

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA DESENVOLVIMENTO
DO RIO GRANDE DO NORTE**

Elvis Joannes Fidelis de Araújo

**ANÁLISE DA IMPLEMENTAÇÃO DE TECNOLOGIAS WEBSERVICE E ESB
EM JAVA**

Natal/RN

2011

ELVIS JOANNES FIDELIS DE ARAÚJO

**ANÁLISE DA IMPLEMENTAÇÃO DE TECNOLOGIAS WEBSERVICE E ESB EM
JAVA**

Monografia apresentada ao curso de especialização em Desenvolvimento de sistemas corporativos como, requisito obrigatório para obtenção de grau de especialista em desenvolvimento de sistemas corporativos.

Orientador: David
David,Msc.

Natal/RN

2011

ELVIS JOANNES FIDELIS DE ARAUJO

ANÁLISE DA IMPLEMENTAÇÃO DE TECNOLOGIAS WEBSERVICE E ESB EM
JAVA

Monografia apresentada ao curso de bacharelado em sistemas de informação e aprovada em ____ de _____ de ____, pela banca examinadora composta pelos seguintes membros

David, Msc.
Orientador

Membro Examinador

Membro Examinador

Resumo

Esta pesquisa analisa as tecnologias *Webservices* e ESB em Java. Para tanto foi desenvolvido dois exemplos utilizando o framework JBossAS e JBossESB, com a finalidade de identificar diferenças relevantes entre essas tecnologias. Para isso foram desenvolvidos exemplos de aplicações para entender os papéis do Web Service fornecendo serviços diretamente para um consumidor e um WebService fornecendo serviços através de um ESB. As tecnologias *Webservices* e ESB desempenham um papel muito importante na integração de sistemas de uma forma padronizada e é bastante implementada devido seu grau de flexibilidade e confiabilidade ajudando organizações a melhorar sua governança corporativa. Com a utilização dessas tecnologias é possível adotar a abordagem SOA como uma alternativa para soluções de integração de sistemas. O Frameworks JBossAS e JBossESB são ferramentas que vem sendo utilizadas cada vez mais no mercado, e por fazer parte da plataforma Java traz inúmeros benefícios de custo e escalabilidade.

Palavras chave: WebServices.ESB.serviços.integração.SOA.governança corporativa.java.escalabilidade

Abstract

This research examines the Web services and ESB technologies in Java. For this purpose two examples were developed using the framework and JBossAS JBossESB, in order to identify significant differences between these technologies. For this example applications were developed to understand the roles of Web Service providing services directly to a consumer and a Web Service providing services through an ESB. Web services and ESB technologies play an important role in the integration of systems in a standardized way and is fairly implemented due to its flexibility and reliability by helping organizations improve their corporate governance. Using these technologies it is possible to adopt the SOA approach as an alternative to systems integration solutions. The Frameworks and JBossAS JBossESB are tools that are being used increasingly in the market, and to be part of the Java platform brings numerous benefits of cost and scalability.

KeyWords: Webservices.ESB.services.integration.SOA.corporate governance.java. scalability

LISTA DE FIGURAS

Figura 1 - Solução ponto-a-ponto.....	10
Figura 2 - Solução Hub-and-spoke.....	11
Figura 3 - Enterprise Message Bus.....	12
Figura 4 - Enterprise Service Bus.....	14
Figura 5 - Acoplamento fraco	21
Figura 6 - Dissociação de fonte de dados	22
Figura 7 - Fonte de dados	23
Figura 8 - Troca de mensagens XML	25
Figura 9 - Tipos de troca de mensagens.....	26
Figura 10 - Papéis <i>Web services</i>	27
Figura 11 - Pilha de protocolos WebService	29
Figura 12 - WS-Security model	30
Figura 13 - Configurações,status e roadmap	32
Figura 14 - Integração baseada em padrões	34
Figura 15 - Orquestração e fluxo de processo	35
Figura 16 - Utilização de XML compartilhando dados entre as aplicações	36
Figura 17 - ESB orientado a protocolo	37
Figura 18 - ESB orientado a API	37
Figura 19 - Unidades de negócio com falta de autonomia.	38
Figura 20 - ESB faz organização realizar operações cooperativamente.	39
Figura 21 - Action pipeline.....	40
Figura 22 - Arquitetura de componentes do Registro.....	41
Figura 23 - Componentes SOA	42
Figura 24 - converterTemp.wsdl.....	45
Figura 25 - Registro do serviço TempWS	45
Figura 26 - Action Pipeline	47
Figura 27 - Registro do serviço HelloWorldWS no JBossWS.....	51
Figura 28 - JBossESB expondo Web Services sobre HTTP,socket e protocolos jms	52
Figura 29: Registro do serviço GoodbyWorldWS.....	53
Figura 30: Saida da classe SendMessage.java.....	53

SUMÁRIO

1 INTRODUÇÃO	8
1.1 SOLUÇÃO PONTO-A-PONTO.....	9
1.2 SOLUÇÃO HUB-AND-SPOKE	10
1.3 ENTERPRISE MESSAGE BUS INTEGRATION	11
1.4 ENTERPRISE SERVICE BUS INTEGRATION	13
1.5 OBJETIVOS	14
1.5.1 Objetivo Geral	14
1.5.2 Objetivos específicos.....	15
1.6 JUSTIFICATIVA	15
2 REFERENCIAL TEÓRICO.....	16
2.1 SOA.....	16
2.1.1 Arquitetura	18
2.1.2 Foco na arquitetura de negócio	19
2.1.3 Contrato de serviço.....	19
2.1.4 <i>Service Level Agreements</i>	20
2.1.5 Acoplamento fraco	20
2.1.6 Fonte de dados para consumidores	22
2.1.7 Qualidade de Serviço	24
2.2 WEB SERVICES	24
2.2.2 Papéis <i>Web services</i>	26
2.2.2.1 Fornecedor de serviço.....	26
2.2.2.2 Solicitante do serviço.....	26
2.2.2.3 Registro de serviço.....	27
2.2.2.4 Pilha de protocolos do <i>Web services</i>	27
2.2.2.5 Serviço de transporte	27
2.2.2.6 Mensagens XML	28
2.2.2.7 Descrição do serviço	28
2.2.2.8 Serviço de descoberta.....	28
2.2.2.9 Segurança.....	29
2.2.3 JBossWS.....	31
2.3 ESB	32
2.3.1 Integração baseada em padrões.....	33
2.3.2 Adoção de XML	35
2.3.3 <i>Message oriented Middleware</i>	36
2.3.4 ESB Orientado a protocolo.....	36
2.3.5 ESB Orientado a API	37
2.3.6 Segurança.....	38
2.3.7 JBossESB	39
2.3.7.1 Criação de serviço no JBossESB.....	40
2.3.7.2 Registro	41
2.3.7.3 Componentes SOA.....	42
2.4 RELAÇÃO ENTRE ESB E WEB SERVICES	43
3 METODOLOGIA ADOTADA	43
4 ANÁLISE DE RESULTADOS	45
4.1 IMPLEMENTAÇÃO WEB SERVICES	45
4.2 IMPLEMENTAÇÃO DE WEB SERVICE EM UM ESB	46
4.2.1 Implementação do projeto <i>WebService consumer</i>	46

4.2.2 Implementação do projeto WebService producer	51
5 CONCLUSÃO	54
6 REFERENCIAS.....	54

1 INTRODUÇÃO

Durante os últimos anos presenciamos uma proliferação de sistemas de informação computadorizados (SI) nas organizações. Ter integrações eficazes entre Sis tornou-se um dos aspectos críticos para operação de ambientes de negócios, sobretudo aqueles de natureza altamente colaborativa que consiste na utilização das tecnologias da internet, como protocolos, padrões e softwares para integrar os processos de negocio a nível organizacional e com eventuais parceiros, fornecedores e clientes. Este crescente desafio do ambiente de negócios motivou a indústria de tecnologia da informação (TI) a desenvolver novos conceitos, técnicas e softwares direcionados exclusivamente a questão de integração entre Sis .

A TI surgiu nas décadas de 1950 e 1960, nessa época existiam especialistas que sabiam como operar máquinas utilizadas para processar informações do negócio de suas empresas. Com elas era possível executar vários cálculos e automatizar alguns processos. Anos depois as empresas começaram a aderir aos ERP's (Enterprise Resource Planning) como solução para automatizar seus processos. Por não existir padrões de integração naquela época, fornecedores de soluções ERP adquiriam os softwares de seus concorrentes para integrar nos seus pacotes. Os fornecedores disponibilizavam várias opções de soluções de acordo com necessidades específicas de cada uma das unidades operacionais das indústrias, e a integração não era colocada como prioridade.

Uma alternativa para a integração entre os setores foram às fusões e compras com uma empresa que tinham unidades que precisavam do mesmo produto, com o intuito de que unidades como RH, faturamento, compras, vendas e administração pudessem reutilizá-lo. A primeira opção foi a de automatizar a integração de aplicações específicas, de modo que o usuário poderia digitar os mesmos dados apenas uma vez para serem aplicados em diversos sistemas, como alternativa, era criado um processo batch que era responsável pela transferência de informações de um sistema para outro. A integração das aplicações dos sistemas da empresa parecia oferecer um apoio sofisticado para criar integrações ponto-a-ponto com softwares dedicados. A necessidade de uma rápida integração dos sistemas de TI em uma empresa comercial exigia unidades com cada vez mais aplicações específicas que por padrão, ainda não estavam nos pacotes de software, a

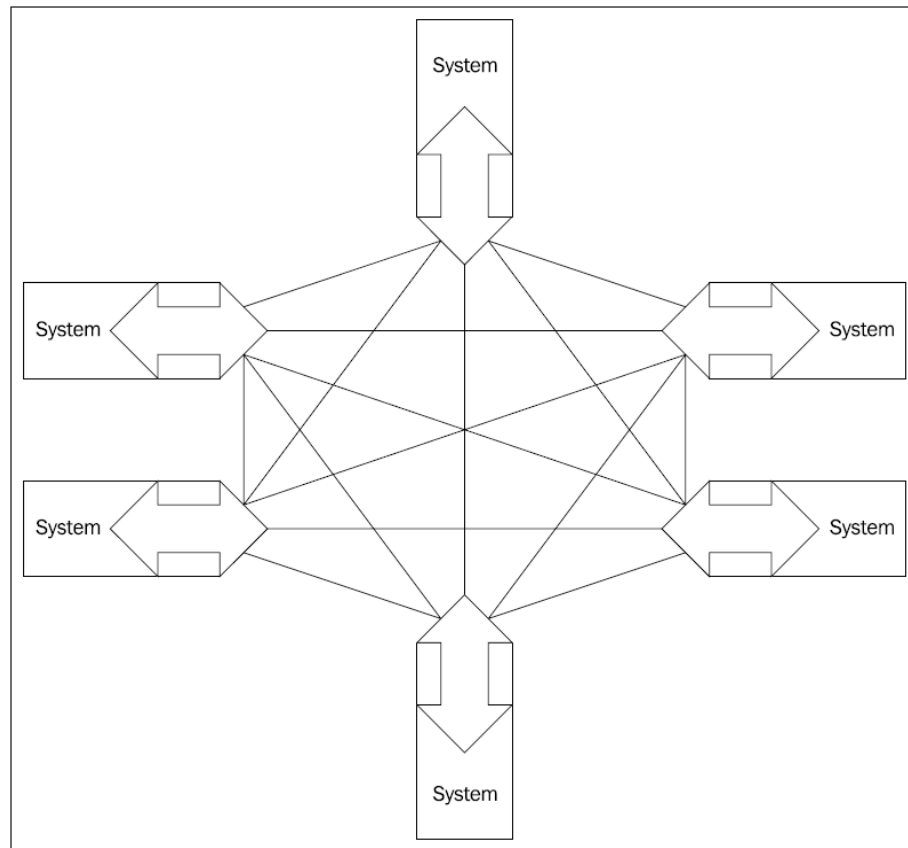
necessidade de cooperação com os novos negócios baseados em Internet (muitas vezes sem padrões de integração) impulsionou a indústria a trabalhar com normas em TI. Sob o nome de *webservice*, os grandes fornecedores de software e uma grande quantidade de pequenas empresas de TI em todo o mundo definiram e apoiaram um conjunto de normas. Com estas normas, os primeiros termos de interoperabilidade foram definidos. Todos os serviços oferecidos através da *World Wide Web* foram bem aceitos, pois definia como o serviço poderia ser encontrado, invocado, os padrões de formatação entre outros. No entanto, isso não significava ainda uma arquitetura utilizada pela TI, exceto para o conceito fundamental de fornecedor-consumidor que define um registro padrão para encontrar *web services* publicados, que podiam ser acessados por unidades consumidoras. Nos anos 90 o Corba (*Common Object Request Broker*) surgiu como uma promessa, que definia um padrão de interface que utilizava um protocolo binário para integrar qualquer sistema, mais isso não se realizou e todos os sistemas baseados em Corba foram descontinuados. Hoje existem diferentes tipos de topologias e precisamos entender a diferença entre elas. Abaixo a descrição das principais arquiteturas:

- Solução Ponto-a-Ponto
- Solução *Hub-and-Spoke*
- *Enterprise Message Bus Integration*
- *Enterprise Service Bus Integration*

1.1 SOLUÇÃO PONTO-A-PONTO

Definimos a solução através da utilização de um par de aplicações. Assim temos dois pontos a fim de serem integrados. Esta é a maneira mais fácil de integrar desde que o volume de integração seja baixo. Normalmente são utilizados API's específicas de tecnologias como FTP, IIOP, interfaces de integração remota para realizar a integração. A vantagem de utilizar essa arquitetura é que entre os dois pontos temos acoplamento forte, isso quando ambos os lados tem conhecimento entre eles. Abaixo uma figura que representa a integração ponto a ponto.

