CENTRO UNIVERSITÁRIO DO TRIÂNGULO
INSTITUTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Tecnologias do padrão J2EE

Geane Maria Pereira

Uberlândia, Dezembro/2003.

CENTRO UNIVERSITÁRIO DO TRIÂNGULO
INSTITUTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Tecnologias do padrão J2EE

Geane Maria Pereira

Monografia apresentada ao Curso de Ciência da Computação do Centro Universitário do Triângulo - Unit, como requisito básico à obtenção do grau de Bacharel em Ciência da Computação, sob a orientação do Prof. Edson Angoti Júnior , Msc.

Uberlândia, Dezembro/2003.

Tecnologias do padrão J2EE

Geane Maria Pereira

Monografia apresentada ao Curso de Ciência da Computação do Centro Universitário do Triângulo - Unit, como requisito básico à obtenção do grau de Bacharel em Ciência da Computação.

Edson Angoti Júnior, Msc.
(Orientador)

Marcos Alberto Lopes da Silva, Msc.
(Avaliador)

Alfen Ferreira de Souza Júnior, Msc.
(Avaliador)

Silvia Fernanda Martins Brandão, Dsc.
(Coordenadora de Curso)

Uberlândia, Dezembro/2003.

Agradecimentos

Agradeço à minha família e esposo, pela compreensão e paciência nos momentos difíceis durante o curso e agradeço também ao meu orientador que compreendeu minhas dificuldades durante o desenvolvimento deste trabalho.

Resumo

Neste trabalho faz-se um estudo da arquitetura de sistemas *J2EE* (*Java 2 Enterprise Edition*) e da tecnologia *Enterprise Java Beans*. A Plataforma *J2EE* foi desenvolvida pela *Sun MicroSystems* para desenvolvimento de aplicações empresariais distribuídas. Por ser baseada na tecnologia Java, tais aplicações rodam nas plataformas Windows, *Linux*, *Unix* e *Mainframes*. Esta plataforma fornece um ambiente integrado para criação de aplicativos Java para múltiplas camadas a nível empresarial.

Foi desenvolvido um estudo de caso da utilização do *J2EE* com a ferramenta de desenvolvimento *Magic.* *J2EE* é utilizado nesta ferramenta através de componentes gerados pelo *Magic*. O *Magic* cria componentes EJB (*Enterprise Java Bean*) usando uma biblioteca de programas existentes, podendo definir objetos de uma aplicação como componente e usá-lo em outra aplicação. Um componente *Magic* é empacotado em um arquivo MCI e um componente EJB é em um arquivo .jar. *Component Builder* do *Magic* gera e empacota os módulos da aplicação, e os salva em um arquivo .jar. O *EJB Interface Builder* habilita uma aplicação Magic a ser distribuída em um *J2EE Enterprise Server*. Por fim é mostrado um estudo de caso para demonstrar a eficácia da ferramenta *Magic*.

Sumário

Lista de Figuras

1 Introdução

Este trabalho tem como finalidade estudar as arquiteturas de sistemas, neste caso a arquitetura *J2EE* e apresentar um estudo de caso que utiliza a integração do *J2EE* como uma ferramenta de desenvolvimento chamada *Magic*, visando um desenvolvimento rápido de uma aplicação, nele encontrará um todo os procedimentos para criação de componentes EJB.

1.1 Divisão dos capítulos do trabalho

Este trabalho está divido em cinco capítulos, Introdução, *J2EE* (*Java Interprise Edition*), *EJB* (*Enterprise Java Beans*), Estudo de caso, Conclusão.

1.1.1 *J2EE (Java 2 Enterprise Edition)*

A plataforma *J2EE* resumi-se em um conjunto de práticas, classificações e tecnologias da plataforma *java*, são administradas pela *Sun MicroSystems*, juntas tendem a oferecer suporte ao *design*, desenvolvimento e *deploy* de aplicações componentizadas baseadas em objetos distribuídos e multicamadas. *J2EE* uma plataforma estável,

voltada para desenvolvimento de soluções corporativas em uma implementação distribuída, através de camada de dados, camada cliente, camada intermediária na qual é responsável por acessar o sistema de informação corporativa. Possui os principais componentes, *Enterprise JavaBeans* (*EJB*), *servlets, JavaServer Pages* (*JSP*), Aplicações Cliente, *Applets*. Na arquitetura *J2EE* existem diversos tipos de *contêiners* como contêiner *EJB*, contêiner *Web*, contêiner de aplicação cliente, contêiner *applet*, para que um componente rode nestes ambientes tem que atender as especificações descritas nos mesmos.

