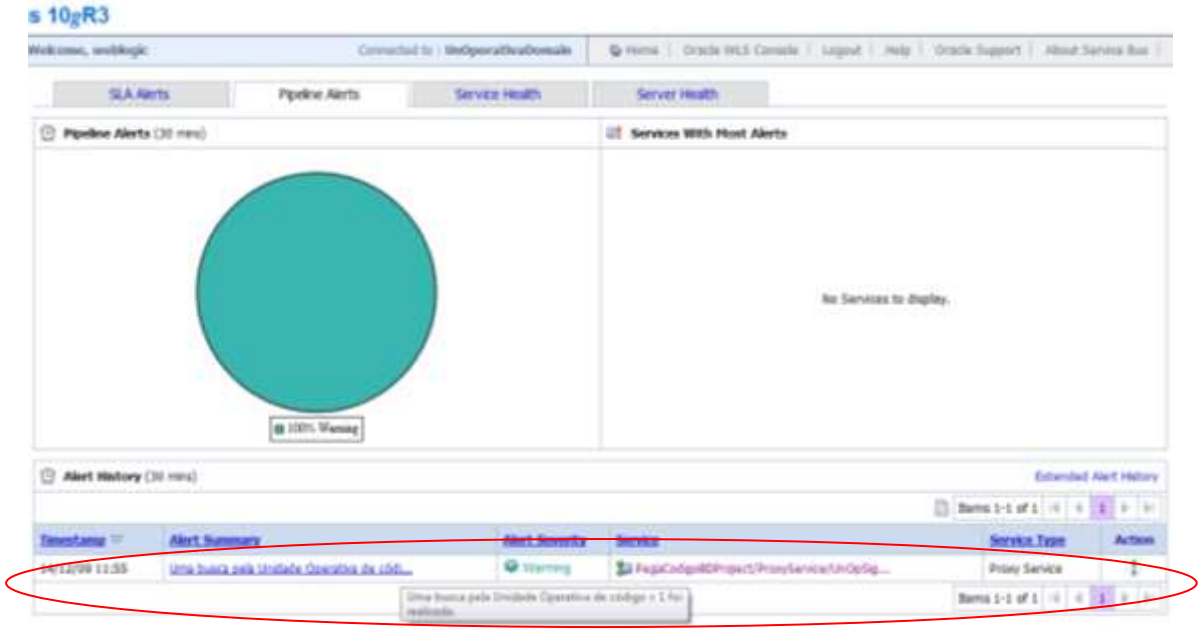

Figura 131 – Registro do alerta de pipeline no Oracle Service Bus Console

Observe o gráfico em "pizza" apresenta a estatística de 100% de mensagens do tipo "*Warning*". Isto ocorreu porque só existe 1 mensagem registrada.

Para obter mais informações sobre o registro, clique no link respectivo ao "*Alert Summary*". A Figura 132 mostra o detalhamento do alerta.

Figura 132 – Detalhamento da mensagem de alerta de pipeline

4.6.2.4 Configuração do *Report*

A ação de *Report* no OSB permite emitir uma mensagem de relatório contendo dados extraídos da mensagem XML que está sendo tratada no fluxo de mensagem. Para poder utilizar a ação de *Report* deverá ser configurado um provedor de relatório (*reporting provider*) para receber as mensagens de relatório.

O OSB entrega os dados da mensagem e dos alertas para um ou mais provedores de relatório. Os dados da mensagem podem ser capturados do "*body*" da mensagem e de outras variáveis associadas com a mensagem, como o "*header*" ou "*inbound variables*".

O OSB oferece o "JMS *reporting provider*" como provedor de relatório padrão. O módulo de relatório (*reporting module*), no OSB console, exibe as informações capturadas pelo provedor de relatório. Caso não se deseje utilizar este provedor, é possível criar um provedor específico utilizando o *"Reporting Service Provider Interface (SPI)"*. Caso seja configurado um provedor próprio, as informações enviadas a ele não serão exibidas no console do OSB, sendo necessário desenvolver uma interface para o usuário. Para obter maiores informações sobre o uso do SPI, consulte o guia de operações (Operations Guide) do Oracle Service Bus [OSB, 2008b].

O provedor padrão JMS *Reporting Provider* é automaticamente configurado quando é criado um domínio no OSB. Todas as mensagens são reunidas e guardadas em um banco de dados do JMS (JMS *Reporting Provider Data Store*) em um formato específico.

Para receber mensagens de relatório do JMS *Reporting Provider*, é necessário criar uma ação de "*Report*" no fluxo de mensagens de um serviço de *Proxy*. A ação de "*Report*" permite extrair informações de cada mensagem e registrar no OSB *Reporting provider*. Para isso, é necessário especificar a informação que se deseja extrair da mensagem e adicionar no OSB *Reporting Data Stream*. Para realizar a configuração de um relatório dentro do fluxo de mensagem do OSB, siga os passos a seguir:

1. Ao editar o fluxo de mensagem no serviço de Proxy alvo no OSB, selecione o estágio onde deseja realizar a configuração de um alerta, e então clique em "Edit Stage" (Figura 128).
2. Adicione a ação no local mais adequado dentro do fluxo de mensagem indo em "Add an Action/Reporting/Report" (Figura 133).

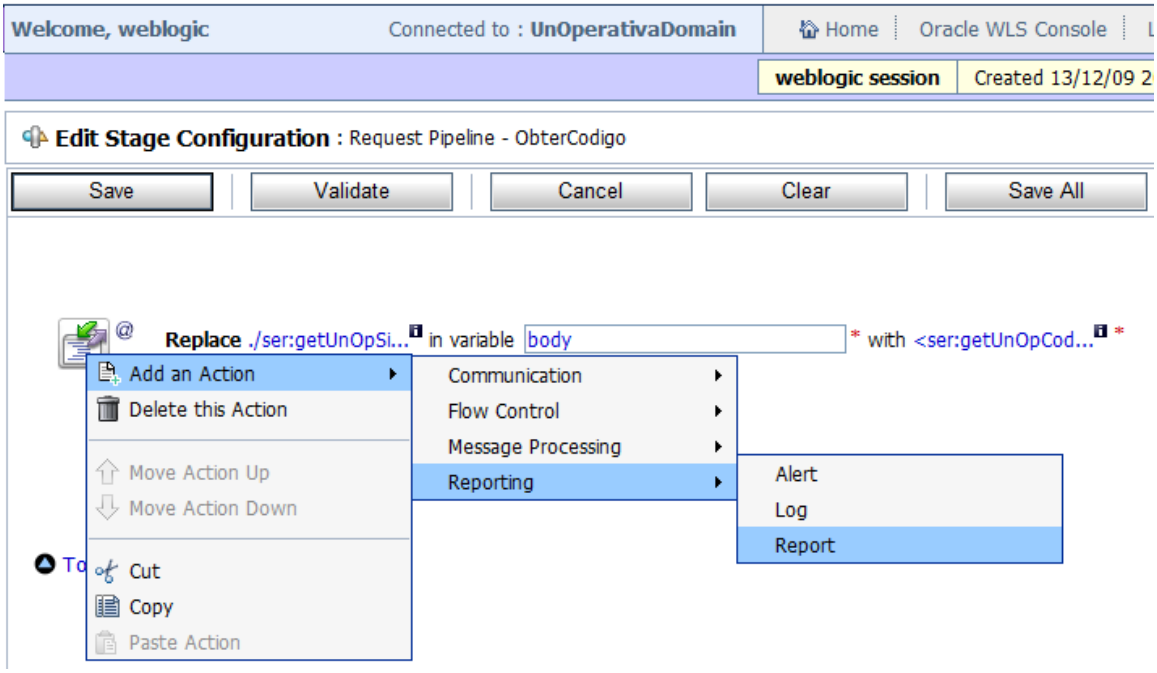

Figura 133 – Adicionando uma ação de "*Report*"

3. Para configurar a ação de "*Report*", clique em <Expression> (Figura 134) para selecionar, utilizando o Xpath, as informações que se deseja incluir na mensagem do relatório. As informações podem ser incluídas não somente a partir da variável "*Body*" da mensagem, mas de qualquer parte da mensagem como, por exemplo, os dados do campo "header". Para este exemplo, foi selecionado todo o conteúdo da variável "*body*" (Figura 135).

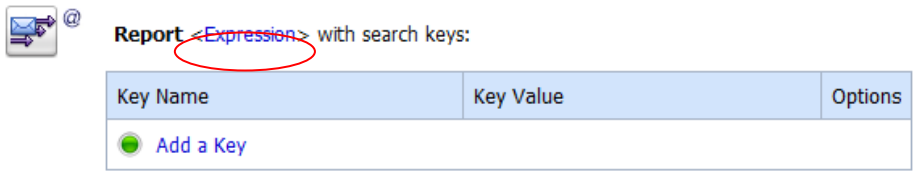

Figura 134 – Link para seleção de partes da mensagem que serão incluidas no *"Report"*

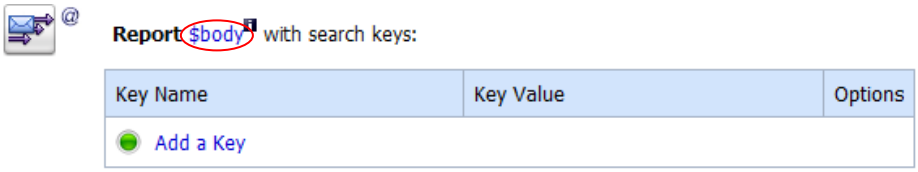

Figura 135 – Conteúdo da variável *"body"* selecionado para constar no relatório

4. O próximo passo é inserir uma chave (*Key Name*) junto com o valor de uma variável da mensagem (*Key Value*) para identificar o relatório. É possível inserir mais de uma chave e seus respectivos valores. Para inserir a chave clique em *"Add key"* (Figura 136).

