UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL**

GABRIEL LAINO MARTINELLI FURLAN

SOA NA PRÁTICA: DESENVOLVIMENTO DE UM SERVIÇO UTILIZANDO HORNETQ

Bandeirantes

2012

GABRIEL LAINO MARTINELLI FURLAN

SOA NA PRÁTICA: DESENVOLVIMENTO DE UM SERVIÇO UTILIZANDO HORNETQ

Trabalho de Conclusão de Curso apresentado à – Universidade Estadual do Norte do Paraná – UENP, *campus* Luiz Meneghel, Centro de Ciências Tecnológicas, como requisito parcial para aprovação na disciplina Projeto Final II do Curso de Sistemas de Informação.

Orientador: Prof. Estevan Braz Brandt Costa

Bandeirantes

2012

GABRIEL LAINO MARTINELLI FURLAN

SOA NA PRÁTICA: DESENVOLVIMENTO DE UM SERVIÇO UTILIZANDO HORNETQ

Trabalho de Conclusão de Curso apresentado à - Universidade Estadual do Norte do Paraná – UENP, *campus* Luiz Meneghel, Centro de Ciências Tecnológicas, como requisito parcial para aprovação na disciplina Projeto Final II do Curso de Sistemas de Informação.

COMISSÃO EXAMINADORA

_____
Orientador: Prof. Estevan Braz Brandt Costa
UENP – c*ampus* Luiz Meneghel

_____
Prof. Me. Ricardo Gonçalves Coelho
UENP – c*ampus* Luiz Meneghel

_____
Prof. Me. Bruno Miguel Nogueira de Souza
UENP – c*ampus* Luiz Meneghel

Bandeirantes, __ de_____ de 2012

Dedico este trabalho à minha família, aos amigos de Piraju e aos meus amigos do NTI - UENP.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a DEUS, pois, independente daqueles que crêem ou não, ELE sempre me deu forças para continuar batalhando. ELE não disse que seria fácil, mas que valeria a pena!

Agradeço à minha família por ter me educado e ensinado a sempre agir para o bem, por ter me apoiado quando eu mais precisei, por ter me sustentado quando eu já era capaz de fazê-lo. Em especial à minha mãe Maria de Fátima, ao meu pai José Roberto, aos meus tios, Maria Alice e José Carlos, aos meus primos, José Renato e Luiz Fernando, à minha avó Diva, e a todos os outros familiares não citados. Obrigado, FAMÍLIA.

Agradeço aos meus amigos, que mesmo longe, deram todo apoio que precisei para dar continuidade aos estudos. Em especial aos *Bullocks*, Mateus, João Paulo, Breno, Rodrigo, Rafael.

Agradeço aos meus amigos de faculdade e toda a minha turma por terem me ajudado em diversos trabalhos, sempre em parceria. Em especial aos amigos de Avaré, Ricardo Crivelli Barbosa, Thiago Gaspar e Fabrício Batista.

Agradeço aos meus irmãos de trabalho, que, mais do que amigos, fizeram parte do profissional que me tornei. Devo a eles toda experiência adquirida nesses quatro anos de universidade e estágio, não apenas pelas diversas situações e problemas enfrentados e, juntos, solucionados, mas como pelas alegrias e diversões diárias, sem as quais não teria feito nossa proximidade tão grande. Em especial ao Luiz Fernando Legore do Nascimento, ao Vinícius Rodrigues Silva, ao Wellington Della Mura, ao Italo Ruy Fernandes, ao Micael Mercadante, e a toda equipe que participara do Núcleo de Tecnologia da Informação da Universidade Estadual do Norte do Paraná.

Agradecimento especial a todos os professores do curso de Sistemas de Informação por toda atenção e apoio dado dentro e fora da sala de aula.

Agradeço ao meu orientador, Estevan, por ter me dado suporte e estado ao meu lado mesmo nas dificuldades e obstáculos encontrados no decorrer deste trabalho.

Obrigado a todos os supracitados e os que, por descuido, não foram citados.

“O único homem que está isento de erros, é aquele que não arrisca acertar.”
(Albert Einstein)

RESUMO

O mercado de sistemas de informação vem crescendo gradativamente devido ao avanço da tecnologia e das necessidades dos negócios. Com isso, a arquitetura orientada a serviços propõe um crescimento integrado dos sistemas, e estes desenvolvidos na forma de módulos de serviços. O presente trabalho demonstra uma breve abordagem sobre a arquitetura orientada a serviços, as formas de torná-la real com algumas das arquiteturas de softwares disponíveis no mercado, as arquiteturas de integração de *softwares* utilizando a camada de mensagens, bem como a plataforma Java EE de desenvolvimento, o servidor de aplicação JBoss e seu *middleware* de mensagens HornetQ. Todo desenvolvimento foi voltado para o desenvolvimento do serviço de dados acadêmicos da Universidade estadual do Paraná.

Palavras-chave: JavaEE, SOA, HornetQ, JBoss AS, MOM.

ABSTRACT

The market for information systems has been increasing gradually due to the advancement in technology and business needs. With that, the service-oriented architecture proposes a growth of integrated systems, and these modules developed in the form of services. The present work demonstrates a brief overview on the service-oriented architecture, ways to make it real with some of the architectures of software available in the market, the integration of software architectures using messaging layer, as well as the Java EE platform development, the JBoss application server and its HornetQ messaging middleware. All development was focused on the development of data service academics from the University of Paraná state.

Keywords: JavaEE, SOA, HornetQ, JBoss AS, MOM.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

AS – *Application Server*

CLM – *Campus* Luiz Meneghel

DAO – *Data Access Object*

EJB – *Enterprise Java Beans*

HTTP – *Hypertext Transfer Protocol*

IDE – *Integrated Development Environment*

Java EE – *Java Platform, Enterprise Edition*

Java SE – *Java Platform, Standard Edition*

JBoss – *Java Beans Open Source Software*

JBossMQ – *JBoss Message Queue*

JDBC – *Java Database Connection*

JDK – *Java Development Kit*

JMS – *Java Message Service*

JNDI – *Java Name and Directory Interface*

JPA – *Java Persistence API*

JRE – *Java Runtime Environment*

JSON – *JavaScript Object Notation*

JVM – *Java Virtual Machine*

MDB – *Message Driven Bean*

MDP – *Message Driven POJO*

MOM – *Message-oriented Middleware*

ORM – *Object-relational Mapping*

POJO – *Plain Old Java Object*

SGBD – Sistema Gerenciador de Banco de Dados

SI – Sistemas de Informação

SOA – *Service-oriented Architecture*

SOAP – *Simple Object Access Protocol*

TI – Tecnologia da Informação

UDDI – *Universal Description, Discovery, and Integration*

UENP – Universidade Estadual do Norte do Paraná

WSDL – *Web Service Description Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Quando se trata de uma instituição, empresa ou corporação existem aplicações ou sistemas que auxiliam as atividades dos diversos setores, como logística, financeiro, vendas, produção, marketing, recursos humanos, etc. No entanto, a busca pela melhoria contínua e agilidade nos processos de negócio atribui ao setor de Tecnologia da Informação a necessidade de que os ativos de *software* da organização sejam tão flexíveis às mudanças do negócio quanto possível, e esse é o principal objetivo da Arquitetura Orientada a Serviço (Sivakumar *et al*, 2010).

A arquitetura SOA é também conhecida como *building blocks*, ou construindo blocos, onde cada bloco é um serviço (Koch, 2006). Tais blocos auxiliam a construção de várias estruturas somente pelo reajuste dos blocos, sendo possível adequar as aplicações com a estratégia da empresa apenas reagrupando os serviços.

Segundo Koch (2006), SOA define que um sistema deve partilhar suas informações de forma reutilizável, onde as disponibiliza utilizando algum meio de comunicação para que outras aplicações se integrem a ele. A integração consiste na combinação de partes que trabalham isoladamente, formando um conjunto que trabalha como um todo, ou o ato de fazer com que algo pertença a um todo. Desta forma, o papel da integração dos ativos de *software* é auxiliar o compartilhamento de dados comuns entre eles, de tal forma que seja possível tanto os sistemas já existentes, quanto os novos sistemas fazer uso de tais dados (Zhao, 2006).

Para a realização dessa integração são utilizados diversos padrões que alinham os sistemas aos negócios da organização. Uma das principais e mais utilizadas formas de comunicação entre as aplicações é troca de mensagens. Nesse caso, há um produtor e um ou mais consumidores, e cabe ao **sistema de mensagens** a tarefa de **garantir a entrega** das mensagens entre um ponto e outro (Hohpe & Woolf, 2003). Esses sistemas são chamados de *Message-oriented Middleware*, e um dos mais utilizados é o HorneQ.

O presente trabalho visa desenvolver um serviço baseado na arquitetura SOA, tornando-o integrável, utilizando o MOM HornetQ como sistema de comunicação por mensagens.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um modelo de solução para o problema da integração entre os diferentes *software* da UENP, integrando a base do sistema acadêmico atual ao serviço de dados acadêmicos (aluno, disciplina e curso), utilizando o *middleware* orientado a mensagens HornetQ.

1.1.2 Objetivos Específicos

1) Descrever os principais padrões para desenvolvimento de aplicações integradas;
2) Estudar o modelo de integração entre a aplicação e o sistema de mensagens;
3) Desenvolver a integração entre a base do atual sistema acadêmico e o sistema de mensagens;
4) Reduzir o custo de desenvolvimento ao empregar a integração entre os sistemas de uma organização.

1.2 JUSTIFICATIVA

O presente trabalho tem a intenção de melhorar a integração entre os diferentes setores da instituição, motivando a criação de diversos serviços que irão auxiliar os processos comuns de negócio, proporcionando flexibilidade na TI e agilidade no negócio.

