Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT
Centro de Aperfeiçoamento Tecnológico - CENATEC

Cristina Corrêa Oliveira

ANÁLISE DOS AMBIENTES DISTRIBUÍDOS J2EE SOB A ÓTICA DO RM-ODP

São Paulo
2006

Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT
Centro de Aperfeiçoamento Tecnológico - CENATEC

Cristina Corrêa Oliveira

AVALIAÇÃO DOS AMBIENTES DISTRIBUÍDOS J2EE SOB A ÓTICA DO RM-ODP

Defesa apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, para obtenção do título de Mestre em Engenharia de Computação.
Área de concentração: Engenharia de Software

Orientador: Prof. Dr. Reginaldo Arakaki

São Paulo
2006

Dedico este trabalho ao meu marido, meu filho e a minha família pela motivação e compreensão durante essa longa jornada e, ao Prof. Dr. Reginaldo Arakaki, incansável orientador.

Meus agradecimentos a todas as pessoas que me apoiaram de uma forma ou de outra. Sem dúvida nenhuma, contribuíram muito para que eu alcançasse este objetivo.

Apólogo

CENA (mata): É um belo e ensolarado dia na floresta e um pequeno coelho com pose de intelectual se encontra sentado fora de sua toca, datilografando a sua máquina de escrever pessoal (ou seria um computador portátil?). À sua frente, perambulando na sua caminhada matinal, se apresenta uma lépida e vivaz raposa, quiçá a mais perspicaz do seu bando.
RAPOSA: — Em que você está trabalhando?
COELHO: — Em minha Tese.
RAPOSA: — Humm.... Do que se trata?
COELHO: — Ah! estou discorrendo sobre como coelhos se alimentam de raposas...
(PAUSA INCRÉDULA!)
RAPOSA: — É ridículo!!! Qualquer tolo sabe que coelhos não comem raposas.
COELHO: — Claro que o fazem, e eu posso provar. Venha comigo. — Ambos desaparecem rumo à toca. Após uns poucos minutos, volta o coelho, sozinho, à sua atividade de escritor e recomeça a trabalhar junto ao seu equipamento. Daí em breve, entra em cena um lobo, em trajes de gala (beca), o qual, mesmo muito açodado, se detém ao ar de compenetrado em que parece se deixar absorver o coelho.
LOBO: — Sobre o que você tanto escreve?
COELHO (absorto): — Eu estou elaborando um tratado sobre como coelhos se nutrem de lobos...
(GARGALHADA DE ESCÁRNIO!)
LOBO: — Você não espera, sinceramente, que tal bobagem seja publicada..., ou espera?
COELHO (muito calmo): — Sem problemas. O senhor quer saber por quê? — Os dois sujeitos, por conseguinte, adentram a toca; e, novamente, o coelho retorna, desacompanhado, ao seu passatempo de criador.
CENA (toca): Em um canto do recinto, há uma pilha de ossos de raposa. Em um outro, uma ruma de fragmentos cadavéricos de lobo. Percorrendo os olhos ao longo da sala, pode-se aperceber, em seu deleite quase extático, um encorpado leão, palitando os dentes, a arrotar.

— FIM —

MORAL:
Parece não ser muito relevante o que você escolheu como assunto de tese.
Parece não ser muito importante o que você escolheu como dados para a prova.
Pois, realmente, o que interessa é quem você escolheu como seu orientador.

Em vista disso, agradeço ao Prof. Dr. Reginaldo Arakaki, por sua amizade, paciência e principalmente por seu apoio

Agradeço também ao Prof Mário Miyake; Prof. Hamilton J. Brumatto; Prof. Ana Lucia Lima Marreiros; Ana Claudia Dantas Ferreira e Rita de Sousa, pela contribuição.

E a todos que, direta ou indiretamente me auxiliaram nesta empreitada.

Resumo

O trabalho apresenta a hipótese de que existe uma relação dos elementos do J2EE ao RM-ODP. Ambos têm sido usados para a definição e construção de aplicações distribuídas.

O padrão J2EE se apresenta como uma plataforma completa e aberta para aplicações distribuídas complexas e corporativas, suportando desenvolvimento de aplicações em camadas utilizando componentes que fornecem serviços específicos.

O RM-ODP se apresenta como norma, considerando sua importância por ser um padrão ISO, para o desenvolvimento de aplicações distribuídas abertas, pois este padrão apresenta subsídios para a especificação de arquiteturas distribuídas de software, explorando principalmente a abertura a padrões; separação de problemas em pontos de vistas, transparências e funções de distribuição.

A partir desta visão, o foco deste trabalho recaiu sobre a análise dos componentes J2EE e sua compatibilidade ao padrão RM-ODP com posterior análise dos produtos Websphere, Tomcat e Sun One e a compatibilidade ao RM-ODP.

Palavras-chave: 1. J2EE; 2. Padrões; 3. Ambiente Distribuído; 4. RM-ODP.

Abstract

This study presents the hypothesis that a relation between the elements of J2EE and RM-ODP exists. Both tools are used for the definition and construction of distributed applications.

The norm J2EE presents itself as a complete and open platform for complex and corporative distributed applications, supporting application development in layers and using components which provide specific services. RM-ODP presents itself as a norm, considering the importance of being an ISO norm, for the development of distributed open applications, for this norm presents subsidies for a specification of distributed software architecture, principally exploring the opening to norms; problem separation in points of view, transparency and distribution functions.

From this point of view, the focus of this study lapses to the analysis of the components J2EE and its adhesion to the norm RM-ODP with the posterior analysis of the tools Websphere, Tomcat and Sun One and the adhesion to RM-ODP.

Key-words*:* 1. J2EE; 2. Standard; 3. Distributed System; 4. RM-ODP.

Ninguém pode vos revelar nada a não ser
o que jaz meio adormecido no âmago do vosso conhecimento.

Khalil Gibran

Lista de Ilustrações

Lista de Abreviaturas

API	Application Program Interface
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architectur
COTS	Commercial-off-the-shelf.
EAR	Enterprise Archive File
EJB	Enterprise Java Beans
ERP	Enterprise Resource Planning
HTML	Hypertext Markup Language
HTTP	Hyper-Text Transport Protocol
IDL	Interface Definition Language
ISO/IEC	International Organization for Standardization/ International Electrotechnical Commission
ITU-T	International Telecommunication Union
J2EE	Java 2 Enterprise Edition
JAI	Java Advanced Image
JAAS	Java Authentication and Authorization Service
JAR	Java Archive
JCA	Java Connector Architecture
JCP	Java Community Process
JDBC	Java Database Connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSP	Java Server Pages
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
OCL	Object Constraint language
ODP	Open Distributed Processing
OMG	Object Management Group
ORB	Object Request Broker

PDA	Personal Digital Assistant
RM-ODP	Reference Model for Open Distributed Processing
RMI	Remote Method Invocation
RMI- IIOP	Remote Method Invocation over Internet Inter-Orb Protocol
RUP	Rational Unified Process
SD	Sistema Distribuído
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UML	Unified Modeling Language (OMG)
WAR	Web Archive
XML	Extensible Markup Language

Sumário

1 INTRODUÇÃO

1.1 Introdução

O ambiente computacional atual vem sofrendo transformações que resulta em ampla diversidade de *hardware* e *software*. As corporações, pressionadas por essa evolução, vêm exigindo soluções rápidas e flexíveis de estruturas que reconheçam e respondam às novas oportunidades.

A corrente tendência demonstra a importância dos ambientes paralelos e distribuídos que vêm se estabelecendo, quase como um padrão, tanto no meio acadêmico quanto comercial. Trata-se de uma coleção de serviços localizados em computadores independentes, conectados por uma rede de interconexão e sem memória compartilhada, se apresentando ao usuário final como um único computador.

Nesse sistema, *hardware* e *software* de máquinas autônomas se comunicam através da rede, coordenando suas ações por meio da troca de mensagens, possibilitando:

a. o aproveitamento de máquinas potencialmente ociosas, oferecendo um melhor custo-benefício;
b. a facilidade da implementação de algumas aplicações, por serem distribuídas pelo sistema por natureza;
c. a sobrevivência do sistema como um todo, em caso de falha de uma máquina, em que apresenta apenas uma degradação de desempenho;

d. um crescimento incremental, pois o poder computacional pode ser aumentado através da inclusão de novos equipamentos;
e. maior flexibilidade em comparação às máquinas isoladas e, por isso, muitas vezes são utilizados até mesmo aqueles que não necessitam de maior desempenho[1].

Os sistemas distribuídos são desenvolvidos em *n-camadas* com a finalidade de compartilhamento de dados, separando a lógica de negócio da lógica da apresentação.

A construção de sistemas heterogêneos e distribuídos introduziu problemas não encontrados em sistemas convencionais ou abordados de maneira diferente, tornando o desenvolvimento de sistemas distribuídos uma atividade complexa quando comparada com os sistemas centralizados. Um aspecto importante a ser levado em consideração são os problemas relacionados com a rede como, por exemplo, nomeação, proteção, compartilhamento e questões relativas a traduções que se tornam necessárias devido à heterogeneidade de componentes; questões relacionadas ao gerenciamento de recursos e contas de usuários a fim de evitar o acesso indevido; sincronização, consistência e recuperação de erros além de outros problemas resultantes da distribuição dos componentes.

Um dos grandes desafios atuais é o de ajudar os desenvolvedores a criar aplicativos amigáveis para a Web. “Existem muitos servidores de aplicativos disponíveis para abrigar aplicativos corporativos, e muitos provedores de serviços estão escrevendo ferramentas modulares para conectar e ampliar a rica funcionalidade” (BOND et al. 2003, p. 3). Um simples navegador Web pode tirar proveito dessa arquitetura.

[1] Fonte: GEYER, et al., 2001, p. 1-2.

A plataforma Java 2 Enterprise Edition (J2EE) é um conjunto de especificações coordenadas que, em conjunto com um guia de práticas, permitem o desenvolvimento, a instalação, a execução e o gerenciamento de aplicações *n-camadas* no servidor, com a finalidade de reduzir a complexidade, o tempo e, conseqüentemente, o custo do desenvolvimento de aplicações corporativas. Essa plataforma fornece um ambiente comum para a construção de aplicativos escaláveis, distribuídas e portáveis (BOND et al. 2003, p. 13).

1.2 Motivação

Sistemas distribuídos eram construídos utilizando-se padrões proprietários e muitas vezes esses sistemas eram construídos sem a utilização do conceito de modularidade.

A motivação relacionada aos fundamentos teóricos utilizados apresenta inúmeros conceitos e elementos estruturais e possuem uma grande importância tanto na área acadêmica quanto na indústria de *software*.

O elemento central dessa dissertação é a norma ISO/IEC RM-ODP (*Reference Model of Open Distributed Processing*) para o desenvolvimento de aplicações distribuídas abertas, pois esse padrão apresenta subsídios para a especificação de arquiteturas distribuídas de *software*, explorando principalmente a abertura a padrões.

O segundo elemento, porém não menos importante, é o padrão aberto J2EE da Sun. Uma plataforma completa para a construção de aplicações distribuídas complexas (AHMED, 2002) e principalmente abertas.

É um desafio muito grande construir aplicações abertas que

geralmente é conseguido através da publicação de interfaces, tornando-a disponível para os usuários (não o final, mas sim o desenvolvedor). Nesse aspecto o J2EE se enquadra perfeitamente, pois todos os elementos que compõe esse padrão são publicados pela Sun.

Este trabalho trata da relação existente entre essas duas abordagens arquiteturais para a construção de sistemas distribuídos abertos. O resultado dessa pesquisa é mostrado nas tabelas do capítulo 4.

A motivação acadêmica deste trabalho é mapear e apresentar a relação e compatibilidade existente entre as duas arquiteturas de sistemas distribuídos utilizando padrões abertos.

1.3 Objetivos

1.3.1 Geral

O presente trabalho tem por objetivo propor um critério de avaliação da especificação que compõe o ambiente distribuído J2EE utilizado no ambiente corporativo, utilizando a norma ISO RM-ODP a fim de verificar o quanto J2EE implementa as especificações do padrão ISO. Essa avaliação pode ser usada como critério ne seleção dos produtos que implementam a plataforma J2EE.

1.3.2 Específicos

São objetivos específicos deste trabalho:

a. Apresentação da hipótese da compatibilidade entre as especificações para construção de sistemas distribuídos que são a norma ISO RM-ODP e os elementos do J2EE;
b. Identificar os elementos do J2EE que correspondem a visões do RM-ODP, confirmando a hipótese de relacionamento utilizando o método comparativo dos elementos do J2EE com os modelos da UML e as visões do RM-ODP;
c. Identificação das transparências do RM-ODP com análise do relacionamento entre os elementos J2EE as essas transparências;
d. Análise da compatibilidade dos produtos compatíveis com a indústria ao RM-ODP utilizando o mapeamento dos elementos do J2EE com RM-ODP.

1.4 Contribuição

O presente trabalho realizou um mapeamento dos elementos J2EE sob a visão do RM-ODP, posto ser este um padrão ISO “que atualmente é considerado o mais completo e atualizado padrão para um modelo de arquitetura para sistemas distribuídos e abertos” (AVELINO, 2005, p. 3).

O Modelo Referencial ODP não impõe nenhuma metodologia ou mesmo uma linguagem de modelagem, e sua estruturação baseia-se na orientação a objeto. Por isso, foram utilizados os diagramas da UML como passo intermediário no mapeamento dos componentes do J2EE. O uso da UML se justifica por sua ampla utilização em desenvolvimento de *software* orientado a objeto e por sua facilidade de uso.

Como a UML foi utilizada apenas de forma ilustrativa, não foram consultados os trabalhos correlatos em relação ao mapeamento do RM-ODP para UML, que aqui está representada de maneira original.

Não há qualquer pretensão em esgotar o tema aqui explorado, antes disso, a proposta é contribuir com discussões com a utilização de padrões e normas em sistemas distribuídos de padrões abertos.

1.5 Metodologia

Para a elaboração desta dissertação, a metodologia aplicada segue os seguintes passos:

1. Pesquisa: a pesquisa tomou grande parte do trabalho, em que os vários assuntos como J2EE, RM-ODP e padrões abertos que compõem a dissertação foram identificados e estudados. Durante o período de pesquisa, as informações foram.
2. Análise das especificações: esta fase consistiu na investigação exploratória realizada em dois procedimentos distintos. O primeiro foi a análise das visões do RM-ODP e suas particularidades com posterior identificação dos modelos da UML correspondentes a cada visão do RM-ODP. O segundo procedimento se resumiu na discriminação, definição e análise dos elementos do J2EE, incluindo comparação entre eles e verificação do relacionamento que se deu pela comparação desses elementos com as visões do RM-ODP.
3. A avaliação correspondeu à análise dos produtos compatíveis com o ambiente distribuído J2EE, confrontados com os propósitos do

Modelo Referencial ODP.

1.6 Organização da Dissertação

Para fazer uma avaliação de ambiente distribuído, analisando o Java 2 Enterprise Edition sob a ótica do RM-ODP, foi necessário conceituar e apresentar alguns componentes e aplicações, além de discorrer sobre alguns processos. Para que o presente trabalho tivesse uma sólida base, utilizou-se a seguinte estrutura:

O Capítulo 1 Introdução trata da introdução do tema, os motivos que nos levaram a escolher o assunto discorrido, a contribuição pretendida para o desenvolvimento da área de conhecimento e esta estrutura.

No Capítulo 2 Conceitos Fundamentais estão apresentados conceitos básicos que serviram de base para os temas tratados, além de apresentar produtos e componentes para melhor situar o leitor no tema.