Figura: Solução ponto-a-ponto
Fonte: Binildas (2008)

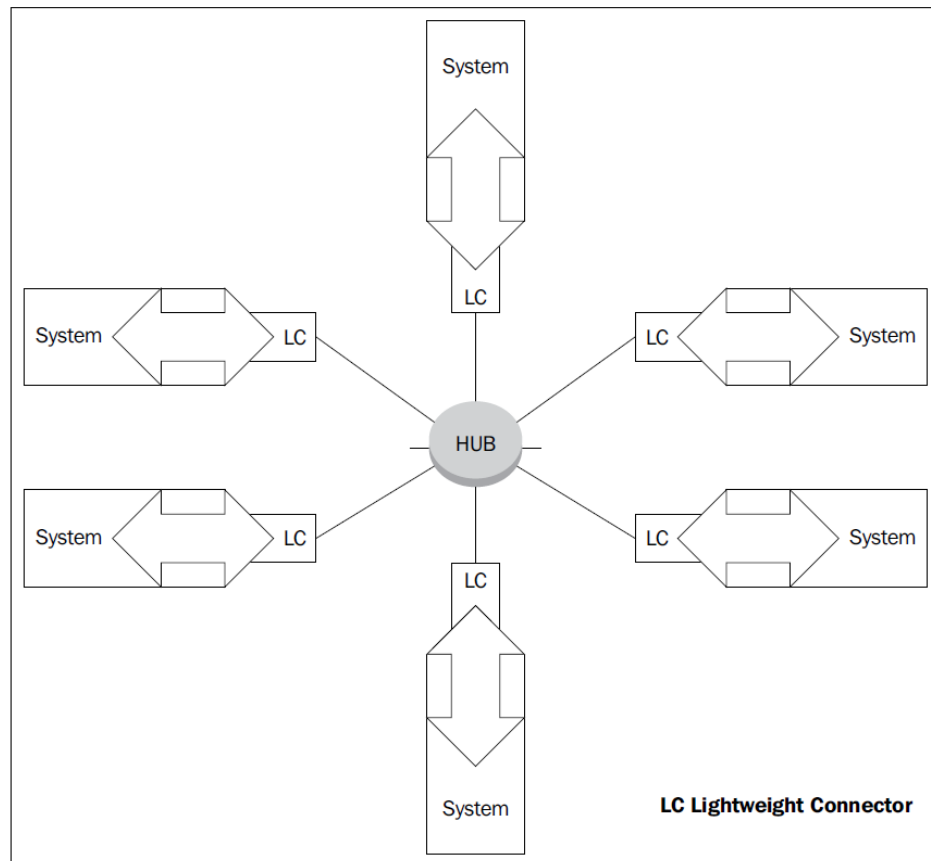


1.2 SOLUÇÃO HUB-AND-SPOKE

A arquitetura Hub-and-Spoke também chamada de corretora de mensagens é semelhante a uma arquitetura ponto-a-ponto em que é fornecido um hub central (corretor) para que todas as aplicações estejam conectadas. Outra característica da arquitetura Hub-and-Spoke é que cada aplicação se conecta com a central através de um conector leve. Os conectores leves facilitam a integração de aplicações com pequenas ou nenhuma mudança com as aplicações existentes. A transformação e o roteamento da mensagem ocorrem dentro do Hub. Com isso há uma diminuição do número de conexões necessárias para a integração. Como as aplicações não conectam diretamente a outras aplicações, elas podem ser removidas da topologia de integração através da remoção no Hub. Há um grande problema com a arquitetura hub-and-spoke. Devido a centralização do hub, ele acaba se tornando o

único ponto de falha. Se o hub falhar a topologia de integração inteira falhará. Uma solução para este problema é agrupar os hubs. Abaixo uma figura representando a arquitetura *Hub-and-Spoke*.

Figura 2: Solução Hub-and-spoke
Fonte: Binildas (2008)



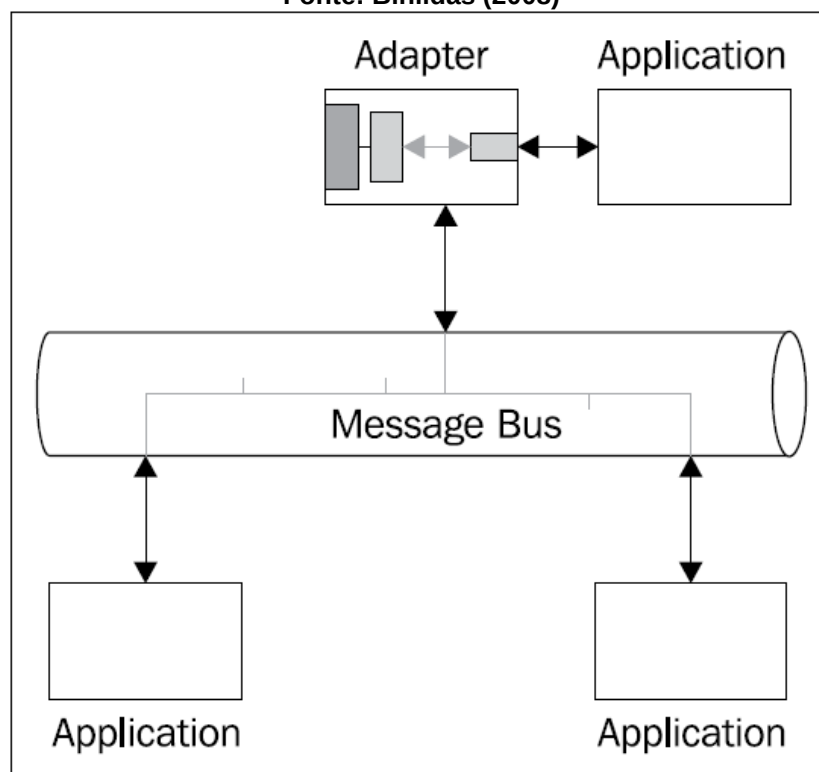
1.3 ENTERPRISE MESSAGE BUS INTEGRATION

Enquanto a arquitetura *hub-and-spoke* faz uso de conectores leves para integração de aplicações em um hub central, muitas vezes as aplicações integradas precisam interagir umas com as outras, de modo que elas possam ser adicionadas ou removidas sem afetar as outras aplicações. Um *Enterprise Message Bus* fornece uma infra-estrutura comum de comunicação para mensagem que atua como um elemento neutro e interpretador das linguagens entre as aplicações.

Esta infra-estrutura de comunicação pode incluir um roteador de mensagens. Assim os aplicativos interagem uns com os outros através do barramento de mensagens com filas de solicitação-resposta. Se um aplicativo do consumidor quer invocar um serviço específico em um aplicativo de um provedor diferente, ele coloca uma mensagem de solicitação adequada formatada na fila de pedidos para esse serviço. Em seguida, ele ouve a mensagem de resposta na fila do serviço resposta. O aplicativo provedor escuta os pedidos na fila do serviço de pedidos, executa o serviço, e em seguida envia qualquer resposta para fila de serviço resposta.

Existem produtos como o IBM WebSphere MQ (WMQ) e o MQ Microsoft (MSMQ), que são as melhores soluções de classes de fila de mensagens, que são utilizadas para integrar aplicações na topologia barramento de mensagens. Abaixo um figura mostra a utilização de um adaptador que pode fornecer conectividade entre as aplicações e as mensagens usando API's proprietárias.

Figura 3: Enterprise Message Bus
Fonte: Binildas (2008)

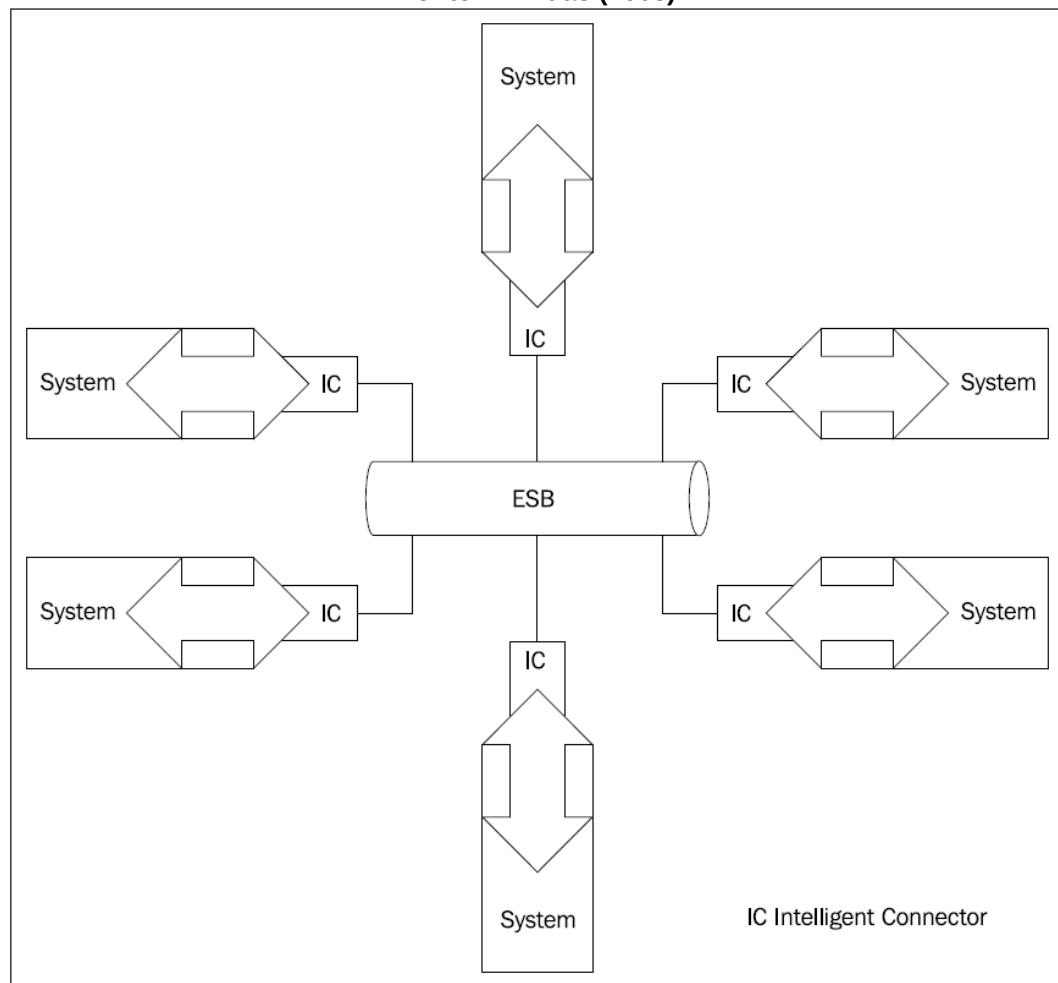


1.4 ENTERPRISE SERVICE BUS INTEGRATION

É a abordagem que se aproxima da integração, que faz uso da tecnologia de pilhas para proporcionar um barramento para integrar aplicações. Diferentes aplicações não comunicam-se diretamente uns com as outras com a finalidade de integração, em vez disso elas se comunicam através de um *middleware Service Oriented Architecture* (SOA) . A arquitetura ESB caracteriza-se pela natureza distribuída da sua topologia de integração. A maioria das soluções ESB são baseadas em tecnologias *Web services description Language* (WSDL), e eles usam formatos *Extensible Markup Language* (XML) para tradução e transformação de mensagens.

ESB é uma coleção de *middlewares* que fornece capacidades de integração. Estes serviços *middlewares* estão no centro da arquitetura ESB onde aplicações são encaminhadas e transformadas. Semelhante a arquitetura *hub-and-spoke*, na arquitetura ESB, os aplicativos também se conectam por meio abstrato, através de conectores inteligentes. Eles são inteligentes, porque tem lógica desenvolvida em conjunto com o ESB para selecionar e interligar serviços em tempo de execução. A figura abaixo representa o esquema de integração ESB.

Figura 4: Enterprise Service Bus
Fonte: Binildas (2008)



1.5 OBJETIVOS

1.5.1 Objetivo Geral

Este trabalho tem como objetivo analisar a implementação das tecnologias WebService e ESB na plataforma Java e identificar seus principais componentes de integração de serviços,

1.5.2 Objetivos específicos

- Analise da abordagem SOA para saber sua influência numa implementação WebService e ESB.
- Análise e desenvolvimento de aplicações *Web service* utilizando a API JBossWS e JBossEsb .
- Análise das diferenças entre as implementações das tecnologias *web services* dentro e fora de um ambiente ESB.

1.6 JUSTIFICATIVA

Este estudo visa comparar a implementação de duas tecnologias que fazem parte da abordagem SOA. Esta pesquisa é de fundamental importância para identificar possíveis soluções para cenários, que utilizam um ambiente com vários sistemas e forte acoplamento. Diante dos resultados desta pesquisa será possível vislumbrar de forma coesa um paradigma de integração que vem evoluindo gradativamente com o passar dos anos. A utilização da SOA com seus respectivos modos de implementação faz parte de uma realidade em que poucas empresas se preocupam com o controle do seu negócio através da SOA. A utilização de implementações SOA através dos *web services* ou ESB elimina a redundância de informações que são geradas por diversos sistemas e os tornam escaláveis e flexíveis a mudança.