Devido às dificuldades, como modificar os sistemas já existentes, dentre eles o sistema acadêmico, o bibliotecário, e o de recursos humanos, com o intuito de extrair suas informações, a falta de padronização no esqueleto das informações e a necessidade de compreender o funcionamento de cada sistema, tomou-se a decisão de utilizar a arquitetura orientada a serviço como modelo de solução.

Para isso, foi necessário escolher a forma pela qual seria feita a integração entre os diversos *softwares*, sendo escolhido o *middleware* orientado a mensagens, o qual traz a vantagem do aumento da disponibilidade para uma aplicação requerer informações à outra, mesmo que esta não esteja disponível no momento da requisição.

1.3 METODOLOGIA

Para a realização deste trabalho, utilizou-se como base os diferentes padrões de integração entre aplicações, as formas de desenvolvimento baseado na arquitetura orientada a serviço e casos similares vivenciados por outras instituições. Assim, com base na análise do serviço proposto, a criação do modelo padrão para a organização dos dados acadêmicos e a criação do pacote padrão para comunicação por mensagens, fazem parte dos resultados desejados.

1.4 ORGANIZAÇÃO DO TRABALHO

O restante do trabalho está organizado da seguinte forma. O capítulo 2 apresenta a fundamentação teórica, onde serão detalhados os padrões de integração, abordando de forma mais detalhada os fundamentos de um *middeware* orientado a mensagens. Neste capítulo ainda consta a arquitetura orientada a serviço, duas formas de criação e utilização de um serviço, *web service* e Java EE, e algumas informações sobre o servidor de aplicação e o sistema de mensagem que foram utilizados.

O capítulo 3 descreve o desenvolvimento do modelo de dados acadêmicos, a coleta dos dados, a preparação do ambiente de execução, a criação do pacote padrão para o envio e recebimento de mensagens, e a criação do serviço. O capítulo 4 apresenta os resultados obtidos, no capítulo 5 as considerações finais e sugestões para futuros trabalhos e finalmente, no capítulo 6, as referências de toda a pesquisa do presente trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta as principais tecnologias, modelos e padrões envolvidos aos ambientes de integração, SOA e o desenvolvimento de sistemas na forma de serviços, bem como o padrão DAO.

2.1 PADRÕES DE ACESSO À DADOS

Armazenar os dados de maneira segura, confiável e torná-la acessível apenas para pessoas ou sistemas autorizados, é uma responsabilidade muito grande e, com a chegada dos sistemas de informação, fora atribuída aos Sistemas Gerenciadores de Banco de Dados. Como a informação é de grande, senão, de maior importância para as organizações, as empresas desenvolvedoras destes sistemas buscam a melhoria de suas técnicas no armazenamento de dados (Date, 2004).

2.1.1 *Data Access Object*

A fim de tornar o acesso aos dados simples, provendo algumas operações específicas e, sem expor detalhes sobre a base de dados, fora criado o padrão *Data Access Object*. Para a aplicação DAO ter acesso a uma base de dados ela necessita de um conector padrão (JDBC) para a manipulação de dados e acesso exclusivo ao SGBD utilizado para gerenciar essa base (Sun Microsystems, 2001-2002).

2.1.2 *Object-relational Mapping*

Devido à necessidade de tornar esse acesso aos dados mais transparente, de fácil reutilização e baixo custo de programação, algumas ferramentas foram desenvolvidas para criar mapeamentos de objetos ligados a uma base de dados relacional. Os *frameworks* ORM, como são chamados, utilizam arquivos de configuração, em sua maioria no formato XML, que contém os campos da base e os seus respectivos atributos de cada classe (Linwood & Minter, 2010). Os mais conhecidos são Hibernate, da JBoss e MyBatis, antigo iBatis, um dos projetos da *Apache Foundantion.*

2.2 PADRÕES DE INTEGRAÇÃO

Considerando o fato das redes de computadores (interna ou externa) serem utilizadas pelas soluções de integração como meio de comunicação, alguns desafios estão acoplados (Hohpe & Woolf, 2003), tais como:

1) Redes são inconfiáveis - pois o transporte de dados entre computadores separados depende da rede, incluindo toda sua arquitetura, compreendendo seus meios de transmissão como linhas telefônicas, roteadores, *switches*, redes públicas, fibras ópticas, conexões via satélite, etc.
2) Redes são lentas – desenvolver uma aplicação largamente distribuída com uma abordagem de aplicação única pode trazer vários problemas de desempenho.
3) Quaisquer duas aplicações são diferentes – os *softwares* são desenvolvidos em diferentes linguagens, plataformas e formato de dados. Uma solução de integração deve estar apta a criar uma interface entre essas diferentes tecnologias.
4) Mudança é inevitável – aplicações mudam o tempo todo, assim as soluções de integração têm de minimizar as dependências entre um sistema e outro utilizando baixo acoplamento entre eles.

Com base nesses desafios, eis algumas abordagens para a troca de informações.

1) Transferência de arquivo – uma aplicação escreve um arquivo para depois outra ler.
2) Base de dados compartilhada – varias aplicações compartilham de uma mesma base de dados, fisicamente única. Assim nenhum dado tem de ser transferido de uma aplicação para outra.
3) Chamada de processo remoto – uma aplicação disponibiliza alguma de suas funcionalidades que pode ser acessada remotamente por outra aplicação. A comunicação ocorre em tempo real e de forma síncrona.
4) Mensagem (*messaging*) – Uma aplicação publica uma mensagem em um canal e outra pode ler a mensagem num momento posterior. As aplicações devem estar de acordo com o canal assim como o formato da mensagem. A forma de comunicação é assíncrona.

Desta forma, as características das principais abordagens para troca de informações demonstram a necessidade de passar o papel da transferência a uma camada inferior às aplicações, conhecida como *middleware.* Tal camada é responsável pela garantia da entrega das mensagens, independentemente dos possíveis problemas ocasionados no caminho percorrido (Hohpe & Woolf, 2003).

2.3 ARQUITETURAS DE INTEGRAÇÃO

Normalmente aplicações interessantes raramente vivem isoladas, e estas aplicações não funcionam de forma isolada quando necessitam de uma informação que outra aplicação gera. Porém, há a necessidade de utilizar um meio de comunicação para obter essa informação. Isso traz à tona a mudança na forma em que o *software* é projetado (Hohpe & Woolf, 2003).

2.3.1 Message-oriented Middleware (MOM)

Um sistema de mensagens gerencia mensagens assim como um sistema gerenciador de banco de dados gerencia a persistência dos dados (Hohpe & Woolf, 2003). Com essa definição é possível compreender a importância de um sistema de mensagens, cuja principal tarefa é mover a mensagem do computador emissor para o computador receptor de uma forma confiável, assim a comunicação entre eles é realizada por meio de um canal.

Dessa forma, a entrega da mensagem envolve algumas fases, componentes e atores.

Atores:

1) Computador 1 – Aplicação emissora;
2) Computador 2 – Aplicação receptora;
3) Canal (fila ou tópico).

Componentes:

1) Dados – os dados que serão transmitidos;
2) Mensagem com os dados – mensagem com os referidos dados;

3) Armazém de mensagens – local, dentro do canal, onde a mensagem fica armazenada.

Fases:

1) Criar – o emissor cria a mensagem e preenche com os dados;
2) Enviar – o emissor adiciona a mensagem ao canal;
3) Entregar – o sistema de mensagens move a mensagem do emissor para o receptor, tornando-a disponível ao receptor;
4) Receber – o receptor recebe a mensagem do canal;
5) Processar – o receptor extrai os dados da mensagem;

Esse processo pode ser mais facilmente compreendido conforme a Figura 1.

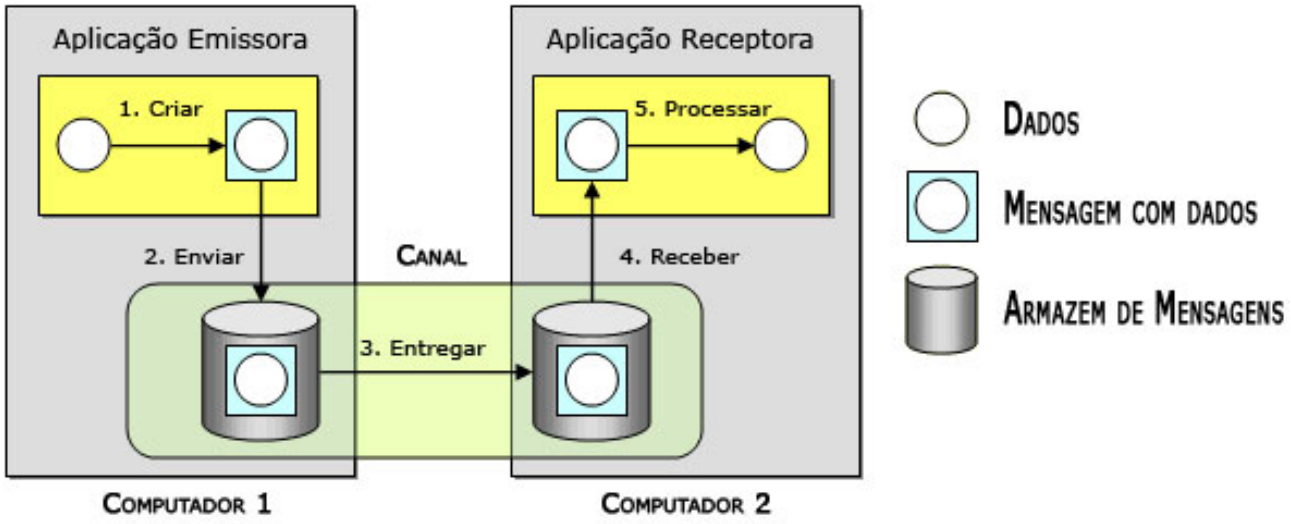

Figura 1 – O passo a passo da transmissão de mensagem.
Fonte: adaptado de Hohpe & Woolf (2003).