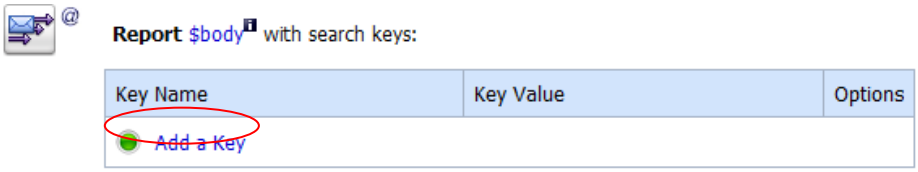

Figura 136 – Adicionando uma chave para o relatório

5. No novo quadro que surgir, insira no campo *"Key Name"* o nome da chave, e no campo *"Key Value"*, o valor da chave a partir do conteúdo da mensagem que está sendo manipulada dentro do estágio, identificando também a variável de onde o valor está sendo extraído (Figura 137).

Figura 137 – Configuração do *"Report"*

6. Para adicionar outra chave, clique no ícone da chave e depois em *"Add Key"*, conforme ilustra a Figura 138:

Figura 138 – Adicionando mais uma chave

Se estiver usando o provedor padrão JMS *Reporting Provider*, as chaves e seus valores associados são mostrados na coluna de índice dos relatórios, na tabela de resumos

das mensagens. Para acessar essa tabela, a partir da tela principal vá em Operations/Message Reports, conforme ilustra a Figura 139.

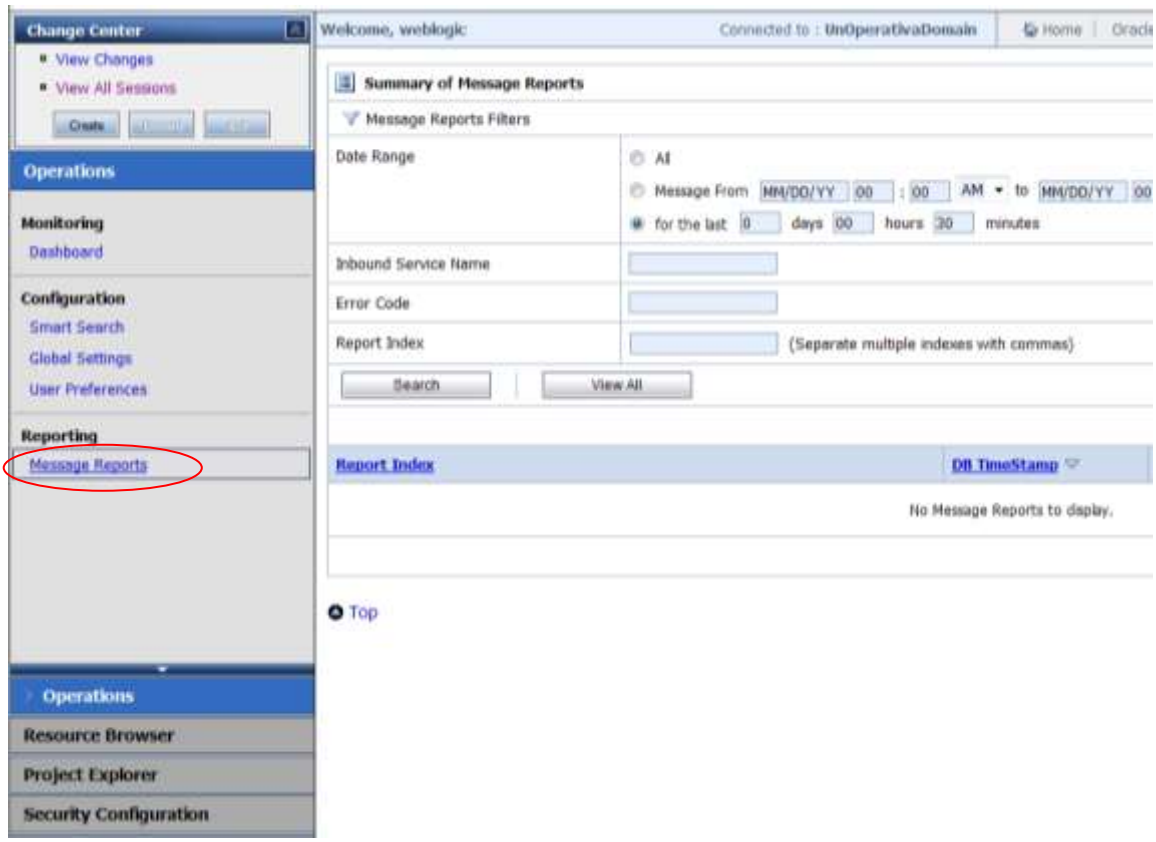

Figura 139 – Tela de resumo das mensagens de relatório

Ao executar o serviço que emite uma mensagem de relatório, a mensagem registrada pelo JMS é exibida na janela de resumo de mensagens de relatório (Figura 140).

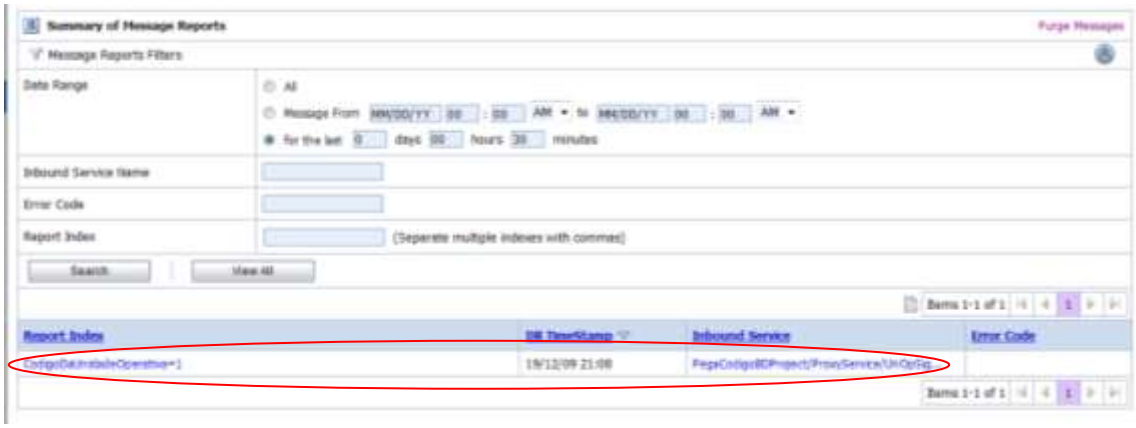

Figura 140 – Registro da mensagem de relatório

No campo *"Report Index"* encontra-se a chave e o valor da chave configurado na ação de *"Report"*. O campo *"DB TimeStamp"* mostra a data e o horário da ocorrência da mensagem. O campo *"Inbound Service"* mostra o caminho do serviço de *Proxy* que originou a mensagem e o campo Error Code somente é preenchido quando a mensagem de relatório for configurada a partir de um estágio do tipo *"Error Handller"*, onde o código do erro é associado à mensagem de relatório.

Para acessar os detalhes da mensagem de report, clique na respectiva chave da mensagem dentro da lista de mensagens de relatório, na coluna *"Report Index"* (Figura 141).

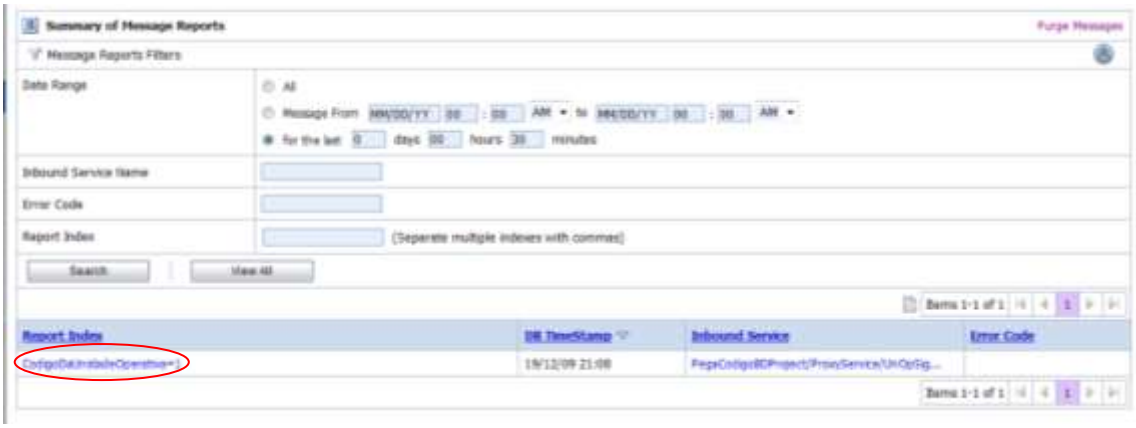

Figura 141 – Link para o detalhamento da mensagem de relatório

A Figura 142 ilustra o conteúdo da mensagem de relatório. Clicando em *"Detail"*, o OSB exibirá o conteúdo programado na ação de *"Report"*. Neste exemplo foi configurada a ação para extrair o conteúdo do *"body"* da mensagem (Figura 143).

Figura 142 – Detalhamento da mensagem de relatório

```
- <ser:getUnOpCodigo xmlns:ser="http://br/com/petrobras/services"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <ser:codigo>1</ser:codigo>
  </ser:getUnOpCodigo>
```

Figura 143 – Conteúdo extraído da mensagem para compor o relatório

4.6.2.5 Resumo de Logs

O OSB possui uma seção de resumo de log (Log Summary) que mostra o resumo das mensagens de log dos servidores associados ao domínio. O arquivo de log do domínio oferece uma localização central dos logs dos servidores associados ao domínio, no qual é possível verificar o estado geral do domínio. Cada instância de servidor envia um subconjunto de suas mensagens para um arquivo de log do domínio.

Por padrão, os servidores emitem somente algumas mensagens de nível de gravidade maior ou igual a "Notice" e não é possível modificar esse conjunto de mensagens que são enviadas. Se for configurada uma ação de *log* no *pipeline,* a mensagem é enviada para o servidor que administra os *logs* (*Admin server log*). É possível visualizar a mensagem de log na sessão *Server Health* do OSB (Figura 144).