2 REFERENCIAL TEÓRICO

2.1 SOA

A arquitetura orientada a serviços (SOA – service-oriented architecture) segundo (JOSUTTIS, 2006) é uma abordagem que ajuda os sistemas a permanecerem escalares e flexíveis enquanto crescem, e que também ajuda a resolver a lacuna do negócio/TI. A abordagem consiste em três elementos principais.

- **Serviços** - representam funcionalidades de negócio independentes e, que podem ser implementados com qualquer tecnologia em qualquer plataforma.
- **Uma infra-estrutura específica, chamada de barramento de serviços corporativos (ESB-Enterprise service bus)** - Permite combinar serviços de maneira fácil e flexível com baixo acoplamento.
- **Políticas e processos** - lidam com o fato de que os grandes sistemas distribuídos são heterogêneos, estão sob manutenção e possuem diferentes proprietários.

Segundo Josuitis (2006) grandes sistemas tem que lidar com legados. Não se pode implementar SOA modelando tudo a partir do zero. Lidar com o fato de que muitos dos sistemas em uso permanecerão em uso significa que estabelecer SOA não é um projeto como modelar um novo sistema. Envolve mudar a estrutura de um sistema existente, o que significa que lidará com plataformas antigas e questões de compatibilidade.

Para projetar softwares que sejam efetivamente orientados a serviços e para que elas façam uso dos benefícios esperados de uma arquitetura SOA, é necessário haver uma atenção especial na identificação e especificação dos serviços que a compõe. Os serviços que apoiarão os processos de negócio e satisfarão os requisitos devem ser definidos de forma adequada para que eles realmente apresentem os atributos de reusabilidade, flexibilidade e estejam em alinhamento com as metas do negócio.

Segundo Novaes (2008) com o incremento da adoção da gestão por processos nas organizações houve a necessidade de recursos de TI para permitir a modelagem, a execução e monitoramento de processos. Além disso, a necessidade de integração de sistemas também aumentou, entre outros motivos pelo fato dos processos de negócio usualmente conterem informações de áreas funcionais diversas. A combinação das ferramentas de workflow com as de (EAI) originou os sistemas conhecidos como BPMS(*Business Process Management System*).

Após o mapeamento dos processos de negócio, o primeiro desafio da automação é definir quais serviços deverão compor a SOA. Essa definição baseia-se na necessidade de informação do *workflow* de processos que serão automatizados. Uma das principais normas para acelerar a adoção de SOA é o (*Bussiness Process Executation Language for Web services*) (BPEL4WS). BPEL foi criado para enfrentar as exigências da composição dos serviços da web em um ambiente orientado a serviços. Nos últimos anos BPEL4WS tem se tornado efetivamente a norma mais importante para elevar a visibilidade de SOA em TI a nível empresarial. BPEL4WS permite que as organizações automatizem seus processos empresariais orquestrando serviços dentro e através de firewalls. Ela obriga as organizações a pensar em termos de serviços. Funcionalidades são expostas como serviços. A comunicação com fornecedores e parceiros externos é feito através de serviços. Os serviços são reutilizados em diferentes aplicações.

Erl(2006) Propõe na concepção de SOA a separação dos serviços em três camadas lógicas: camada de aplicação, camada de negócios e a camada de orquestração. A camada de negócios deve conter apenas lógica de negócios em seus serviços; a camada de aplicação faz a ligação do SOA as tecnologias específicas (banco de dados, sistemas legados, etc.) e a camada de orquestração representa, como o nome já diz a orquestração de serviços, como é o caso da utilização do BPEL4WS. Segundo ERL(2006), essa separação dos serviços em camadas lógicas promove agilidade. A independência entre a lógica de negócios e de aplicações é a grande responsável pelo ganho de agilidade. Por exemplo, um serviço da camada de aplicação que faz acesso a um banco de dados “x” pode passar a acessar o banco de dados “y” sem que a camada de negócios sofra impacto.

É possível harmonizar uma grande variedade de fornecedores, plataformas e aplicações com a utilização de vários protocolos do mercado baseados em *Web*

services. Entre eles o XML (*Extensible Markup language*) , que tem como o principal função a descrição de informações para a troca de dados em ambiente internet e que serve de base para os demais protocolos utilizados pela arquitetura. O SOAP (*Simple Object Access Protocol*) serve para troca de informações em ambiente distribuído, enquanto o WSDL (*Web service description language*), atua como uma linguagem de descrição dos serviços; e a UDDI (*Universal description , Discovery e integration*) é um conjunto de registro com função de “paginas amarelas” dos serviços disponíveis. Aplicações SOA são construídas a partir de serviços web fracamente acoplados. Os padrões JWS são muito importantes. Eles são o alicerce para o desenvolvimento de SOA em Java. As aplicações SOA fracamente acopladas são críticas para competitividade das empresas porque permitem as empresas serem flexíveis e se adaptarem a rápida evolução dos mercados.

2.1.1 Arquitetura

Muitos executivos e empresários, embora conscientes da definição de sua estratégia, gestão de suas operações e condução de seus negócios não utilizam o conceito de arquitetura. Quer serviço orientado ou não, a arquitetura é algo que os profissionais de TI estão preocupados. (JOSUTTIS,2008)

Esta situação hoje está mudando. Existe a aceitação subjacente da idéia de que muitos dos princípios que são aplicados pelos arquitetos de TI para os sistemas de informação pode, e deve, também ser aplicado por arquitetos de negócios para sistemas empresariais.

A arquitetura é necessária para restringir todas as possibilidades de SOA de tal maneira que leve a um sistema funcional e de fácil manutenção. Conceitos SOA, ferramentas SOA, e padrões SOA criam um espaço em que tenham que tomar decisões para evitar a desorganização. É necessário classificar os diferentes tipos de serviços, decidir a respeito da quantidade de acoplamento fraco, restringir o modelo de dados das interfaces dos serviço, definir as políticas, regras e padrões, esclarecer os papéis e as responsabilidades, decidir a respeito da tecnologia da infra-estrutura, decidir que padrões vai usar, e versões. (JOSUTTIS,2008)

A arquitetura orientada a serviço representa um modelo arquitetônico que visa aumentar a agilidade e o custo – benefício de uma empresa e reduzir a carga global da TI na organização. SOA suporta orientação a serviços na realização de objetivos estratégicos associados à computação orientada a serviços. Como uma forma de arquitetura. Uma implementação da tecnologia SOA pode consistir de uma combinação de tecnologias, produtos, API's, extensões de infra-estrutura de apoio, e várias outras peças. Como resultado, a construção de uma arquitetura tecnológica ao redor da orientação de serviço, estabelece um ambiente arquitetônico propício para a lógica de solução que foi concebido em conformidade com os princípios de design de orientação de serviços. (BINILDAS,2008)

2.1.2 Foco na arquitetura de negócio

Há não muito tempo, os sistemas de informação eram vistos como suporte para as organizações empresariais e metas do negócio. Sistemas computacionais eram introduzidos para executar tarefas manuais de forma rápida e mais precisa. Hoje, essa situação tem evoluído ao ponto em que sistemas de informação não só melhoram a eficiência das empresas, mas desempenham um papel crucial na estratégia corporativa para alcançar vantagem competitiva em mercados cada vez mais complexos. (JOSUTTIS,2008)

2.1.3 Contrato de serviço

Um contrato de serviço é composto de um ou mais documentos publicados que expressam meta informações sobre um serviço. A parte fundamental de um contrato de serviço consiste nos documentos que expressam sua interface técnica. Estes constituem o contrato de serviço técnico que basicamente define uma API para a funcionalidade oferecida pelo seus serviços através de sua capacidade. (CHAPPEL,2008)

2.1.4 Service Level Agreements

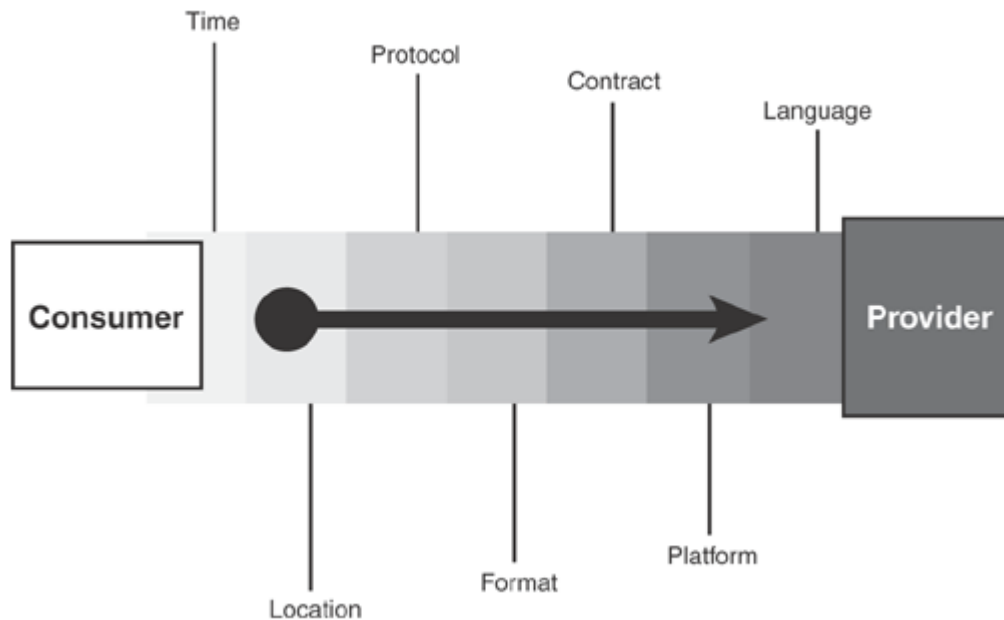
Um acordo de nível de serviço (SLA) é um documento que dá a informação necessária para decidir e invocar um determinado serviço a partir de outro software. A presença da utilização de uma SLA varia por fornecedor de SOA e usuário corporativo. Se presente o SLA.

- Inclui um acesso elementar e propõe detalhes do contrato na maioria dos casos.
- Pode ser escrito por design de softwares para ajudar a negociar quais funcionalidades serão incluídas em determinado serviço.
- Pode comunicar planos para utilizadores potenciais de serviços e outras partes interessadas.
- Pode ser à base de um documento jurídico que indica o nível de confiabilidade do serviço oferecido. (Por exemplo, quantas horas por semana o serviço está disponível).

2.1.5 Acoplamento fraco

O conceito de acoplamento fraco é introduzido, como é aplicado nas SOA's. Múltiplas dimensões incluindo tempo, localização, protocolo, formato e linguagem são plataformas que devem ser consideradas para disponibilizar o serviço entre consumidores e fornecedores em uma SOA.

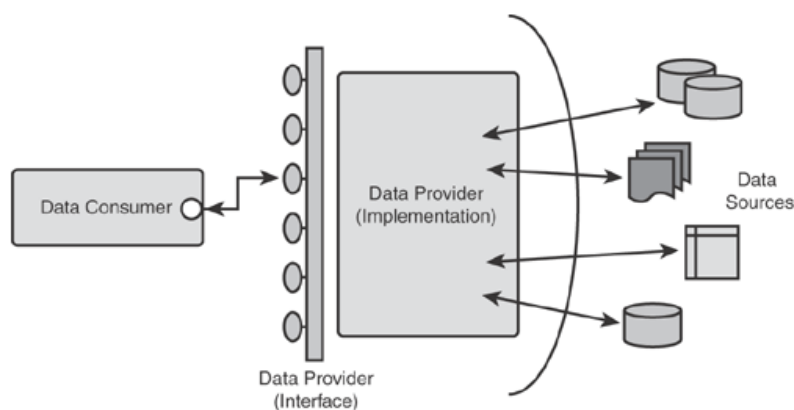
Figura 5: Acoplamento fraco
Fonte: Josuttis(2006)



Quando se permite o acoplamento entre consumidores e fornecedores, se constrói dependências indesejáveis e introduz uma complexidade e uma fragilidade. Esta é a base para uma boa aceitação da SOA. O acoplamento fraco é o conceito de minimizar as dependências. Quando as dependências estão minimizadas, as modificações têm os efeitos minimizados e os sistemas ainda executam quando parte deles estão quebradas ou indisponíveis JOSUTTIS(2006)

Na maioria das implementações SOA, nas empresas o número e a variedade de fonte de dados que devem ser incorporadas é grande e complexa. Do mesmo modo o número de interfaces e fornecedores de serviços também é grande, por isso esse ponto crítico é a melhor forma de fornecer dados que podem vir das várias fontes de dados de forma coerente e flexível. Como mostra a figura abaixo.