Outros dois conceitos importantes sobre o padrão de integração de aplicações por mensagens, ilustrados na Figura 1, são o *send and forget* e o *store and forward* (Hohpe & Woolf, 2003), e ocorre no passo Enviar, que é quando a aplicação emissora envia a mensagem ao canal.

O *send and forget* (enviar e esquecer/desprezar) ocorre assim que a mensagem é completamente enviada ao canal. O sistema de mensagens transmite-a em background, assim a aplicação emissora pode seguir com seu trabalho, não se obrigando a aguardar até que a aplicação receptora receba a mensagem.

O *store and forward* (armazenar e seguir em frente/despachar) ocorre assim que o sistema de mensagens recebe a mensagem do emissor. O sistema de mensagem armazena a mensagem no computador emissor na memória (RAM ou HD) e despacha a mensagem para o computador receptor, passo entregar. Assim que a mensagem chega ao computador receptor ela é armazenada nele. Esse processo pode ser repetido várias vezes até que a mensagem seja completamente movida de um computador a outro.

Esses dois métodos podem parecer uma sobrecarga redundante e desnecessária. Contudo, o fato de armazenar a mensagem no sistema de mensagens faz com que a aplicação emissora delegue ao sistema de mensagens a responsabilidade de entregar a mensagem.

A forma pela qual as mensagens são manipuladas depende do tipo de canal utilizado, seja este uma fila ou um tópico. No canal do tipo fila, o produtor envia a mensagem e ela é entregue a um e, somente um consumidor, como mostra a Figura 2.

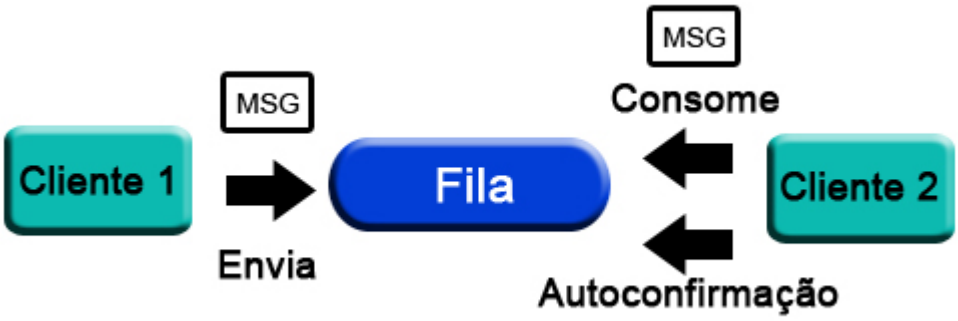

Figura 2 - Troca de mensagens utilizando fila.
Fonte: adaptado de Suconic, *et al* (2012).

Já no canal do tipo tópico, todos os consumidores que assinam o tópico recebem a mensagem, demonstrado na figura 5.

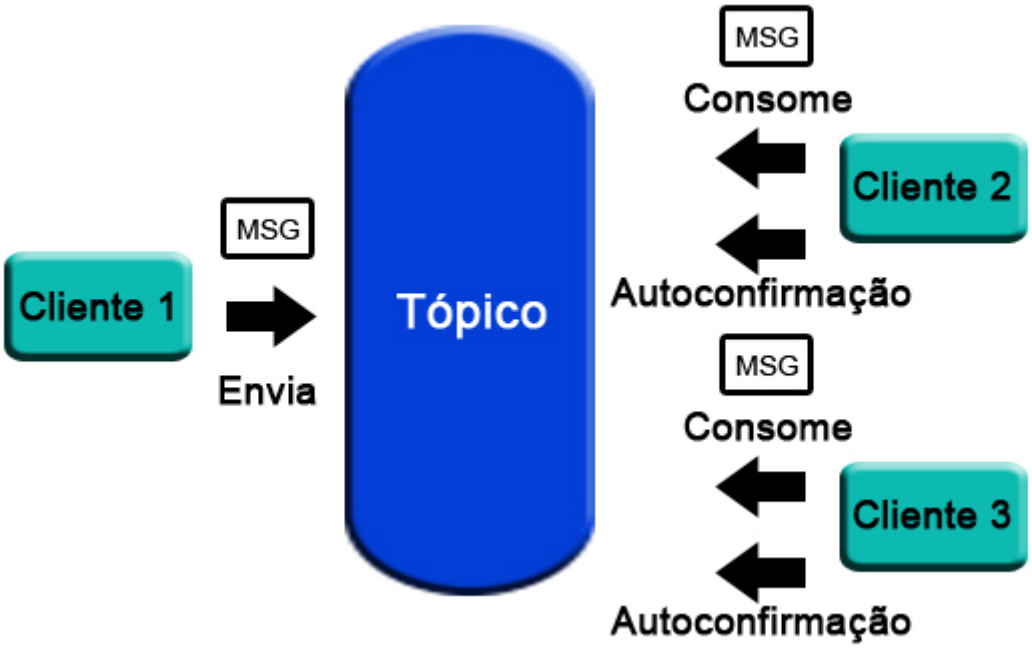

Figura 3 - Troca de mensagens utilizando tópico.
Fonte: adaptado de Suconic, *et al* (2012).

De acordo com tais características vários sistemas de integração foram desenvolvidos baseados na troca de mensagens.

2.4 ARQUITETURA ORIENTADA A SERVIÇO (SOA)

A Arquitetura Orientada a Serviço é um paradigma de integração de *softwares* que são desenvolvidos na forma de módulos de serviço baixamente acoplados. Visa a reutilização destes *softwares* para prover maior agilidade na implementação de novos sistemas. Desse modo, SOA baseia-se na ideia de que o serviço é parte do *software* a qual é construída de modo a facilitar seus vínculos a outros componentes de *software* (Koch, 2006).

Marzullo (2009, p. 123) apresenta uma definição de SOA quanto o seu comportamento ligado ao negócio, alegando que:

> [...] SOA representa uma nova abordagem para a utilização dos recursos de TI em apoio ao negócio da organização. Apesar de ainda carecer de certa maturidade, SOA apresenta a renovação, em níveis técnico e organizacional, da forma como a TI deve ser usada para apoiar o processod e alinhamento da organização.

A Figura 4 apresenta o modelo geral do alinhamento entre o negócio e a TI utilizando SOA.

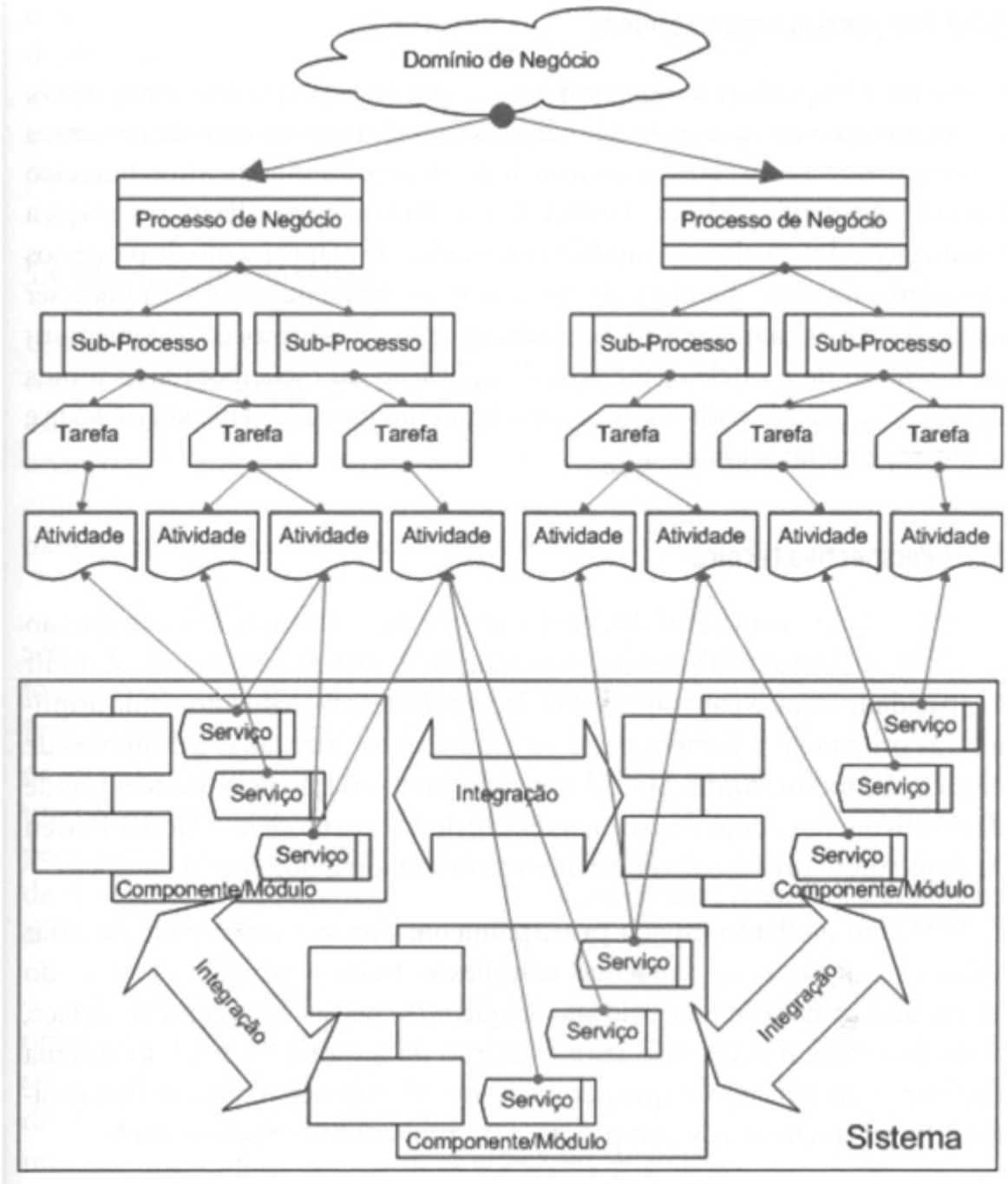

Figura 4 - Modelo geral de alinhamento entre o negócio e a TI utilizando arquitetura orientada a serviços.