O Capítulo 3 A Plataforma Java 2 Enterprise Edition apresenta o J2EE e seus componentes, discorrendo acerca de sua funcionalidade e apresentando separadamente os conceitos de seus aplicativos.

O Capítulo 4 Visões do RM-ODP e o Padrão J2EE traça um paralelo entre o padrão ISO RM-ODP e a compatibilidade que J2EE permite, com análises concernentes, que servem de base para o capítulo 5.

No Capítulo 5 Análise dos produtos confrontados com RM-ODP apresenta a análise dos produtos que implementam o ambiente distribuído J2EE, apontando semelhanças e diferenças entre elas e estudando sua compatibilidade ao RM-ODP. Os produtos analisados são o Websphere, Tomcat e Sun One.

O Capítulo 6 Conclusões encerra o trabalho, apresentando as

considerações finais sobre o trabalho e as comparações entre os produtos.

Ao final do trabalho segue a lista de obras e literatura consultada para a realização deste. Esta dissertação ainda conta com uma lista de abreviaturas e lista de figuras.

2 CONCEITOS FUNDAMENTAIS

A construção e a definição de sistema distribuído (SD) é uma tarefa desafiadora. Eles são grandes, complexos e considerações diferentes que influenciam seu projeto podem resultar em um substancial corpo de especificação, que necessita ser estruturado para ser controlado com sucesso. Ao se projetar um SD os desafios da distribuição são os seguintes: resolução do negócio, qualidade de serviços, comunicação, segurança, integridade e transferência de dados e integração com sistemas legados.

Uma boa estrutura permite que diferentes partes de um projeto sejam trabalhadas separadamente se elas são independentes, mas deve haver uma identificação clara dos pontos que limitam o projeto.

O negócio é o principal guia para a construção de um sistema. Os desafios relacionados ao negócio vão desde a definição do escopo, objetivos da solução, regras de negócio, comportamento esperado da solução com suas restrições.

Este capítulo apresenta os conceitos básicos de sistemas distribuídos; os estilos arquiteturais de sistemas distribuídos; os padrões utilizados para a construção de sistemas distribuídos;, buscando definir desde uma linguagem de modelagem, processo de desenvolvimento, linguagem de marcação até um padrão de desenvolvimento de sistemas distribuídos.

2.1 Sistemas Distribuídos

Um Sistema Distribuído, SD, é um conjunto de computadores –

processadores e memórias independentes – ligados em rede que se comunicam e interagem através da troca de mensagens. Esses computadores podem estar separados geograficamente por qualquer distância (COULOURIS, 2001).

Essa definição traz consigo algumas características de um SD, como ausência de sistema operacional padrão, pois os computadores são autônomos e não compartilham a memória principal, mas cooperam entre si enviando mensagens assíncronas, utilizando uma rede de comunicação.

2.1.1 Características Principais

As principais características de um sistema distribuído, segundo Couloris (2001), são:

- heterogeneidade;
- abertura a padrões;
- segurança;
- escalabilidade;
- tolerância à falhas;
- concorrência;
- compartilhamento;
- transparência.

Cada uma dessas características diz respeito a um aspecto relevante do sistema distribuído, como se pode observar:

- **Heterogeneidade:** um sistema distribuído é composto de diferentes tipos de redes, linguagens de programação e sistemas operacionais. Um protocolo de comunicação consegue ocultar essas diferenças de redes e um *middleware* pode tratar as outras diferenças. O SD deve permitir o compartilhamento de recursos entre os sistemas de uma empresa, principalmente os legados.
- **Abertura a padrões:** a abertura de um sistema a padrões é uma característica que determina se ele pode ou não ser estendido e implementado de várias formas, mas para que isso seja possível, é preciso publicar as interfaces dos componentes.
- **Segurança:** as informações compartilhadas em SD são valiosas, portanto é necessário um especial cuidado com a segurança. Dessa característica podem ser destacados três componentes: confidencialidade (proteção contra alteração ou manipulação da informação por usuários não autorizados); disponibilidade (proteção contra interferências com o objetivo de obter acesso a um recurso); integridade (proteção contra alteração e corrupção de dados).
- **Escalabilidade:** sistemas distribuídos são escaláveis, ou seja, permitem um aumento na demanda de processamento ou usuários do sistema, permanecendo eficazes, sem perder a performance. Permitem ainda que os sistemas e/ou aplicações sejam expandidos sem uma mudança estrutural ou uma alteração dos algoritmos das aplicações.
- **Tolerância à falha:** SDs são tolerantes à falha quando se explora o potencial de replicação de *software* e *hardware*.

- **Concorrência:** vários processos podem ser executados ao mesmo tempo em diferentes computadores de uma rede, dispensando a utilização de interfaces. Podem ser processos cooperativos, que se comunicam um com o outro para executar uma tarefa mais rapidamente, ou processos que não cooperam entre si.
- **Compartilhamento:** no sistema distribuído, um conjunto de computadores interage entre si, compartilhando informações, recursos, processos e objetos.
- **Transparência:** é definida como a ocultação dos mecanismos de distribuição do sistema, tanto para o usuário quanto para o desenvolvedor.

Segundo Galasini (2004), sistemas distribuídos apresentam as seguintes desvantagens:

- **Complexidade:** maior complexidade em relação a sistemas centralizados, pois ocorre mais identificação das variáveis que tornam os sistemas indisponíveis.
- **Gerenciamento de falha:** há um esforço maior na gerência e na manutenção de um sistema distribuído, pois erro em uma máquina pode proporcionar erros em outras.
- **Imprevisibilidade:** SDs são não determinísticos, pois eles dependem da infra-estrutura de comunicação e funcionamento de várias máquinas para desempenho perfeito. Sem isso não há como prever o tempo de resposta, o que pode variar entre os usuários.

2.1.2 Vantagens de um sistema distribuído

Compartilhamento de recursos é uma das vantagens do uso de um sistema distribuído, pois recursos de alto custo podem ser usados por vários usuários ao mesmo tempo, como informações em arquivos, dispositivos de alto custo e motores de busca.

O desempenho de um sistema distribuído é maior que o de um centralizado. Segundo Deitel (2005), esses sistemas são projetados para melhorar a capacidade de processamento e armazenamento, com a confiabilidade de uma única máquina, pois requisições dos usuários podem ser enviadas a diferentes servidores que trabalham em paralelo para aumentar o desempenho.

A estrutura de um SD é composta por muitas máquinas que trabalham em conjunto tornando o sistema tolerante a falhas, pois não depende de uma única máquina.

A falha de um ou mais recursos em máquinas isoladas não afeta a disponibilidade dos recursos do sistema, pois outro servidor, através de comunicação grupal, pode atender os serviços de um servidor com falha.

O crescimento incremental de um sistema distribuído permite sua extensibilidade ou escalabilidade, adicionando mais máquinas ao sistema conforme a necessidade, sem afetar as aplicações e os usuários existentes.

2.1.3 Exemplos de sistemas distribuídos

Exemplos clássicos de sistemas distribuídos são os sistemas

operacionais Amoeba[2] e Mach[3], por seu valor histórico e acadêmico, sistemas distribuídos mais específicos como Globo[4], AFS[5], Jini[6], projeto SETI (SETI, 1999).

Para efeito de exemplos, este trabalho apresenta uma pequena descrição apenas do CORBA, seguindo a linha dos padrões abertos para a construção de aplicações distribuídas.

2.1.3.1 CORBA

CORBA (Common Object Request Broker Architecture) é uma especificação-padrão de arquitetura de sistemas distribuídos, concebida na década de 1990 pelo Object Management Group (OMG), sendo um padrão aberto elaborado para habilitar a interoperabilidade entre programas em sistemas heterogêneos, suportando objetos tanto na passagem de parâmetros como no retorno na chamada a procedimentos durante a comunicação entre processos. Essa especificação é independente da linguagem de programação e do sistema operacional, operando através de um núcleo comum da arquitetura CORBA.

Este modelo, baseado em objetos, permite que métodos de objetos sejam ativados remotamente, utilizando uma rede, através de um elemento intermediário chamado ORB (Object Request Broker) situado entre o objeto propriamente dito e o sistema operacional. O ORB é o componente mais importante da arquitetura, pois permite que objetos façam e recebam requisições de métodos transparentemente em um ambiente distribuído e heterogêneo.

---

[2] AMOEBA WWW HOME PAGE. Disponível em: <www.cs.vu.nl/pub/amoeba/>.
[3] The mach Porject Home Page. Disponível em: <http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html>.
[4] The GLOBE Project. Disponível em: <http://www.cs.vu.nl/globe/>.
[5] Welcome to the home of OpenAFS. Disponível em: <http://www.openafs.org/frameless/main. html>.
[6] Jini Architecture Specification: Version 1.2. Disponível em: <http://www.sun.com/software/jini/specs/jini1.2html/jini-title.html>.

O CORBA fornece aos usuários independência de linguagem por intermédio da IDL (Linguagem de Definição de Interface), permitindo que programadores definam os procedimentos que podem ser chamados no objeto.

2.2 Estilos arquiteturais de sistemas distribuídos

Um modelo arquitetural de sistemas distribuídos primeiro simplifica e abstrai as funções de cada elemento, segundo Coulouris (2001), considerando:

- a localização dos componentes através da rede – define padrões úteis para a distribuição de dados e o balanceamento de carga;
- o relacionamento entre os componentes – procura definir as regras funcionais e os padrões de comunicação entre eles.

Uma simplificação inicial é obtida pela classificação dos processos ou componentes como: Processos Servidores, Processos Clientes e Processos Peer. Esta classificação de processos identifica as responsabilidades de cada processo, auxiliando na definição de suas cargas de trabalho e, principalmente, para determinar o impacto de falhas em cada um deles. Os resultados dessa análise podem ser utilizados para determinar o local dos processos, de forma que sejam encontrados os objetivos de performance e confiabilidade do sistema resultante.

Sistemas dinâmicos são construídos como uma variação de modelo cliente/servidor:

- a possibilidade de mover o código de um processo para outro permite que um delegue tarefas para o outro, por exemplo: o cliente

pode fazer *download* de código de um servidor e executá-lo localmente (código móvel). Objetos e o código que os acessam podem ser movidos para reduzir a espera de acesso, minimizando o tráfego de comunicação;

- alguns SDs são implementados para possibilitar que computadores e dispositivos móveis sejam adicionados ou removidos facilmente, permitindo que descubram os serviços disponíveis e oferecendo seus serviços a outros.

Existem vários padrões arquiteturais que podem ser utilizados para a alocação de trabalho em um sistema distribuído e que tem um importante impacto na performance e na eficiência do sistema resultante.

2.3 Padronização

Os padrões expandem os mercados e reduzem a fricção que impede a transação. Renaud (1994) defende que padrões promovem o crescimento de produtos competitivos, aumentando a faixa de alternativas disponíveis.

Os padrões permitem que as empresas se focalizem em problemas comerciais específicos, ao invés de se preocupar com problemas técnicos complexos. Eles oferecem uma língua franca – uma linguagem comum que permite a qualquer empresa, em qualquer lugar e a qualquer momento, fazer parte do mercado. Em um ambiente totalmente padronizado, os vendedores diferenciam seus produtos pelo preço, desempenho e qualidade.

A tendência de padronização está nos levando a um ambiente de

sistemas “fácil”. A capacidade que uma grande faixa de produtos “plugar e usar” (*plug and play*) nos permitem planejar sistemas de forma bastante modular. A tecnologia cliente/servidor se encaixa naturalmente nesse ambiente “fácil”.

Padrões coerentes, largamente aceitos na computação empresarial são importantes, principalmente com o advento da Internet, já que representa, neste novo paradigma, a sobrevivência na nova economia. Quanto mais uma empresa puder usar padrões para se conectar de modo eficaz com seus clientes, fornecedores e parceiros, maior será a eficácia com que poderá participar no mercado e mais competitiva será na economia em rede.

Os padrões têm a mesma facilidade do dinheiro: eles facilitam a troca, oferecendo um canal comum, pelo qual os negócios são realizados.

2.3.1 UML

A UML, ou *Unified Modeling Language*, é um padrão aberto, controlado pelo OMG, um consórcio de empresas que apostaram em um padrão que suportasse interoperabilidade, especificamente de sistemas orientados a objetos. A UML, considerada uma linguagem de modelagem, define uma notação e um meta-modelo. A notação representa a sintaxe da linguagem composta por elementos gráficos. Um meta-modelo é um diagrama de classes.

Ela pertence a uma família de notações gráficas que, com um meta-modelo único, ajuda na descrição e no projeto de sistemas de *software*, “particularmente daqueles construídos utilizando o estilo orientado a objetos” (FOWLER, 2005, p. 25).

Nos diagramas utiliza-se o conceito de modelos de elementos que

representam definições comuns da orientação a objetos como as classes, objetos, mensagem, relacionamentos entre classes incluindo associações, dependências e heranças.

Os diagramas são os gráficos que descrevem o conteúdo em uma visão. A UML possui 13 tipos de diagramas que são usados em combinação para prover todas as visões do sistema.

A OMG, em 1997, adotou a especificação UML como uma linguagem gráfica para visualizar, especificar, construir e documentar artefatos de *software*. Segundo Silva (2003, p. 30), apesar de contribuir sensivelmente para o processo de desenvolvimento de *software*, a UML não define nenhuma metodologia ou processo para desenvolver *software*. Em sua versão 1.4, a especificação apresenta, em sua estrutura:

a. a semântica da linguagem UML,
b. um guia da notação (sintaxe da linguagem UML),
c. alguns exemplos de Perfis UML (mecanismo de extensão),
d. intercâmbio de Modelos (via XML e IDL),
e. e uma Linguagem de restrições em Objetos (OCL – *Object Constraint Language*).

As partes que compõem a UML são: visões; modelagem de elementos, mecanismos gerais e diagramas, conforme apresentadas por Esmin (1999, p.02). Na UML, as visões mostram diferentes aspectos do sistema que está sendo modelado. Não se trata de um gráfico, mas sim de uma abstração, composta por uma série de diagramas.

Os mecanismos gerais provêm comentários suplementares,

informações, ou semântica sobre os elementos que compõem os modelos.

A UML se adequa às cinco fases de desenvolvimento de *software*: análise de requisitos, análise, projeto, implementação e testes. Estas fases não necessariamente devem ser executadas na ordem seqüencial. As fases podem ser descritas como:

a) *Análise de requisitos:*

Durante essa fase, são capturadas as necessidades dos usuários e o comportamento do sistema através de análise de casos de uso chamados “Use Case”. As entidades externas ao sistema, ou atores externos, que interagem com o sistema são modeladas entre as funções que eles necessitam. O diagrama de caso de uso é usado para identificar como o sistema se comporta em várias situações que podem ocorrer durante sua operação;

b) *Análise:*

Identificam-se, nesta fase, as classes, objetos e mecanismos presentes no domínio do problema. As classes são modeladas e interligadas através de relacionamentos utilizando o diagrama de classe. Na análise, só são modeladas classes que pertencem ao domínio do problema;

c) *Projeto* (*design*)*:*

O resultado da análise é expandido nesta fase em soluções técnicas. Novas classes são adicionadas para prover uma infra-estrutura técnica: a interface do usuário e de periféricos, gerenciamento de banco de dados, comunicação com outros sistemas, entre outros. As classes do domínio do problema modeladas na fase de análise são mescladas nessa nova infra-estrutura, tornando possível alterar tanto o domínio do problema quanto a infra-estrutura. O projeto é denominado como *design*, que resulta no detalhamento das especificações para a fase seguinte.

d) *Implementação:*

Nesta fase, as classes são convertidas para código real em uma linguagem orientada a objetos, pois procedural não é recomendado. Dependendo da capacidade da linguagem usada, essa conversão pode ser uma tarefa fácil ou não.

e) *Testes:*

Idêntico a qualquer outro método de modelagem. Dividida em testes de unidades, testes de integração, teste de sistema e testes de aceitação.