Para acessar as mensagens de log no seção *"Log Summary",* vá em "Operations/Dashboard/Server Health".

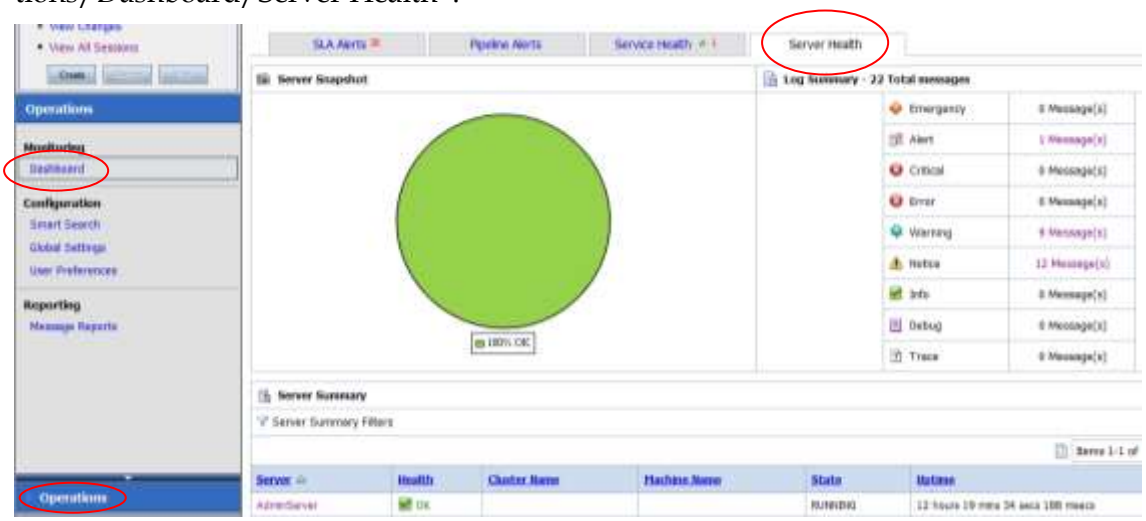

Figura 144 – Seção de resumo de *log*

Somente os usuários que possuírem o privilégio de administrador poderão visualizar o resumo de log. A Tabela 3 descreve os níveis de gravidade que agrupam as mensagens no resumo de *log.*

Tabela 3 – Descrição dos níveis de gravidade no resumo de *log*

Nível de gravidade	Descrição
Alert	Este nível indica que um serviço particular está em um estado instável enquanto outras partes do sistema continuam funcionando normalmente. A recuperação automática não é possível, logo a atenção imediata do administrador é necessária para resolver o problema.
Critical	Este nível indica que o sistema ou erros de serviços ocorreram. O sistema pode recuperar-se, mas podem existir perda momentânea ou permanente degradação do serviço.

Emergency	Este nível indica que o servidor está em um estado em que ele não pode ser usado. Isso indica que ocorreu uma falha severa no sistema.
Error	Este nível indica que um erro de usuário ocorreu. O sistema ou aplicação pode manipular este tipo de erro sem interrupção. Degradações limitadas no serviço podem ocorrer.
Info	Este nível reporta operações normais com uma mensagem informativa.
Notice	Esta é uma mensagem informativa com um alto nível de importância em relação ao nível *info*.
Warning	Este nível indica que uma operação suspeita ou configuração foi realizada. No entanto, as operações normais não são afetadas.

4.6.2.5.1 Configuração do Log

Para configurar uma ação de log que será registrada no "Log summary", siga os passos abaixo:

1. Ao editar um estágio (*stage*) em um fluxo de mensagens, adicione a ação indo em "Reporting/Log" (Figura 145).

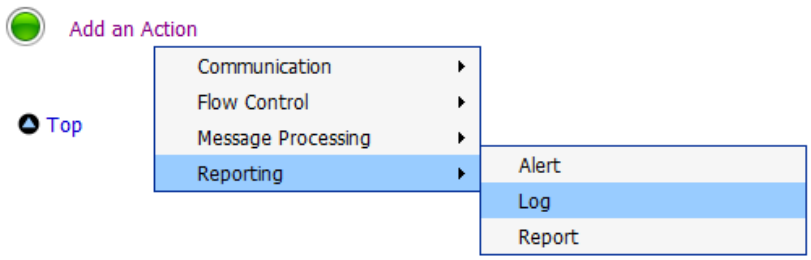

Figura 145 – Inserindo uma ação de *log*

2. Clique no *link* <Expression> para selecionar o conteúdo (utilizando o XPath) da mensagem o qual deseja anexar ao registro do log (Figura 146).

Figura 146 – Anexando o conteúdo da mensagem no registro de *log*

3. Insira uma anotação no campo de texto para constar no *log* e escolha um nível de gravidade para o este registro de *log* (Figura 147). Salve as alterações e ative a sessão.

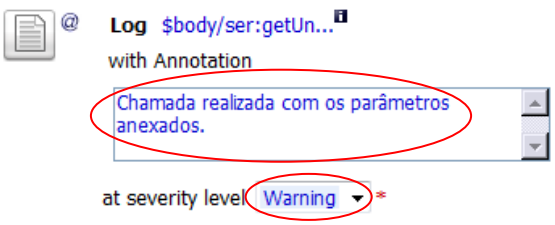

Figura 147 – Registrando anotação e nível de gravidade no registro de *log*

Ao executar o respectivo serviço onde foi configurado o *log*, o registro é inserido no resumo de *log (Log Summary),* respeitando o tempo de agregação. Para visualizar o registro do *log,* entre na seção de resumo de *log* e clique no respectivo nível de gravidade para o qual o log foi configurado (Figura 148). As mensagens estão agrupadas pelo nível de gravidade e cada linha da tabela mostra o número de *logs* registrados.

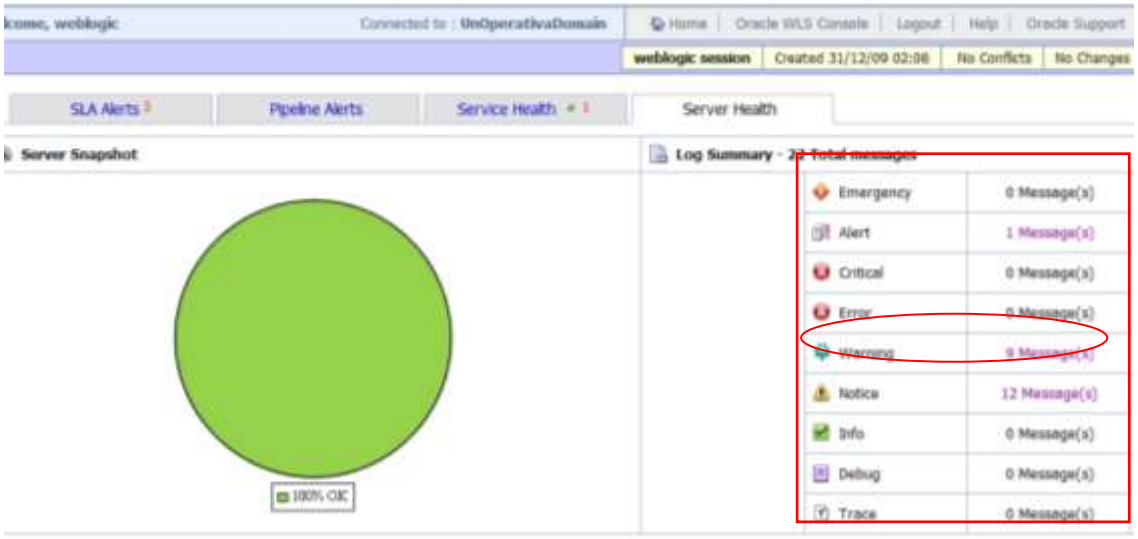

Figura 148 – Mensagens no resumo de *Log*

A próxima janela mostra a lista de *logs* que estão registrados detalhando alguns campos como data, subsistema de origem, nível de gravidade, identificação da mensagem e a própria mensagem (Figura 149).

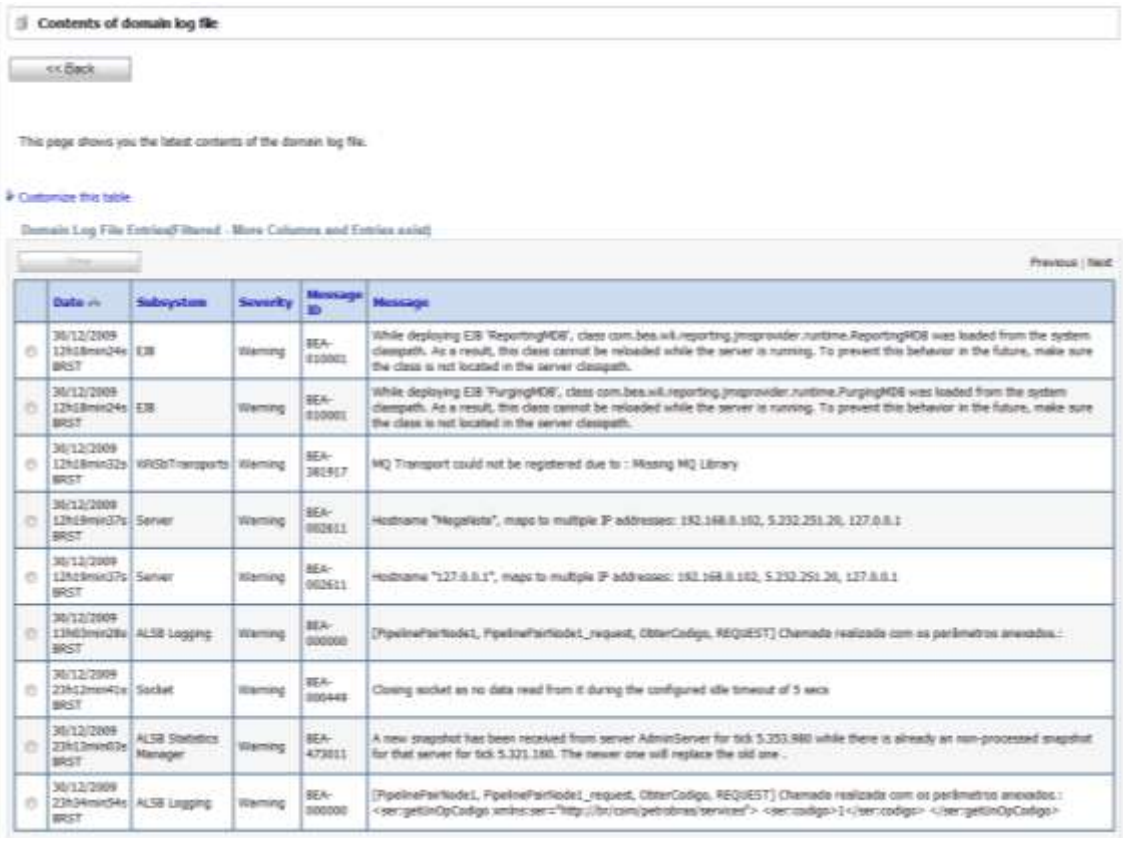

Figura 149 – Lista de registros de *log*

Para acessar detalhes do *log*, selecione o registro e clique em *"View"* (Figura 150).