Fonte: Marzullo (2009, p. 129).

O modelo da Figura 4 está relacionado ao que demonstra a Figura 5 que, na visão de Krafzig, Banke, & Slama (2005), são as principais peças que compõe a estrutura idealizada pela Arquitetura Orientada a Serviços.

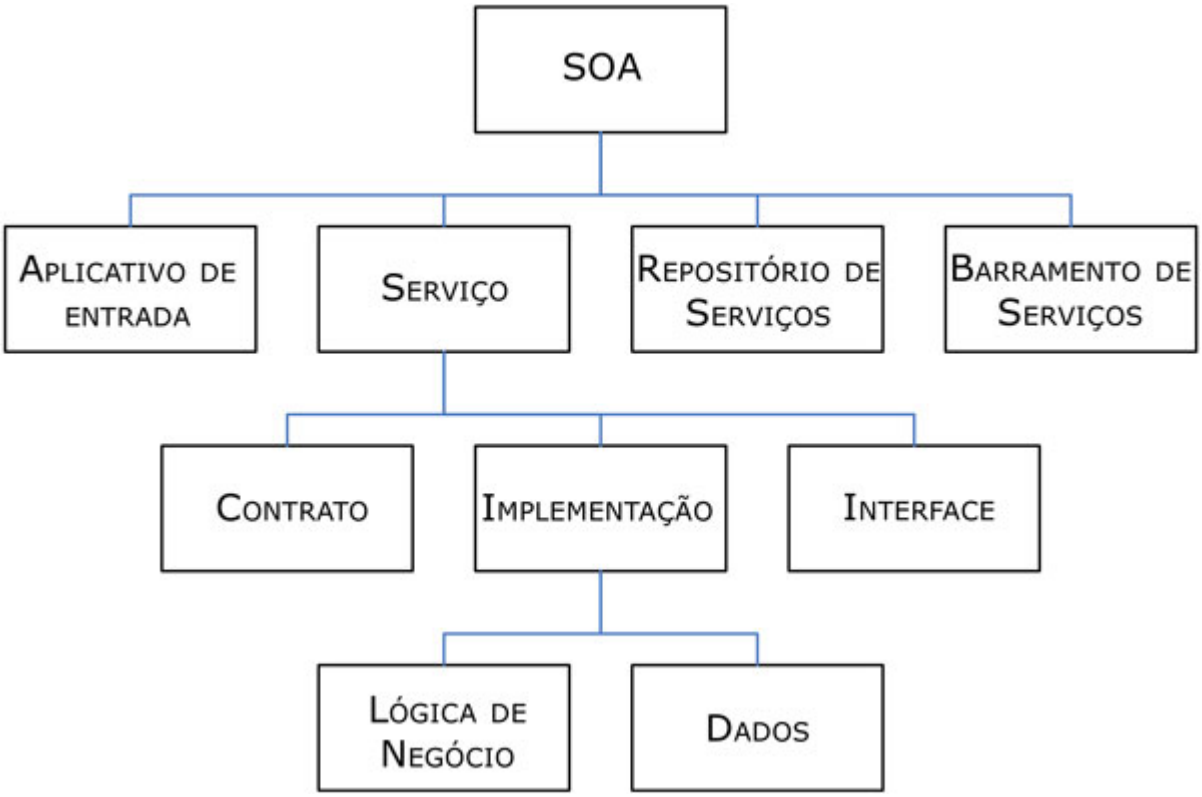

Figura 5 - Estrutura idealizada pela Arquitetura Orientada a Serviço.
Fonte: adaptada de Krafzig, Banke, & Slama (2005).

Segundo Koch (2006), o serviço deve ser descrito e estruturado de forma compreensível aos responsáveis pela camada de negócios. Soma-se a isso que, segundo Josuttis (2008), estes serviços operam entre si com base em uma definição formal (ou contrato, por exemplo, WSDL) que é **independente da plataforma e da linguagem de programação**.

Figura 6 - Tecnologias envolvidas no barramento de serviços.
Fonte: Marzullo (2009).

O que torna real a consideração em destaque feita por Josutiis (2008) é a criação de um barramento de serviços, o qual integra e gerencia tanto os serviços quanto as aplicações, onde a Figura 6 demonstra alguns dos componentes tecnológicos.

2.5 WEB SERVICE

Com base nas definições da SOA, *web service* é uma materialização de um serviço que é disponibilizado na Internet, tornando possível seu acesso de qualquer lugar (Marzullo, 2009). Na prática, um *web service* (produtor) utiliza alguns protocolos (assinaturas de comunicação) para se comunicar com o cliente (consumidor). A comunicação pode ocorrer nas seguintes formas:

1) Síncrona – a aplicação consumidora fica sem ação até que receba a resposta do produtor em modo requisição/resposta;
2) Assíncrona – a aplicação consumidora pode executar outras tarefas sem ter que aguardar a resposta do produtor. A resposta aciona um mecanismo quando chega. Assim, a aplicação pode processar a informação requisitada.

Entretanto, *Web Service* deve ser visto como uma convergência de algumas tecnologias essenciais, como mostra a Figura 4:

1) Protocolo HTTP – protocolo padrão para transmissão de dados pela Internet;
2) XML – é o formato padrão para troca de informações;
3) Protocolo de acesso a simples objetos *(SOAP)* – cria uma estrutura padrão de empacotamento para o transporte de documentos XML;
4) Linguagem de descrição de *Web Service (WSDL)* – XML que descreve a interface de um *Web Service* de forma padronizada;
5) Integração, Descoberta e Descrição Universal *(UDDI)* – descreve um registro mundial de serviços.

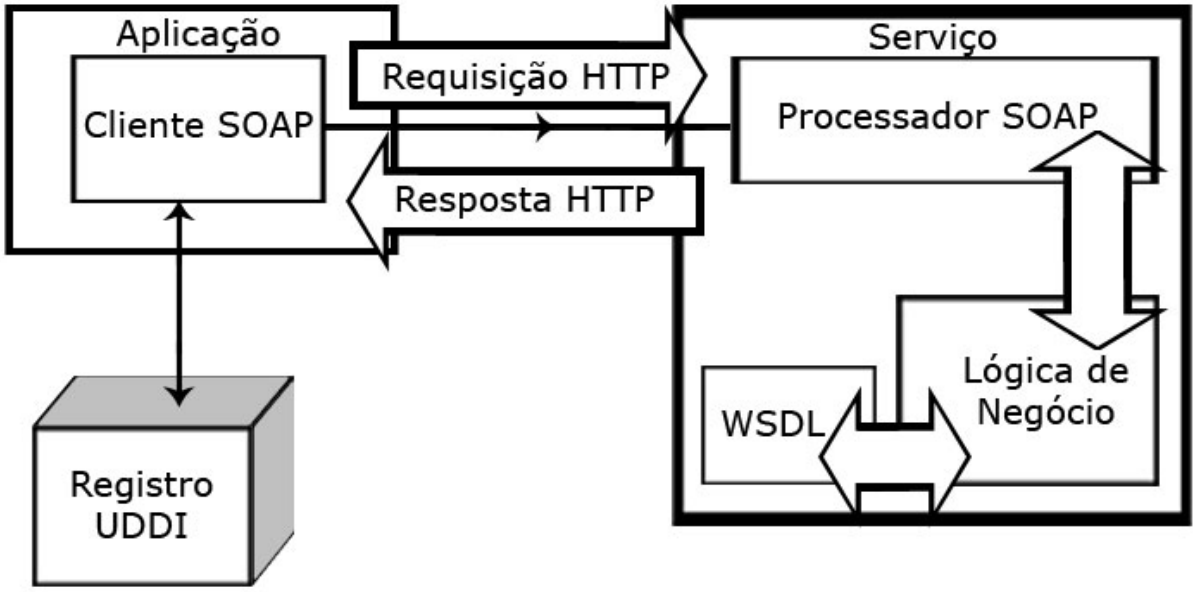

Figura 7 - Visão simplificada da infraestrutura necessária para execução de um *web service*.
Fonte: adaptado de Marzullo (2009, p. 153).

De fato, as regras de comunicação com um sistema *web service* ficam por conta da aplicação. Isso faz com que ela fique presa à forma de comunicação utilizada por ele, no caso uma mensagem dentro de um envelope SOAP, através do protocolo HTTP (Marzullo, 2009).

2.6 JAVA PLATFORM, ENTERPRISE EDITION

A Java EE versão 6 é uma plataforma de computacional para *softwares* corporativos na linguagem de programação Java, a qual provê um conjunto de APIs e ambiente de execução para desenvolvimento e funcionamento, incluindo rede e *web services*. A Figura 8 apresenta a proposta de um modelo de infraestrutura em Java EE (JSR 316, 2009).