2.4 RM-ODP

O padrão *Reference Model for Open Distributed Processing* (RM-ODP), criado em parceria pelo ISO/IEC e ITU-T, consiste de uma série de atividades de padronização para a descrição de sistemas de processamento distribuído, para desenvolvimento de uma estrutura de suporte à distribuição, interconexão, interoperabilidade e portabilidade de sistemas computacionais. Possui uma metodologia orientada a objetos para especificação dos requisitos deste domínio de aplicação.

2.4.1 Conceitos básicos

Segundo Varoto (2002), a arquitetura do padrão RM-ODP fornece:

a. Interoperabilidade: troca de informações e uso conveniente de funcionalidades através de sistemas distribuídos, quando do

estabelecimento das interfaces de comunicação/utilização.

b. Transparência na distribuição: oculta conseqüências de distribuição da programação, reduzindo sua complexidade.
c. Portabilidade e interoperabilidade: de aplicações, compartilhadas por plataformas heterogêneas.

Portanto, conforme afirmado anteriormente, pode-se dizer que a proposta do padrão RM-ODP se baseia em um conjunto de objetivos básicos, como alcançar a portabilidade de aplicações entre plataformas heterogêneas; levar à interoperabilidade de sistemas ODP, garantindo uma troca significativa de informações e o uso conveniente de todas as funcionalidades por todo o sistema distribuído; e proporcionar uma transparência de distribuição, escondendo as conseqüências da distribuição tanto do programador de aplicações como do usuário.

Para esse fim, Coelho (1998, p. 13) aponta os componentes da especificação do RM-ODP:

1. *Um guia de uso e de visão geral do modelo de referência*[7]*:*

Contém a motivação do padrão, fornecendo o escopo, a justificativa e a explicação dos conceitos-chave, além de algumas linhas gerais sobre a sua arquitetura. Explica como o ODP deve ser entendido e aplicado por usuários, arquitetos e padronizadores de sistemas concordantes.

2. *Modelo Descritivo*[8]*:*

[7] ITU-T Rec. X.901 | ISO/IEC 10746-1 - *Basic Reference Model of Open Distributed Processing: Part 1: Overview and Guide to Use*, 1998.
[8] ITU-T Rec. X.902 | ISO/IEC 10746-2 - *Basic Reference Model of Open Distributed Processing: Part 2: Descriptive Model*, 1996.

*A*presenta a definição, a estruturação e a notação de conceitos para uma descrição normatizada, provendo um vocabulário comum para um sistema distribuído e aberto. Compreende um nível de detalhe no suporte ao Modelo Prescritivo e no estabelecimento de requisitos a novas técnicas de especificação.

3. Modelo Prescritivo[9]*:*

Traz o conjunto das características que qualificam um sistema de processamento distribuído como aberto. Descreve os requisitos de um SD aberto e se baseia nas cinco visões do Modelo de Referência ODP.

4. Semântica da Arquitetura[10]*:*

É a formalização dos conceitos básicos de modelagem ODP definidos no Modelo Descritivo. A formalização é atingida ao interpretar cada conceito em termos da construção de diferentes técnicas padronizadas de descrição formal.

Conforme a filosofia proposta pelo RM-ODP, ao invés de se lidar com toda a complexidade de um sistema distribuído, pode-se observá-lo sob *pontos de vista* distintos, cada qual refletindo um conjunto de atributos diferentes, com ênfase sobre um determinado assunto. Cada ponto de vista representa um nível de abstração particular do sistema distribuído original, suprimindo a necessidade de se criar um grande modelo que descreva todos esses níveis em detalhes.

Segundo Coelho (1998, p. 13), a caracterização do sistema sob a ótica de cada uma das visões permite a análise tanto de sua descrição geral como da consistência entre as diferentes abstrações possíveis. Não se trata de um modelo hierárquico, pois não impõe dependência na seqüência nem entre as diferentes visões, estas consideradas ortogonais.

---

[9] ITU-T Rec. X.903 | ISO/IEC 10746-3 - *Basic Reference Model of Open Distributed Processing: Part 3: Prescriptive Model*, 1996.
[10] ITU-T Rec. X.904 | ISO/IEC 10746-4 - *Basic Reference Model of Open Distributed Processing - Part 4: Architectural Semantics*, 1998.

2.4.2 Visões

No RM-ODP, os problemas específicos de sistemas distribuídos são identificados pelas transparências e pelas cinco visões. Cada visão, segundo Beitz (apud VAROTO, 2002) possui conceito, estrutura e regras:

a) *Visão da empresa:*

Que trata do propósito, escopo e restrições que regem o negócio, como regras do usuário em relação ao sistema e ao ambiente com o qual interage. Utiliza regras e política da empresa para implementar recursos no sistema; note-se que aqui ao termo “empresa” não se refere a uma única organização, mas sim sua condição, posto que o modelo descreve a interação entre um grande número de organizações distintas e, segundo Coelho (1998, p. 14), esta visão procura caracterizar os requisitos empresariais.

b) *Visão da informação:*

Que trata da semântica e do processo de informação, provendo uma interpretação geral e consistente do sistema e cobrindo as fontes, os destinos e os fluxos dos dados. Utiliza "esquemas" para descrever a informação requisitada pelo sistema, especificando o estado e a estrutura dos objetos. Inclui o conceito de objetos compostos. As regras e restrições que governam a manipulação da informação;

c) *Visão computacional*:

Que especifica a decomposição das funcionalidades de uma aplicação ODP de forma transparente, considerando o particionamento lógico do sistema

distribuído em vários objetos que interagem entre si. Ainda encapsula dados e processamento e oferece uma ou várias interfaces para interação com outros objetos. Este modelo é representado por interface; atividade; e contrato de ambiente. Segundo Varoto (2002), a reusabilidade é alcançada nesta visão em função da distribuição dos componentes que constituem a aplicação;

d) *Visão de engenharia:*

Que trata da infra-estrutura necessária para suportar a distribuição. Descreve o projeto (*design*) dos aspectos orientados a distribuição do sistema ODP, mas não se concentra na semântica da sua aplicação, exceto para determinar seus requisitos de distribuição e transparência. O modelo correspondente a esta visão inclui objetos e canais e salienta os mecanismos e funções requeridas para o suporte à interação distribuída entre os objetos do sistema;

e) *Visão tecnológica:*

Trata da escolha de tecnologias, descrevendo a implementação do sistema ODP e considerando os aspectos de identificação, distribuição e instalação de tecnologias de *hardware* e *software* específicas que suportam e constituem o sistema, fornecendo detalhes de componentes e *links* com os quais um sistema distribuído é construído.

O padrão RM-ODP utiliza as visões para dividir a grande e complexa especificação de sistema distribuído em pequenas partes, gerenciáveis pelos participantes do desenvolvimento do sistema (VAROTO, 2002).

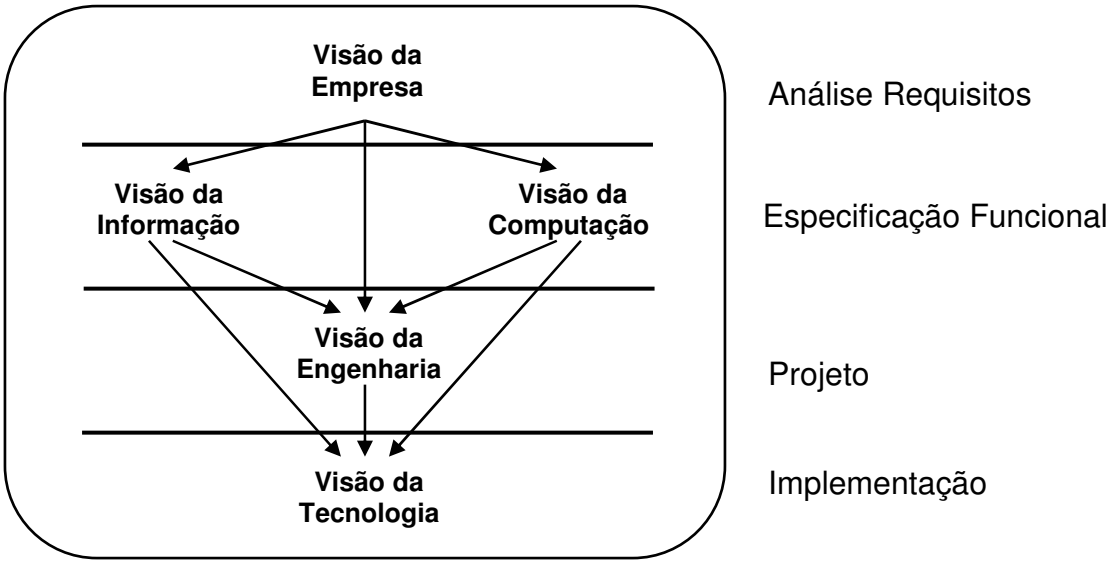

Figura 1 Visões de RM-ODP

Fonte: VAROTO, 2002.

Por ser específico para ambientes de processamento distribuído, segundo Varoto (2002), considera as visões como uma completa descrição auto-contida dos requisitos do sistema em diferentes níveis de detalhamento, específicos para as necessidades de cada um dos profissionais envolvidos no processo. Para Varoto (2002), o esquema é assim constituído não para padronizar componentes ou influenciar na escolha da tecnologia, mas sim para organizar partes do sistema ODP em um todo coerente; o autor aponta que o termo “modelo de visão”, utilizado na descrição das visões, não indica nem sintaxe nem semântica específicas, antes fornece uma terminologia básica, necessária para modelar os interesses de cada uma das visões.

No RM-ODP, um conceito estendido de objeto, oriundo de técnicas tradicionais de orientação a objetos, é utilizado para atender às complexidades particulares do domínio do sistema.

A visão da empresa compreende os aspectos relacionados ao negócio e suas restrições, como, por exemplo, questões como a solução distribuída deve se adequar às questões políticas e organizacionais da empresa para a qual o sistema está sendo construído. Segundo Varoto (2002), essa visão se localiza na etapa de modelagem do processo, na fase de engenharia de requisitos, quer dizer, é a primeira atividade da arquitetura.

2.4.3 Transparências

As transparências permitem ocultar as complexidades inerentes a qualquer sistema distribuído, quando for especificada, permitindo uma maior capacidade de abstração. Podem ser definidas como a ocultação dos mecanismos de distribuição do sistema, tanto para o usuário quanto para o desenvolvedor**.** Segundo o padrão RM-ODP (ISO, 1996), são classificadas em transparências que podem ser fornecidas pelos sistemas distribuídos:

a) *Acesso:*

Esconde os diferentes mecanismos que garantem que recursos (objetos de informação) locais e remotos sejam acessados, usando operações idênticas, ou seja, protocolos de rede habilitam a comunicação entre computadores, fornecendo um meio universal de acesso a dados armazenados em diferentes formatos em todo o sistema. Provê a interoperabilidade entre arquiteturas de computadores e linguagens de programação heterogêneas.

b) *Localização:*

Em que os recursos são utilizados sem que se saiba a sua localização, aproveitando a transparência de acesso para ocultar a localização de recursos no

sistema distribuído e mascara a localização de interfaces. Segundo Couloris (2001), a transparência da localização e de acesso são as mais importantes atualmente.

c) *Falha:*

Oculta possíveis falhas e erros que possam ocorrer em objetos, providenciando, também, sua recuperação, para permitir certos aspectos de tolerância a falhas, possibilitando que usuários/aplicações consigam cumprir suas tarefas. Se um ou mais recursos falharem, os usuários perceberão apenas uma redução de desempenho. Esta transparência é, muitas vezes, implementada utilizando a replicação, na qual um sistema fornece vários recursos que executam a mesma função, ou por ponto de verificação/recuperação, que consistem em armazenar periodicamente o estado do sistema de um objeto (como um processo) que possa ser restaurado na presença de uma falha.

d) *Replicação:*

Providencia múltiplas instâncias de um mesmo serviço para proporcionar confiabilidade e disponibilidade, sem que isso seja percebido pelos usuários ou programas-cliente, que não devem saber o número de cópias existente de um recurso. Todos os acessos a um grupo de recursos replicados ocorrem como se houvesse um só recurso disponível.

e) *Persistência:*

Esconde a distinção entre componentes ativos e passivos. Mascara a alocação/desalocação e a ativação/desativação de recursos para os componentes do sistema; oculta a informação sobre o lugar onde esse recurso está armazenado, se em memória primária ou disco.

f) *Migração*

Encobre a heterogeneidade dos componentes do sistema a fim de

possibilitar a migração de funções e aplicações; oculta a movimentação de componentes através de um sistema distribuído, mascarando a movimentação de um arquivo de um servidor para ou outro, ou mesmo de um servidor para um cliente;

g) *Transação:*

Permite que um sistema obtenha consistência, mascarando a coordenação, supervisão e recuperação das transações entre objetos de forma externa, sem ter que empacotar os próprios objetos, muitas vezes utilizando as técnicas de replicação ou ponto de verificação. Possibilita a manutenção da integridade e consistência dos dados, aumentando a confiabilidade e a tolerância a falhas, e a melhoria do desempenho via a execução de operações em paralelo.

h) *Relocação*

Esconde a relocação de uma interface de outras interfaces. A relocação permite que o sistema continue operante mesmo quando a migração ou relocação de alguns objetos criarem uma inconsistência na visão do usuário.

O Modelo Referencial ODP define uma série de funções e estruturas para realizar tais transparências. Contudo, como aponta Moreno (2001, p. 04), cada uma delas pode ser associada a um compromisso entre a qualidade de seu funcionamento requerido e seu custo (seja de tempo ou de recursos); embora nem sempre todas as transparências sejam relevantes para todos os casos. Portanto, ODP não obriga a suportar todas estas transparências, embora seja indicado que, no caso de suportar alguma delas, deve ser utilizada de acordo com as normas para garantir a conformidade com este padrão, e a integração com outros sistemas ODP.

Coelho (1998, p. 15) aponta, como suas funções:

- *Gerência*, para controlar o ciclo de vida dos objetos, os quais

são agrupados numa hierarquia formada por *nós*, *cápsulas*, *clusters* e *objetos básicos de engenharia*;

- *Coordenação*, para orquestrar as atividades de grupos de objetos distribuídos;
- *Repositório*, para manutenção de bases de dados gerais e específicas de informação, provendo serviços de armazenagem e persistência de objetos; e
- *Segurança*, para o estabelecimento de um marco padrão de segurança que garanta restrição de acesso aos objetos.

O RM-ODP prevê que a interação entre os objetos aconteça através do estabelecimento de uma conexão denominado canal. O conceito de canal é importante para aplicações multimídia distribuídas, pois tal mecanismo pode transportar fluxos contínuos de dados, ter suas grandezas (tais como taxa de transmissão e atraso) monitoradas, ter seu fluxo sincronizado com o de outro de mesma categoria, oferecendo, ainda, comunicação multiponto (de uma fonte para muitos destinos). Como o RM-ODP é um padrão de referência, este somente especifica quais funcionalidades devem estar disponíveis às aplicações e não como estas devem ser implementadas. Para isso, padrões adicionais devem surgir.