Figura 6: Dissociação de fonte de dados
Fonte: Josuttis(2006)



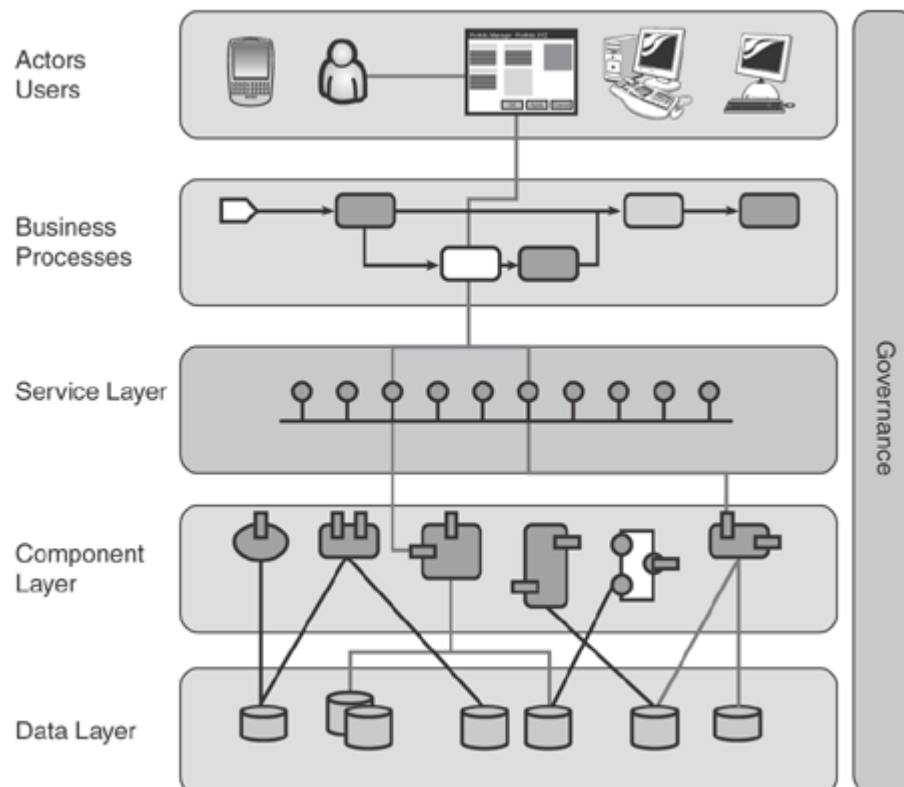
O consumidor de serviço é vinculado apenas ao contrato oferecido pelo fornecedor do serviço, na sua interface e descrição. Isto não é limitar a lógica do fornecedor de serviço para as fontes de dados que podem ou não ser acessadas para fornecer um serviço de resposta. Isto implica que a lógica do fornecedor de serviços pode ser substituída a qualquer momento e que as fontes de dados também podem ser alteradas a qualquer momento, sem que o consumidor tenha conhecimento dessas alterações.

2.1.6 Fonte de dados para consumidores

As fontes de dados são bastante variadas como planilhas e documentos na web, filas de mensagens, simples arquivos e banco de dados relacionais com as empresas com posse dos dados em sistemas de gerenciamento de conteúdo, e-mail e outros servidores que estão disponíveis para os consumidores através de interfaces de serviços. Com o advento da SOA, tem havido uma forte ênfase na identificação, organização, composição e reutilização da atividade comercial, no entanto até recentemente não houve um foco correspondente em dados do negócio.

Os componentes que fornecem serviços em execução para o componente de uma camada SOA dependem de fontes de dados que, na maioria dos sistemas empresariais reais são muitos e variados em natureza.

Figura 7: Fonte de dados
Fonte: Josuttis(2006)



A camada de dados da SOA implica em uma separação ampliada entre os utilizadores, os processos empresariais, serviços, componentes e fontes de dados. Alguns dos mesmos aplicam esses princípios para organizar os dados da camada que se aplicam as outras camadas. Para os dados estarem na primeira classe, na SOA deve existir uma separação clara entre os dados dos consumidores e fornecedores de serviços, os mesmos devem ser distintos e separados em uma SOA. Esta separação deve ser delimitada por uma interface, ou contrato, em ambas as partes, os fornecedores e consumidores. (JOSUTTIS,2008)

2.1.7 Qualidade de Serviço

Uma interface de serviço define a interação entre um serviço e um requerente em um sentido estreito, mas a interação pode ter muitas características adicionais. Em alguns contextos, a qualidade do serviço refere-se apenas a garantias, tais como o percentual de tempo em que um serviço contratado ficará disponível. A qualidade do serviço também inclui aspectos avançados de processamento em tempo real.

- Garantias de acesso
- Mecanismos de segurança
- Serviço de coordenação, incluindo a operação de controle de transação,
- Atualização de endereço em tempo real, conexão e conteúdo de mensagem.

Um determinado serviço pode ser oferecido com características de QoS diferentes, como quando uma empresa cobra uma taxa diferente em troca de um outro nível de confiabilidade. JOSUTTIS(2008)

2.2 WEB SERVICES

É visto como um servidor que pode ser interrogado com base em um conjunto de operações. Ela publica suas operações através de um arquivo XML, conhecido como WSDL (*Web Service Description Language*). Normalmente este documento é gerado e analisado pela máquina. As entidades que acessam os *web services* são conhecidos como clientes web service . SOAP é usado como protocolo de comunicação entre clientes e servidores. (LEONARD,2009)

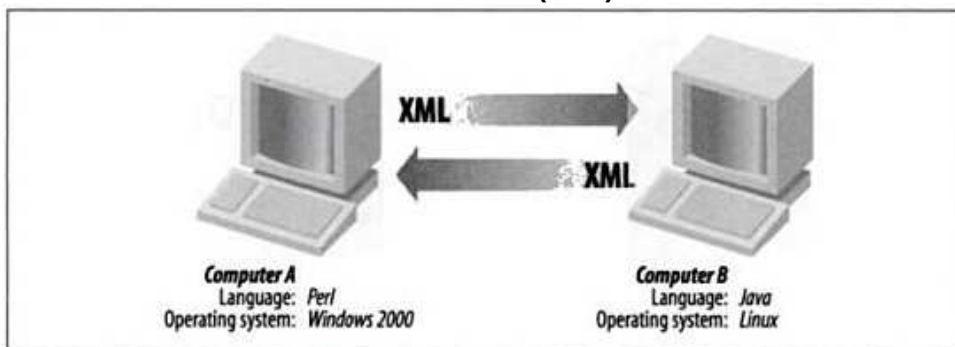
Uma plataforma *web services* é um conjunto de ferramentas para invocar e implantar serviços web utilizando uma determinada linguagem de programação. A plataforma tem componentes do lado do servidor e componentes do lado do cliente. Os componentes do lado do servidor são geralmente encapsulados dentro de algum

container (por exemplo , um servidor de aplicações Java EE). Do lado do cliente componentes são geralmente encapsulados como ferramentas para acessar interfaces de instancias que são utilizadas pelo *webservice*.

Segundo Chappel(2008) *Webservice* é uma tecnologia distribuída recente. Eles fornecem a base tecnológica para assegurar a interoperabilidade de software, sistemas operacionais e linguagens de programação. Do ponto de vista tecnológico os *web services* são o próximo passo evolutivo em arquiteturas distribuídas.

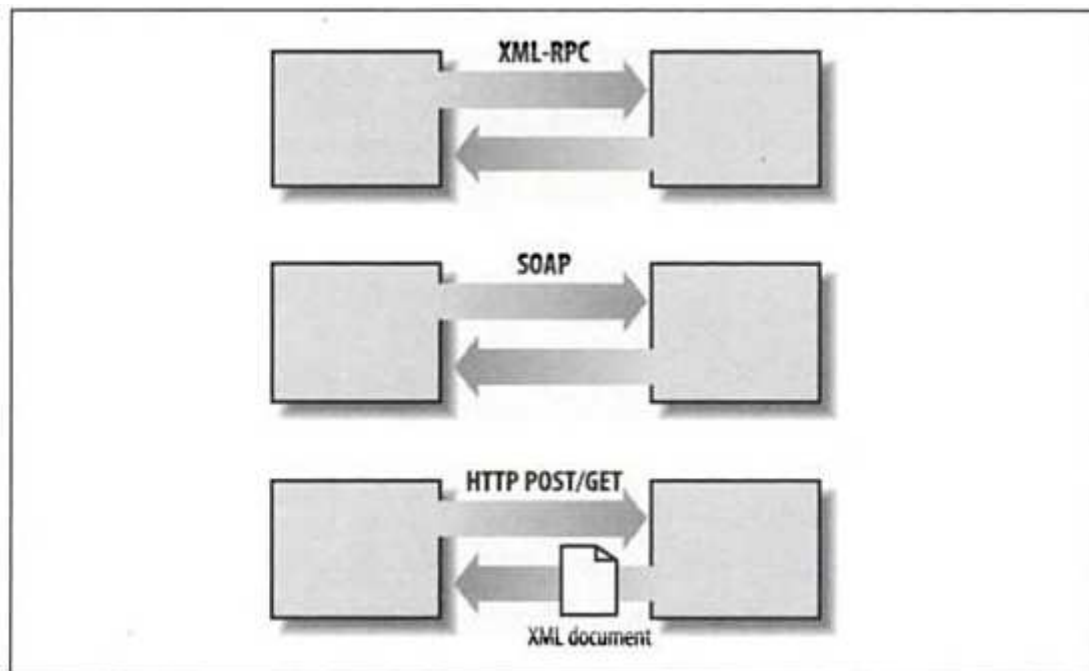
Segundo Cerami(2002) um *web service* é qualquer serviço que está disponível sob a internet, utilizando padrões de sistemas de mensagem XML e não está vinculado a nenhum sistema operacional ou linguagem de programação.

Figura 8: Troca de mensagens XML
Fonte: Cerami(2002)



Estas são as principais alternativas para mensagens XML. Poderia – se utilizar chamadas remotas usando (XML – RPC) ou SOAP. Podendo utilizar apenas HTTP GET/POST passando documentos XML arbitrários.

Figura 9: Tipos de troca de mensagens
Fonte: Cerami(2002)



2.2.2 Papéis *Web services*

2.2.2.1 Fornecedor de serviço

O fornecedor de serviços executa o serviço e torna-o disponível na internet

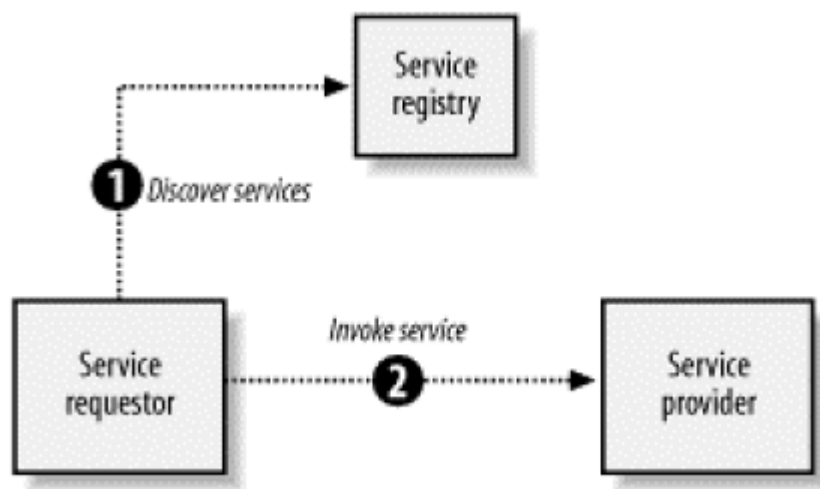
2.2.2.2 Solicitante do serviço

Este é o consumidor de qualquer serviço da web. O solicitante utiliza serviços web existentes através da abertura de uma conexão de rede e envia uma solicitação XML.

2.2.2.3 Registro de serviço

Esta é uma central lógica de serviços de diretórios. O registro fornece um local central onde os programadores podem publicar ou encontrar novos serviços existentes.

Figura 10: Papéis Web services
Fonte: Cerami(2002)



2.2.2.4 Pilha de protocolos do Web services

Abaixo é mostrado uma breve descrição das principais camadas de um *web services*.

2.2.2.5 Serviço de transporte

Esta camada é responsável pelo transporte das mensagens entre as aplicações. Atualmente esta camada inclui HTTP (*Hypertext Transfer Protocol*) ,

Simple Mail Transfer Protocol (SMTP) , *FTP (File Transfer Protocol)* e protocolos mais recentes , tais como *Blocks Extensible Exchange Protocol* (BEEP). (CERAMI,2002)

2.2.2.6 Mensagens XML

Esta camada é responsável por codificar as mensagens em um formato comum XML, a fim de que as mensagens possam ser entendidas em ambas as extremidades. Atualmente esta camada inclui XML-RPC e SOAP. (CERAMI,2002)

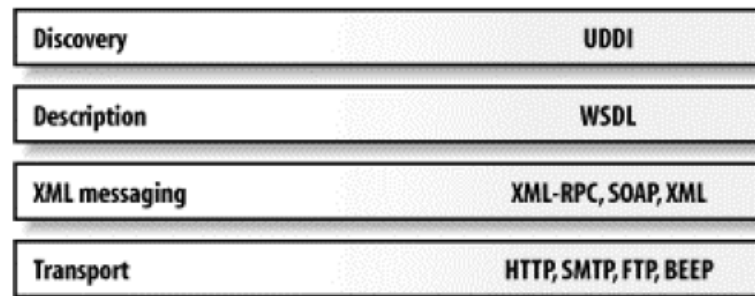
2.2.2.7 Descrição do serviço

Esta camada é responsável por descrever a interface publica para um determinado serviço da Web. Atualmente, o serviço é tratado através de uma descrição do *Web Service Description Language* (WSDL). Cerami(2002)

2.2.2.8 Serviço de descoberta

Esta camada é responsável pela centralização de serviços em um registro comum proporcionando publicar facilmente ou encontrar a funcionalidade. Atualmente, o serviço de descoberta é manuseado através do *Universal Description, Discovery, e Integration* (UDDI). Cerami(2002)

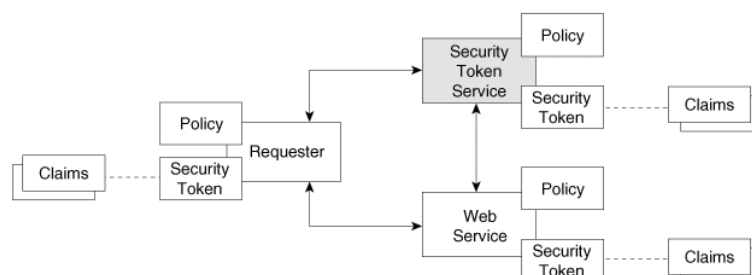
Figura 11: Pilha de protocolos WebService
Fonte: Cerami(2002)