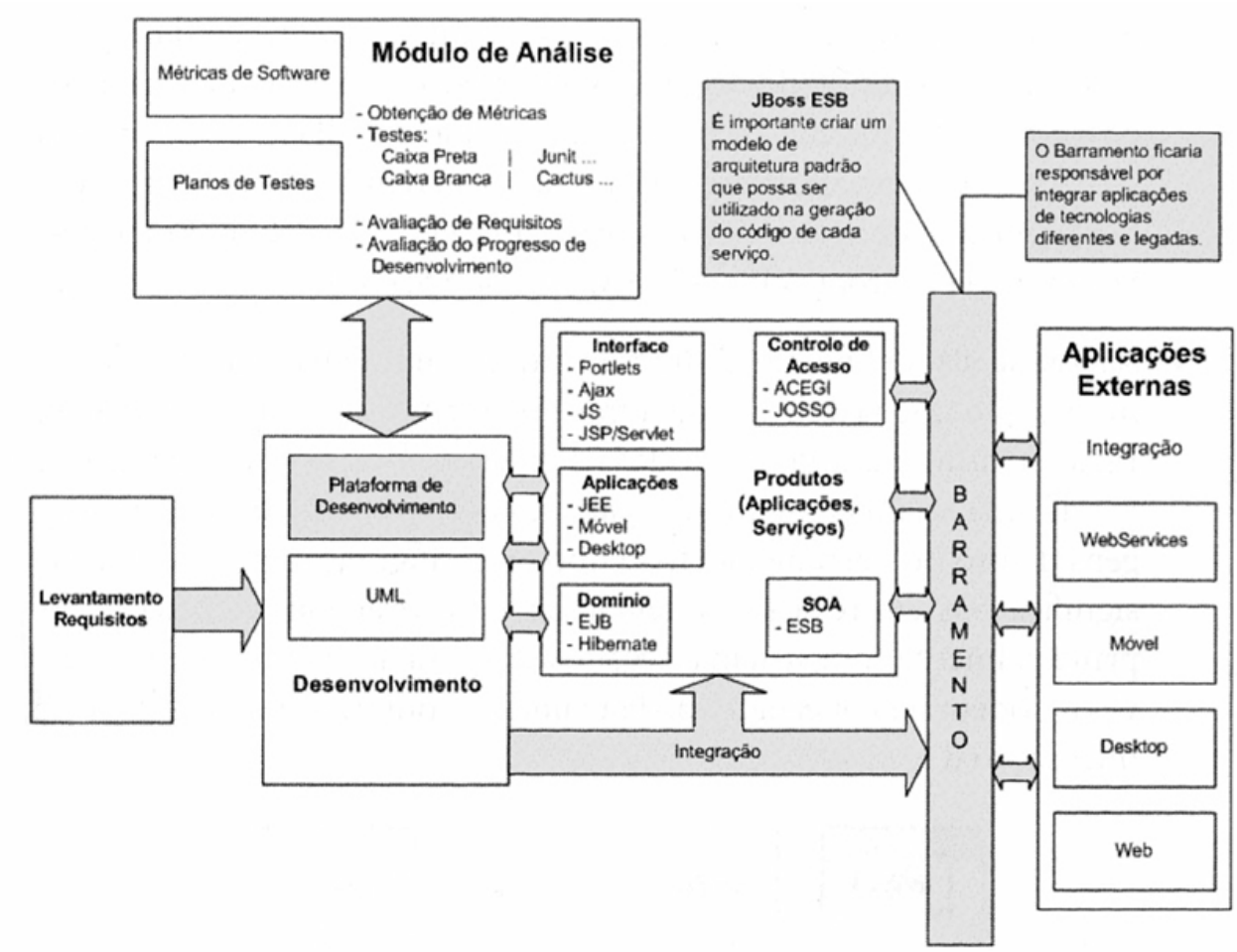

Figura 8 - Proposta de um modelo de infraestrutura JEE.
Fonte: Marzullo (2009, p. 109).

A plataforma Java EE é uma extensão da plataforma *Java Platform, Standart Edition*, e seus projetos são baseados em módulos de componentes executados em um

servidor de aplicação. O desenvolvimento nesta plataforma proporciona características como aplicações em rede em grande escala, multicamadas, escaláveis, confiáveis e seguras. (Overview - The Java EE 6 Tutorial, 2012)

2.6.1 Java Database Connectivity API

JDBC é uma API para a conectividade entre a aplicação Java e a base de dados, independente do SGBD utilizado (JDBC Overview, 2012).

2.6.2 Java Persistence API

JPA é um *framework* na linguagem Java voltado para o gerenciamento de dados relacionais em aplicações Java SE e Java EE. Esta API é definida pelo pacote *javax.persistence* (Biswas & Ort, 2006).

2.6.3 Java Message Service API

JMS é uma API que define uma forma comum para os *softwares* desenvolvidos em Java criarem, enviarem, receberem e lerem mensagens de sistemas gerenciadores de mensagens. Um provedor JMS é para um MOM o que um JDBC é para uma base de dados (JSR 914, 2003).

2.6.4 Java Naming and Directory Interface

JNDI é um padrão Java que provê uma API uniforme para o acesso à uma variedade de diretórios e nomeações de serviços. Em outros aspectos, JNDI é parecido com o JDBC. JNDI permite a escrita de código independente do provedor JMS, assim como o JDBC permite a escrita de código independente do SGBD (JNDI Overview, 2010).

2.6.5 Enterprise Java Beans

EJB é uma especificação padrão de componente gerenciável, executado no lado do servidor, desenvolvido para a construção modular de aplicações organizacionais, cuja função é encapsular a lógica de negócio. Sua atual versão é a 3.1 (JSR 318, 2009).

2.6.5.1 Message Driven Beans

Um MDB é um EJB cujo funcionamento é baseado no processamento de eventos, estes que são disparados por **requisições assíncronas de mensagens**. Para isso, ele deve implementar a interface *MessageListener*, a qual contém o método *onMessage*. Para sua execução ele deve assinar uma fila ou tópico, e assim receber suas mensagens (JSR 318, 2009).

Assim como os *web services* que podem ser desenvolvidos em outras plataformas e linguagens de programação, a arquitetura Java EE, através de suas APIs, traz outra forma, além dos *web services,* de desenvolver SOA na prática. O EJB é uma boa alternativa para desenvolver serviços, dentre eles, o MDB, o qual utiliza a API JMS para comunicação. A Figura 9 demonstra um exemplo prático da utilização da plataforma Java EE.

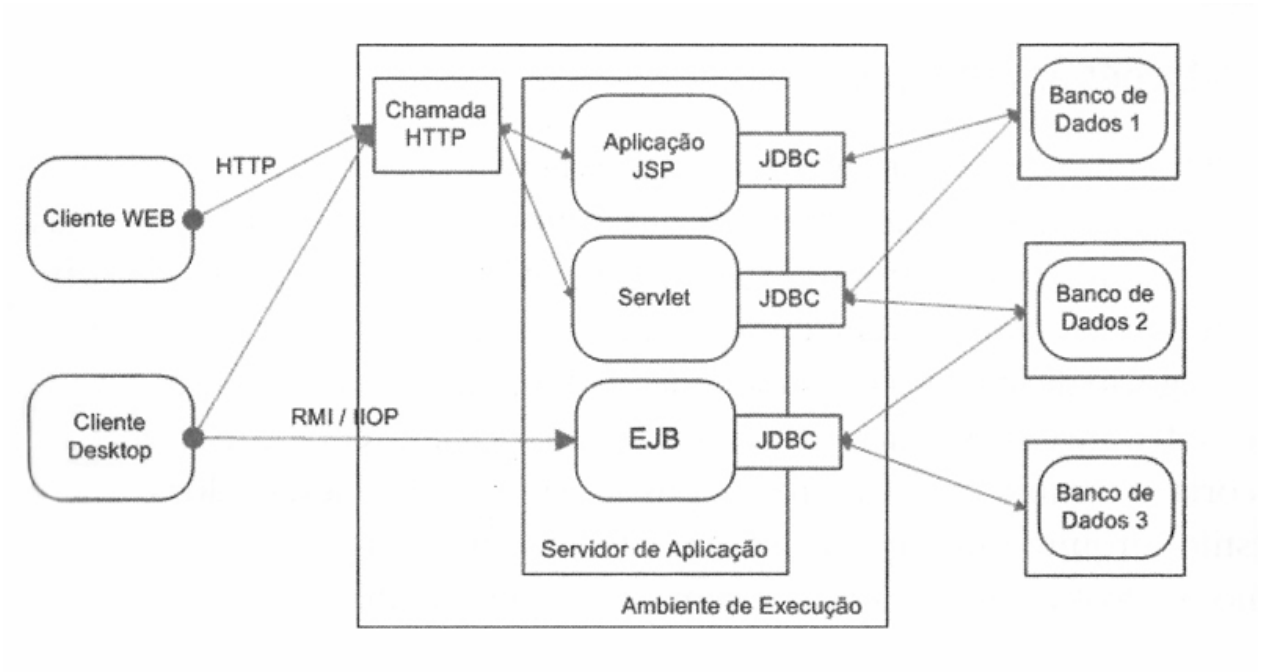

Figura 9 - Exemplo de integração entre os diferentes tipos de componentes dentro de uma arquitetura JEE.

Fonte: Marzullo (2009, p. 59).

2.7 SERVIDOR DE APLICAÇÃO

> Application servers, ou servidores de aplicação, são softwares que fornecem a infraestrutura de serviços para a execução de aplicações distribuídas. Os servidores de aplicação são executados em servidores e são acessados pelos clientes através de uma conexão de rede (Iweb, 2003).

Um servidor de aplicação fornece componentes que permitem que o desenvolvedor se concentre na camada de negócio. Os AS contém serviços que diminuem a complexidade do desenvolvimento, incrementam o desempenho, gerenciam segurança e controlam o fluxo de dados (Iweb, 2003).

2.7.1 *JBoss Enterprise Application Server*

A JBoss AS é um dos mais utilizados e mais completos servidores de aplicação que implementa a Java EE, assim desenvolvido em Java devido a sua capacidade de ser multiplataforma. A *JBoss Company* é uma divisão da Red Hat Inc., especialista em desenvolver *softwares middlewares* de código aberto, e é a desenvolvedora do JBoss AS, cuja versão, até o presente momento, é a 7.1 (JBoss Org, 2011).