Figura 150 – Selecionando o registro de *log* para acessar seu detalhamento

A janela seguinte mostra o conteúdo do detalhamento da mensagem (Figura 151).

This page shows you an entry from the domain log file.

Message:	[PipelinePairNode1, PipelinePairNode1_request, ObterCodigo, REQUEST] Chamada realizada com os parâmetros anexados.: <ser:getUnOpCodigo xmlns:ser="http://br/com/petrobras/services"> <ser:codigo>1</ser:codigo> </ser:getUnOpCodigo>
Date:	30/12/2009 23h34min54s BRST
Subsystem:	ALSB Logging
Message ID:	BEA-000000
Severity:	Warning
Machine:	MegaNote
Server:	AdminServer
Thread:	[ACTIVE] ExecuteThread: '9' for queue: 'weblogic.kernel.Default (self-tuning)'
User ID:	<anonymous>
Transaction ID:	(No value specified)
Context ID:	(No value specified)
Detail:	(No value specified)
Cause:	(No value specified)
Action:	(No value specified)

Figura 151 – Detalhamento do registro de *log*

4.6.2.5.2 Visualização de detalhes do servidor

É possível acessar a página de detalhes do servidor clicando no nome do servidor abaixo do resumo do servidor (*Server Summary)* (Figura 152).

Figura 152 – Acesso a visualização de detalhes do servidor

As informações da próxima página (Figura 153) são mostradas em um subconjunto de abas de monitoramento do OSB. A Tabela 4 descreve as informações disponíveis.

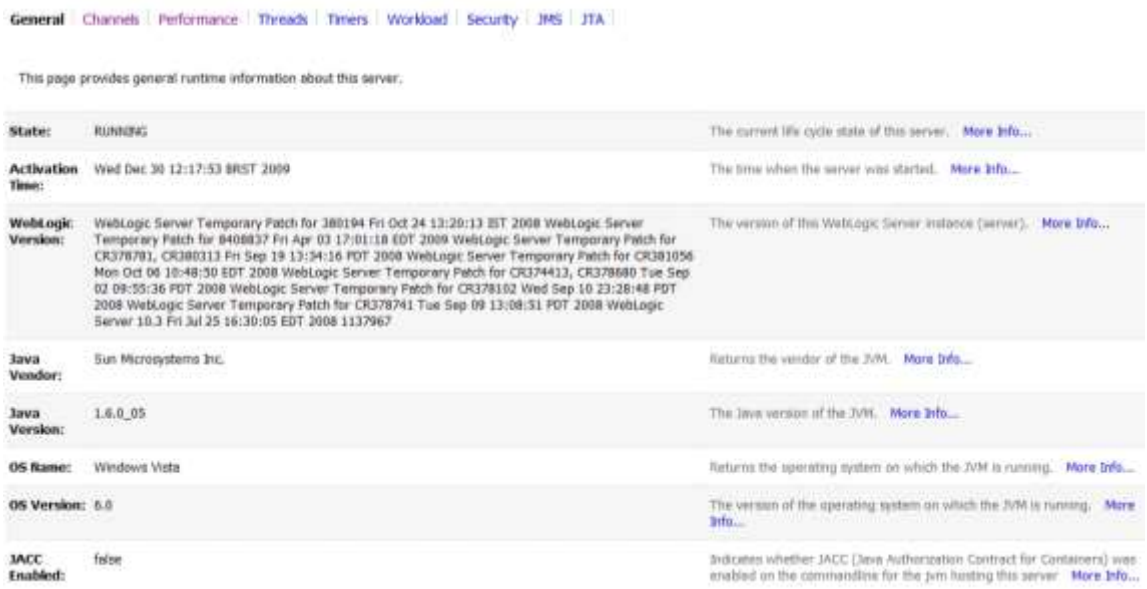

Figura 153 – Janelas de informações sobre o servidor

Tabela 4 – Detalhamento das informações sobre o servidor

Aba	Descrição
General	Esta aba oferece informações gerais em tempo de execução sobre o servidor.
Channels	Esta aba oferece informações de monitoramento sobre os canais.
Performance	Esta aba oferece informações sobre o desempenho do servidor.
Threads	Esta aba oferece características em tempo de execução e estatísticas das filas que estão sendo executadas no servidor no momento.

Timers	Esta aba oferece informações sobre o temporizador utilizado pelo servidor.
Workload	Esta aba oferece estatísticas de "*Work managers*", restrições e políticas configuradas no servidor.
Security	Esta aba permite o monitoramento de estatísticas dos bloqueios de acesso não autorizado a usuários, realizados no servidor.
JMS	Esta aba permite o monitoramento das informações sobre JMS no servidor.
JTA	Esta aba permite um resumo de toda informação de transação para todos os tipos de recursos no servidor.

4.6.2.6 Configuração *Tracing*

O OSB permite rastrear mensagens sem ter que desligar o servidor. Essa funcionalidade permite descobrir problemas e diagnosticar fluxos de mensagens em um ou mais serviços de *proxy*.

Por exemplo, se um serviço de *proxy* estiver falhando e o usuário desejar descobrir em qual estágio (*stage*) o problema se localiza, é possível habilitar o rastreamento para um serviço de Proxy. Após habilitar esta opção, o sistema registra vários detalhes extraídos do fluxo de mensagem como nome do estágio, nome do *pipeline* e nome do nó da rota.

Todo o contexto das mensagens também é registrado, incluindo o conteúdo dos "*Headers*" e "*Message body*". Quando algum problema ocorre no fluxo de mensagens, detalhes adicionais como o erro do código e a razão do problema também são registrados. O rastreamento ocorre no início e fim de cada componente no fluxo de mensagem, incluindo estágios, *pipelines* e nós. As ações (*Action*) não são rastreadas individualmente.

Para habilitar a função de rastreamento (*Tracing)* siga os passos abaixo:

1. Identifique o serviço de negócio e clique nele, conforme mostra a Figura 154, para acessar suas opções.

Figura 154 – Serviços de proxy para configuração do *Tracing*

2. Vá na guia "*Operational Settings*".

Nesta guia se encontram as opções "*Execution Tracing*" e "*Message Tracing*". Habilitando a opção "*Execution Tracing*", o OSB registrará no log diversos detalhes selecionados a partir do contexto do fluxo de mensagem (*message flow context)* e do contexto da mensagem (*message context*). Estes detalhes incluem: nome do estágio (*stage*), nome do *pipeline* ou nó da rota e contexto da mensagem corrente.

Habilitando a opção "*Message Tracing*", o OSB registrará no log as trocas de mensagens entre o fluxo de pipeline e o serviço de proxy (requisições de entrada e resposta à requisição de entrada, bem como requisições de saída e resposta da requisição de saída), ou seja, no momento em que a mensagem entra no pipeline para ser processada (requisição de entrada), quando ela sai do fluxo para ser enviada para o serviço de ne-

gócio (requisição de saída), quando ela retorna do serviço de negócio para o serviço de proxy (resposta de saída) e quando a resposta é enviada para o cliente (resposta da entrada). Quando aplicável, os registros de log de mensagens de saída podem incluir o número de tentativas, código de erro e mensagem de erro.

Ao habilitar a opção "*Message Tracing*", deverá ser configurado o parâmetro "*Detail Level*", que pode ser definido como "*Tarse*", "*Headers*" e "*Full*". Essa opção especifica o nível de detalhe do registro no log.

Selecionando a opção "*Tarse*", será registrado a data, hora, tipo do serviço, nome do serviço e o URI.

Selecionando a opção "*Headers*", serão registradas as informações da opção "*Tarse*" juntamente com a representação em XML dos metadados mensagem.

Selecionando a opção "*Full*", serão registradas as informações da opção "*Tarse*" juntamente com conteúdo da mensagem e os anexos, se houver. Ao escolher esta opção, deverá ser especificado o valor máximo do tamanho (em *Kilobytes*) da mensagem que será registrada no log, preenchendo o campo "*Payload Tracing Limit*". O campo "*Default Enconding*" é opicional. Deixando-o em branco, o OSB usará o codificador padrão da estação (*host)*. Se o valor especificado não puder ser usado (por exemplo, o valor não é uma opção de configuração válida), então o OSB utilizará o codificador Base64 no conteúdo que será registrado.

3. Habilite as opções desejadas nos campo "*Execution tracing*"e "*Message tracing*". Neste teste ambas as opções foram habilitadas, conforme mostra a Figura 155.