2.2.2.9 Segurança

Segundo Weerawarana (2005) os serviços Web podem ser acessados através do envio de mensagens SOAP para serviços de parâmetros identificados como WS-Addressing. Estas mensagens solicitam ações específicas a partir do servidor e muitas vezes desencadeiam respostas de mensagens SOAP. Dentro deste contexto, o grande objetivo de assegurar os serviços Web dentro de duas metas: fornecendo facilidades para assegurar a integridade e confidencialidade das mensagens, e assegurando que o funcionamento do serviço para solicitações de mensagens que expressam as reivindicações exigidas pelas políticas de segurança. O desafio para o WS-Security é permitir a integração segura de sistemas extremamente diversificados e infra-estruturas. A figura abaixo mostra a estrutura do WS-Security. (JURIC,2005)

Figura 12: WS-Security model
Fonte: Juric,M(2005)



O WS-Security fornece uma maneira de representar o conjunto de requisições. Isso é representado como um token de segurança, e o WS-Security define um formato XML padrão para transportar esses tokens de segurança. Alguns sinais podem ser emitidos por uma terceira parte confiável. Por exemplo um certificado X.509 normalmente é emitida por uma autoridade de certificação (CA). No modelo WS-Security, a terceira parte confiável é chamada de símbolo de segurança Pública (STS), porque uma das suas tarefas é a segurança através dos tokens. (WEERAWARANA,2005)

Segundo Weerawarana (2005) A implementação do web service, tem um conjunto de requisitos para o pedido ser aceito, como “X.509 Based a autenticação é necessária” e a mensagem deve ser criptografada por uma chave AES e ter um comprimento de 128 bits. Estes requisitos são descritas como uma política de serviço . Nessa política de segurança particular, o requerente é conhecido através de UDDI ou WSMetadataExchange. Com base na política de segurança do serviço, o solicitante decide qual regra deve ser usada, isto é, que tokens de segurança para propagar a mensagem e que tipo de proteção é necessária para o pedido.

O WS-Security define um formato padrão para o transporte de tokens de segurança e seu mapeamento para cabeçalhos SOAP; a especificação inclui um mesmo esquema XML para o efeito. WS-Security também reconhece determinados tipos de tokens de segurança, que é comum para a infra-estrutura de segurança. Atualmente o WS-Security tem cinco tipos de tokens de segurança reconhecidos como perfis no WS-Security.

- Um usuário token (<wsse: UsernameToken>) é uma forma de identificação. Ele pode ter uma senha, no caso o token também representa uma afirmação de que o requerent saiba a senha.
- Um certificado X.509 (<wsse: BinarySecurityToken>) é um direito sobre um ligação entre uma chave publica e seu conteúdo, apoiados por uma terceira parte confiável.
- Um ticket Kerberos (<wsse: BinarySecurityToken>) é uma solicitação que afirma: “Eu, o dono da chave da sessão contidas nesse ticket, que estou autorizado a fazer um pedido aos serviços especificados.

- Um *Security Assertion Markup Language* (SAML) a afirmação usa a sintaxe genérica da linguagem SAML para representar as afirmações relacionadas à segurança, tais como identidade, atributo e informações
- O *Rights Expression Language* (REL) é uma licença que representa a ISO / IEC 21000 – 5 expressões padrões com relação à especificação WS-Security.

O WS-Security também define um modelo de processamento de assinatura e criptografia de tokens de segurança de mensagens SOAP. Isso apóia a integridade dos tokens, impede sua falsificação e impede o acesso não autorizado às suas informações. (WEERAWARANA,2005)

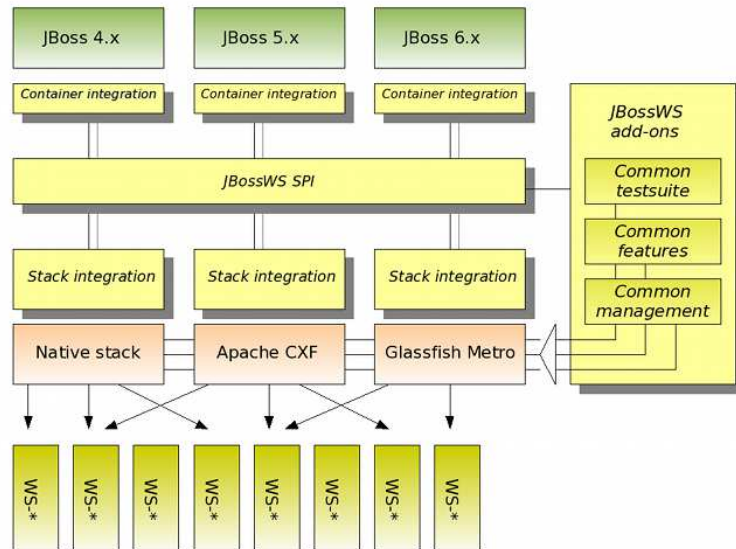
Para McGovern (2003) Vários algoritmos padrões da indústria, podem ser escolhidos para um mecanismo de criptografia personalizado. Um dos elementos a considerar é se os dados criptografados precisam ser decifrados, uma vez criptografado. Alguns algoritmos são apropriados para uma forma de criptografia, geralmente é usado em cenários de login e autenticação.

2.2.3 JBossWS

JBossWS é um framework *webservice* desenvolvido como parte do servidor de aplicações JBoss. Ele implementa a especificação JAX-WS, que define um modelo de programação e arquitetura em tempo de execução para implementação de *web services* em Java, direcionada para a plataforma Java, Enterprise Edition 5 (Java EE 5).

Como características o JBossWS incluem muitas especificações / implementações de normas bem como recursos adicionais, opções, ferramentas para melhorar a facilidade de uso, gerenciamento de *endpoints* e monitoramento.

Figura 13: Configurações, status e roadmap
Fonte: [HTTP://www.jboss.org/jbossws](http://www.jboss.org/jbossws)



2.3 ESB

Segundo Chappel (2004) Um ESB aplica serviços Web e outras normas complementares, combinando-os com conceitos de tecnologia e melhores práticas aprendidas com consultores de EAI. Serviços ESB podem ser configurados e não codificados. Fluxo de processo e invocações de serviço de forma transparente dentro de um barramento distribuído. Um ESB fornece um ambiente de integração altamente distribuído que se estende muito além do alcance de arquiteturas *hub-and-spoke* e uma clara separação de lógica de negócio e lógica de integração, tal como roteamento e transformação de dados.

Uma das responsabilidades do ESB é possibilitar aos consumidores chamar os serviços oferecidos pelos fornecedores. O papel principal do ESB é prover interoperabilidade.

Segundo Chappel (2007) um *Enterprise Service Bus* (ESB) é uma infraestrutura de software atuando como uma camada intermediária de *middleware* que atende aos requisitos estendidos, que geralmente não podem ser satisfeitos por web services e outras tecnologias de *middleware* e produtos, com maior nível de

dependência, robustez e segurança, gerenciamento e controle de serviços e sua comunicação.

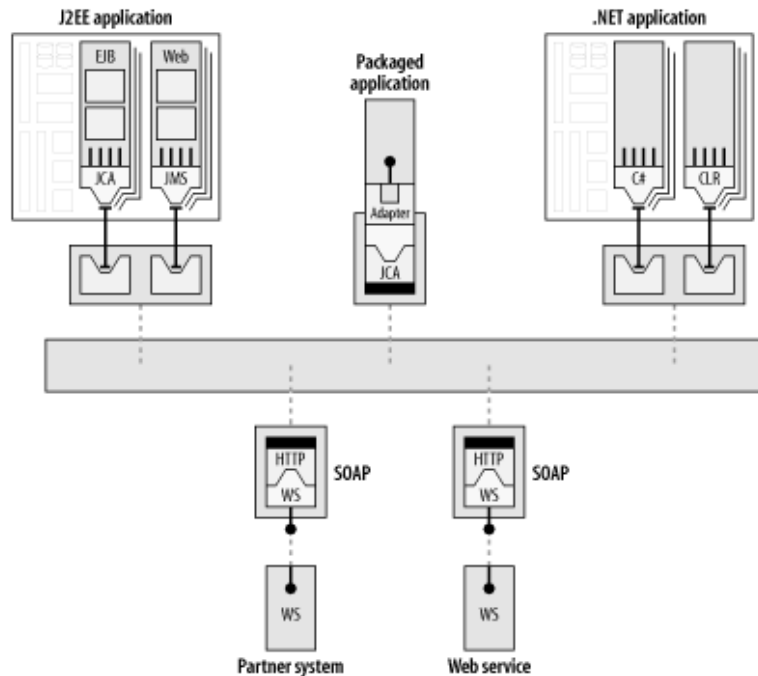
O ESB pode ser heterogêneo segundo Josuttis (2008) um aspecto que as tecnologias ESB diferem é a quantidade de acoplamento que elas provêem para as conexões físicas.

Aplicações dentro do barramento, conforme necessário são capazes de ter a visibilidade e compartilhar dados com outros aplicativos ou serviços que estão ligados no barramento. Apesar das interfaces *web services* serem uma parte integral de uma arquitetura ESB todos os aplicativos não precisam ser modificados para se tornarem *web services* verdadeiros que participam da ESB. A conectividade é conseguida através de vários protocolos, tecnologias API de clientes, ambientes de mensagens legadas, e adaptadores de aplicativos fornecidos por terceiros.

2.3.1 Integração baseada em padrões

Integração baseada em padrões é um conceito fundamental de um ESB. Para conectividade, um ESB pode utilizar componentes J2EE, como *Java message service* (JMS) para conectividade MOM, e *J2EE Conector architecture* (JCA ou J2CA) para conectar a adaptadores e aplicações. A ESB também é integrado a aplicações criadas com o .NET , COM, C# e C / C++. Abaixo a figura exemplifica um ambiente com os diferentes componentes.

Figura 14: Integração baseada em padrões
Fonte: Chappel,D (2004)

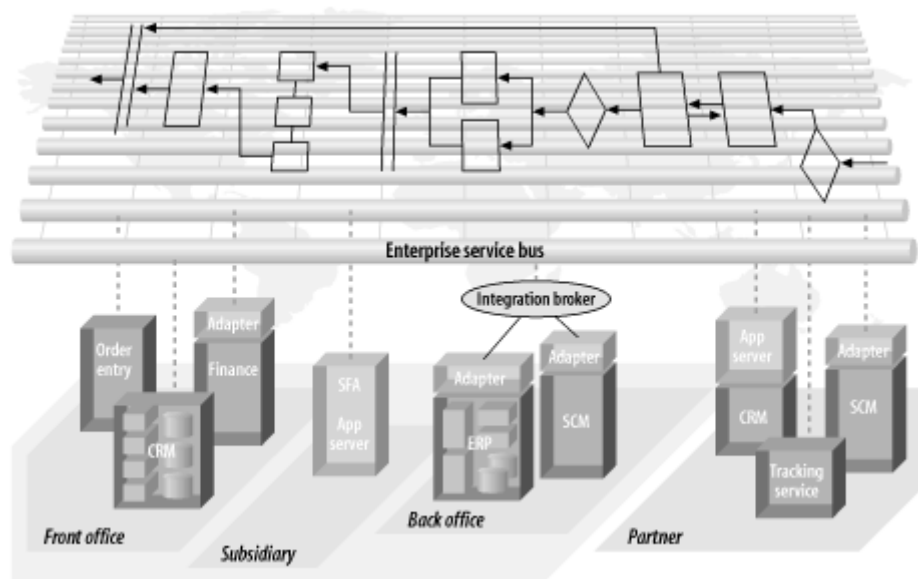


As capacidades do fluxo de processos de um ESB variam de simples seqüências de passos finitos para uma sofisticada orquestração dos processos de negócio com caminhos de processos paralelo que executam uma condicional que divide e une. Estes podem ser controlados por mensagens simples ou através da utilização de uma linguagem de orquestração como BPEL4WS.

O fluxo de processo também pode envolver a integração de serviços especializados que realizam roteamento inteligente de mensagens baseadas em conteúdo.

Como o fluxo do processo distribuído é construído em cima da SOA, também é capaz de abranger tecnologias de implantação altamente distribuídas como mostra a figura abaixo.

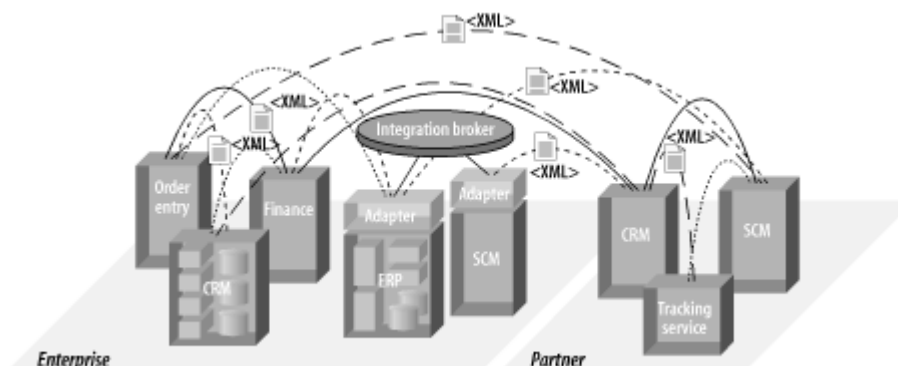
Figura 15: Orquestração e fluxo de processo
Fonte: Chappel,D (2004)



2.3.2 Adoção de XML

De acordo com (ESB) a utilização do formato XML como meio de intercâmbio de dados entre aplicações pode ter inúmeros benefícios. É possível fornecer um isolamento global e fazer alterações em estruturas e as interfaces das arquiteturas que necessitam de ajustes de uma só vez. Abaixo uma figura mostra a troca de dados XML entre as aplicações. (CHAPPEL,2004)

Figura 16: Utilização de XML compartilhando dados entre as aplicações
Fonte: Chappel,D (2004)



2.3.3 Message oriented Middleware

Segundo (ESB) *Enterprise Message* é o núcleo central de uma arquitetura ESB. A MOM é uma peça chave da arquitetura uma vez que fornece as bases da rede de canais virtuais que utiliza um ESB para encaminhar mensagens se estendendo por toda empresa e mais além.