1.1.2 *EJB* (*enterprise JavaBeans*)

A tecnologia *Enterprise JavaBeans*(*EJB*) de componentes, dispõe um ambiente para gerenciamento de componentes. Reduzindo o processo de desenvolver aplicações escaláveis, portáveis e reusáveis no seu ambiente de negócios. Oferece ao desenvolvimento de aplicações baseadas em uma arquitetura de objetos distribuídos em várias camadas (*multitier*), o qual maioria da parte lógica da aplicação é levada do cliente até o servidor. A lógica da aplicação é dividida em uma ou mais objetos que são disponibilizados em um servidor de aplicação.

Um componente *EJB* pode ser de três tipos: *Entity Beans, Session Beans* e *Message Driven Beans. Entity Beans* representa conceitos de negócios que podem ser representados por nomes. *Session Beans* são uma extensão da aplicação cliente e são responsáveis por representar processos ou ações. *Message Drive Beans* são um tipo de *Bean* que parecem com *Session Bean*, são acessado a partir de um sistema de mensagens pelo Java *Message Service* (JMS).

1.1.3 Estudo de Caso

Objetivo deste trabalho está no estudo de caso, trabalhar com uma ferramenta de desenvolvimento que reduz a complexidade e o tempo do desenvolvimento de aplicações corporativas. Com *Magic Developer* permite ao desenvolvedor criar componentes *EJB* a partir de programas existentes ou a partir de novos programas.

1.1.4 Conclusão

Para concluir o trabalho neste capítulo foi feito uma análise da abordagem de todo trabalho, demonstrando a praticidade de trabalhar com uma ferramenta rápida de desenvolvimento.

2 *J2EE* (*Java 2 Enterprise Edition*)

Com as novas versões do *java* surgiu o *J2EE* desenvolvida pela *Sun MicroSystems* anunciada em abril de 1997 que é o nicho de tecnologia que vem despertando um grande interesse. Apesar do pouco tempo de existência *J2EE* vem gerando muito controvérsias. Em seu lançamento sua execução era no browser, através dos *applets*. Dependendo do browser o código *applet* rodava de forma totalmente diferente. Como em todo lançamento havia muita inconsistência em sua versão original, aproveitando deste momento os opositores e os incrédulos questionaram o *J2EE*. Neste momento muitas inconsistências foram sanadas tornando o *J2EE* mais maduro e confiável a tal ponto de gerar novos nichos de mercado a cada dia como: cartões java, telefones celulares e aplicativos desktop estão sendo criados pelo mundo.

No início o *java* no servidor era restrito aos *servlets engines.* Devido o sucesso de tecnologias como *PHP* e *ASP* a tecnologia *JSP* foi agrupada pelo *java* no qual foi anexado com um diferencial, ela serve como desacoplamento entre código (*servlet*) e a apresentação (*JSP*).

A plataforma *J2EE* foi desenvolvido pela S*un MicroSystems* para desenvolvimento de aplicações, nele inclui várias *API´s* (*Application Programming Interfaces,* interfaces de programação de aplicações) para construção de aplicações *java*, incluindo *EJB*, *Servlets*, *JNDI*, *JMS*, transações e outros e para desenvolvimento de aplicativos empresariais utilizando tecnologias *java*. Sua arquitetura é Cliente/Servidor de n-camadas, ele roda em qualquer hardware, como *mainframes*, *linux*, *Unix* e windows.

2.1 O que é Plataforma *J2EE*

Uma plataforma confiável, estável, possui alta performance, desenvolvida para soluções corporativas. É um conjunto de classificações coordenadas que permite execução, instalação e gerenciamento de aplicações n-camadas no servidor. Esta plataforma tem como principal objetivo reduzir o custo, tempo e a complexidade no progresso de aplicações corporativas.

A plataforma *J2EE* não é um produto. Nas especificações de *J2EE* há vários produtos que o implementa veja alguns destes: *Apache*, *Applets*, *ATG*, *Borland, Bull, IBM*.