2.7.2 HornetQ

O JBoss AS vem incorporando a API JMS desde sua versão 4.3, e seu sistema de mensagens era inicialmente chamado de JBoss MQ que posteriormente veio a ser chamado de *JBoss Messaging*, cuja versão 1.0, a qual foi lançada em 29 de março de 2006, tornava-o executável com o sem o JBoss AS (JBossMQ | JBoss Community, 2004). O *JBoss Messaging* foi o provedor JMS padrão do *JBoss SOA Platform* e do JBoss AS versão 5 (JBossMessaging | JBoss Community, 2010). Em 24 agosto de 2009, HornetQ foi lançado baseado no *JBoss Messaging 2.0* e, este foi colocado no modo correção de erros.

O HornetQ é o atual projeto de *software* livre e de código aberto da JBoss, desenvolvido e mantido como o MOM padrão do JBoss AS desde a versão 6. Algumas características o fizeram um dos mais competitivos dentre os MOMs *open source* disponíveis no mercado, como:

1) Um sistema de mensagens multi-protocolos;

2) Incorporável;
3) Alto desempenho;
4) Suporta o protocolo STOMP;
5) Agrupável (clustered);
6) Possui 100% de conformidade com a API JMS;
7) Sistema mestre/escravo para tolerância a falhas;
8) Pontes entre servidores HornetQ;
9) Alta desempenho utilizando conector Netty NIO sobre modos TCP, SSL e Servlet;
10) Simples configuração de XML;

O HornetQ é um projeto licenciado pela *Apache Software License v2.0*, e em 2 de abril de 2010 quebrou *records* no SPECjms2007 no recebimento de mensagens por segundo (Suconic *et al*, 2012).

3 DESENVOLVIMENTO

Para o trabalho proposto foi necessário desenvolver alguns módulos antes de iniciar a comunicação por mensagens, como o modelo dos dados acadêmicos em sua devida hierarquia, o módulo de conexão ao banco de dados e o módulo para ter acesso aos dados e transformá-los em objetos.

Posteriormente, o ambiente de execução do *middleware* de mensagens *HornetQ* foi preparado utilizando o *JBoss Enterprise Application Server* versão 6 e configurado com um EJB criado no IDE Eclipse Indigo.

Assim que as filas e tópicos estavam funcionando no servidor de aplicação, a fase de construção do módulo de serviço de mensagens foi iniciada, criando classes padrões para serem estendidas por serviços futuros que queriam utilizar essa plataforma.

O serviço de dados acadêmicos foi desenvolvido com base nas classes de mensagens desenvolvidas, como exemplo de reutilização de parte do trabalho desenvolvido.

3.1 MODELO DE DADOS ACADÊMICOS

Como fase inicial do desenvolvimento houve a necessidade de estudar o ambiente acadêmico e as informações que a ele pertencem. Com base nas reuniões e análises feitas com os responsáveis pela área acadêmica, como a Pró-Reitoria de Graduação e a Secretaria Acadêmica do *Campus* Luiz Meneghel, foi desenvolvido o modelo de dados acadêmicos de modo que pudesse abranger toda a universidade.

Tal modelo coleta informações desde o *campus* de origem até o aluno, onde cada *campus* tem um conjunto de centros, tendo cada centro um conjunto de cursos. Cada curso pode ter uma ou mais matrizes curriculares e, tanto um curso, quanto uma matriz curricular são regidos por um ato legal de criação e outros atos de modificação.

Uma matriz curricular é dividida em serializações e cada uma delas é composta por um conjunto de disciplinas. As serializações dependem da sua matriz curricular, pois podem ter duração de um semestre ou um ano.

A partir de cada disciplina são definidas as turmas, as quais são separadas dependendo da sua serialização (semestre ou ano), contendo o professor responsável pela referida disciplina naquele período, como por exemplo, o primeiro semestre de 2012, e um conjunto de matrículas que se refere a cada aluno matriculado na disciplina e período desta turma.

Cada matrícula é referente a um aluno, e contém os dados referentes ao desempenho deste aluno na disciplina, como o status de matrícula, que informa se o aluno é regularmente matriculado, dependente ou retido, a média parcial, a nota do exame, a média final, a porcentagem da freqüência, e o resultado final, o qual informa se o aluno foi aprovado ou reprovado.

O modelo de aluno tem as informações pessoais identificadoras dele e um conjunto de matrículas, o qual contém todas as disciplinas que o aluno estava matriculado durante o todo o curso. Este modelo é apresentado na Figura 10.

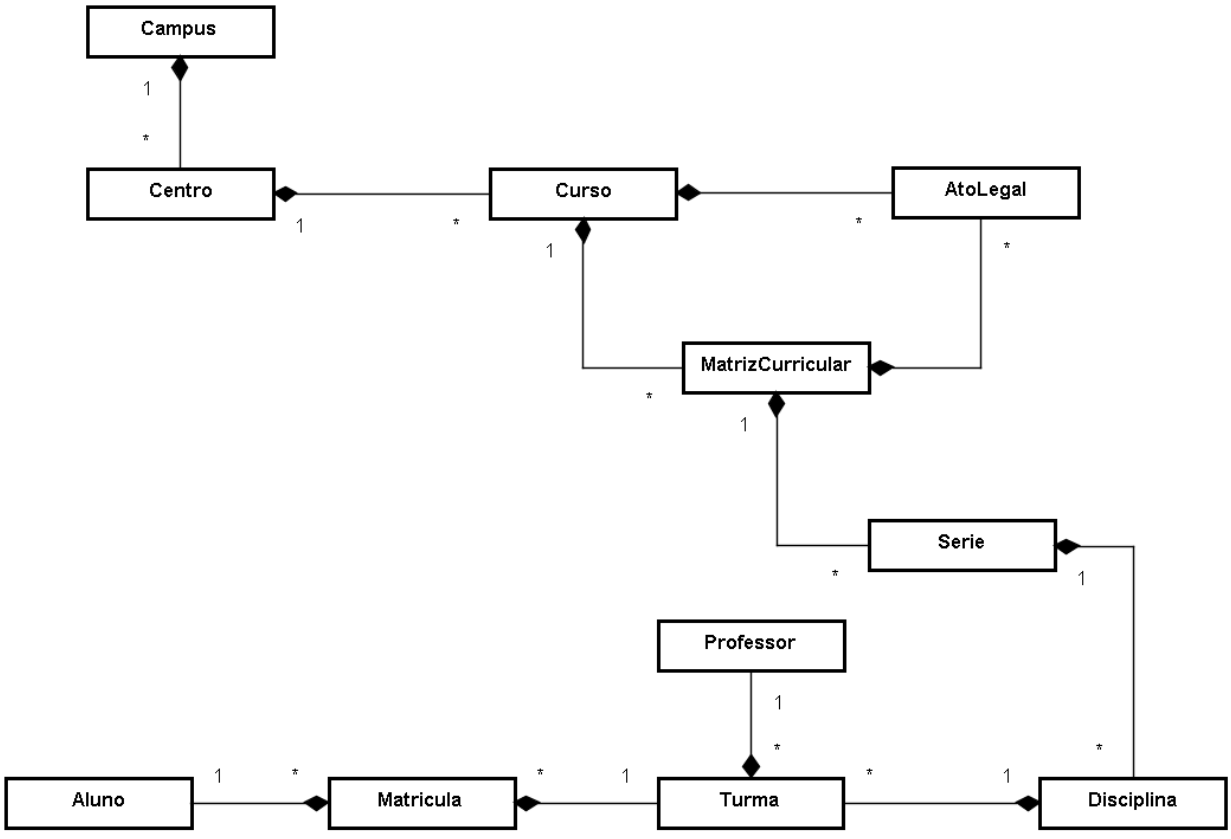

Figura 10 - Diagrama de Classes de Dados Acadêmicos.