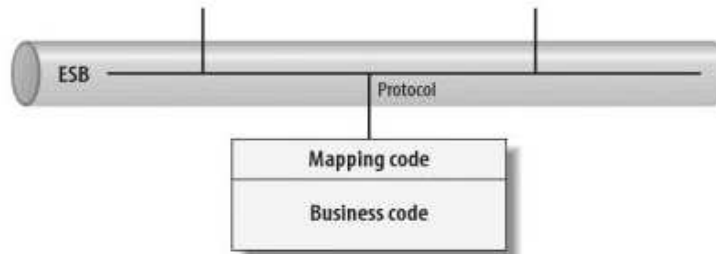
O sistema de mensagens é responsável por gerenciar os pontos de conexão entre várias mensagens clientes, e para o gerenciamento dos múltiplos canais de comunicação entre os pontos de comunicação. O sistema de mensagens geralmente é implementado como um processo de software comumente conhecido como servidor de mensagens ou corretor de mensagens. Servidores de mensagens são normalmente capazes de ser agrupadas para formar grupos que oferecem recursos avançados de balanceamento de carga, tolerância a falhas, e roteamento sofisticado utilizando gerenciamento de domínios de segurança.

2.3.4 ESB Orientado a protocolo

Segundo Josuitis (2006) o ponto de vista dos consumidores e fornecedores são importantes na utilização inicial de um ESB. Com a abordagem orientada a protocolo, o ESB define um protocolo e os consumidores e fornecedores enviam e

recebem as mensagens de acordo com esse protocolo. *WebService*, que requerem um protocolo SOAP, podem ser um exemplo dessa abordagem.

Figura 17: ESB orientado a protocolo
Fonte: Josuttis(2006)



2.3.5 ESB Orientado a API

Com a abordagem orientada a API, o ESB define API's específicas das plataformas (tais como Java), e os fornecedores e os consumidores utilizam essas API's para implementação de serviços e chamadas de serviços. Josuttis (2006)

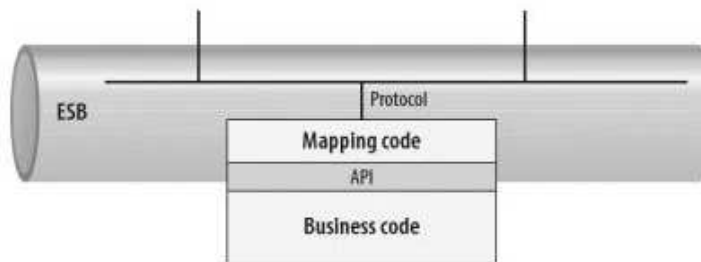


Figura 18: ESB orientado a API
Fonte: Josuttis(2006)

Quando um ESB define um protocolo, os detalhes de como os processos de consumo e fornecimento são desenvolvidos estão fora do escopo do ESB. A equipe de infra-estrutura não precisa fornecer as ferramentas e as bibliotecas para mapeamento das API's responsáveis pelas chamadas de serviços e as implementações de serviços para o protocolo requerido. (JOSUTTIS,2006)

Quando o ESB é responsável pelas API's o problema de mapeamento das chamadas de serviços e as implementações de serviços para as mensagens corretamente formadas que os participantes podem enviar através do ESB são

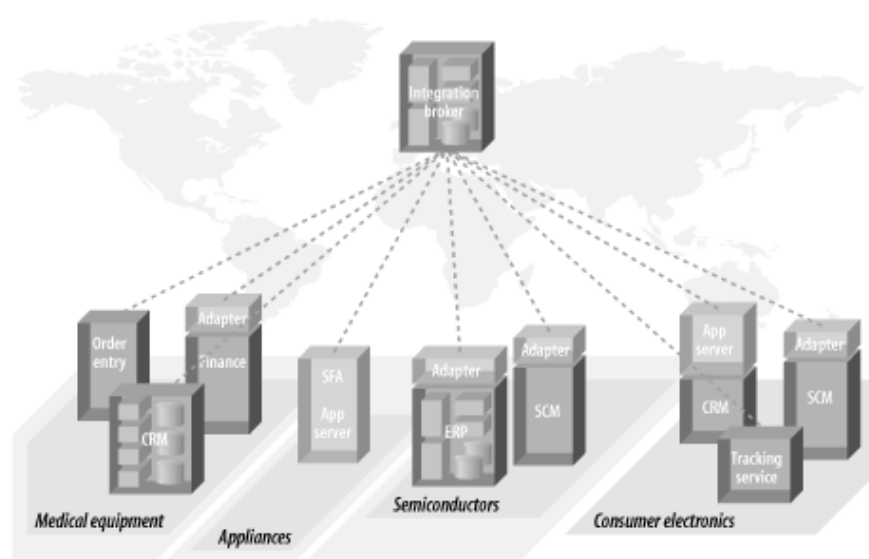
tarefas da equipe que provê o ESB. Os fornecedores e consumidores podem se concentrar nas funcionalidades do negócio. (JOSUTTIS,2006)

2.3.6 Segurança

As conexões entre os nós da ESB estão por trás do firewall. A segurança entre os aplicativos e o ESB , e até mesmo entre os nós do ESB, são capazes de estabelecer e manter as mais rígidas autenticações, gerenciamento de credenciais e controle de acesso. (CHAPPEL,2004)

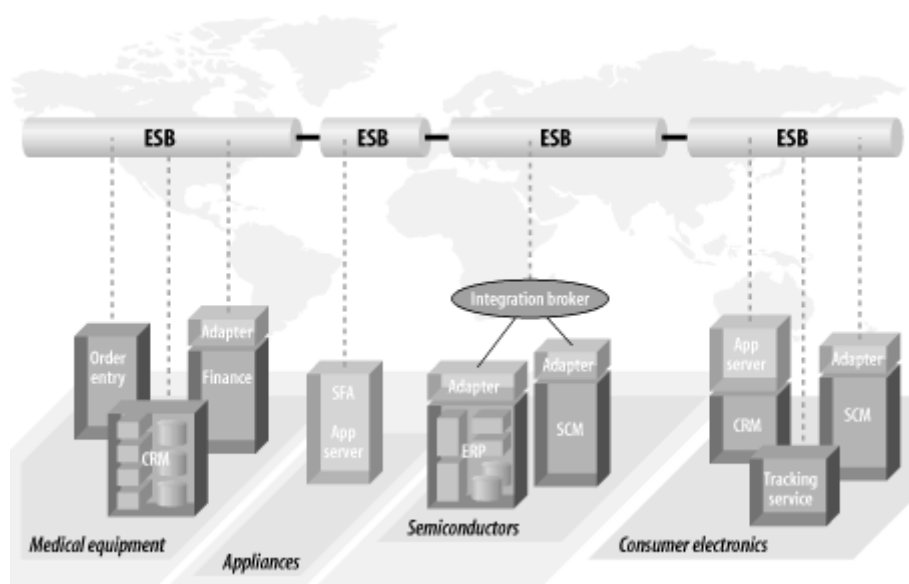
Chappel (2004) relata que em ambientes corporativos que centralizam a integração de todos os seus departamentos em um único ponto, é necessário que essa cultura seja mudada fazendo com que cada departamento ou unidade de negócio torne-se independente uma da outra. Em um ambiente de unidades de negócio flexíveis não faz sentido que os fluxos dos processos de negócios entre aplicações localizadas, ou domínios de segurança possam ser geridos por um único departamento de TI. Abaixo um exemplo desse ambiente.

Figura 19: Unidades de negócio com falta de autonomia.
Fonte: Chappel,D (2004)



Unidades de negócios locais e departamentos de TI precisam ter controle sobre os seus próprios recursos de TI, tais como aplicações hospedadas em seu site. O ESB fornece esse modelo de implantação permitindo o tráfego de mensagens locais, componentes de integração, adaptadores para serem instalados localmente, podendo ser capaz de ligar em conjunto os domínios de integração em uma rede mais abrangente com um modelo de segurança integrado. Abaixo uma figura detalhando esse ambiente. (CHAPPEL,2004)

Figura 20: ESB faz organização realizar operações cooperativamente.
Fonte: Chappel,D (2004)



2.3.7 JBossESB

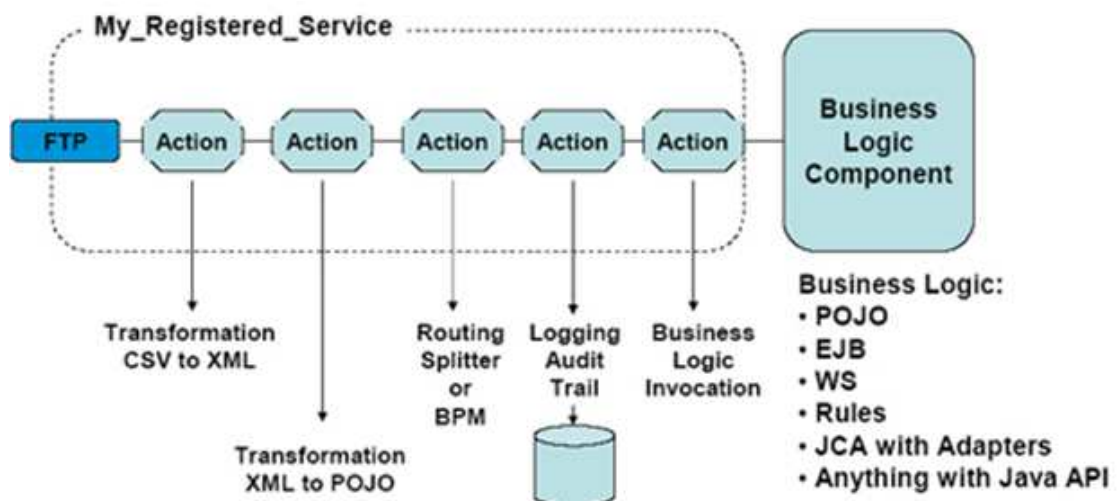
A equipe JBOSS criou seu próprio serviço de Barramento. JBOSS *Enterprise Service Bus* é uma solução SOA robusta projetada desde a versão 4.2 com *build-in fail-over*, entrega e reentrega de mensagens e balanceamento de carga. Permite distribuir as instâncias do serviço em vários nós. Cada nó pode ser uma máquina virtual ou física que executa uma ou mais instâncias do JBossESB.

2.3.7.1 Criação de serviço no JBossESB

No serviço JbossEsb é definida uma lista de classes de ação que processam uma mensagem ESB de forma seqüencial. Esta lista de classe de ação é conhecida como “*pipeline action*”. Um serviço pode definir uma lista de *listeners*, que funcionam como roteadores de entrada para o serviço, roteando mensagens para o *pipeline action*.

Figura 21: Action pipeline

Fonte: <http://www.mastertheboss.com/jboss-esb/78-jboss-esb.html>



JbossEsb suportam duas formas de configuração de *listener*:

- *Gateway Listeners*: Estas configurações do *listener* configuram *Gateway Endpoint*. Estes tipos de *Endpoint* podem ser usados para receber mensagens de um barramento ESB. Ele é responsável por adaptar a carga de mensagem por envolvimento dentro de uma mensagem ESB antes de enviá-lo para o *pipeline action*.
- *ESB Aware Listeners*: Esta configurações do *listener* configuram um *ESB Aware Endpoint*. Estes tipos de *EndPoint* são utilizados para trocar

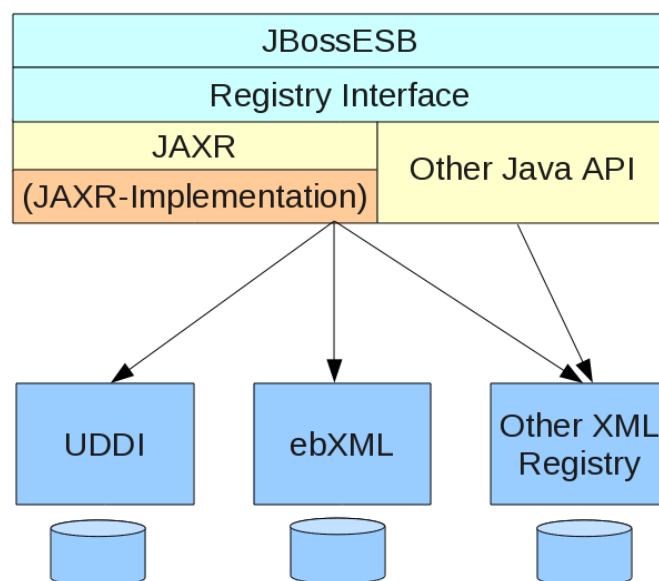
mensagens ESB entre os componentes ESB *Aware*, ou seja, troca de mensagens no barramento.