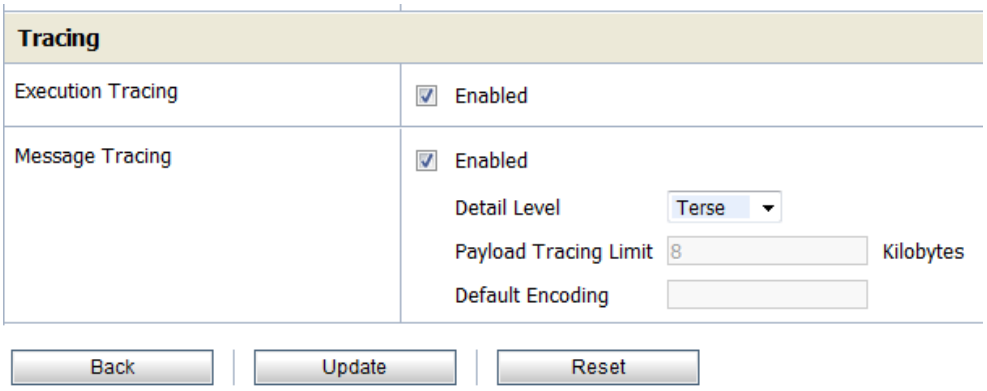

Figura 155 – Habilitando o *Trancing* de serviços no OSB

4. Configure o campo "*Detail Level*" como "*Full*" e defina um valor máximo em *kilobytes* para o tamanho do conteúdo gerado pelo rastreamento. O campo "*Default Encoding*" não é obrigatório. Após a configuração, clique em "Update" e ative a seção (Figura 156).

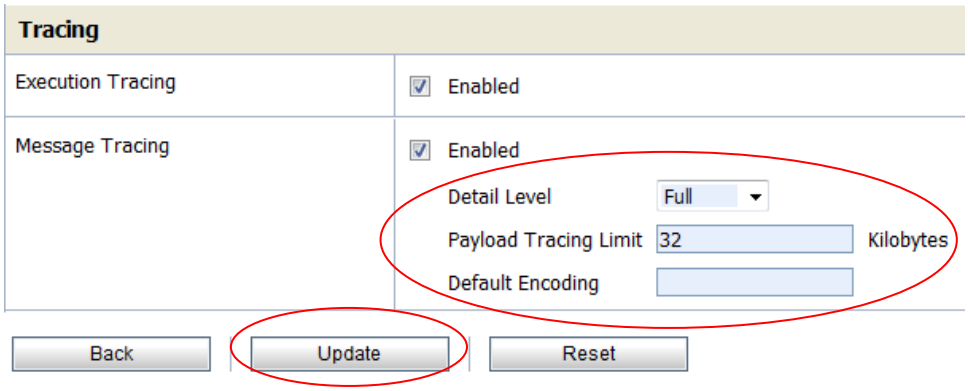

Figura 156 – Definindo a configuração do *Tracing* no OSB

O conteúdo gerado pelo "*Tracing*" é armazenado no diretório de log do servidor. Por exemplo, no domínio criado como exemplo, que pode ser instalado juntamente com o OSB, o caminho para o conteúdo gerado no arquivo de log é: <*OSB_HOME*>\samples\domains\servicebus\servers\xbusServer\logs\servicebus.log. OSB_HOME é o diretório no qual foi instalado o OSB. A Figura 157 mostra, como exemplo, o conteúdo do arquivo de log gerado pelo "*Tracing*".

```
####<24/12/2009 00h37min52s BRST> <Info> <Log Management> <MegaNote> <AdminServer> <[ACTIVE] ExecuteThre
Kernel>> <> <> <1261622272403> <BEA-170025> <Initialized Domain Logging. Domain log events will be writt
D:\bea\user_projects\domains\UnOperativaDomain\servers\AdminServer\logs/UnOperativaDomain.log.>
####<24/12/2009 00h37min52s BRST> <Notice> <Log Management> <MegaNote> <AdminServer> <[ACTIVE] ExecuteTh
Kernel>> <> <> <1261622272406> <BEA-170027> <The Server has established connection with the Domain level
####<24/12/2009 00h37min52s BRST> <Info> <Diagnostics> <MegaNote> <AdminServer> <[ACTIVE] ExecuteThread:
<> <> <1261622272583> <BEA-320077> <Initialized the Diagnostic Accessor Service.>
####<24/12/2009 00h37min52s BRST> <Info> <WebLogicServer> <MegaNote> <AdminServer> <[ACTIVE] ExecuteThre
Kernel>> <> <> <1261622272594> <BEA-000256> <Invoking com.bea.wli.sb.transports.JCATransportPostActivati
####<24/12/2009 00h37min52s BRST> <Info> <JMX> <MegaNote> <AdminServer> <[ACTIVE] ExecuteThread: '0' for
<1261622272667> <BEA-149512> <JMX Connector Server started at service:jmx:iiop://192.168.0.102:7001/jndi
####<24/12/2009 00h37min52s BRST> <Notice> <WebLogicServer> <MegaNote> <AdminServer> <main> <<WLS Kernel
####<24/12/2009 00h37min52s BRST> <Notice> <WebLogicServer> <MegaNote> <AdminServer> <main> <<WLS Kernel
####<24/12/2009 00h37min52s BRST> <Info> <netuix> <MegaNote> <AdminServer> <[ACTIVE] ExecuteThread: '1'
<1261622272830> <BEA-423101> <[sbconsole] Initializing the NetUIx container>
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272896> <BEA-149059> <Module Dummy of application ALSB Cluster Singleton Marker Application i
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272896> <BEA-149060> <Module Dummy of application ALSB Cluster Singleton Marker Application s
AdminServer.>
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272897> <BEA-149059> <Module Dummy of application ALSB Domain Singleton Marker Application is
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272897> <BEA-149060> <Module Dummy of application ALSB Domain Singleton Marker Application su
AdminServer.>
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272897> <BEA-149059> <Module Dummy of application ALSB Framework Starter Application is trans
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272897> <BEA-149060> <Module Dummy of application ALSB Framework Starter Application successf
AdminServer.>
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272898> <BEA-149059> <Module xbus_dynamic of application XBus Kernel is transitioning from ST
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272898> <BEA-149060> <Module xbus_dynamic of application XBus Kernel successfully transitione
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272898> <BEA-149059> <Module /httpTransport of application XBus Kernel is transitioning from
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272898> <BEA-149060> <Module /httpTransport of application XBus Kernel successfully transitio
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
<> <1261622272899> <BEA-149059> <Module Servidor of application _auto_generated_ear_ is transitioning fr
####<24/12/2009 00h37min52s BRST> <Info> <Deployer> <MegaNote> <AdminServer> <[STANDBY] ExecuteThread: '
```

Figura 157 – Exemplo do conteúdo do arquivo de log

4.6.2.7 Configuração do *Throttling*

O Oracle Service Bus oferece opções de configuração de concorrência de mensagens. As restrições impostas ao fluxo de mensagens para um serviço de negócio são conhecidas como "*Throttling*".

Para configurar o "*Throttling*" para um serviço de negócio, é necessário habilitar a função nas configurações operacionais do serviço de negócio (*Operational Settings*). Para isso, marque a opção "*Throttling State*" como "*Enabled*" (Figura 158). Observe que é necessário possuir uma seção ativa.

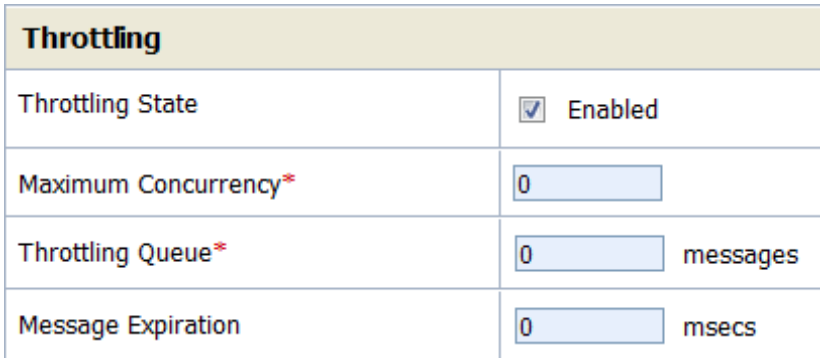

Figura 158 – Habilitando a opção de *Throttling*

Após habilitar o "*Throttling*" para um serviço de negócio, é necessário especificar um valor para o campo "*Maximum Concurrency*". Também podem ser definidos valores para os campos "*Throttling Queue*" e "*Message Expiration*". Abaixo se encontra o detalhamento de cada um dos parâmetros de configuração do "*Throttling*":

- **Maximum Concurrency:** Restringe o número de mensagens que podem ser processadas concorrentemente por um serviço de negócio. O valor deve ser um inteiro positivo. Quando o valor for alcançado por um serviço de negócio, todas as mensagens de chegada para o serviço de negócio serão colocadas em uma fila (*Throttling queue*) até o serviço de negócio poder aceitar mais mensagens.

  Qualquer mudança neste parâmetro afetará tanto as novas mensagens quanto aquelas que estiverem na fila (*throttling queue*). Quando o valor é aumentado, o OSB permite que mais mensagens sejam enviadas para o serviço de negócio após processar aquelas que estavam na fila antes. Quando o valor é reduzido, o OSB põe qualquer mensagem nova em uma fila até o valor definido para que a concorrência fique abaixo do configurado, se estiver definido um valor para a fila (*throttling queue*). Caso não tenha definido um valor para a fila (*throttling queue*), a mensagem será descartada.

  Em um ambiente de *cluster*, o número de mensagens que podem ser processadas concorrentemente pelos serviços de negócio é dividido igualmente entre os servidores gerenciados.

- **Throttling queue:** É uma fila de prioridades na qual as mensagens são enquadradas quando um serviço de negócio alcança seu número máximo de concorrência. Mensagens com prioridade maior são processadas primeiro. Mensagens são processadas baseadas na regra "*First in First out*". Somente uma fila pode ser configurada para um serviço de negócio. Um "*throttling queue*" é uma fila gravada na memória. Mensagens que são colocadas na fila não são recuperáveis se o servidor falhar ou for reiniciado. Quando se apaga ou renomeia um serviço de negócio, todas as mensagens no "*throttling queue*" são descartadas.