2.5 Desfecho

Um sistema distribuído necessita ser tratado de um modo diferente de um sistema centralizado. Atualmente, um fator importante para o sucesso dos negócios é um sistema que possibilite a integração de vendedores, fornecedores e empregados através da Internet. Hoje as soluções de negócios vão desde

disseminação de informações e serviços ao cliente, para comércio eletrônico totalmente automatizado.

A tecnologia da Internet possibilita essa nova geração de aplicações Web, construídas na Internet para fornecer uma poderosa capacidade de integração com uma plataforma-neutra. Os servidores de aplicação baseados em Java são componentes da estratégia baseada na Web para implementação desta nova geração da tecnologia da informação.

Ahmed (2002, p. 13) aponta que o suporte ao *hardware* e a independência operacional que o J2EE oferece permite o acesso aos serviços através do Java e dele próprio, não necessitando das APIs do sistema subjacente, o que faz com que sistemas comerciais que utilizam essa arquitetura possam ser facilmente portados entre diversos sistemas de *hardware* e sistemas operacionais.

Uma justificativa para a utilização do padrão J2EE é que ele próprio é formado por padrões abertos como o XML, HTML e SOAP; outra muito importante é a construção de aplicações em camadas, separando a lógica de negócio da lógica da apresentação, possibilitando um desenvolvimento mais rápido, pois componentes de *software* podem ser adquiridos para compor uma nova aplicação.

3 A PLATAFORMA JAVA 2 ENTERPRISE EDITION

A plataforma J2EE é uma especificação amplamente aceita pelo mercado e mundialmente utilizado no desenvolvimento de aplicações multicamadas distribuídas. Trata-se de um padrão dinâmico, utilizado para a produção de aplicativos corporativos seguros, escaláveis, independentes de plataforma e altamente disponíveis.

O J2EE define uma arquitetura para desenvolver aplicações Java comerciais complexas e distribuídas, e consiste em regras de construção para desenvolver aplicações comerciais usando o J2EE; uma implementação de referência para fornecer uma visão operacional do J2EE; um conjunto de testes de compatibilidade para ser usado por terceiros para verificar a compatibilidade de seus produtos com o J2EE; várias APIs para permitir o acesso genérico para recursos comerciais e infra-estrutura e tecnologias para simplificar o desenvolvimento Java comercial

3.1 Tecnologias da plataforma J2EE

A plataforma Java 2 Enterprise Edition (J2EE) é um conjunto de práticas, especificações e tecnologias da plataforma Java, gerenciadas pela Sun Microsystems, e que, juntas, visam oferecer suporte ao *design*, desenvolvimento e *deploy* de aplicações de missão crítica componentizadas, baseadas em objetos distribuídos e multicamadas. Trata-se de um esforço de padronização da Sun e um consórcio de empresas que, juntos, fizeram com que a plataforma J2EE tenha se firmado como referência de qualidade e um padrão da indústria de *software*.

Algumas das tecnologias de maior importância da plataforma J2EE são:

a. Servlets e JSP´s (Java Server Pages), responsáveis por aspectos de apresentação e tratamento de eventos;
b. Enterprise JavaBeans (EJB), responsáveis pelo mapeamento da lógica de negócio em componentes reutilizáveis baseados em tecnologia de objetos distribuídos.
c. JDBC é responsável pela transparência no acesso a bancos de dados;
d. JNDI (Java Naming and Directory Interface), que fornece um serviço de nomes e diretórios para o registro e obtenção de referências a recursos.
e. JVM (Java Virtual Machine) fornece um ambiente de execução para aplicativos serem executados em diferentes plataformas.
f. Applet é um pequeno programa que é armazenado em um computador remoto que os usuários se conectam através de um navegador Web, sendo descartado após completar a execução.

Segundo Santos Júnior (2002, p. 31), em adição às tecnologias da Plataforma Standard Edition, foram criadas algumas tecnologias que servem de suporte à plataforma J2EE:

a. CORBA (Common Object Request Broker Architecture); É um padrão aberto para sistemas heterogêneos, fornecendo um *framework* para utilização de objetos distribuídos, permitindo interoperabilidade com outras linguagens de programação e

tecnologias.

b. ECperf - Composto por uma especificação e um kit de *software* tendo como finalidade medir a performance e escalabilidade de servidores de aplicação J2EE. Foi desenvolvido através do Java Community Process em parceria com fabricantes de servidores J2EE.

A plataforma permite focar na lógica do negócio, deixando os detalhes de infra-estrutura para o ambiente de execução que dá suporte a J2EE. Além disso, as aplicações construídas pelas tecnologias da plataforma J2EE obtêm transparentemente duas vantagens importantes: independência de plataforma, seja ela o sistema operacional ou *hardware*, através da Máquina Virtual Java (JVM) e independência de Implementação, uma característica da especificação que garante o desenvolvimento e implantação em qualquer ambiente que forneça suporte e esteja certificado à plataforma, através da API (Application Programming Interfaces) JDBC e tecnologia CORBA para a interação com sistemas legados.

A máquina virtual Java fornece um ambiente de execução eficiente através das diversas plataformas de *hardware* e é, segundo Radhakrishnan (2000), a base da tecnologia Java, permitindo que o código seja escrito uma única vez e executado em qualquer lugar.

A plataforma J2EE fornece um ambiente comum para a construção de aplicativos corporativos seguros, escaláveis e portáveis. Os requisitos do ambiente exigem padrões abertos para construir aplicativos.

A arquitetura da plataforma J2EE permite o desenvolvimento de aplicações divididas em camadas com funcionalidades específicas. Uma aplicação

Web típica seria composta por uma camada de apresentação, geralmente baseada em Servlets ou JSP, que por sua vez obtém e fornece dados para a camada de negócios, baseada em Enterprise JavaBeans, persistindo dados através de objetos de persistência. Esta camada intermediária acessa a camada de informação, composta por bases de dados relacionais, aplicações legadas, sistemas ERP, entre outros.

As aplicações distribuídas mais simples geralmente possuem uma camada de cliente no *desktop*, além de uma camada de servidor em uma máquina separada, acessível a vários clientes. As mais complexas podem ser configuradas, oferecendo camadas de lógica comercial nas camadas intermediarias e acrescentando uma camada de banco de dados no *back-end*. Aplicações transacionais são aquelas que envolvem a modificação e a atualização de dados a partir de diversas fontes, com operações que precisam ser concluídas ou canceladas como um todo, são as chamadas transações atômicas, pois são indivisíveis.

A camada do cliente de uma aplicação distribuída normalmente é executada em um *browser* no *desktop* do usuário. Os clientes também podem ser aplicações isoladas ou outros processos. Eles podem ser executados em outro dispositivo, como telefones celulares ou PDAs (os computadores de mão).

A camada da Web normalmente é executada em um servidor centralizado ou em servidores localizados dentro da empresa, oferecendo conteúdo a vários clientes ao mesmo tempo. A camada da Web pode realizar operações como manter informações de estado sobre cada usuário que acessa as páginas no servidor e acessar outras camadas da aplicação.

A camada de lógica comercial geralmente entra em ação quando o servidor da Web precisa acessar comportamentos específicos que se aplicam às

diretrizes comerciais para o gerenciamento de uma empresa ou serviço *on-line*.

O banco de dados, inclusive fazendo parte de um sistema legado, oferece armazenamento e acesso básico para os dados da organização. A camada da origem de dados pode consistir em vários sistemas, adquiridos em épocas diferentes para finalidades diferentes.

A tecnologia Java no servidor começou de modo muito simples com o JDBC, que permitiu que clientes escritos na linguagem Java acessassem banco de dados no servidor usando APIs padrão. Java Servlets foi a primeira tecnologia específica de servidor para Web, projetada para substituir programas CGI (Commom Gateway Interface) escritos em um modo independente da plataforma por uma tecnologia que oferecia as capacidades “escrever uma vez e executar em qualquer lugar“ da tecnologia Java.

As aplicações em rede são essencialmente aplicações multicamadas, baseadas em um servidor, aceitando interações entre uma série de sistemas. Essas aplicações são executadas em vários dispositivos diferentes, incluindo *mainframes* para o acesso aos dados no *back-end*, servidores para o suporte a Web e monitoração de transação na camada intermediária e vários dispositivos clientes gordos — aplicações isoladas no *desktop* — e clientes magros, como aplicações rodando em um *browser* no *desktop*. Para aplicações de empresa a empresa, a computação distribuída envolve conexões ponto a ponto entre sistemas servidores dispersos.

A proliferação de sistemas e dispositivos e a extensão dos serviços oferecidos pelo servidor aumentaram a complexidade do projeto, desenvolvimento e distribuição de aplicações distribuídas, que são cada vez mais solicitadas para integrar a infra-estrutura existente, incluindo sistemas de gerenciamento de BD,

sistemas de informações coorporativas e aplicações e dados legados, e também para projetarem esses recursos em um ambiente em evolução, com clientes diversificados em locais diferentes.

O J2EE possui um padrão para a criação de aplicações distribuídas, isto é, aplicações que são executadas em vários sistemas de computador ao mesmo tempo, as quais são complementares entre si. Trata-se de diferentes aplicações executando processos diversos que, juntos, serão responsáveis pela efetivação de uma negociação qualquer entre empresas. Essas aplicações, atuando em conjunto, necessitam ser efetivadas ou canceladas como um todo, dependendo da efetivação da negociação.

Elas podem ser executadas em camadas diferentes, com sistemas, plataformas e máquinas diferentes.

Uma aplicação em Java, por meio da plataforma J2EE, é formada pela união de diversos componentes que se comunicam entre si, cada um desempenhando uma funcionalidade especifica. Segue abaixo alguns exemplos de componentes:

a. componente do tipo cliente — um applet é um componente do tipo cliente
b. componente do tipo servidor — um servlet é um componente do tipo servidor, da mesma forma que um applet
c. componente de negócio — os componentes de negócio são criados a partir da tecnologia Enterprise Java Beans e englobam todas as funcionalidades para um determinado tipo de negócio.

Componentes são programas que podem ser reutilizados quantas vezes forem necessárias e possuem certas características e comportamentos que possibilitam sua comunicação com outros componentes. Um componente é um pedaço de *software* que possui uma certa funcionalidade e permite criar um sistema a partir da união de diversos deles.

Os componentes encapsulam serviços ou aplicações semanticamente significativas, sendo que os componentes padrões, segundo Krieger (1998), apontam como construir e interconectar componentes de *software*, especificando como deve se apresentar independente de sua implementação interna e é isso que distingue esse sistema da comunicação convencional.

A plataforma J2EE possui uma série de componentes que permite agilizar o processo de desenvolvimento de um sistema, ocultando diversos detalhes de implementação já embutidos nos componentes.

Segundo Bond et al. (2003,13), o padrão J2EE define quais serviços devem ser fornecidos pelos servidores que suportam J2EE. Esses servidores fornecerão contêineres J2EE nos quais os componentes J2EE serão executados. Os contêineres fornecerão um conjunto definido de serviços para os componentes. A especificação J2EE disponibiliza uma definição por meio da qual fornecedores corporativos podem produzir servidores de aplicativos J2EE, nos quais os aplicativos compatíveis com J2EE podem ser implementados.

Embora a especificação J2EE defina um conjunto de serviços e tipos de componentes, ela não contém informações sobre como organizar a arquitetura lógica em máquinas físicas, ambientes ou espaços de endereços.

3.1.1 Contêineres

Um conceito muito importante para o entendimento da arquitetura de aplicações J2EE é o de contêiner. Em cada camada existem contêineres, que são interfaces entre um componente e uma plataforma específica, possibilitando definir ou restringir as ações de certos componentes. Antes que um componente possa ser executado, ele deve ser inserido em um contêiner. Os contêineres podem ser comparados a um ambiente em que o componente está inserido e deve seguir certas regras de funcionamento. Essa característica facilita a criação de aplicações, uma vez que só é preciso se preocupar com o funcionamento da aplicação em si e não em detalhes de implementação ou segurança, agilizando o processo de desenvolvimento.

O contêiner refere-se a uma entidade de *software* responsável por gerenciar a execução de componentes. Estes residem e atuam dentro de contêineres, que fornecem um contexto compartilhado para que ocorra interação entre componentes, oferecendo acesso a serviços comuns entre eles, mesmo quando implementados de forma a serem aninhados em outros contêineres (KRIEGER, 1998).

A plataforma J2EE é baseada no conceito de componentes, os quais representam unidades de *software* interdependentes e reutilizáveis, sendo necessária apenas a utilização do contêiner compatível com a especificação do tipo de componentes para publicação e execução destes, que representam as funcionalidades exigidas pelas aplicações coorporativas e a combinação destes componentes forma a aplicação em si, sendo que alguns podem ser reutilizáveis em outras aplicações e até mesmo por empresas diferentes.

A plataforma J2EE está organizada nos seguintes tipos de contêineres:

a. *J2EE Server* — representa a infra-estrutura básica do lado servidor da plataforma J2EE, englobando os contêineres EJB e Web;

b. *Contêiner Enterprise Java Beans (EJB)* — gerencia a execução dos componentes EJB;

c. *Contêiner Web* — gerencia a execução de componentes Web Servlets e JSPs;

d. *Application* — gerencia a execução de aplicações Java, ficando na camada cliente;

e. *Applet* — gerencia a execução de applets Java, ficando na camada cliente associado ao programa de navegação (*browser*).

A figura 2 apresenta um esquema dos componentes nos contêineres J2EE.

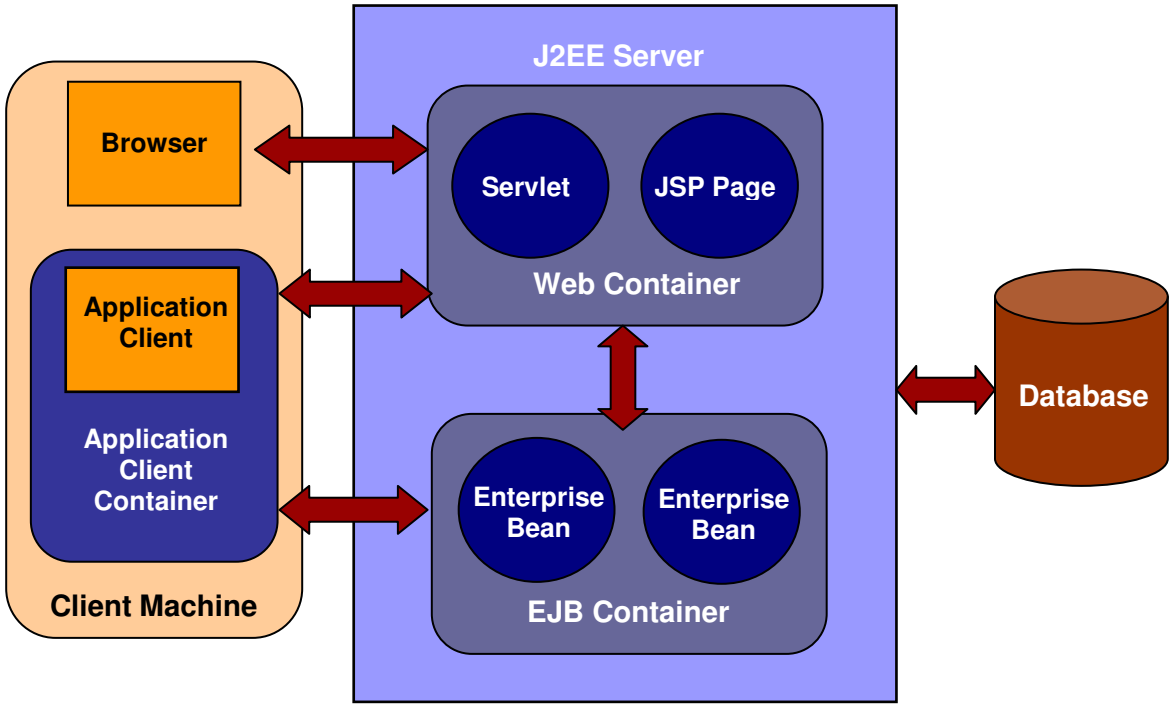

Figura 2 Contêineres e componentes J2EE

Fonte: SANTOS JÚNIOR, 2002, p. 38.