Cada classe deste modelo foi criada na forma de POJO, implementando a interface *Serializable* e com anotações *XmlRootElement* e *XmlType(propOrder)* para exportação dos dados nos formatos XML(Figura 11) e JSON.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<curso>
    <codigo>4</codigo>
    <nome>GRADUA□□O EM SISTEMAS DE INFORMA□□O</nome>
    <habilitacao>LICENCIATURA EM COMPUTA□□O  - BACHAREL EM SISTEMAS DE
    <graduacao>BACHAREL EM SISTEMAS DE INFORMA□□O</graduacao>
    <duracaoMaxima>16</duracaoMaxima>
    <matrizCurricular>
        <codigo>8</codigo>
        <nome>Grade Nova</nome>
        <serie>
            <codigo>50</codigo>
            <nome>BLOCO 8</nome>
            <disciplina>
                <codigo>380</codigo>
                <nome>AUDITORIA DE SISTEMAS</nome>
                <cargaHorariaPratica>72</cargaHorariaPratica>
            </disciplina>
            <disciplina>
                <codigo>381</codigo>
                <nome>EMPREENDEDORISMO</nome>
                <cargaHorariaPratica>72</cargaHorariaPratica>
            </disciplina>
            <disciplina>
                <codigo>382</codigo>
                <nome>PROJETO FINAL II</nome>
                <cargaHorariaPratica>180</cargaHorariaPratica>
            </disciplina>
            <disciplina>
                <codigo>383</codigo>
                <nome>LEGISLA□□O EM INFORM□TICA</nome>
                <cargaHorariaPratica>36</cargaHorariaPratica>
            </disciplina>
            <disciplina>
                <codigo>384</codigo>
                <nome>INFORM□TICA E SOCIEDADE</nome>
                <cargaHorariaPratica>72</cargaHorariaPratica>
            </disciplina>
        </serie>
    </matrizCurricular>
</curso>
```

Figura 11 - Dados Xml de uma Classe.

3.2 COLETA DOS DADOS

Assim que o modelo de dados acadêmicos estava pronto, a fase de coleta de dados foi iniciada com base no banco de dados do sistema acadêmico utilizado no CLM, cujo SGBD é o *Firebird* na versão 1.5. Com base nas análises das tabelas desta base e com a busca dos dados necessários para completar cada uma das classes do modelo desenvolvido, foi possível criar um conjunto de seleções complexas entre as tabelas, chegando ao resultado desejado, como mostra a Figura 11.

3.3 PREPARAÇÃO AMBIENTE DE EXECUÇÃO

Para iniciar a criação do serviço utilizando mensagens foi necessário preparar o ambiente de execução. Primeiramente, foi criada uma pasta com todas as aplicações necessárias, como a JDK na versão 7, o *JBoss Enterprise Application Server 6*, o IDE Eclipse Indigo e o IDE Netbeans versão 7.1.1.

Executando o IDE Eclipse foi necessário adicionar um *plugin* chamado *JBoss Tools*, o qual auxilia o IDE no gerenciamento dos componentes desenvolvidos pela *JBoss*. Assim, foi adicionado ao IDE um servidor de aplicação JBoss versão 6, selecionando a pasta onde estava instalado o JBoss e a configuração de execução como *default*.

Um novo projeto do tipo EJB foi criado para configurar permissões e níveis de acessos de cada usuário a persistir mensagens nas filas e tópicos JMS do JBoss AS, utilizando o HornetQ, apenas com arquivos XML de configuração, sendo eles o *hornetq-configuration.xml, o hornetq-jms.xml e o hornetq-users.xml*. Isso faz com que seja possível adicionar uma fila ou tópico ao servidor de aplicação apenas modificando o arquivo *hornetq-jms.xml*, como mostra a Figura 12.

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="urn:hornetq"
    xsi:schemaLocation="urn:hornetq /schema/hornetq-jms.xsd" >

    <queue name="Requests" >
        <entry name="queue/Requests" />
    </queue>

    <topic name="Curso" >
        <entry name="topic/Curso" />
    </topic>

    <topic name="Aluno" >
        <entry name="topic/Aluno" />
    </topic>

</configuration>
```

Figura 12 - Configuração de filas e tópicos no arquivo *hornetq-jms.xml.*

Assim que o container EJB estava pronto ele foi adicionado ao servidor de aplicação. Ao iniciar o servidor de aplicação, é possível verificar que a configuração definida nos arquivos presentes no container estava correta, como mostra a Figura 13.

```
12:53:07,194 INFO  [HornetQServerImpl] trying to deploy queue jms.queue.ExpiryQueue
12:53:07,248 INFO  [HornetQServerImpl] trying to deploy queue jms.queue.DLQ
12:53:07,301 INFO  [HornetQServerImpl] trying to deploy queue jms.queue.Requests
12:53:07,313 INFO  [HornetQServerImpl] trying to deploy queue jms.topic.Aluno
12:53:07,360 INFO  [HornetQServerImpl] trying to deploy queue jms.topic.Curso
12:53:07,402 INFO  [service] Removing bootstrap log handlers
12:53:07,503 INFO  [org.apache.coyote.http11.Http11Protocol] Starting Coyote HTTP/1.1 on
12:53:07,565 INFO  [org.apache.coyote.ajp.AjpProtocol] Starting Coyote AJP/1.3 on ajp-0.0
12:53:07,565 INFO  [org.jboss.bootstrap.impl.base.server.AbstractServer] JBossAS [6.1.0.F
```

Figura 13 - Inicialização do Servidor de Aplicação.

3.4 CRIAÇÃO DAS CLASSES DE MENSAGENS

Utilizando o IDE Netbeans, o pacote de classes *br.edu.uenp.academic.message* foi criado com o intuito de que as classes existentes nele fossem padrões para o envio e recebimento de mensagens em filas e tópicos, bastando apenas preencher os três

parâmetros do construtor destas classes: JNDI da fila ou tópico, usuário e senha. Além disso, é necessário que a aplicação tenha um arquivo de configuração chamado *jndi.properties* com os parâmetros do ambiente de execução, como o endereço do AS. Esse arquivo existe para que haja portabilidade das aplicações sem a necessidade de alterar o código fonte das classes de execução.

O arquivo *jndi.properties* é utilizado pelo método construtor para criar uma instancia do objeto *ConnectionFactory* que serve para criar conexões e sessões de filas ou tópicos, e a partir dele enviar e receber mensagens. Caso haja alteração no servidor de mensagens, basta alterar as informações deste arquivo de configuração.

A classe *Requester* foi criada para enviar mensagens de texto a uma fila, enquanto a classe *Producer*, para enviar mensagens de POJOs a um tópico. Para receber as mensagens de texto de uma fila de mensagens, a classe *Listener* foi criada, enquanto para receber as mensagens contendo POJOs, foi criada a classe *Subscriber*, já que é voltada para tópicos. Desta forma, uma aplicação que queira receber os POJOs de um tópico deve estender a classe *Subscriber* e sobrescrever o método *onMessage* para fazer alguma operação diferente com o POJO recebido. O método padrão apenas exibe o resultado do método *toString* do objeto.

3.5 DESENVOLVIMENTO DO SERVIÇO

Para o serviço de dados acadêmicos fora criado, inicialmente, uma fila chamada *Requests* para receber as requisições e dois tópicos, um chamado Curso e outro chamado Aluno, para que cada tópico armazenasse as mensagens referentes a cada uma das entidades. Uma mensagem de requisição para a fila *Requests* deve ser feita da seguinte forma: o nome da entidade desejada mais o código identificador da entidade, sendo separadas por ponto e vírgula. Um exemplo seria se fosse desejado obter as informações do Curso de código quatro (4). A mensagem correta a ser enviada à fila de requisições seria "Curso;4".

Como implementação do serviço foi criada a classe *ListenerRequests,* a qual estende a classe *Listener* e sobrescreve o método *onMessage*. Quando uma

mensagem é enviada à fila *Requests*, o método *onMessage* é acionado, a mensagem de texto é dividida no ponto e virgula e cada parte da mensagem é armazenada em uma posição de um vetor de texto (*String)*. Assim que a mensagem de requisição é separada neste vetor, o método *getAcademicObject* é acionado, passando por parâmetro, respectivamente, a primeira e segunda posições do vetor.

O método chamado *getAcademicObject* recebe dois parâmetros do tipo *String*, onde um representa o nome da entidade desejada e o outro o código identificador dessa entidade. Com isso, inicia-se uma verificação, comparando o nome da entidade com as possibilidades existentes. Caso seja encontrada, uma instancia da entidade desejada é criada, buscando por código na respectiva classe DAO e o resultado é armazenado nessa instancia. Desta forma, o método *writeObject* é acionado para publicar o POJO resultante no seu respectivo tópico.

O método *writeObject* recebe por parâmetro duas variáveis, uma do tipo *String* contendo o JDNI do tópico e outra do tipo *Serializable* contendo o POJO a ser postado. Então o método cria uma instancia da classe *Producer* com o nome do tópico e usuário e senha padrões, publica o POJO e depois aciona o método *close*. Por fim, retorna uma mensagem dizendo que foi enviado ao tópico o referido POJO.

4 RESULTADOS OBTIDOS

Este trabalho teve como intuito iniciar o barramento de serviços da UENP, criando o serviço de dados acadêmicos. Como resultado fica o modelo de dados acadêmicos proposto, que pode ser utilizada não só na UENP, mas também em outras universidades, podendo ser expandido a nível estadual ou nacional, e as classes para comunicação por mensagem. Aplicações que queriam utilizar a comunicação por mensagens, basta importar o pacote criado, demonstrado na Figura 14.

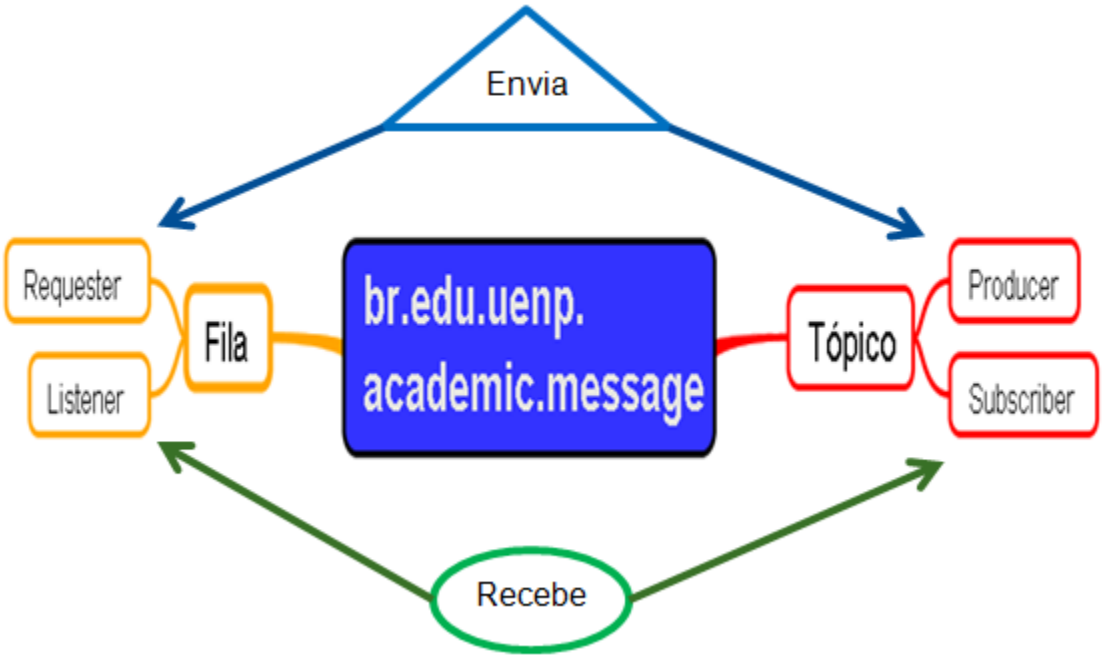

Figura 14 - Classes de mensagens.