  O parâmetro "*throttling queue*" restringe o número de mensagens na fila ("*throttling queue*"). O tamanho da fila deve ser um inteiro positivo. Todas as mensagens de chegada acima do máximo do limite de concorrência para os serviços de negócio são colocadas na fila ("*throttling queue*"). Quando a fila está cheia, a mensagem integrante da fila com o mais baixo valor de prioridade será removida da fila e a nova mensagen que possui um nível de prioridade maior será inserida.

Se seu tamanho estiver configurado como zero, nenhuma fila será criada para o serviço de negócio. Qualquer mudança neste parâmetro é automaticamente aplicada. Quando se reduz o valor deste parâmetro, todas as mensagens acima do novo valor são descartadas. Em um ambiente de "*Cluster*", as mensagens são divididas igualmente entre os servidores gerenciados.

- **Message Expiration:** Restringe o tempo máximo (em milisegundos) em que uma mensagem pode permanecer na fila ("*throttling queue*") de um serviço de negócio. O valor para este parâmetro deve ser um inteiro positivo. Quando o tempo configurado terminar, a mensagem é removida da fila. Estas mensagens são referenciadas como "mensagens expiradas".

  Se o parâmetro configurado por zero, as mensagens na fila não irão expirar. Caso aumente o valor do parâmetro configurado, o tempo de expiração para novas mensagens e para as mensagens que já estão presentes na fila serão aumentados. Caso reduza o valor do parâmetro configurado, todas as mensagens que excedem o novo valor serão imediatamente descartadas.

Após configurar o "*Throttling*", clique em "Update" e depois ative a seção (Figura 159).

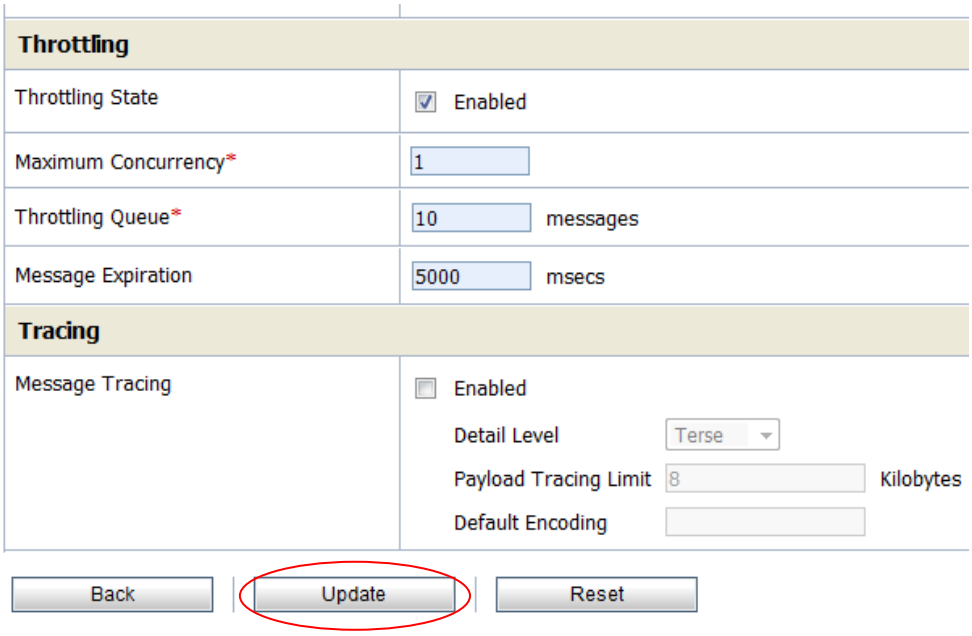

Figura 159 – Salvando a configuração do *"Throttling"*

4.6.2.7.1 Métricas disponíveis para "*Throttling*"

Algumas métricas são disponibilizadas ao habilitar a função de "Throttling". Elas podem ser visualizadas na janela de detalhamento do monitoramento do serviço que é acessada indo em "operations/dashboard/service health", e então clicando no serviço de negócio desejado (Figura 160 e Figura 161).

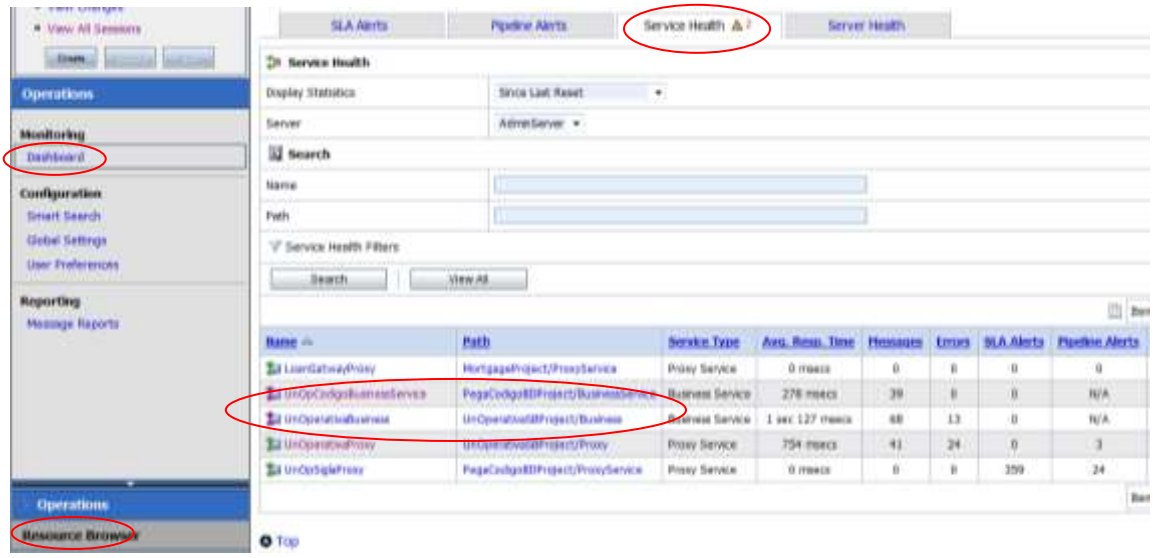

Figura 160 – Acessando métricas para *Throttling*

General	
SLA Alert Count	0
Min Response Time	60 msecs
Max Response Time	2 secs 660 msecs
Overall Avg. Response Time	294 msecs
Message Count	34
Error Count	0
Failover Count	0
Success Ratio	100%
Failure Ratio	0%
WS Security Errors	0
Validation Errors	N/A
Throttling	
Min Throttling Time	0 msecs
Max Throttling Time	198 msecs
Average Throttling Time	5 msecs

Figura 161 – Métricas do serviço de negócio

4.6.2.8 Gerenciamento de *Endpoint* URI

Um *"Endpoint URI"* é uma URL de um serviço externo que é acessado pelo serviço de negócio. No OSB, é necessário definir no mínimo um *"Endpoint URI"* para um serviço de negócio. Quando se define múltiplos *"endpoints URI"* para um serviço de negócio, é necessário configurar um algoritmo de balanceamento de carga entre as seguintes opções: *"Round robin", "Random", "Random-weighted"* ou *"none"*.

Os algoritmos de balanceamento controlam a forma na qual os serviços de negócio acessarão os "*Endpoints URI*". Um "*Endpoint URI*" pode permanecer *online* ou *offline*. O OSB permite configurar o número de tentativas de acesso a um "*Endpoint URI*" que se encontra *offline*. Para realizar essa configuração, siga os passos abaixo:

1. Na janela de configuração do serviço de negócio, clique em "*Edit*" (Figura 162).

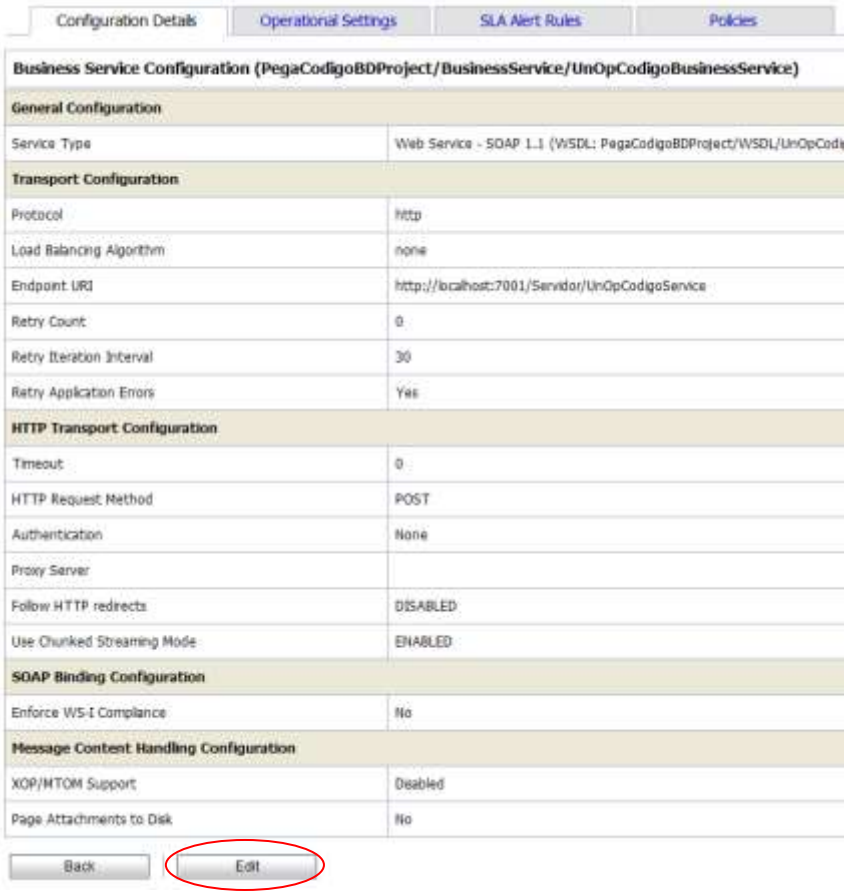

Figura 162 – Janela de configuração do serviço de negócio

2. Clique em "***Next***" na próxima janela (Figura 163).

Figura 163 – Janela de configuração geral do Serviço de negócio

3. Na opção "*Retry Count*" (Figura 164) deve-se especificar o número máximo de tentativas que o serviço de negócio deverá executar para acessar o "*Endpoint URI*" após alguma falha inicial. Na opção "*Retry Iteration Interval*", configure o intervalo, em segundos, entre as tentativas de acesso ao "*Endpoints URI*". Nesta janela ainda é possível escolher um algoritmo de balanceamento de carga, selecionando uma opção em "*Load Balancing Algorithm*". Caso deseje desabilitar as tentativas de chamada ao "*Endpoint URI*" em ca-

so de erro na aplicação, marque "*no*" na opção "*Retry Applications Errors*". Ao finalizar a configuração, clique em "*Last*" e, na próxima janela, em "*Save*". Ative a sessão para aplicar as configurações.