Dessa forma, é possível desenvolver aplicações J2EE a partir da combinação de componentes, reduzindo a complexidade e aumentando a produtividade no desenvolvimento de *software*. Os contêineres representam um ambiente operacional, fornecendo os recursos e infra-estrutura necessária à publicação e execução de componentes. Assim, os contêineres encapsulam a complexidade do ambiente coorporativo e fornecem independência de plataforma aos componentes, pois cada tipo de contêiner possui um conjunto de regas e serviços a serem fornecidos aos componentes.

Os serviços oferecidos por um contêiner podem ser resumidos da seguinte forma:

a. conectividade: os contêineres devem suportar conectividade com outros componentes e com os clientes do aplicativo;
b. serviço de diretório: os serviços J2EE são obrigados a fornecer serviços de atribuição de nomes, nos quais os componentes podem ser registrados e descobertos;
c. acesso a dados e persistência: o acesso a dados é fornecido através da API JDBC que, em se tratando de aplicativo, faz a interface com banco de dados e também com provedores de serviços que constroem *drivers* para banco de dados específicos.
d. conectividade legada: a Java Connector Architecture (arquitetura de conectores do Java, JCA ou Connectors) fornece suporte para J2EE na integração de servidores de informações corporativas e sistemas legados, como o processamento de transação de computador de grande porte e sistemas ERP (Enterprise Resource Planning,

planejamento de recursos corporativos). Esse suporte se estende aos provedores de serviços J2EE que escrevem adaptadores para conectar outros sistemas com a arquitetura corporativa do J2EE;

e. segurança: tanto APIs como o JAAS (Java Authentication and Authorization Service, serviço de autenticação e autorização do Java), ajudam o aplicativo corporativo J2EE na imposição das verificações de autenticação de segurança para os usuários;
f. suporte para XML: modelos de API suportam a análise de documentos XML;
g. transações: um servidor J2EE fornece serviços de transação para seus componentes, com limites especificados pelo contêiner ou pelo aplicativo;
h. troca de mensagens e e-mail: o JMS (Java Message Service, serviço de mensagens do Java) permite aos componentes enviar e receber mensagens assíncronas, normalmente dentro de um limite organizacional.

Todo servidor compatível com J2EE deve suportar os serviços definidos, expostos abaixo. A Figura 3 mostra a arquitetura J2EE com os serviços disponíveis para seus contêineres:

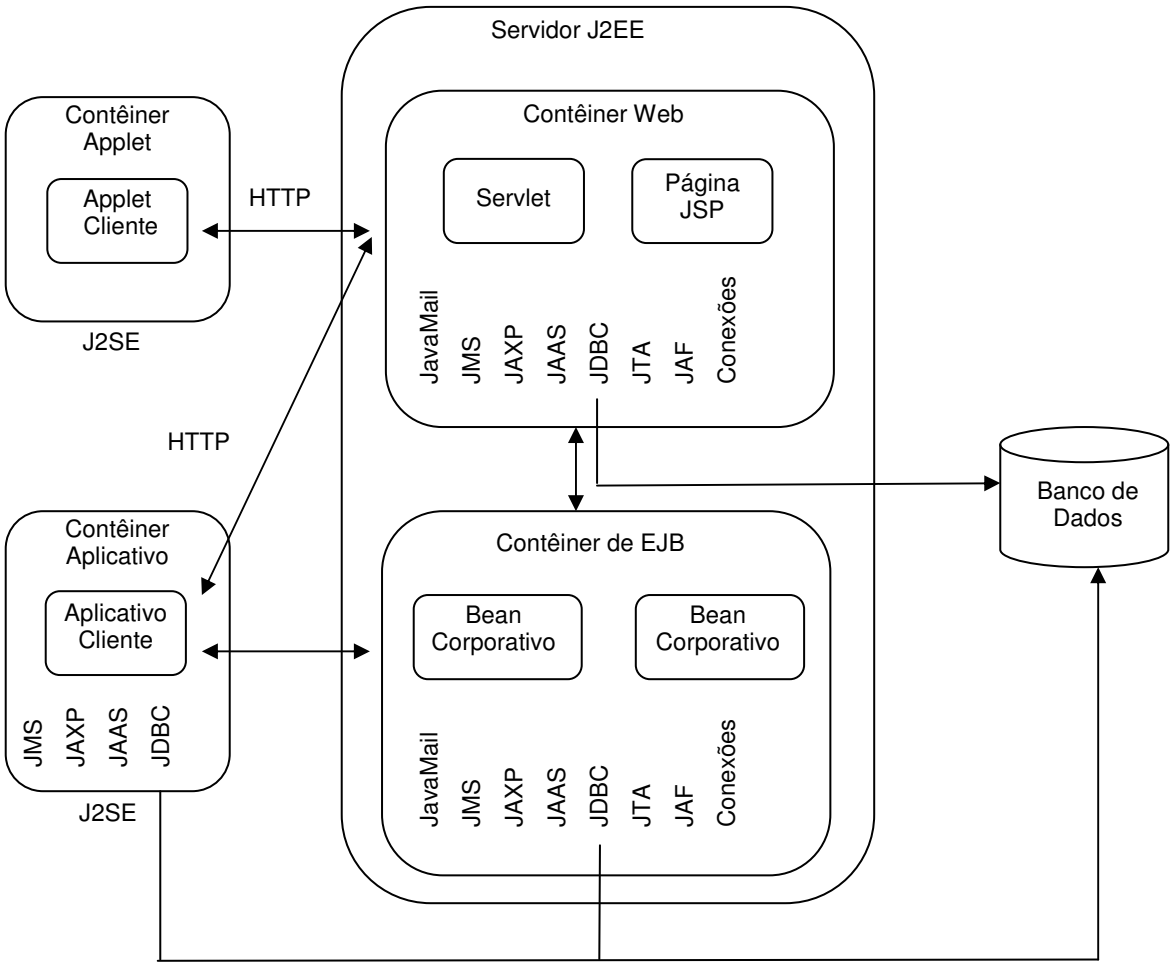

Figura 3 A Plataforma J2EE com os serviços disponíveis

Fonte: BOND et al, 2003, p.16.

Os contêineres representam a interface entre os componentes e serviços de baixo nível fornecidos pela plataforma e necessários à execução do componente, sendo assim, os componentes devem ser implementados seguindo as especificações do contêiner. Para que um componente possa ser utilizado, o mesmo deve ser publicado em um contêiner, necessitando de uma operação chamada *deploy* (configuração e preparação para execução). No *deploy*, o componente é configurado, através de arquivos XML e/ou da geração de classes de serviços. Esses arquivos XML irão conter informações relativas à execução do componente tais como: segurança, controle transacional, persistência de dados, escalabilidade,

associação com outros componentes, serviço de nomes, etc.

Após a operação de *deploy*, o componente pode então ser publicado no contêiner e, a partir deste momento, este fica responsável pela execução e gerência do componente. Toda e qualquer comunicação com os componentes passa pelo contêiner, fornecendo assim uma arquitetura flexível, na qual os componentes podem ser configurados de acordo com as necessidades, permitindo que um mesmo componente possa ter configurações diferentes, bastando para isso realizar uma nova operação de *deploy*, ajustando os parâmetros de configuração.

O resultado da operação de *deploy* é um arquivo JAR (Java Archieve) contendo o(s) componente(s), os arquivos XML com suas configurações, e as classes de serviços geradas automaticamente.

3.1.2 Java Servlets

As servlets são utilizadas para a criação de conteúdo dinâmico na Web, uma vez que podem agir de forma diferente, dependendo das ações do usuário, devendo ser inserido em um contêiner. Uma servlet é a versão para servidor de applets clientes - componentes para serem usados em um servidor Web.

Servlets são classes/objetos que funcionam como programas CGI. Com o suporte de um Web contêiner, uma servlet tem acesso a dados de formulários e tem o compromisso de produzir um código HTML que será devolvido ao cliente. Para cumprir sua função, a servlet pode fazer uso de qualquer API ou código Java necessário, incluindo JDBC, JavaMail, Java Advanced Image (JAI), etc.

Servlets são destinadas a substituir o antigo CGI, uma das primeiras extensões para servidores Web simples, que tornaram possível a criação de

conteúdo HTML dinâmico. Um programa CGI primeiro obtém uma requisição de uma página HTML (geralmente através de um formulário), processa essa requisição, e finalmente envia uma página HTML completa de volta ao *browser*. Um programa CGI pode ser escrito em diversas linguagens - Perl e C são algumas das mais populares.

As servlets Java são usadas com um servidor da Web para manipular requisições http. Elas gerarão respostas HTML para serem exibidas por meio de um navegador. São componentes no lado do servidor, e podem ser usados para estender a funcionalidade de qualquer servidor compatível com Java, mas são, normalmente, utilizadas para escrever aplicativos Web em servidor Web, geralmente quando o conteúdo não é estático, ou seja, é ideal para páginas Web que dispõe de variáveis de acessibilidade e mudam de acordo com a entrada do usuário, as chamadas páginas dinâmicas, como aponta Bond et al. (2003, p. 455).

Bond et al. (2003, p. 456) apontam algumas vantagens e recursos das servlets, como:

a. A API de servlets fornece uma interface personalizada para aplicativos Web, e as servlets genéricas têm uso limitado;
b. As servlets são independentes do servidor e da plataforma. Isso as torna altamente portáveis entre diferentes sistemas operacionais e implementações de servidor, além de serem reutilizáveis;
c. As servlets são eficientes e flexíveis, ou seja, depois de carregadas, normalmente residirão na memória do servidor. Com isso, apenas uma servlet possibilita acessos concomitantes, criando, para isso, múltiplas linhas de execução, evitando sobrecarga e economizando

memória. Como permanecem na memória do servidor, as servlets podem manter referência a outros objetos Java;

d. As servlets são executadas dentro do servidor; portanto, elas podem delegar certas funções para serem executadas pelo servidor em seu nome, como autenticação do usuário. Para uma servlet ser portável para outras plataformas e ambientes, ela acaba ficando limitada, não sendo possível tirar proveito de todos os recursos do servidor, pois, como aponta Bond et al. (2003, p. 457), pelo fato de interferir na segurança, o servidor as executará, com freqüência, no *sandbox*, ou caixa de areia, um ambiente controlado, projetado para proteger o servidor de uma servlet mal-intencionada ou mal-escrita.

3.1.3 JavaServer Pages (JSP)

A tecnologia JavaServer Pages se baseia na tecnologia Java Servlet com a finalidade de simplificar o desenvolvimento de conteúdos dinâmicos na Web. Embora as servlets sejam capazes de gerar dinamicamente páginas da Web, acaba também gerando muita informação de formatação HTML dentro do Java. Para Bond et al. (2005, p. 505), as JavasServer Pages “são servlets escritas em HTML Na verdade, há um pouco mais do que isso, mas o código Java em JSP é normalmente inexistente ou muito simples”. Não precisa ser programador para entender a JSP.

A JSP cria um modelo HTML estático que possibilita a inserção de apontamentos que serão substituídos por conteúdo dinâmico no momento da criação. Enquanto as servlets geram o código HTML, de código Java, as JSPs incorporam o código Java em código HTM.

As JSPs precisam ser convertidas em servlet Java antes da sua execução. Isto ocorre da seguinte forma: o texto JSP é traduzido em Java e, depois, o código Java é compilado em servlet.

As fases por quais a JSP passa são executadas pelo servidor Web, ou seja, ele traduz e compila automaticamente uma JSP.

Para Bond et al. (2005, p. 546), comparadas as servlets, as JSPs são:

a. mais rápidas para escrever e desenvolver;
b. focalizam o *layout* da página e delegam lógica para suportar JavaBeans e tags personalizadas;
c. diferenciam a apresentação da página Web (HTML) da lógica subjacente (Java);
d. suportam um mecanismo de relato de erro-padrão, por meio da diretiva de página de erro.

3.1.4 Enterprise JavaBeans (EJB)

Trata-se da arquitetura de componentes do lado servidor para a plataforma Java 2 Platform Enterprise Edition (J2EE) que encapsulam a lógica de negócios de uma aplicação, ou seja, o código que satisfaz o objetivo da aplicação.

Com a tecnologia Enterprise JavaBeans (EJB) é possível um desenvolvimento rápido e simples de aplicações distribuídas, transacionais, seguras e portáveis baseadas na tecnologia Java.

O modelo de componentes EJB, segundo Gomes (2005, p. 11-12), é fornecido pela Sun. Tem sua especificação aberta voltada para o desenvolvimento e

instalação de aplicações Java do domínio de negócios. O contêiner EJB, um ambiente de tempo de execução alocado no interior do servidor J2EE, é o local onde os componentes enterprise beans implementam a tecnologia e são executados. De forma transparente para o desenvolvedor de aplicações, os contêineres EJB oferecem serviços como suporte a transações e persistência.

A implementação do EJB se divide em duas partes: contêiner EJB, que hospeda a gerencia um enterprise beans do mesmo modo que o contêiner Web hospeda o servlet; e o servidor EJB.

Conforme Gomes (2005, p. 13-14), são três tipos de enterprise bean:

i. Sessão (Session): Executa uma tarefa para um cliente;
ii. Entidade (Entity): Representa um objeto de entidade de negócios que existe em armazenamento persistente;
iii. Orientado a mensagem (Message-Driven): Atua como um listener para a API Java Message Service (JMS) processando mensagens assincronamente.

3.1.4.1 Beans de sessão

É como se representasse a um único cliente, o J2EE. A relação ocorre da seguinte forma: “o cliente invoca as operações do bean de sessão para acessar uma aplicação implementada no servidor. Ou seja, o bean esconde de seu cliente a complexidade da lógica de negócios” (GOMES, 2005, p. 14).

O bean de sessão pode ser de dois tipos, sendo uma que contém informações de estado, ou *stateful* e outra sem ela, ou *stateless*. No bean com

informações de estado, as variáveis de instância representam o estado de uma única sessão cliente-bean. Posto que o bean “conversa” com o cliente, denomina-se seu estado de conversacional, que é mantido no período de duração da sessão cliente-bean.

Por outro lado, quando o bean invocado não possui informação de estado, as variáveis de instância do bean podem conter um estado, mas apenas enquanto durar a chamada. Exceto, segundo Gomes (2005, p. 14), durante a chamada do método, na qual “todas as instâncias de um bean sem informação de estado são equivalentes, permitindo que o contêiner atribua uma instância a qualquer cliente”. Assim, tais beans podem oferecer escalabilidade para aplicações que requerem um grande número de clientes.

3.1.4.2 Beans de entidade

Um EJB de entidade, ou simplesmente bean de entidade, é a representação de um dado corporativo e, geralmente, representa os principais dados manipulados durante processos corporativos. Quando um bean de entidade é possuidor de dados dinâmicos, ele tem a funcionalidade associada e pode ser compartilhado entre vários clientes a qualquer momento.

Por sua interface síncrona, os beans de entidade acessam fontes de dados subjacentes, como bancos de dados ou sistemas de ERP, reunindo todas as informações corporativas que representam, fornecendo, dinamicamente, métodos para atualizá-las e recuperá-las de várias maneiras. Bond et al. (2003, p. 36) enfatizam que os beans de entidade são usados com freqüência junto com beans de sessão a fim de fornecer a funcionalidade do negócio de um sistema.