O processo de envio de mensagem segue alguns passos básicos que são comuns, tanto as classes de fila, quanto as de tópico, sendo eles:

1) Criar o produtor de mensagens;
2) Iniciar o produtor de mensagens;
3) Enviar uma mensagem.

Da mesma forma, o processo de recebimento de mensagens de filas é parecido com o de tópicos. Neste caso, a aplicação deve estender a classe para receber

mensagens, sobrescrever o método *onMessage* e manipular a variável *Message* recebida como parâmetro por este método.

Para o envio de mensagem do tipo texto a uma fila, uma aplicação precisa criar um objeto do tipo *Requester* e inicializá-lo, e em seguida executar o método *writeMessage* passando o texto da mensagem por parâmetro. Para enviar uma mensagem de um objeto do tipo POJO (classe que implementa *Serializable*) a um tópico, uma aplicação deve criar um objeto do tipo *Producer* e inicializá-lo, e em seguida executar o método *writeMessage* passando o objeto por parâmetro.

Uma aplicação que queira receber mensagens do tipo texto de uma fila, precisa estender a classe *Listener* e sobrescrer o método *onMessage*, manipulando a variável do tipo *Message*, recebida por parâmetro, a fim de obter o resultado desejado. Da mesma forma, para uma aplicação receber mensagens de um tópico, é necessário que ela estenta a classe *Subscriber*, sobrescreva o método *onMessage* e manipule a variável do tipo *Message*, recebida por parâmetro, para obter o resultado desejado.

5 CONSIDERAÇÕES FINAIS

O modelo de solução para o problema de integração entre os diversos sistemas da UENP foi baseado em SOA. Tal paradigma se dá pela realização do compartilhamento de informações entre aplicações, que é realizado utilizando rede de computadores, onde as aplicações são baixamente acopladas, sendo divididas e desenvolvidas na forma de módulos de serviço. Assim, cada serviço pode ser disponibilizado na forma de *web service* ou container movido a mensagens, e um conjunto de serviços pode construir uma ou mais aplicações.

Concluí-se que o desenvolvimento em SOA não é simples de ser aplicado, e exige muito estudo e comprometimento da equipe envolvida. A arquitetura de *software* Java, utilizada no presente trabalho, é uma das que dispõe uma vasta gama de ferramentas e *frameworks* para este tipo de desenvolvimento. O fato é que um estudo detalhado do escopo dos serviços e, das pretensões futuras de cada organização, auxilia na escolha das ferramentas a se utilizar no desenvolvimento, entrega e continuidade do serviço.

Como peça chave da SOA, tem-se a integração entre os serviços e as aplicações, que normalmente não estão concentradas em um único ponto. Devido à utilização da rede de computadores para a realização desta integração, há a possibilidade de ocorrerem falhas na comunicação, o que faz com que um servidor de aplicação, que contenha diversos serviços, torne-se indisponível por um determinado momento. Isso traz uma redução na disponibilidade de utilização dos serviços, fazendo com que as requisições não cheguem ao destinatário.

Para solucionar este problema, foi apresentada a tecnologia de integração por *middleware* de mensagens, que é, atualmente, uma boa alternativa para a comunicação e integração das aplicações e serviços de uma organização, devido à comunicação assíncrona.

Neste caso, as mensagens são adicionadas ao canal de comunicação, ficando armazenadas dentro do *middleware*, aguardando o momento de entrega, que é quando a aplicação produtora esta disponível. Assim, uma aplicação consumidora poder fazer

requisições, mesmo que a aplicação produtora não esteja disponível, trazendo a vantagem delas não terem que aguardar a disponibilidade uma da outra.

A utilização da comunicação assíncrona traz o aumento da independência no momento das requisições, tornando as aplicações independentes da disponibilidade do servidor de aplicação onde se contra o serviço, aguardando a chegada da resposta a qualquer momento.

O HornetQ, *middleware* orientado a mensagens utilizado neste trabalho, apresenta um dos melhores desempenhos dentre os outros MOMs disponíveis no mercado, tendo como principais vantagens a licença de uso gratuita e a replicação dos canais de mensagem formando um cluster de MOMs, aumentando ainda mais a disponibilidade dos canais de comunicação.

5.1 TRABALHOS FUTUROS

Com os componentes desenvolvidos no presente trabalho, é possível dar continuidade ao projeto de criação do barramento de serviços da UENP e automatizar o serviço de dados acadêmicos, utilizando algum *framework,* dentre os disponíveis no mercado, que atenda as necessidades deste e de novos serviços.

Um *framework* que vem sendo muito utilizado no desenvolvimento de aplicações corporativas é o Spring, dentre as características encontram-se o acesso a dados, segurança e integração. Para a fase de coleta de dados pode ser utilizado o mapeamento objeto relacional, com ferramentas como Hibernate, MyBatis. Para a automação dos módulos, além do uso de Spring podem ser utilizados EJBs, como os MDBs que podem utilizar os canais de comunicação criados no HornetQ.

6 REFERÊNCIAS

JSR 318. (10 de 12 de 2009). *JSR 318: Enterprise JavaBeansTM 3.1*. Acesso em 24 de 10 de 2012, disponível em The Java Community Process: http://jcp.org/aboutJava/communityprocess/final/jsr318/index.html

JSR 914. (02 de 12 de 2003). *JSR-000914 JavaTM Message Service (JMS) API* . Acesso em 24 de 10 de 2012, disponível em Java Community Process: http://jcp.org/aboutJava/communityprocess/final/jsr914/index.html

Biswas, R., & Ort, E. (05 de 2006). *The Java Persistence API - A Simpler Programming Model for Entity Persistence*. Acesso em 11 de 10 de 2012, disponível em Oracle: http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html

Caetano, G. F. (2008). *Arquitetura Orientada a Serviço: maior interação entre estratégia de negócio e a tecnologia da informação* . Viçosa, Minas Gerais, Brasil: Universidade Federal de Viçosa.

Date, C. J. (2004). *Introdução a sistemas de bancos de dados.* Elsevier Brazil.

Hohpe, G., & Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.*

Ibrahim, N. M., Hassan, M. F., & Balfagih, Z. (2011). Agent-based MOM for Interoperability cross-platform communication of SOA Systems. *International Symposiom on Humanities, Science and Engineering Research* , 40-45.

IWeb. (10 de 2003). *APPLICATION SERVERS*. Acesso em 27 de 10 de 2012, disponível em iweb: http://www.iweb.com.br/iweb/pdfs/20031008-appservers-01.pdf

JBoss AS7 vs AS6 | JBoss Community. (12 de 05 de 2011). Acesso em 02 de 10 de 2012, disponível em JBoss Org: https://community.jboss.org/wiki/JBossAS7VsAS6

JBossMessaging | JBoss Community. (20 de 10 de 2010). Acesso em 20 de 09 de 2012, disponível em JBoss Org: https://community.jboss.org/wiki/JBossMessaging

JBossMQ | JBoss Community. (10 de 02 de 2004). Acesso em 20 de 09 de 2012, disponível em JBoss Org: https://community.jboss.org/wiki/JBossMQ

JDBC Overview. (16 de 07 de 2012). Acesso em 24 de 10 de 2012, disponível em Oracle: http://www.oracle.com/technetwork/java/overview-141217.html

JDK 6 Java Database Connectivity (JDBC)-related APIs &. (2011). Acesso em 24 de 10 de 2012, disponível em Oracle: http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/

JNDI Overview. (16 de 07 de 2010). Acesso em 24 de 10 de 2012, disponível em Oracle: http://www.oracle.com/technetwork/java/overview-142035.html

Josuttis, N. M. (2008). *SOA na Prática: a arte de modelagem de sistemas distribuidos.* Jacaré: Alta Books.

Koch, C. (17 de 07 de 2006). *ABC da SOA*. Acesso em 02 de 08 de 2012, disponível em CIO: http://cio.uol.com.br/tecnologia/2006/07/17/idgnoticia.2006-07-17.3732358054

Krafzig, D., Banke, K., & Slama, D. (2005). *Enterprise SOA.* Prentice Hall.

Linwood, J., & Minter, D. (2010). *Beginning Hibernate.* APRESS.

Marzullo, F. P. (2009). *SOA NA PRÁTICA: Inovando seu negócio por meio de soluções orientadas a serviços.* São Paulo: Novatec Editora Ltda.

Overview - The Java EE 6 Tutorial. (06 de 2012). Acesso em 24 de 10 de 2012, disponível em Oracle: http://docs.oracle.com/javaee/6/tutorial/doc/bnaaw.html

Sivakumar, G., Abrahms, F., Hogg, K., & Hartley, J. (2010). SOI (Service Oriented Integration) and SIMM (Service Integration Maturity Model) - an Analysis. *IEEE 6th World Congress on Services* , 178-182.

Suconic, C., Taylor, A., Fox, T., Mesnil, J., & Gao, H. (13 de 04 de 2012). *Putting the buzz.* Acesso em 29 de 07 de 2012, disponível em HornetQ User Manual: http://docs.jboss.org/hornetq/2.2.2.Final/user-manual/en/html_single/index.html

Sun Microsystems, I. A. (2001-2002). *Core J2EE Patterns - Data Access Object*. Fonte: http://www.oracle.com: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 316. (10 de 12 de 2009). Acesso em 24 de 10 de 2012, disponível em Java Community Process: http://jcp.org/en/jsr/detail?id=316

Zhao, Y. (2006). Enterprise Service Oriented Architecture (ESOA) Adoption Reference. *IEEE International Conference on Services Computing (SCC'06)* .