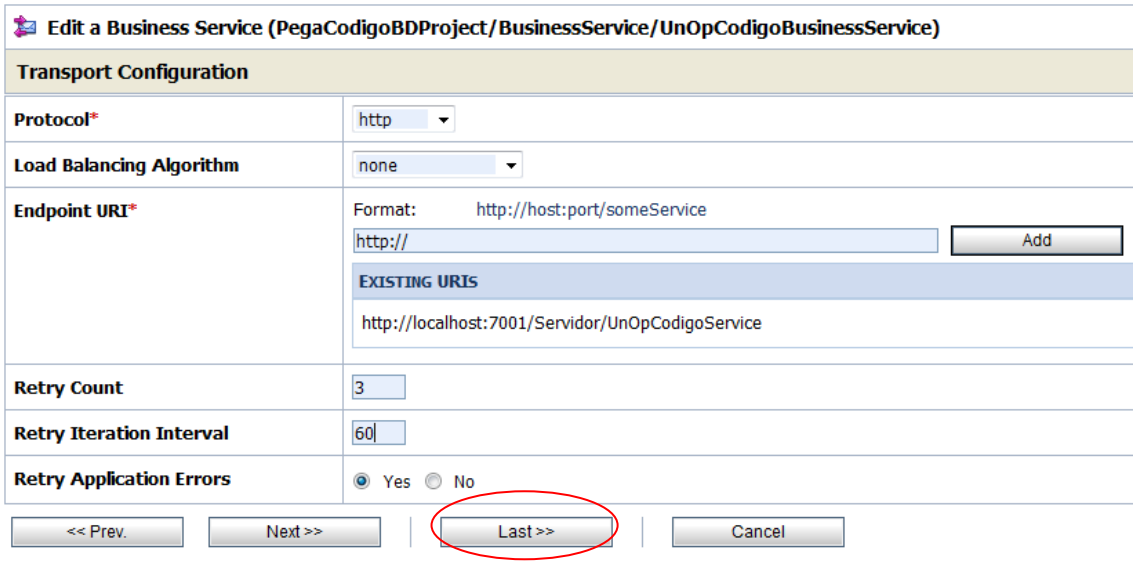

Figura 164 – Janela de edição da configuração do serviço de negócio

Também é possível configurar "*Endpoints URI*" com o estado de *offline* permanentemente ou temporariamente em um serviço de negócio. Um erro de comunicação ocorre cada vez que um serviço de negócio tenta acessar um "*Endpoint URI*" que não responde. Configurando um "*Endpoint URI*" como *offline*, é possível prevenir que um serviço de negócio continue tentando acessá-lo caso ele não responda alguma vez e, por conseguinte, não emitirá erros de comunicação.

Para configurar um "*Endpoint URI*" como "*offline*" permanentemente ou termporariamente em um serviço de negócio, siga os passos abaixo:

1. Na janela de configurações operacionais do serviço de negócio (Figura 165), marque a opção "*Enable With Retry Interval*"e insira o intervalo de tempo. Quando esta opção é selecionada, o serviço de negócio remove os "*Endpoints URI*" que não respondem, colocando-os "*offline*", em tempo de execução, durante o intervalo tempo configurado. Para tornar os "*Endpoints*" permanentemente *offline,* configure o intervalo de tempo como 0 hora, 0 minuto e 0 segundo. Logo, somente as URIs que respondem são usadas para novas tentativas de acesso e para processamento das requisições subseqüentes. Neste caso, para habilitar novamente as URIs marcadas como "*offline*", é necessário fazer isso manualmente. Esta opção é útil quando se deseja verificar o motivo do "*Endpoint*" ficar *offline* por um tempo.

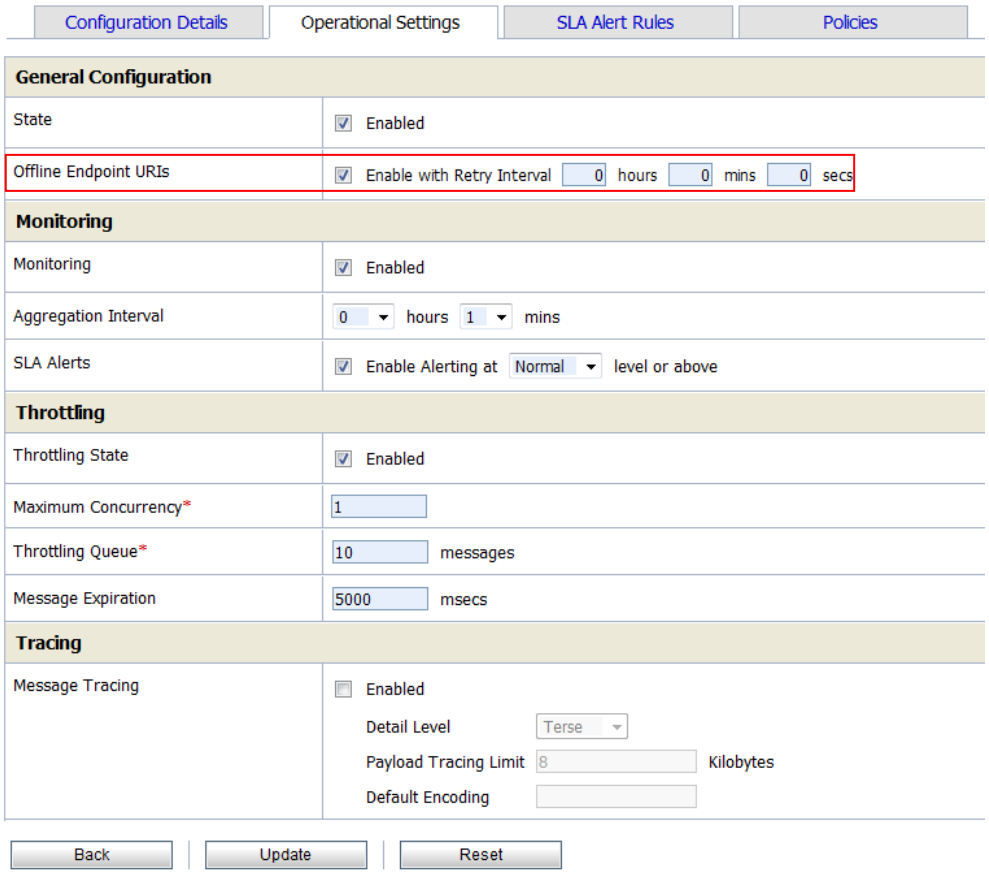

Figura 165 – Janela de configurações operacionais

4.7 Conclusão

O monitoramento de serviço do OSB oferece um conjunto de funcionalidades que permitem ao administrador configurar ações que emitem avisos sobre diversas variáveis relativas às mensagens dos serviços e a saúde do servidor. Essas mensagens são programadas para serem emitidas caso ocorra algum evento que seja relevante no contexto dos fluxos de mensagens e no Acordo de Nível de Serviço (SLA).

Isso permite ao administrador um acompanhamento detalhado do que ocorre no servidor, facilitando a tomada de decisão em tempo hábil. Ainda é possível configurar funcionalidades que executarão procedimentos automáticos que irão corrigir problemas ou minimizar possíveis impactos no sistema.

O OSB fornece um conjunto de parâmetros e estatísticas que demonstram detalhadamente o que ocorre no barramento. Cada chamada de serviço pode ser rastreada e seus dados registrados em log para futuras análises. Durante os testes a ferramenta realizou suas funcionalidades sem apresentar problemas, mostrando-se plenamente estável e confiável.

5 Conclusão

O Enterprise Service Bus (ESB) é a principal infra-estrutura de uma Arquitetura Orientada a Serviços. Hewit [2009] aponta que não existe um padrão para ESB. No entanto, é consenso entre os fornecedores de ferramentas quais são as responsabilidades de um

ESB, as quais incluem: prover conectividade; transformação de dados; roteamento de mensagens baseado em conteúdo; tratar segurança; tratar confiabilidade; gerência de serviços; monitoramente e *log* de serviços; balanceamento de carga etc.

Este relatório apresentou os principais conceitos de ESB e analisou de forma prática a implementação de ESB da Oracle, o Oracle Service Bus (OSB). As principais características da ferramenta foram apresentadas, sendo detalhada a arquitetura da ferramenta. A análise prática abordou as principais funcionalidades para implantação de um ESB em uma organização focando na:

- Publicação de serviços no barramento;
- Efeitos da evolução de serviços sobre o barramento e como este pode ser utilizado para tratar versionamento de serviços;
- Uso do barramento para definição de fluxos (em serviços de *proxy*) para tratar processos de negócio simples;
- Validação de tipos de dados definidos nas mensagens utilizando o barramento;
- Funcionalidades para monitoramento de serviços no barramento.

Os resultados obtidos com nossos experimentos demonstraram que o OSB é uma ferramenta que atende às características principais de um ESB no que concerne à implantação de SOA em uma organização.

6 Referências

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J., **Design Patterns: Elements of Reusable Object-Oriented Software**, Addison-Wesley, 1994.