O EJB de entidade mapeia uma combinação de dados (ou conceito de aplicativo equivalente) e a funcionalidade associada. Os beans de entidade são, normalmente, baseados em um repositório de dados subjacente, pois são criados com base em dados neles existentes.

3.1.4.3 Beans baseado em mensagens

Um bean baseado em mensagens, embora semelhante a um bean de sessão, somente é ativado quando chega uma mensagem assíncrona. Criando uma interface assíncrona, o bean baseado em mensagens permite que clientes interajam com ele.

É associado a uma determinada fila de mensagens que serão distribuídas para uma estância dele. Bem como os beans de sessão, os dirigidos por mensagens destinam-se a abrigar a lógica do negócio, mas acessando os dados exigidos através de JDBC ou de beans de entidade.

Geralmente os beans dirigidos por mensagens são criados para consumirem mensagens, mas, como outros EJBs, podem ser usados para criar e enviar mensagens. Este bean não possui um contexto de segurança próprio e fica alojado inteiramente em um contêiner. Quando implantado, é associado a uma fila ou tópico em particular, sendo acionado pelo contêiner quando da chegada de uma mensagem pela qual é responsável.

Um bean dirigido por mensagens tem, como características, segundo Bond et al. (2003, p. 390), ser anônimo, invisível ao cliente, sendo que nenhum estado é mantido para o cliente; todas as instâncias de um bean dirigido por mensagens, em particular, são equivalentes; o contêiner pode colocar as instâncias

em *pool*; não possui interface local ou remota; é chamado de forma assíncrona pelo contêiner; seu ciclo de vida e seu ambiente são gerenciados pelo contêiner, que também fornece a ele funcionalidade para segurança, concorrência, transações, entre outros. Como outros tipos de beans, o dirigido por mensagens tem uma classe bean e um descritor de implantação XML. Além disso, componentes JSP ou aplicativos não-J2EE podem utilizar beans dirigidos por mensagens. Como o beans dirigido por mensagens não podem ser acionados diretamente pelo cliente, não correm o risco de serem usados inadequadamente.

3.1.5 API – Application Programming Interface

Uma Application Programming Interface, ou simplesmente API, é um conjunto de rotinas e padrões estabelecidos por um *software* para utilização de suas funcionalidades. De modo geral, a API é composta por uma série de recursos, acessíveis somente por programação, que permitem utilizar características do *software* menos evidentes ao usuário tradicional.

O J2EE é um conjunto integrado de APIs Java que possibilita a criação de aplicações cliente servidor na Web.

Com a revolução da Internet, a Sun continuou utilizando e melhorando as APIs, adicionando recursos de interface Web e as remodelando para melhorar o desenvolvimento orientado a objeto. Atualmente, as APIs evoluíram de versões anteriores, tornaram-se estáveis e compatíveis, sem mudanças radicais que impossibilitam a boa programação Java.

Java tem muitas APIs, com diferentes versões. As APIs J2EE são referenciadas como J2EE Services. Todo servidor J2EE comercial deve suportar

este conjunto de APIs.

Em J2EE, encontram-se tanto APIs exigidas, como também as opcionais. As APIs exigidas são básicas e são importantes pois, além de terem um propósito geral, são usadas nas aplicações J2EE. A aplicação que suportam estas APIs estão em conformidade com o J2EE.

A figura abaixo apresenta uma lista das APIs exigidas e as opcionais para as aplicações, sendo que algumas delas já foram descritas em seções anteriores deste trabalho.

Exigidas	**Opcionais**
HTTP	EJB
IDL	Servlet
JDBC	JMS
RMI-IIOP	JTA
JNDI	JavaMail
JSP	JAF
-	XML
-	JCA
-	JAAS

Figura 4 Lista de APIs exigidas e opcionais em J2EE

Fonte: Criada pela autora deste trabalho com base em Nilsson (2003)

3.2 Desfecho

Um dos motivos da escolha do J2EE para esta análise é a independência de *hardware*, pois uma aplicação é escrita somente uma vez e pode ser executada em diferentes plataformas. Uma técnica muito importante na construção de aplicações é a separação entre negócio, visão e objetos de controle. Esta separação permite que equipes de desenvolvimento trabalhem em diferentes

pedaços ao mesmo tempo. Segundo Zhang (2003), o J2EE simplifica o desenvolvimento de sistemas distribuídos, fornecendo serviços prontos para serem usados, tais como transações e segurança. A tecnologia de componentes do J2EE é considerada uma das mais avançadas em tecnologia de *middleware* COTS[11].

Uma das vantagens principais do J2EE é garantida pelas várias APIs disponibilizadas, que garantem a compatibilidade entre diversas versões de aplicativos. Como arquitetura aberta, o J2EE ainda conta com a facilidade de ter uma extensa gama de fornecedores e produtos, também devido às suas APIs.

O J2EE, utilizado no desenvolvimento de aplicativos corporativos, tem ajudado as empresas a fornecer rapidamente aplicativos escaláveis e seguros capacitados para a WEB, além de permitir a produção e desenvolvimento de produtos e serviços que podem ser conectados.

Uma observação importante, que deve ser considerada quando do desenvolvimento de sistemas utilizando o J2EE, é o efeito desejado. Para que sejam alcançadas transparências, é necessário o uso de componentes EJB, pois com os servlets isso não é possível. Contudo, o uso de servlets garante um melhor desempenho do sistema, que não é conseguido com o uso de EJB.

[11] Commercial Off-The-Shelf – "Software que existe, está disponível para o público e pode ser comprado ou licenciado" (OBERNDORF, apud SOUSA et al, p. 2). Trata-se de *softwares* modulares, baseados em componentes, dos quais as funcionalidades necessárias de um sistema são adquiridas de terceiros, separadamente ou não.

4 VISÕES RM-ODP E O PADRÃO J2EE

O objetivo deste capítulo é apresentar o mapeamento entre o RM-ODP e os elementos do J2EE, identificando os elementos que correspondem a cada visão do ODP a fim de demonstrar a compatibilidade entre os dois padrões.

O ambiente corporativo vem sofrendo mudanças fundamentais com o surgimento e uso intenso de novas tecnologias aliado ao panorama político, econômico e social. Nas últimas décadas, as mudanças se tornaram mais acentuadas onde a competitividade entre elas alcança níveis nunca antes experimentados.

Para se manterem neste novo ambiente, as empresas necessitam reformular constantemente sua estrutura de negócios para poderem competir com as demais. As alterações necessárias para este ambiente visam a utilização de sistemas (de processamento) distribuídos e abertos que permitem que diferentes aplicações distribuídas, desenvolvidas em ambientes diferentes, possam interagir considerando a integração de sistemas legados com os novos sistemas implementados na Web.

O rápido crescimento do processamento distribuído levou à necessidade de uma estrutura de coordenação para a padronização de processamento aberto distribuído. O RM-ODP fornece tal estrutura, criando uma arquitetura dentro da qual suporta a distribuição, interação e a portabilidade. Ele foi escolhido por ser um padrão aberto para a construção de aplicações distribuídas.

Como o modelo RM-ODP se baseia em objetos, não definindo nenhuma metodologia ou mesmo uma linguagem de modelagem, e atualmente a UML é o padrão de modelagem visual orientado a objeto, traçou-se um paralelo

entre as visões do RM-ODP com os diagramas da UML.

O J2EE define uma especificação e não um produto, os recursos e funcionalidades de cada versão do J2EE estão em conformidade por meio do Java Community Process JCP[12]. O JCP, sendo uma organização aberta, define as especificações Java assegurando a compatibilidade entre plataformas e garantindo a contínua estabilidade que permite desde o funcionamento de centenas de milhões de dispositivos, dos computadores *desktop* a robôs industriais.

A plataforma J2EE, por ser um padrão de fato, é implementada por fornecedores e produtores de aplicativos servidores como a BEA, IBM, Oracle, JBOSS. Com essa flexibilidade pode-se escolher um produto com o padrão J2EE de qualquer fornecedor, com base na qualidade, no suporte ou mesmo na facilidade de uso. A Sun fornece um ambiente de teste para garantir que esses fornecedores vendam produtos dentro das especificações do J2EE. Os produtos aprovados neste teste recebem um certificado de compatibilidade com J2EE (J2EE compliant)[13].

O J2EE foi desenvolvido para trabalhar apenas com orientação a objeto e não obriga nenhum modelo específico, recomendando o uso da UML para a definição do sistema conforme citado por Bond et al. (2003,113) e justificado por Ahmed (2002,26).

4.1 RM-ODP e os Modelos UML

O RM-ODP estabelece cinco visões que são estruturadas como subdivisões da especificação de um sistema completo. Essas visões não são completamente independentes, itens chaves em cada visão são identificados como

[12] Cf: JAVA COMMUNITY PORCESS.
[13] Cf: SUN DEVELOPER NETWORK (SDN).

relacionados a itens de outras visões.

Na análise do RM-ODP foi feito um mapeamento com base nos paradigmas da engenharia de *software*, utilizando o ciclo de vida espiral que define as atividades de planejamento, análise de riscos, engenharia e avaliação do cliente. Foram utilizados os modelos da abordagem do processo RUP e da Linguagem de Modelagem Unificada (UML) que se trata de um padrão no RUP.

O mapeamento foi concebido como uma proposta auxiliar entre as visões do RM-ODP e os componentes do J2EE, não havendo a pretensão de esgotar este tema. Comparando-se definições das visões do RM-ODP com as definições dos modelos UML e do RUP, nota-se que existem vários pontos em comum, o que induz a considerar a possibilidade da adequação da UML e do RUP as visões do RM-ODP.

A **visão da empresa** define objetivos, políticas e restrições do sistema. Nessa visão foram definidos os requisitos que são mapeados através do modelo de caso de uso da UML. Esse modelo permite compreender as exigências funcionais e as não funcionais do sistema. Modelo de atividades também permite definir os processos de negócio. Pode-se analisar a visão empresa utilizando o RUP, pois ele possui o modelo de caso de uso de negócio e o modelo de objeto do negócio.

A **visão da informação** expressa o comportamento do sistema face às transformações das informações. Ele descreve a informação requisitada pelo sistema, através de representações, denominadas esquema, que especificam a estrutura do objeto. Nessa visão foram mapeados os modelos de classe, objetos, estados e os de interação como seqüência e comunicação da UML por representarem os relacionamentos e comportamento dos objetos e, segundo a OMT, esses modelos compreendem as visões estáticas e dinâmicas do sistema.

A **visão computacional** está relacionada com os processos de alteração de dados nos modelos de informações, capturando os aspectos dinâmicos do sistema. Nesta visão foram mapeados os seguintes modelos de Atividade, Interação e de Estados da UML por fornecerem uma representação de como os objetos podem interagir na realização de cada uma das funcionalidades e por representarem as seqüências de interação de cada um destes objetos, capturando o comportamento dinâmico de cada objeto.

A **visão da engenharia** está relacionada ao modo de interação dos objetos, com especificação das interfaces internas e com a infra-estrutura, foram mapeados os modelos de Atividade, Interação e de Estados.

O RM-ODP não indica nenhuma metodologia na visão da tecnologia, definindo apenas como a tecnologia suporta os sistemas distribuídos. Essa visão provê uma ligação entre as visões e a implementação real. Os modelos da UML para essa visão são os modelos de Pacotes, Componentes e Distribuição por representarem os componentes de *software* e *hardware* de implementação.

4.2 Análise do J2EE sob a ótica do RM-ODP

Essa seção apresenta a comparação entre os elementos do J2EE com as visões do RM-ODP. Foram utilizados os modelos mapeados na seção anterior com o objetivo de auxiliar no mapeamento dos elementos do J2EE.

4.2.1 J2EE e a Visão Empresa

Essa visão expressa os objetivos, políticas e restrições do sistema. Consistem nos seguintes modelos:

a. De requisitos

b. Casos de uso

c. Atividades

d. Modelo de caso de uso de negócio e objetos do negócio

No RM-ODP, a visão da empresa refere-se diretamente ao propósito, escopo e restrições que regem o negócio, extraindo requisitos e estruturas da organização a partir das ações desempenhadas que mudam a política e as restrições do negócio, tal como criar uma obrigação ou revogar uma permissão.

O componente EJB (Enterprise Java Bean), do contêiner J2EE, reside no servidor e compreende a lógica do negócio, assim como as entidades mapeadas na UML e os casos de usos.

A lógica do negócio será encapsulada convenientemente pelo componente EJB, pois este possui dados corporativos ativos e soluciona problemas como escalabilidade, gerenciamento de ciclo de vida e gerenciamento de estados. EJBs fazem uso de fontes de dados, como banco de dados e sistemas de computador de grande porte, para executar a lógica do negócio exigida.

Uma das vantagens em se usar EJB é que ele permite que a funcionalidade do negócio seja desenvolvida e depois implantada, independente da camada lógica de apresentação, o que significa que interfaces com os usuários possam ser desenvolvidas ou alteradas a qualquer hora, sem a necessidade de desenvolver novamente o aplicativo inteiro.

O componente EJB é um componente de engenharia que se adapta a essa visão por conter a lógica de negócio e por ser definido numa classe Java. O *package* se adapta a essa visão por representar classes que implementam serviços

e a lógica de negócio.

A compatibilidade entre o J2EE e o RM-ODP é que essa visão representa processos e lógica de negócio que são implementados por EJB e *packages*.

4.2.2 J2EE e a Visão Informação

Refere-se à semântica e processo da informação, sendo usada para descrever as interações entre os objetos que representam o mundo real, incluindo seus relacionamentos e comportamento. Complexidades do mundo são representadas por composições de objetos. Composta pelos seguintes modelos:

a. Modelo de Classes
b. Modelo de Objetos
c. Modelo de Estados
d. Modelo de Interação: seqüência e colaboração

Essa visão, no RM-ODP, é usada para descrever a informação requisitada pelo sistema através de representações denominadas “esquemas”, que especificam o estado e a estrutura dos objetos. Têm-se aqui três tipos de “esquemas”: estático (captura o estado e estrutura de um objeto em alguma instância), invariante (restringe o estado e estrutura de um objeto no tempo) e dinâmico (define uma mudança permitida no estado e estrutura de um objeto, restringindo-se ao esquema invariante).

O padrão J2EE permite que funcionalidades possam ser divididas em

classes, que são agrupadas em pacotes ou componentes, que podem ser utilizadas em diferentes partes do aplicativo.

Bond et al. (2003, p. 8) apontam que a modelagem orientada a objetos promove bastante a modularidade, pois os objetos encapsulam seus dados ou estados e, através de suas interfaces, oferecem funcionalidades. Os autores afirmam ainda que, quando projetadas da maneira correta, as dependências entre objetos diferentes podem ser minimizadas, significando que os objetos estão fracamente acoplados, sendo mais fáceis de manter e evoluir.

O J2EE é totalmente compatível com essa visão, pois ele utiliza a orientação a objeto, não oferecendo nenhuma restrição ou conflito de nenhuma natureza à essa especificação do RM-ODP, o que torna conveniente seu uso em conjunto.

4.2.3 J2EE e a Visão Computacional

Esta visão está diretamente ligada à distribuição de processos através da decomposição funcional do sistema em objetos que interagem entre si, mas não com os mecanismos de interação que possibilitam que a distribuição ocorra.