2.3.7.2 Registro

Um registro permite o registro do serviço, descoberta de metadados e classificação de entidades em categorias pré-definidas. Ao contrário de um repositório, ele não tem capacidade de armazenar definições de processos de negócio ou WSDL ou quaisquer outros documentos que são necessários para acordos comerciais. Um registro é essencialmente um catálogo de itens, enquanto que um repositório guarda estes itens.

O registro pode ser configurado de várias maneiras. A figura abaixo mostra em azul uma cópia de todos os componentes do registro. De cima para baixo podemos ver na figura toda interação JBossESB com o registro através da interface de Registro. Por padrão ele chama a implementação JAXR desta interface. A API JAXR necessita de uma aplicação, que por padrão é a *Scout*. A implementação JAXR chama um registro jUDDI. No entanto existem muitas outras opções de configuração.

Figura 22:Arquitetura de componentes do Registro
 Fonte: http://docs.jboss.org/jbossesb/docs/4.9/manuals/html/Services_Guide



2.3.7.3 Componentes SOA

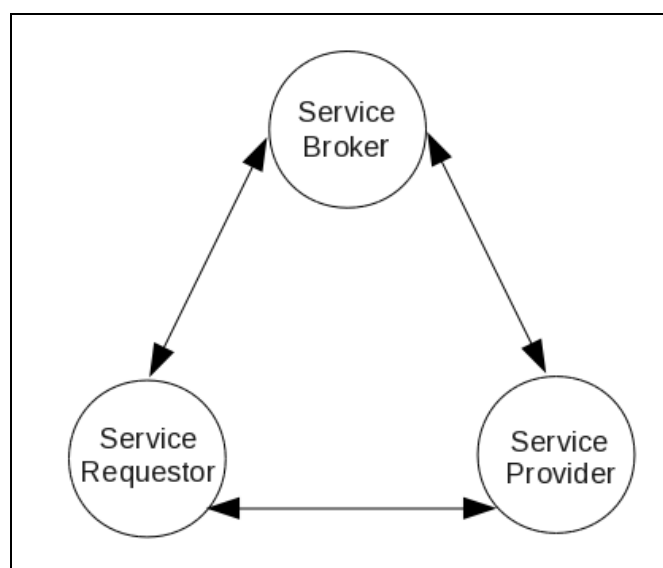
Os componentes chave de uma arquitetura orientada a serviços são as mensagens que são trocadas, os agentes que atuam como solicitantes e provedores de serviços, e mecanismos de transporte compartilhados que permitem o fluxo de mensagens. Uma descrição de serviço que existe dentro de uma SOA é apenas a descrição do caminho de troca de mensagens entre seus usuários. Dentro SOA existem três regras que são críticas.

Service Provider – Permite o acesso aos serviços, cria uma descrição de serviço e publica para o *Service Broker*.

Service Broker – Hospeda um registro de descrições de serviço. É responsável pela ligação entre um solicitante e para um *service provider*.

Service Requester – É responsável por descobrir um serviço, procurando através das descrições de serviço entregue pelo *service broker*. O solicitante também é responsável pela ligação com os serviços fornecidos pelo *service provider*.

Figura 23: Componentes SOA
Fonte: http://docs.jboss.org/jbossesb/docs/4.9/manuals/html/Services_Guide



2.4 RELAÇÃO ENTRE ESB E WEB SERVICES

Para (Josuttis 2006) o roteamento inteligente de um ESB é um recurso que não existe dentro de um *web service*.

O *Web service* acaba fazendo parte da infra-estrutura ESB por trabalhar fisicamente com conexões ponto-a-ponto a partir de um consumidor. Utilizando um *Web service* sozinho com falhas e sobrecarregado não existe opções para rotear a mensagens para *web services* alternativos.(JOSUTTIS,2008).

O ESB orientado a protocolo tem a vantagem de que cada consumidor e fornecedor podem usar ferramentas individuais para chamar e implementar serviços, ou seja, do ponto de vista do fornecedor e do consumidor, o ESB deve ser orientado a API. Qualquer modificação como introduzir interceptadores, mudar a versão do WSDL ou SOAP e outros, torna-se então transparente.

O que um ESB acrescenta é um ponto comum intermediário ou pontos que fazem com que o individuo possa evitar a rigidez das conexões ponto-a-ponto. Neste caso o ESB é efetivamente um switch que ligam vários aplicativos *Web services* e aplicações de legado, ou mesmo outros recursos como, banco de dados ou servidores de aplicações.

Binildas (2008) relata que para um consumidor a descrição do serviço (WSDL) pode ser vinculado a um ESB. E para isso, durante o tempo da conexão do serviço é necessário mais informações como protocolos de transporte.

2 METODOLOGIA ADOTADA

Para o desenvolvimento deste estudo, realizou-se uma pesquisa aplicada com abordagem qualitativa e com objetivo experimental. Segundo Prodanov e Freitas (2009), do ponto de vista de sua natureza, a pesquisa aplicada tem como objetivo gerar conhecimento para aplicação prática, a fim de solucionar problemas específicos, envolvendo verdade e interesses locais. Já do ponto de vista de seus objetivos, a pesquisa experimental quando “determinamos um objeto de estudo,

selecionamos as variáveis que seriam capazes de influenciá-lo, definimos as formas de controle e de observação dos efeitos que a variável produz no objeto” (PRODANOV e FREITAS, 2009, p.21). Segundo Prodanov e Freitas (2009, p.81), na abordagem qualitativa, a pesquisa tem o ambiente como fonte direta de dados. O pesquisador tem contato direto com o ambiente e o objeto de estudo em questão”.

Este trabalho realizará a avaliação de duas tecnologias: ESB e Web Service, relatando suas formas de integração, vantagens e desvantagens, além de identificar suas principais diferenças no ambiente. Para desenvolvimento de uma estrutura ESB em Java foi utilizada a ferramenta de desenvolvimento eclipse com a versão 4.9 do framework JBossESB, de forma que o resultado retratasse uma infraestrutura básica de uma implementação de ESB . Para desenvolvimento da estrutura web service foi utilizada a versão 6.0 do JBoss Application Server.

3 ANÁLISE DE RESULTADOS

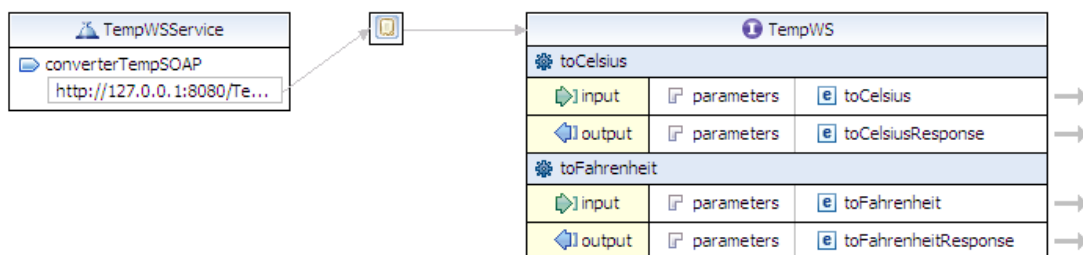
Foram realizadas implementações de aplicações Web Service em um ambiente Web Service e dentro de um ESB em Java, de forma que fossem identificadas seus principais componentes de acesso aos serviços.

4.1 IMPLEMENTAÇÃO WEB SERVICES

Para implementação do Web service, foi criado o projeto TempWS que cria um serviço dentro do servidor de aplicação JbossAS que tem a finalidade de transformar temperaturas de Celsius para Fahrenheit através de métodos desenvolvidos dentro do contrato Web service converterTemp.wsdl.

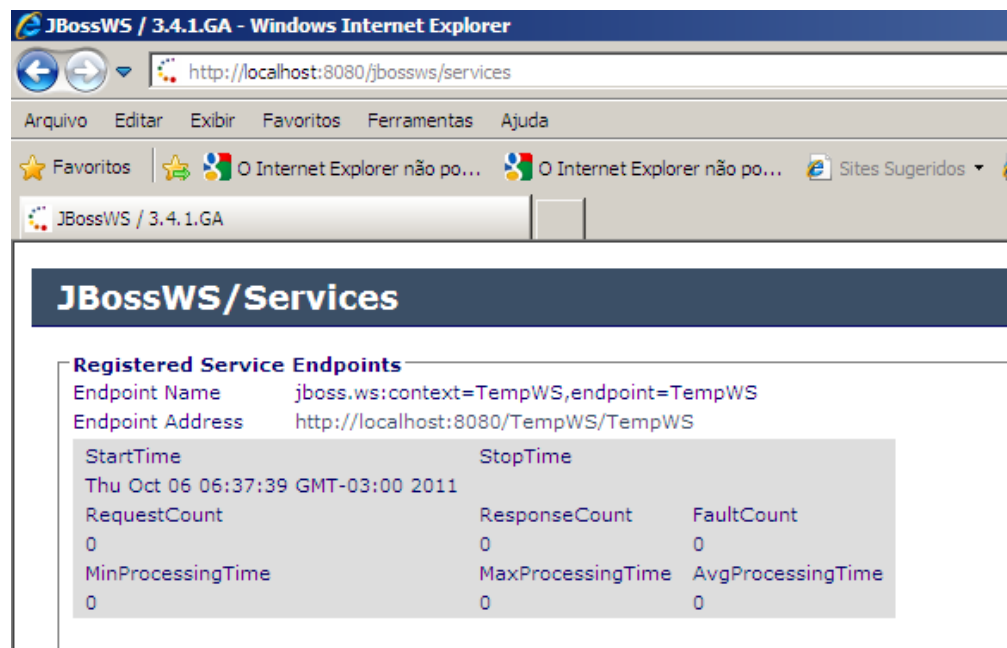
Dentro do contrato foram definidas as entradas e saídas para as chamadas feitas ao Web service.

Figura 24: converterTemp.wsdl
Fonte: fonte: Leonard, A (2009)



Após a instalação do serviço no JbossWS instalado no servidor Jboss Application Server 6.0 é possível visualiza-lo pela console do navegador pelo endereço <http://localhost:8080/jbossws/services> como mostra a figura abaixo:

Figura 25: Registro do serviço TempWS
Fonte: Leonard, A (2009)



The screenshot shows a web browser window titled "JBossWS / 3.4.1.GA - Windows Internet Explorer". The address bar shows "http://localhost:8080/jbossws/services". The page content is titled "JBossWS/Services" and displays "Registered Service Endpoints".

Endpoint Name	jboss.ws:context=TempWS,endpoint=TempWS		
Endpoint Address	http://localhost:8080/TempWS/TempWS		
StartTime	StopTime		
Thu Oct 06 06:37:39 GMT-03:00 2011			
RequestCount	ResponseCount	FaultCount	
0	0	0	
MinProcessingTime	MaxProcessingTime	AvgProcessingTime	
0	0	0	

4.2 IMPLEMENTAÇÃO DE WEB SERVICE EM UM ESB

Para representar a utilização de um webservice dentro de um ESB, foram desenvolvidos os projetos `webservice_consumer` e `web_service producer`.

4.2.1 Implementação do projeto webserver consumer

O Jboss Enterprise Service Bus (ESB) oferece a capacidade de construir solicitações SOAP e consumir respostas SOAP sem ter que gerar o código do proxy. Esta funcionalidade pode ser usada para invocar *endpoints* de *Webservices* externos dentro do ESB. **Cliente SOAP:** A presente ação cliente do cliente SOAP permite o consumo de *webservices* hospedados em *endpoints* externos. **SOAPProcessor:** A ação SOAPProcessor permite expor *endpoints* JBossWS através de terminais (*listeners*) em execução no ESB ("SOAP no barramento"). O

Webservice é representado no projeto `webService_consumer1` pela classe `HelloWorldWS.java`, como é mostrado na figura abaixo.

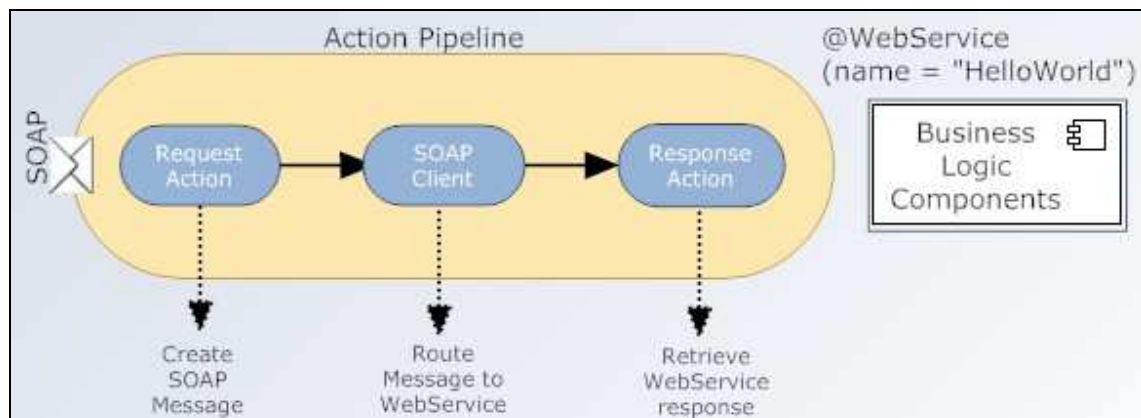
```
@WebService(name = "HelloWorld", targetNamespace =
"http://webService_consumer1/helloworld")
public class HelloWorldWS
{
    @WebMethod
    public String sayHello(@WebParam(name = "toWhom")
        String toWhom)
    {
        String greeting = "Hello World Greeting for '" + toWhom + "' on " +
new java.util.Date();

        return greeting;
    }
}
```

Este é apenas um simples *WebService* JAX-WS o *WebMethod* “sayHello” exposto. Para invocar o serviço é preciso fornecer três *actions* como mostrado na figura abaixo.