GU, Q., LAGO, P., **A stakeholder-driven Service Life Cycle Model for SOA**, ACM, IW-SOSWE'07, Dubrovnik, Croácia, 3 de setembro, 2007. Dísponível em http://portal.acm.org/ft_gateway.cfm?id=1294930&type=pdf&coll=GUIDE&dl=GUIDE&CFID=80367577&CFTOKEN=59281546. Acesso em 10 Jul. 2008.

HOHPE, G., WOOLF, B., **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**, Addison-Wesley, 2004.

HEWITT, E., **Java SOA Cookbook**, O'Reilly, 2009.

JOSUTTIS, N. M., **SOA in practice: The Art of Distributed System Design**. O'Reilly, 2007.

OSB, **Concepts and Architecture 10g Release 3 (10.3)**, 2008a. Disponível em http://download-llnw.oracle.com/docs/cd/E13159_01/osb/docs10gr3/concepts/index.html. Acesso em 14 Out. 2009.

OSB, **Operations Guide 10g Release 3 (10.3),** 2008b. Disponível em http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/pdf/operations.pdf. Acesso em 18 Dez. 2009.

OSB, **Oracle Service Bus**. 2009a. Disponível em http://www.oracle.com/technology/products/integration/service-bus/index.html. Acesso em 12 Set. 2009.

OSB, **Modelling message flow**. 2009b. Disponível em http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/userguide/modelingmessageflow.html. Acesso em 12 Set. 2009b.

OSB, **XQuery mapper**. 2009c. Disponível em http://download.oracle.com/docs/cd/E13160_01/wli/docs10gr3/dtguide/index.html. Acesso em 12 Set. 2009c

OSB, **Security**. 2009d. Disponível em http://download-llnw.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/index.html. Acesso em 12 Set. 2009.

OSB, **User Guide**. 2009e. Disponível em

http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/userguide/index.html. Acesso em 30 Out. 2009.

PAPAZOGLOU, MIKE P., HEUVEL, WILLEM-JAN, **Service oriented architectures: approaches, technologies and research issues**, VLDB Journal, Springer-Verlag, 2007

SOUZA, J., AZEVEDO, L., BAIAO, F., SANTORO, F. **Estudo de ferramentas da BEA para SOA**. Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT-0015/2009, 2009. Disponível (também) em: http://seer.unirio.br/index.php/monografiasppgi.

Apêndice 1 - Diferenças entre OSB e ALSB

Esta seção apresenta as diferenças observadas entre o ALSB e OSB de acordo com as funcionalidades avaliadas por Souza *et al*. [2009].

A publicação de serviços no barramento compreende duas etapas. Em primeiro lugar, é necessário realizar o *deploy* das aplicações no servidor de aplicação WLS (agora renomeado para Oracle WebLogic Server) e, em seguida, publicar as interfaces de serviços (WSDL) no OSB. O primeiro passo é necessário, uma vez que o serviço será executado no mesmo OWLS em que está a sua interface. Observe que, no caso de uma aplicação distribuída, o *deploy* do serviço (seu código executável) será realizado em qualquer outro servidor de aplicação (JBoss, WebSphere, etc), e o WSDL do serviço será publicado no OSB.

Neste trabalho estamos utilizando o OWLS e o OSB disponíveis "http://<IP ou local host>:7001/console" e "http://localhost:7021/sbconsole/", respectivamente, ou a partir dos links do menu "Iniciar".

- WLS × OWLS
    - Botão Lock & Edit

      Para realizar o *deployment*, é necessário entrar no console de administração do OWLS (http://<IP do computador ou localhost>:7001/console/console.portal). Diferentemente do WLS no OWLS não existe mais o botão Lock & Edit. No entanto, após iniciar o OWLS a partir do OSB, aparecem botões para ativar as mudanças (*Activate* e *Undo changes*).

    - Instalação de aplicação no OWLS

      O passo a passo para instalação de aplicação no OWLS é o mesmo realizado no WLS exceto pelo fato de que após o registro da aplicação, esta é iniciada diretamente. Anteriormente, no WLS após a publicação da apli-

cação esta ficava no estado "Preparada para inicialização" e era necessário executar alguns passos para inicializá-la.

- ALSB × OSB
    - Assim como no ALSB, no OSB é necessário criar uma sessão para gerenciar o OSB, bem como ativá-la no final da realização das alterações.
    - Em ambos os ambientes, os recursos (WSDL, XSD, etc) são agrupados por projetos, que podem ser visualizados no Project Explorer.

Apêndice 2 - Elementos de um WSDL

Esta seção descreve os elementos de um WSDL [OSB, 2009e].

Um WSDL possui os elementos apresentados na Tabela 5.

Tabela 5 – Elementos de um WSDL

Elemento	Descrição
<types>	Definições de tipos para o conteúdo das mensagens.
<message>	Definição abstrata do tipo de dados sendo transferido. Uma mensagem consiste de partes, as quais descrevem o conteúdo lógico e abstrato de uma mensagem.
<portType>	Coleção abstrata de operações suportadas pelo serviço.
<operation>	Descrição abstrata de uma ação suportada pelo serviço
<binding>	Define o protocolo concreto e especificação do formato de dados para um <portType>
<port>	Um *endpoint* simples, consistindo de endereço de rede e um *binding*.
<service>	Coleção de portas ou *endpoints*.

Um exemplo de seção <types> do WSDL de um serviço que provê cotas de ações, com os termos *TradePriceRequest* e *TradePrice* é apresentado na Figura 166.

```
<types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2001/XMLSchema">
        <element name="TradePriceRequest">
            <complexType>
                <all>
                    <element name="tickerSymbol" type="string"/>
                </all>
            </complexType>
        </element>
        <element name="TradePrice">
            <complexType>
                <all>
                    <element name="price" type="float"/>
                </all>
            </complexType>
        </element>
    </schema>
</types>
```

Figura 166 – Exemplo de seção *types* de um WSDL

Um exemplo de seção <message> é apresentado na Figura 167, onde existem quatro tipos de mensagens (*sellerInfoMessage*, *buyerInfoMessage*, *response*, e *negotiationMessage*), cada uma tem uma ou mais partes.

```
<message name="sellerInfoMessage">
    <part name="inventoryItem" type="xsd:string"/>
    <part name="askingPrice" type="xsd:integer"/>
</message>
<message name="buyerInfoMessage">
    <part name="item" type="xsd:string"/>
 <part name="offer" type="xsd:integer"/>
</message>
<message name="response">
    <part name="result" type="xsd:string"/>
</message>
<message name="negotiationMessage">
    <part name="item" type="xsd:string"/>
    <part name="price" type="xsd:integer"/>
    <part name="outcome" type="xsd:string"/>
</message>
```

Figura 167 – Exemplo de seção *message* de um WSDL

O elemento <*portType*> define um conjunto de operações suportadas por um ou mais *endpoints* (os quais são definidos pelo elemento <*port*>). No padrão WSDL 2.0, o nome do elemento (<*portType*>) foi alterado para <*interface*>. O tipo de porta provê a interface pública para as operações providas pelo serviço. Cada operação é definida em um elemento <*operation*>, onde cada uma é uma descrição abstrata de uma ação suportada pelo serviço. A Figura 168 apresenta um exemplo que define um tipo de porta com uma operação, *GetLastTradePrice*, a qual tem como entrada a mensagem *GetLastTradePriceInput* e como saída a mensagem *GetLastTradePriceOuput*. As descrições concretas destas mensagens são definidas no binding WSDL, como demonstrado no subelemento <*soap:operation*> da Figura 169.

```
<portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
    </operation>
</portType>
```

Figura 168 – Exemplo de portType e operação do WSDL

A seção *Binding* (Figura 169) [Josuttis, 2007] é onde a definição do serviço passa da descrição abstrata para a concreta. Esta seção especifica detalhes de implementação de um serviço definido de forma abstrata no *portType*, e é composta da seguinte forma:

1) Transport: Protocolo de comunicação (HTTP ou SMTP – Simple Mail Transport Protocol): protocolo para transportar as mensagens SOAP
2) Style: pode assumir os valores rpc ou document
3) Formato de dados: literal ou encoded. Encoded (segue regras de codificação segundo SOAP 1.1), e não está de acordo com o padrão para web services.

O atributo *SoapAction* é usado para informar o propósito da requisição HTTP SOAP, seu valor é uma URI, identificando a intenção. Ele pode ser usado por firewalls, para filtrar as requisições SOAP feitas usando HTTP. A Figura 169 apresenta um

exemplo de *Binding* para o tipo de porta *StockQuotePortType*, o qual é provido pelo valor do atributo *type*. O subelemento <*soap:binding*> é atrelado ao formato de protocolo SOAP. Neste subelemento, o atributo *style* especifica que o formato dados é do estilo *SOAP document*, e o atributo *transport* especifica que o protocolo de transporte é HTTP.

```
<binding name="StockQuoteSoapBinding"
type="tns:StockQuotePortType">
    <soap:binding style="document"
           transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
        <soap:operation
            soapAction="http://example.com/GetLastTradePrice"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
    </operation>
</binding>
```

Figura 169 – Exemplo de elemento *Binding* do WSDL

A seção <*service*> especifica um ou mais *endpoints* na qual a funcionalidade do serviço está disponível. Tecnicamente seção <*service*> lista um ou mais *ports*. Cada *port* consiste de um *portType* (*interface*) e um *binding* (implementação), incluindo endereço físico para acesso ao serviço. A Figura 170 define duas portas, *StockQuotePort* e *StockQuotePortUK*. Ambos usam o mesmo *binding*, *tns:StockQuoteSoapBinding*, o que é definido de forma concreta no atributo <*binding*>, mas tem endereços diferentes *http://example.com/stockquote* e *http://example.uk/stockquote*. Estas são portas alternativas para o serviço.

```
<service name="StockQuoteService">
    <port name="StockQuotePort"
binding="tns:StockQuoteSoapBinding">
       <soap:address
location="http://example.com:9999/stockquote"/>
    </port>
    <port name="StockQuotePortUK"
binding="tns:StockQuoteSoapBinding">
       <soap:address
location="http://example.uk:9999/stockquote"/>
    </port>
</service>
```

Figura 170 – Exemplo de seção *service* do WSDL