Ela está relacionada com *o que* e *por que* os objetos interagem. Consistem nos seguintes modelos:

a. Modelo de Atividade
b. Modelo Interação
c. Modelo de Estados

A visão computacional é representada por:

a. Interface, que compreende três formas de interação entre os objetos: operacional, que é um modelo cliente-servidor para computação distribuída; orientada a cadeias contínuas de informação, seja áudio ou vídeo, entre as partes; e orientada a sinal, com ações de comunicação em níveis muito baixos.
b. Atividade, na qual as ações que um objeto pode sofrer são: criação e destruição do objeto; criação e destruição de uma interface; ligação com a interface; leitura e escrita do estado do objeto; invocação de uma operação na interface operacional; produção e consumo de um fluxo na interface em cadeia; iniciação ou resposta a um sinal na interface orientada a sinal. Tais ações podem ser seqüencializadas ou paralelizadas.
c. Contrato de ambiente, que pode ser expresso em termos do nível de transparência na distribuição e da qualidade de serviços oferecidos pelas interfaces (segurança no transporte de informações, por exemplo).

RM-ODP utiliza como base um modelo de objetos comum para as distintas especificações, bem como técnicas de orientação a objetos para modelar o sistema geral, a partir de cada uma das visões.

Esses métodos se mostram muito apropriados pelas vantagens do encapsulamento e de abstração que proporcionam. A abstração permite ressaltar os detalhes interessantes do sistema sobre aqueles que não se consideram relevantes a um determinado nível, e descrevê-los de forma independente de qualquer

implementação concreta. O encapsulamento permite definir serviços e funções, ocultando detalhes que não são necessários sobre sua implementação, sua possível heterogeneidade, ou os mecanismos de provisionamento dos serviços concretos utilizados pelos clientes e servidores.

O J2EE é compatível a essa visão através dos serviços implementados pelos contêineres dos servidores J2EE para o contrato de ambientes do RM-ODP; através da máquina virtual Java (JVM) para suportar a portabilidade do processamento distribuído especificado na norma RM-ODP e, por último, utilizando componentes EJB para a interação entre objetos e controle do ciclo de vida dos mesmos.

Ao projetar um SD, um grande número de interesses torna-se aparente, o que é resultado direto da distribuição: os componentes do sistema são heterogêneos; quando ocorrem falhas, estas, por serem independentes, não afetam outros aspectos do uso do aplicativo e nem outros usuários; eles estão em posições diferentes e, possivelmente, variando, e assim por diante. Tais interesses podem ou ser resolvidos diretamente como a parte do projeto da aplicação, ou as soluções padrão podem ser selecionadas, baseado na melhor prática.

Se o mecanismo padrão for escolhido, tratar-se-á de uma aplicação baseada nesse interesse particular; o mecanismo padrão fornece uma transparência da distribuição.

A abordagem da transparência da distribuição pode levar diretamente ao reuso do *software*. A seleção de transparências da distribuição na especificação de sistema pode conduzir à incorporação automática de execuções bem estabelecidas das soluções padrão pelos produtos de construção do sistema no uso, tal como compiladores, nos *links* e nos gerenciadores da configuração. Podem-se

expressar exigências do sistema no formulário de uma indicação simplificada do sistema requerido e das propriedades da transparência da distribuição que deve possuir.

As oito transparências propostas pela RM-ODP são compatíveis no J2EE através do contêiner EJB e os serviços oferecidos por ele como, por exemplo, serviços de resolução de nomes para implementar a transparência de acesso e localização; através do uso da JVM para implementar a transparência de migração; utilizando os mecanismos de transação e persistência para implementar a respectivas transparências; e utilizando o EJB para as demais transparências

4.2.4 J2EE e a Visão de Engenharia

Mecanismos para interação distribuída, com foco nos mecanismos e funções exigidas para suportar a interação distribuída entre objetos do sistema. Essa visão está relacionada em *como* os objetos interagem. Os conceitos e regras são suficientes para permitir a especificação das interfaces internas com a infra-estrutura, permitindo a definição de pontos distintos de conformidade para diferentes transparências, e a possibilidade de padronização de uma infra-estrutura genérica em que módulos de transparências padrões podem ser classificados. A visão da engenharia trata com os objetos básicos e com vários outros objetos da engenharia que o suportam.

Essa visão de RM-ODP se refere à infra-estrutura necessária para suportar a distribuição. É usada para descrever o projeto (*design*) dos aspectos orientados a distribuição do sistema ODP, mas não se concentra na semântica da aplicação ODP, exceto para determinar seus requisitos de distribuição e transparência.

O modelo correspondente a esta visão inclui objetos e canais. Os objetos podem ser divididos em duas categorias: objetos básicos (correspondem aos objetos na especificação da visão computacional) e objetos de infra-estrutura (por exemplo, protocolos). Basicamente essa visão trata do serviço orientado a comunicação, onde um canal corresponde a uma ligação ou objeto de ligação na especificação da visão computacional, ou seja, são mecanismos de comunicação e contêm ou controlam as funções de transparência. Essa visão foi parcialmente modelada, pois não há uma correspondência direta entre os conceitos apresentados no RM-ODP e os componentes do J2EE.

Os elementos do J2EE mapeados nessa visão são os serviços oferecidos pelos contêineres, já explicados no Capítulo 3, como por exemplo os serviços de controle de transação, conectividade com sistemas legados, protocolo de transferências de hipertexto, conexão com banco de dado.

4.2.5 J2EE e a Visão de Tecnologia

Essa visão refere-se aos padrões a serem seguidos e adotados quando se escolhe uma tecnologia para a implementação de um sistema, levando-se em consideração aspectos de identificação, distribuição e instalação de tecnologias de *hardware* e *software* e provendo informações para teste. É importante ressaltar que o RM-ODP não recomenda ou obriga nenhuma metodologia em particular.

Este modelo está intimamente ligado ao padrão J2EE por ser um padrão aberto, voltado para aplicações distribuídas, com a utilização do EJB, construindo aplicações em *n-camadas*. Os modelos da UML para essa visão são os modelos de Componentes e Distribuição. A modelagem dos componentes é usada

para estabelecer como várias partes do *software* são reunidas fisicamente e a modelagem de distribuição é usada para mapear o *layout* da aplicação distribuída. O J2EE é compatível a esta visão, pois aqui o RM-ODP não especifica itens. Modelos:

a. Componentes
b. Distribuição

O modelo correspondente a esta visão serve mais para justificar do que descrever subsídios para seleção de tecnologias de mercado.

A compatibilidade entre o J2EE e o RM-ODP se dá pela implementação e especificações dos aplicativos que utilizam os servidores e os serviços dos contêineres do J2EE.

Os Descritores de Distribuição são baseados em XML, que especifica a gramática para os descritores (AHMED, 2002, p.249). Os Descritores de Distribuição são compostos por:

a. Descritor de distribuição Web: usado para os servlets e os JSP. Os arquivos são nomeados como web.jar
b. Descritor de distribuição EJB: esse arquivo contém os detalhes dos EJB e são nomeados como ejb-jar.xml
c. Descritor de distribuição Aplicação: contém detalhes da aplicação e são armazenados nos arquivos application.xml
d. Descritor de distribuição específico do revendedor.

4.3 Desfecho

A arquitetura mais comum de desenvolvimento de aplicações comerciais utiliza três camadas físicas. Para o J2EE, a arquitetura se compõe de camada de apresentação ou Web, camada intermediária ou do negócio e camada de integração ou de dados, que são usados para implementar sistemas distribuídos comerciais, componentizáveis e escaláveis.

Segundo Ahmed (2002,15), pode-se utilizar os componentes Servlets para a construção de conteúdo dinâmico. Esse componente fornece mecanismos de interação entre as camadas da apresentação e de negócio, porém, essa alternativa não é recomendada quando se utiliza transação com banco de dados ou mesmo se a pretensão é a construção de uma aplicação que tenha as características de uma aplicação distribuída, como transparência.

Para a construção de uma aplicação comercial que seja caracterizada como distribuída, devem-se utilizar os componentes EJB, pois eles encapsulam a lógica do negócio e dados coorporativos ativos, conforme afirmado por BOND et al. (2003), proporcionando escalabilidade, gerenciamento do ciclo de vida e gerenciamento de estados.

A performance e a qualidade sistêmica aumentam quando são utilizados padrões de projetos[14] (BOND et al, 2003, p. 753). As qualidades sistêmicas ou os requisitos não funcionais estão diretamente relacionados com as características de uma aplicação distribuída que são escalabilidade, disponibilidade, entre outras. Aplicando padrões, as aplicações apresentam melhoras na qualidade sistêmica (BOND et al, 2003, p. 724). Pode-se utilizar o componente EJB de

[14] Padrões de projetos oferecem exemplos de projetos, descrevendo maneiras comuns de fazer as coisas e são coletados os temas repetitivos em projetos UML destilado (FOWLER, 2005, p. 46).

entidade com o padrão Session Façade[15] impedindo o cliente de acessar diretamente um EJB. Esse padrão aumenta a flexibilidade, segurança com diminuição do desempenho, pois há um aumento das chamadas a métodos.

O XML é um padrão formado por um conjunto de regras para projetar formatos de texto que descrevam a estrutura de dados; ele representa os dados portáveis, sendo legível para o cliente, simples, flexível e principalmente de uso gratuito, facilitando a construção de aplicações comerciais distribuídas. Na plataforma J2EE, a utilização do XML facilita o compartilhamento de dados legados internamente entre departamentos e o compartilhamento de todos os dados com outras empresas, facilitando principalmente o processamento de dados nos aplicativos escritos em Java.

O uso de padrões em aplicações de sistemas distribuídos traz benefícios relevantes, pois qualquer alteração fica facilitada, além de aumentar a capacidade de interação, entre outras vantagens de melhora de desempenho.

O quadro a seguir contém um resumo do mapeamento dos elementos do J2EE com as respectivas visões do RM-ODP e modelos.

[15] Session Façade é um padrão da camada de negócio do J2EE, sua função é isolar os EJBs de entidade do acesso direto do cliente e para ocultar o esquema de dados do cliente.

Visões			Modelos	J2EE
Empresa	Comunidades (uma configuração dos objetos e dados formada para encontrar objetivo)		Modelo de caso de uso Modelo de Atividades Modelo de caso de uso de negócio Modelo de objeto do negócio	EJB;Packages
	Regras de Objetos (permissão, proibição e obrigação)			EJB;Packages
Informação	Esquemas	Invariante	Modelo de Classes Modelo de Objetos Modelo de Estados Modelo de Interação	O J2EE é compatível por ser naturalmente aderente á visão de informação, por oferecer conceito de objeto
		Estático		
		Dinâmico		
	Relações			
	Objetos Compostos			
Computacional	Interação entre objetos		Modelo de Atividade Modelo Interação Modelo de Estados	EJB
	Atividade – ciclo de vida			
	Contrato de ambientes – transparências na distribuição			Serviços oferecidos pelo contêiner EJB
	Portabilidade			JVM

Figura 5 Resumo das visões com J2EE

<table>
<tr><th colspan="3">Visões</th><th>Modelos</th><th>J2EE</th></tr>
<tr><td rowspan="9">Engenharia</td><td colspan="2">Orientação a objeto</td><td rowspan="9">Modelos de Atividades
Modelo de Interação
Modelo de Estados</td><td rowspan="9">Utilização da orientação a objeto

Serviços oferecidos pelos contêineres EJB</td></tr>
<tr><td rowspan="8">Canais</td><td>Cluster</td></tr>
<tr><td>Capsule</td></tr>
<tr><td>Node</td></tr>
<tr><td>Channel</td></tr>
<tr><td>Interface</td></tr>
<tr><td>Binder</td></tr>
<tr><td>Nucleus</td></tr>
<tr><td>Objeto de protocolo</td></tr>
<tr><td>Stub</td></tr>
<tr><td>Tecnologia</td><td colspan="2">Sem definição pelo RM-ODP, implementação da tecnologia</td><td>Modelo Pacotes
Modelo de Componentes
Modelo Distribuição</td><td>• Classes Java
• Descritor de distribuição e implementação do aplicativo
• JAR - Java Archive File
• EAR – Enterprise Archive File
• WAR – Web Archive
• Especificação de Servidores
· WebServer
· ApplicationServer
· DatabaseServer</td></tr>
</table>

(cont) Figura 5 Resumo das visões com J2EE

5 ANÁLISE DOS PRODUTOS CONFRONTADOS COM RM-ODP

O objetivo deste capítulo é apresentar a comparação entre os produtos compatíveis com o ambiente distribuído J2EE confrontados com os propósitos do Modelo Referencial. Os produtos analisados foram o Tomcat, Websphere e Sun One, sendo usadas as tabelas do capítulo anterior confrontando os produtos com o padrão RM-ODP.

Os critérios de seleção dos produtos se basearam na utilização destas no âmbito corporativo, levando-se em consideração custo, robustez e uso corporativo. O critério custo baseou-se na utilização de *software* livre e de código aberto, visando diminuir custos de um projeto. Nessa linha os produtos escolhidos foram o Tomcat e o Sun One.

O segundo critério recaiu na escolha de uma ferramenta que apresentasse robustez em aplicações de missão critica, pois os usuários corporativos exigem a utilização de plataformas e ambientes de execução que forneçam suporte a altos requisitos de segurança, disponibilidade e escalabilidade. Esses requisitos são essenciais para corporações que necessitam oferecer serviços de acesso rápido a grandes comunidades de clientes, fornecedores, colaboradores, utilizando a Internet como canal de comunicação.

5.1 Tomcat

O Tomcat é um servidor de aplicações Java para Web. É um *software* livre, de código aberto, desenvolvido dentro do conceituado projeto Apache Jakarta, endossado oficialmente pela Sun, como a Implementação de Referência (RI) para as

tecnologias Java Servlets e JavaServer Pages (JSP). O Tomcat é robusto e eficiente o suficiente para ser utilizado mesmo em um ambiente de produção. Para fins desse trabalho foi utilizada a versão 5.5 do Tomcat para as comparações entre os produtos.

Tecnicamente, o Tomcat é um contêiner Web; parte da plataforma J2EE que abrange as tecnologias Servlets e JSP, incluindo tecnologias de apoio relacionadas como segurança, JNDI e JDBC e acessam a família inteira das APIs Java. O Tomcat tem a capacidade de atuar também como servidor Web/HTTP, ou pode funcionar integrado a um servidor Web dedicado, como o Apache httpd ou o Microsoft IIS.

O Tomcat não implementa um contêiner EJB. Para aplicações J2EE que utilizam EJB, deve-se utilizar um servidor de aplicações J2EE completo, como JBoss AS, IBM WebSphere (tema do próximo tópico), BEA WebLogic, Oracle AS, ou Sun Java System Application Server Platform Edition, anteriormente conhecido como Sun One. Esses produtos contêm uma implementação de referência para a plataforma J2EE completa (Web e EJB), entre outros.

5.2 Websphere

Websphere Application Server ou Servidor de Aplicações Websphere foi projetado pela IBM para permitir aplicações de alto volume de transação e aplicações de missão crítica utilizando a arquitetura J2EE.

Websphere é uma ferramenta de desenvolvimento que suporta a arquitetura J2EE, contém aplicações, serviços que ajudam o desenvolvimento e execução das APIs do J2EE, desde applets até soluções Web baseada nos

componentes J2EE, baseada no SOA[16] e foi lançada no início de 2001 como uma estratégia de evolução do VisualAge para Java. Ela está em conformidade com o padrão J2EE.

A ferramenta Websphere suporta todos os contêineres, desde um contêiner Web até mesmo um contêiner EJB. Nesse trabalho foi usada a versão 6.