Figura 26: Action Pipeline

Fonte: <http://www.mastertheboss.com/jboss-esb/133-jboss-esb-ws-consumer.html>



A primeira *action* necessária para consumir o serviço é a *Request Action*: que se encarrega de converter a mensagem em um mapa de solicitações *WebService* como mostrado no código abaixo, que está no arquivo `jboss-esb.xml`.

```
<action name="request-mapper"
        class="org.jboss.soa.esb.samples.quickstart.webservice_consumer1.MyRequestAction">
```



```

    System.out.println("#####\n");
}
}

```

A segunda *action* no *pipeline* é a *action* `SoapClient` que roteia as mensagens para o `WebService`. É necessário definir duas propriedades para utilizar a *action* `SoapClient`.

- A url WSDL
- O `SOAPAction` (que é o `@WebMethod` exposto no `WebService`).

```

<action name="soapui-client-action"

    class="org.jboss.soa.esb.actions.soap.SOAPClient"><property
name="wsdl"
value="http://127.0.0.1:8080/Quickstart_webservice_consumer1/HelloWorldWS?w
sdl" /> <!--property name="wsdlTransform" value="/consumer1-wsdl-
transform.xml" / -->
    <property name="responseAsOgnlMap" value="true" />
<property name="SOAPAction" value="sayHello"/></action>

```

A classe `MyResponseAction` servirá para recuperar a mensagem soap.

A ultima ação do *pipeline* *action* é o *Response Action*..

```

<action name="response-mapper"

    class="org.jboss.soa.esb.samples.quickstart.pwebservice_consumer1.My
ResponseAction">
</action>

```

A classe abaixo não é diferente da *RequestAction*. O método *callback* captura o processo e a saída da resposta *WebService*.

```

package org.jboss.soa.esb.samples.quickstart.webservice_consumer1;

import org.jboss.soa.esb.actions.AbstractActionLifecycle;
import org.jboss.soa.esb.actions.ActionUtils;
import org.jboss.soa.esb.helpers.ConfigTree;
import org.jboss.soa.esb.message.Body;
import org.jboss.soa.esb.message.Message;
import java.util.Map;

public class MyResponseAction extends AbstractActionLifecycle
{
    protected ConfigTree _config;

    public MyResponseAction(ConfigTree config)
    {
        _config = config;
    }
}

```

```

public Message noOperation(Message message)
{
    return message;
}

/*
 * Retrieve and output the webservice response.
 */
public Message process(Message message) throws Exception
{
    logHeader();

    // The "responseLocation" property was set in jboss-esb.xml to
    // "helloworld-response"
    Map responseMsg = (Map) message.getBody().get(Body.DEFAULT_LOCATION);
    System.out.println("Response Map is: " + responseMsg);

    logFooter();
    return message;
}

public void exceptionHandler(Message message, Throwable exception)
{
    logHeader();
    System.out.println("!ERROR!");
    System.out.println(exception.getMessage());
    System.out.println("For Message: ");
    System.out.println(message.getBody().get());
    logFooter();
}

// This makes it easier to read on the console
private void logHeader()
{
    System.out.println("#####\n");
}

private void logFooter()
{
    System.out.println("#####\n");
}
}

```

Após o servidor JBossESB 4.9 ser iniciado é possível acessar a informação do serviço no registro do JBossWS pelo endereço <http://localhost:8080/jbossws/services> . Abaixo uma figura ilustrando o registro do serviço.

Figura 27: Registro do serviço HelloWorldWS no JBossWS

Endpoint Name	jboss.ws:context=Quickstart_webservice_consumer1,endpoint=HelloWorldWS		
Endpoint Address	http://localhost:8080/Quickstart_webservice_consumer1/HelloWorldWS?wsdl		
StartTime	StopTime		
Tue Jul 05 13:03:37 GMT-03:00 2011			
RequestCount	ResponseCount	FaultCount	
0	0	0	
MinProcessingTime	MaxProcessingTime	AvgProcessingTime	
0	0	0	

Abaixo o contrato WSDL que pode ser acessado pelo link *Endpoint Address* http://localhost:8080/Quickstart_webservice_consumer1/HelloWorldWS?wsdl.

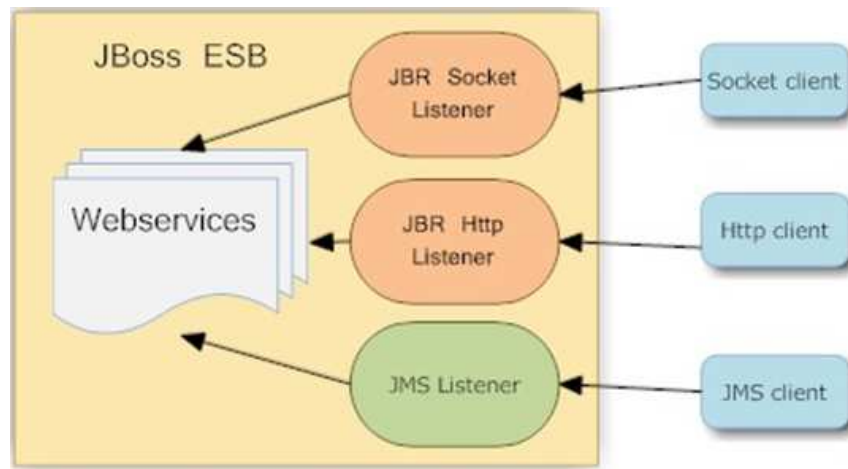
4.2.2 Implementação do projeto WebService producer

O *Soapprocessor action*, conhecido como JBossAdapter, permite que seja exposto o JBossWS nas extremidades superiores dos *endpoints* dos *Web Services* através da execução de *listeners* (SOAP no barramento).

O JBossESB utiliza *JBoss Remoting Interfaces* como provedor para requisições HTTP e transporte via *socket*. A fim de receber os pedidos HTTP / *socket* é necessário instalar um *listener* para estas interfaces. Isto é conseguido com a tag `<jbr-listener>`.

```
<listeners>
<jms-listener name="JMS-Gateway" busidref="quickstartGwChannel" is-
gateway="true"/>
<jbr-listener name="Http-Gateway" busidref="Http-1" is gateway="true"/>
<jbr-listener name="Socket-Gateway" busidref="Socket-1" is-gateway="true"/>
<jms-listener name="JMS-ESBListener" busidref="quickstartEsbChannel"/>
</listeners>
```

Figura 28: JBossESB expando Web Services sobre HTTP,socket e protocolos jms
 Fonte: <http://www.mastertheboss.com/jboss-esb/137-jboss-esb-webservice-producer.html>



As medidas para chamar um *Webservice* são simples. A *action* requer apenas uma propriedade com valor obrigatório, que é o “*endpoint*” propriedade do JBossWS . O SOAPProcessor estará invocando os nomes contidos nas propriedades no JBossWS. O *endpoint* do *WebService* é implantado como aplicativo web no arquivo .Esb . É composto de uma classe com três *WebMethods* expostos, como mostrado no código abaixo.

```
@WebService(name = "GoodbyeWorldWS",
targetNamespace="http://webservice_producer/goodbyeworld")
// @SOAPBinding(style = SOAPBinding.Style.RPC)
public class GoodbyeWorldWS {

    @WebMethod
    public String sayGoodbye(@WebParam(name="message") String message) {

        Message esbMessage = SOAPProcessor.getMessage();
        if(esbMessage != null) {
            System.out.println("**** SOAPRequest perhaps mediated by
ESB:\n" + esbMessage.getBody().get());
            // System.out.println("\n" + esbMessage.toString() + "\n");
        }
        System.out.println("Web Service Parameter - message=" + message);
        return "... Ah Goodbye then!!!! - " + message;
    }

    @WebMethod
    public String sayAdios(String message) {
        Message esbMessage = SOAPProcessor.getMessage();
        if(esbMessage != null) {
            System.out.println("**** SOAPRequest perhaps mediated by
ESB:\n" + esbMessage.getBody().get());
            // System.out.println("\n" + esbMessage.toString() + "\n");
        }
    }
}
```

```

    }
    System.out.println("Web Service Parameter - message=" + message);
    return "... Adios Amigo!!!! - " + message;
}

@WebMethod
@Oneway
public void sayGoodbyeWithoutResponse(@WebParam(name="message") String
message) {

    Message esbMessage = SOAPProcessor.getMessage();
    if(esbMessage != null) {
        System.out.println("**** SOAPRequest perhaps mediated by
ESB:\n" + esbMessage.getBody().get());
    }
    System.out.println("Web Service Parameter - message=" + message);
}
}

```

Abaixo o registro do serviço GoodbyeWorldWS.

Figura 29: Registro do serviço GoodbyeWorldWS

Endpoint Name	jboss.ws:context=Quickstart_webservice_producer1,endpoint=GoodbyeWorldWS		
Endpoint Address	http://localhost:8080/Quickstart_webservice_producer1/GoodbyeWorldWS?wsdl		
StartTime	StopTime		
Tue Jul 05 13:03:38 GMT-03:00 2011			
RequestCount	ResponseCount	FaultCount	
0	0	0	
MinProcessingTime	MaxProcessingTime	AvgProcessingTime	
0	0	0	

Abaixo a o resultado da execução da classe SendMessage.java

Figura 30: Saida da classe SendMessage.java

```

Calling JBoss Remoting Listener using locator URI: http://localhost:8765
JBR Class: java.lang.String
Response from JBoss Remoting Listener 'http://localhost:8765' was '-i00ysr0%org.jboss.remoting.InvocationResponseDa
|

```

5 CONCLUSÃO

Um WebService por si mesmo trata de uma estrutura que disponibiliza serviços individualmente através de contratos feitos entre o cliente e o fornecedor do serviço, não oferecendo os mesmos recursos de um ESB para integração de sistemas. Após a implementação de exemplos *webservice* na plataforma JBoss foi possível observar todos os recursos oferecidos pelo framework se mostra compatível com a abordagem SOA. Nele é possível integrar várias aplicações de diferentes plataformas em um barramento ESB. Uma estrutura que é responsável pelo recebimento, roteamento e transformação da mensagem. Nas implementações dos projetos *WebService_consumer* e *Webservice_producer* podemos ver duas alternativas de utilização de *Webservice* em um barramento ESB. As implementações de segurança também não são as mesmas para o *Webservice* e o ESB. Sendo assim em um ESB podemos ter vários *webservices* funcionando como *endpoints* consumindo e fornecendo serviços, além de oferecer integração com aplicações através de outros protocolos como HTTP, JMS, FTP e outros. O JBossESB disponibiliza conexões para WebService e outras aplicações através dos *listeners*, que são a porta de entrada para o barramento ESB.

REFERENCIAS

BINILDAS,C,A.**Service Oriented Java Business Integration**: Enterprise Service Bus integration solutions for Java Developers, UK: Packt Publishing, 2008.

CERAMI,E. **Web Services Essentials**. USA: O'Relly, 2002.

CHAPPEL,D. **Entrerpise Service Bus**: Theory in Practice.USA:O'Relly, 2004.

CHAPPEL,D. **Web service Contract Design e Versioning for SOA**. USA: Boston,Pearson Education,2008.

Erl,T.**SOA Principles of service design**. USA:Prentice Hall. 2006.

HOHPE,G; WOOLF,B. **Enterprise Integration Patterns**: Design, Building, and Deployng Messaging Solutions. USA: Addison-Wesley Professional,2003

JOSUTTIS,N. **SOA na prática**: A Arte da modelagem de Sistemas distribuídos. São Paulo: Orelly,2008.

JBoss ESB Webservice Producer. Disponível em <http://www.mastertheboss.com/jboss-esb/137-jboss-esb-webservice-producer.html>> Acesso em 05 mai. 2011

JBossWS. Disponível em: <[HTTP://www.jboss.org/jbossws](http://www.jboss.org/jbossws)> Acesso em 05 mai. 2011.

JBoss ESB Tutorial. Disponível em:<http://www.mastertheboss.com/jboss-esb/78-jboss-esb.html>> Acesso em 05 de mai. 2011

JBoss ESB Service Guide. Disponível em : http://docs.jboss.org/jbossesb/docs/4.9/manuals/html/Services_Guide/> Acesso em 04 de Jul. 2011

JBoss ESB Programmers Guide. Disponível em:
http://docs.jboss.org/jbossesb/docs/4.9/manuals/html/Programmers_Guide> Acesso
 em 04 de Jul. 2011

JURIC,M; et al. **SOA Approach to Integration**: XML, Web Services, ESB, and BPEL in real-world SOA projects.USA: Packt Publishing, 2007

LEONARD,A. **JBoss Tools 3 Developer's Guide**. Birmingham, UK: Packt Publishing, 2009.

MCGOVERN,J;et al. **Java Web Service Architecture**. USA:Elsevier Science, 2003.

Novaes, M.**DESAFIOS DO DESENVOLVIMENTO DE ARQUITETURA ORIENTADA A SERVIÇOS (SOA) PARA AUTOMAÇÃO DE PROCESSO DE NEGÓCIO UTILIZANDO BPEL4WS** (Faculdade ETAPA, São Paulo, Brasil, IPT : Instituto de Pesquisas Tecnológicas, São Paulo, Brasil, 2008) .

WEERAWARANA,S;et al. **Web Services Platform Architecture**: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. USA:Pearson Education, 2005.