5.3 Sun One

O Sun One, rebatizado de Sun Java System Application Server Platform Edition 8, é um servidor de aplicação da Sun. Ele é compatível com J2EE 1.4, fornecendo uma plataforma completa para serviços Web incluindo API para o padrão XML. Construída para ter alto desempenho, oferecendo mecanismo para que JMS (Java Message Service) e HTTP tenham alto desempenho.

Outra característica muito importante do Sun One é o tamanho da ferramenta e o consumo de memória é pequeno. Possibilitando assim que servidores mais simples possam utilizar essa plataforma.

5.4 Análise dos produtos

A partir do mapeamento apresentado no capítulo anterior apresentamos, neste capítulo, uma comparação entre os produtos compatíveis com o ambiente distribuído J2EE e o padrão RM-ODP, cujo objetivo é demonstrar a compatibilidade entre eles.

Na visão **Empresa,** que corresponde a comunidades e regras de

[16] SOA (Arquitetura Orientada a Serviços) não é uma nova arquitetura, segundo CUOMO (2005) um serviço é um componente de *software* que encapsula uma função com uma interface bem definida. Esse serviço é disponibilizado em redes de computadores (Internet) que se comunica através de padrões abertos.

objetos, foi mapeada o Modelo de caso de uso, Modelo de atividades, Modelo de caso de uso de negócio e Modelo de objeto do negócio. Os elementos do J2EE correspondentes a essa visão são o EJB e os *packages*, que são classes que contém a lógica do negócio. O Websphere e o Sun One são compatíveis a essa visão. O Tomcat apresenta uma compatibilidade parcial a essa visão pois esse produto não implementa um contêiner EJB, implementando somente contêiner Servlet.

Na visão **Informação**, que corresponde aos esquemas invariante, estático e dinâmico; relações e objetos compostos, foi mapeado o modelo de classes, modelo de objetos, modelo de estados e o modelo de interação. Todos os elementos do J2EE são compatíveis a essa visão. O Websphere e o Sun One são totalmente compatíveis com essa visão, o Tomcat é parcialmente compatível com essa visão, pois ele não implementa o contêiner EJB.

Na visão **Computacional,** que corresponde à interface (Interação entre objetos), ciclo de vida dos objetos, contrato de ambientes – transparências na distribuição e portabilidade; foi mapeado o modelo de atividade, o modelo de interação e o modelo de estado. Os elementos do J2EE compatíveis com essa visão são o EJB para a interface e para o ciclo de vida dos objetos; a máquina virtual Java (JVM) para a portabilidade e os serviços de invocação de método remoto (RMI), o protocolo de transferência de hipertexto (http), os componentes de acesso ao banco de dados (JDBC), o controle de transação do Java (JTA), o serviço de transação para interações com interfaces-padrão Corba (JTS) e o serviço de conectividade com sistemas legados (JCA). Os três produtos são compatíveis a essa visão considerando a portabilidade com a máquina virtual Java (JVM). Com relação ao restante da visão o produto, o Tomcat não é compatível com essa visão pelo

aspecto do contêiner EJB.

Na visão **Engenharia**, que corresponde à orientação a objeto e canais, que é um serviço orientado a comunicação com suas especificações, foi mapeado o modelo de atividade, o modelo de interação e o modelo de estados. O J2EE é compatível com a visão engenharia, pois os dois padrões trabalham com o conceito de orientação a objeto. Os elementos do J2EE compatíveis com a visão engenharia são os serviços implementados pelos contêineres já vistos no capítulo sobre o J2EE e seus contêineres. Os produtos Websphere, Tomcat e Sun One são compatíveis com essa visão.

O RM-ODP não faz recomendações ou mesmo impõe uma metodologia para a visão **Tecnologia**. Os modelos que correspondem a essa visão são os modelo de pacotes, modelo de componentes e o modelo de distribuição, pois são modelos que correspondem as várias partes do *software*, onde pode-se modelar o *layout* da aplicação distribuída. No J2EE, os elementos correspondentes a essa visão são as classes Java, que podem conter desde a lógica do negócio, elementos e serviços, os descritores de distribuição que definem os requisitos da aplicação. Os descritores de distribuição são JAR (Java Archive File), o EAR (Enterprise Archive File), o WAR (Web Archieve) e as especificações de servidores como o WebServer, ApplicationServer e o DatabaseServer. Os produtos são compatíveis com essa visão excetuando o Tomcat quando a comparação envolver o contêiner EJB.

5.5 Desfecho

Neste capítulo foram apresentadas as comparações entre os produtos que e as visões do RM-ODP, demonstrando a compatibilidade dos produtos às visões.

6 CONSIDERAÇÕES FINAIS

Este capítulo apresenta considerações baseadas nos resultados do trabalho de pesquisa efetuado mostrando as conclusões e procurando resumir as observações feitas durante as pesquisas.

6.1 Conclusões

O RM-ODP é um padrão que provê práticas de projeto de propósito geral para definição formal de uma arquitetura de sistema de processamento distribuído. Essas práticas são adequadas a vários tipos de negócios.

As conclusões desse trabalho estão apresentadas a partir dos seguintes tópicos: compatibilidade do ambiente distribuído J2EE com o padrão RM-ODP, compatibilidade dos produtos que implementam o padrão J2EE com o RM-ODP e análise dos produtos para a construção de aplicações comerciais distribuídas.

O ambiente distribuído J2EE é um ambiente quase totalmente compatível ao padrão RM-ODP. Os produtos que utilizam o padrão J2EE possuem a mesma compatibilidade do padrão J2EE em relação ao RM-ODP.

A princípio, o Tomcat parece ser a ferramenta menos aplicável para Web, pois não implementa um contêiner EJB, impossibilitando a criação de aplicações distribuídas e que implemente as transparências. Contudo, essa ferramenta, por sua versatilidade e custo, é uma boa opção de escolha como servidor Servlets e JSP.

O Websphere e o Sun One são equivalentes quando se precisa

construir uma aplicação Web que seja distribuída e possua as características do RM-ODP. Deve-se levar em consideração a disponibilidade de recursos adicionais, performance, disponibilidade desses produtos para a plataforma desejada, consumo de *hardware* e, finalmente, o custo na escolha desses dois produtos.

A contribuição desse trabalho está na análise dos ambientes distribuídos J2EE com o padrão RM-ODP e a comparação dos produtos com essa análise servindo como critérios para escolha de uma ferramenta.

REFERÊNCIAS BIBLIOGRÁFICAS

AGOSTINI, Marcelo N., DECKER, Ildemar C., SILVA, Aguinaldo S. e. Desenvolvimento e implementação de uma base computacional orientada a objetos para aplicações em sistemas de energia elétrica. **Sba Controle & Automação**. Vol.13, no.2, p.181-189, maio/ago, 2002. ISSN 0103-1759.

AHMED, Khawar Zaman e UMRYSH, Cary R. **Desenvolvendo aplicações comerciais em Java com J2EE e uml.** São Paulo: Ciência Moderna, 2002.

AVELINO, V. F. **Merusa: metodologia de especificação de requisitos de usabilidade e segurança orientada para arquitetura.** 2005. 250 fl. Tese (Doutorado) Escola Politécnica da Universidade de São Paulo, São Paulo.

BOND, Martin; HAYWOOD, Dan; LAW, Debbie; LONGSHAW, Andy; ROXBURGH, Peter. **Aprenda J2EE em 21 dias**: com EJB, JSP, Servlets, JNDI, JDBC e XML. São Paulo: Pearson Education do Brasil e Makron Books, 2003.

COELHO, André Luís Vasconcelos. **Caracterização de um serviço de gerência distribuído para objetos multimídia persistentes**. 1998. 159 fl. Tese (Mestrado) Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Distributed System:** concepts and design. 3ª ed. Inglaterra: Pearson Education, 2001.

CUOMO, Gennaro. **IBM SOA "on the Edge".** ACM, SIGMOD June 2005.

DEITEL, Harvey M.; DEITEL, Paul J.; STEINBUHLER, Kate. **Operation Systems.** 3ª ed. Pearson Education Inc., 2005.

ESMIN, Ahmed Ali Abdalla. Modelando com UML: Unified Modeling Language. **INFOCOMP Journal of Computer Science**, vol. 01, N. 01, Anais da II SECICOM, ISSN: 1807-4545, Lavras, MG, Brazil, November, 1999.

FOWLER, Martin. **UML essencial:** um breve guia para a linguagem-padrão de modelagem de objetos. Porto Alegre: Bookman, 2005.

FRANKEL, D. **Applying EDOC and MDA to the RM-ODP Engineering and**

Technology Viewpoints: An Architectural Perspective. EDOC/MDA and RM-ODP Engineering and Technology v01-00. Out., 2003. Disponível em <http://www.net.intap.or.jp/e/odp/odp-techguide.pdf>. Acesso em: 07 jan. de 2006.

FURTADO JÚNIOR, Miguel Benedito. **XML - Extensible Markup Language**. Tutorial criado sob orientação do professor Otto Carlos Muniz Bandeira Duarte, da Universidade Federal do Rio de Janeiro. Disponível em: <http://www.gta.ufrj.br/grad/00_1/miguel/index.html>. Acessado em 12 de jan. de 2006.

GALASINI, Marcelo. **O método ODP-UP para a definição de arquiteturas de sistemas distribuídos.** 2004. 126 fl. Tese (Mestrado) - Escola Politécnica, Universidade de São Paulo.

GEYER, C. F. R.; BORGES, C.; VARGAS, P. K.; FERRARI, D. N.; ÁVILA, C. O.; YAMIN, A. C.; LIMA, C. C.; DALMOLIN, L. C.; LEYSER, C.; BALINSKI, R. Projeto sistemas avançados para comunicação eletrônica - software aberto de correio, agenda e catálogo. In: II WORKSHOP SOBRE SOFTWARE LIVRE. 2001, Porto Alegre. **Anais**... Porto Alegre: UFRGS, 29-31 de maio de 2001. p. 25-28.

GOMES, Wander Euclides Carneiro Pimentel. **Uma plataforma de desenvolvimento de software baseado em componentes para dispositivos móveis Faculdade de Engenharia Elétrica e de Computação**. 2005. 93 fl. Tese (Mestrado). Faculdade de Engenharia Elétrica e de Computação Campinas. São Paulo.

History of CORBA®. **OMG:** Object Manegement Group. Disponível em: <http://www.omg.org/gettingstarted/history_of_corba.htm>. Acesso em dez. 2005.

ISO - INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Recommendation X.902/ISO/IEC 10746-2**: information Technology - Open Distributed Processing – Referencence Model: Foundantions, 1996

Jini Architecture Specification: Version 1.2. **Sun Developer Network (SDN)**. Dec. de 2001. Disponível em: <http://www.sun.com/software/jini/specs/jini1.2html/jini-title.html>. Acesso em dez. 2005.

KLEINROCK, Leonard. **Distributed Systems**, Communications of the ACM, November, 1985.

KRIGER, David. ADLER,Richard M. **The Emergence of Distributed Component Plataforms**. IEEE, March 1998.

KRUCHTEN, P. **Introdução ao RUP:** Rational Unified Process. Rio de Janeiro: Ciência Moderna, 2003.

MORENO, Antonio Vallecillo. RM-ODP: el modelo de referencia de ISO para el procesamiento abierto y distribuido. **DINTEL Edition on Software Engineering**. Nº 3, March, p. 69-99. Málaga, 2001. ISBN: 84-931933-2-1.

NILSSON, Dale. R., MAUGET, Louis. E. **Building J2EE Applications with IBM Websphere.** Indianapolis, Indiana: Wiley Publishing Inc, 2003.

RADHAKRISHNAN, Ramesh et.al. **Java Runtime Systems: Characterization and Architetural Implications.** IEEE Transactions on Computer, Feb 2001.

RAYMOND, Kerry. **Reference Model of Open Distributed Processing - RM-ODP:** Introduction. ICODP'95, 1995. Disponível em: <http://magda.elibel.tm.fr/refs/ UML/odp.pdf> Acesso em: 12 de fev. de 2006.

RENAUD, Paul E. **Introdução aos Sistemas Cliente/Servidor:** Guia prático para profissionais de sistema. Rio de Janeiro: Infobook, 1994.

SANTOS JÚNIOR, José Maria Rodrigues. **Ferramenta para aumento da produtividade no desenvolvimento de aplicações web sobre a plataforma J2EE.** 2002. 138 f. Tese (Mestrado). Universidade Federal da Paraíba. Campina Grande.

SILVA, Cláudio Rodrigues Muniz da. **Um modelo independente de plataforma para a execução de processos de negócios em arquiteturas baseadas em modelos**. Tese (Doutorado). Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 2003.

SINTEF, Arne-Jorgen Berre. **Norwegian use of UML and RMODP for C4I Interoperability specifications.** Washington DC, OMG C4I Coalition days, 1999. Disponível em: <OMG C4I Coalition days>. Acesso em: 12 de set de 2005.

SOUSA, Fabrízia M. de; ALENCAR, Fernanda M. R. de; CASTRO, Jaelson F. B. O Impacto dos COTS no Processo de Engenharia de Requisitos. WER 99, II WORKSHOP ON REQUIREMENTS ENGINEERING. **Anais eletrônicos...** Buenos

Aires, Argentina, 9-10 de set. de 1999. Disponível em: <http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER99/souza.pdf>. Acessado em: 19 de jan de 2006.

TANENBAUM, Andrew. S. Stenn, M. V. **Distributed System:** principles and paradigms. p 42-54. New Jersey: Printice-Hall Inc., 2002.

The GLOBE Project. **Afdeling Informatica: faculteid der exacte wetenchappen**. Disponível em: <http://www.cs.vu.nl/globe/>. Acesso em dez. 2005.

The mach Project Home Page. **Carnegie Mellon:** School of computer science. Disponível em: <http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html>. Acesso em dez. 2005.

VALLECILLO, Antonio. **RM-ODP:** The ISO Reference Model for Open Distributed Processing. Málaga: ETSI Informática - Universidad de Málaga. <http://www. enterprise-architecture.info/Images/Documents/RM-ODP.pdf> Acesso em: 10 de jan. de 2006.

VAROTO, Ane Cristina. **Visões em arquitetura de software.** 2002. 108 fl. Tese (Mestrado). Universidade de São Paulo. São Paulo.

Welcome to the home of OpenAFS. **OpenAFS Info**. Disponível em: <http://www.openafs.org/frameless/main.html>. Acesso em dez. 2005.

ZHANG, Yan; LIU, Anna; QU, Wei. Comparing industry benchmarks for J2EE application server: IBM's trade2 vs Sun's ECperf. In: CONFERENCE IN RESEARCH AND PRACTICE IN INFORMATION TECHNOLOGY (Proceedings of the twenty-sixth Australasian computer science conference. **Anais eletrônicos…** Adelaide, Australia, 2003. p. 199-206. Disponível em: <http://beta.reviews.com/browse/browse_topics3. cfm?ccs_id=2308>. Acesso em: 09 de dez de 2005.

SITES:

AMOEBA WWW HOME PAGE. Disponível em: <www.cs.vu.nl/pub/amoeba/>. Acesso em dez. 2005.

JAVA COMMUNITY PORCESS. Disponível em: <http://www.jcp.org/en/home/index>.

Acessado em: 12 de fev. de 2006.

SETI@HOME. 1999. Disponível em: <http://www.setiathome.ssl.berkeley.edu/>. Acesso em: 20 de set. de 2005.

SUN DEVELOPER NETWORK (SDN). Disponível em: <http://www.jcp.org/en/home/index>. Acesso em: 12 de fev. de 2006.

THE APACHE SOFTWARE FOUNDATION. Disponível em: <http://tomcat.apache. org/>. Acesso em: 18 de fev. de 2006.