Definição da *Sun*, segundo *Sun* em 2002 a *plataforma Java 2 Enterprise Edition* (*J2EE*) define um padrão para o desenvolvimento de aplicações multicamadas. Nesta arquitetura, a camada que contém as regras de negócio, normalmente implementada utilizando *Enterprise JavaBeans*, pode ficar concentrada no servidor de aplicações, sendo compartilhada com diversas aplicações clientes. As aplicações clientes não contêm a lógica do negócio, atendo-se somente à camada de apresentação. Na camada de apresentação normalmente são utilizadas as tecnologias de *Servlets* e *JavaServer Pages*.

2.2 Plataforma *J2EE* (*Java 2 Enterprise Edition)*

Com uma arquitetura de aplicação multicamada, a plataforma *J2EE* foi criada para suportar os rigorosos pedidos do ambiente corporativo atual e padronização ao desenvolvimento em múltiplas camadas de aplicações corporativas. Ambiente corporativo é aquele que exige algumas soluções tais como, segurança, maturidade, alta disponibilidade. O desenvolvimento desta aplicação corporativa é simplificada através da plataforma *J2EE* devido uma padronização feita por ela. Para soluções de ambiente corporativo á uma implementação distribuída, através de camada de dados, camada cliente, camada intermediária na qual é

responsável por acessar o sistema de informação corporativa. Também é uma plataforma da tecnologia *java* para distribuição, desenvolvimento de informações para múltiplas camadas bem como à nível empresarial. A plataforma *J2EE* possui algumas vantagens, tecnologia *CORBA*, independência de plataforma, independência de servidor de Banco de Dados, e uma de suas principais vantagens é a arquitetura distribuída, sistema aberto (interação com tecnologia e aplicativos) e soluções em camadas, disponível no site http://www.aberium.com. Veja a arquitetura distribuíd

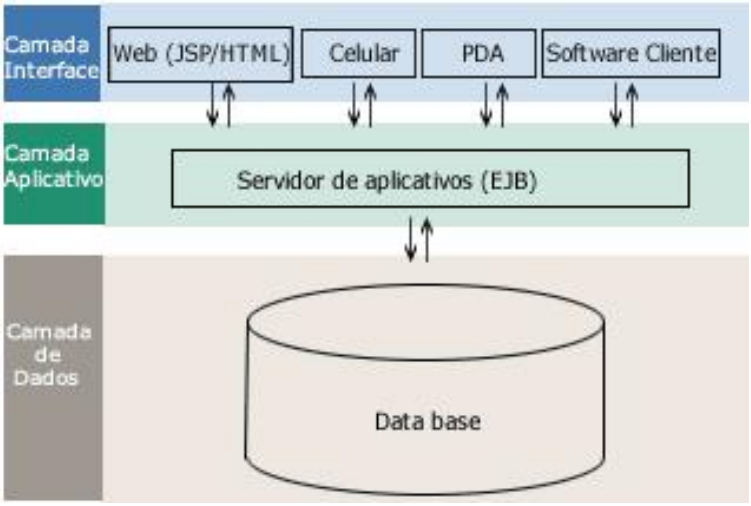


Figura 2.1 - Arquitetura distribuída [05]

Corba (*Common Object Request Broker Architeture*):

Uma arquitetura padrão definida pela *OMG*(*Object Management Group*). Este sistema autoriza que aplicações distribuídas trocam informações umas com as outras estando em rede local, mesmo se estiver rodando em plataforma de hardware diferentes e criadas em linguagem de programação distintas, o usuário nem percebe o que está ocorrendo. *CORBA* possui uma vantagem em ser utilizada por várias empresas em todo o mundo como por exemplo *IBM*, *HP*, *ORACLE*.

2.3 Aplicações Distribuídas e Multicamada

A plataforma *J2EE* utiliza um modelo de aplicação multicamada. Uma aplicação *J2EE* possui vários componentes, que estão distribuídos em máquinas diferentes, na qual o componente da aplicação *J2EE* multicamada dividi-se nas seguintes camadas.

- Componentes da camada cliente são executados na máquina cliente;
- Componentes da camada Web são executados no servidor *J2EE*;
- Componentes da camada de negócios são executados no servidor *J2EE*;
- O software da camada *EIS* (*enterprise information system*, sistema de informações empresariais) é executado no servidor *EIS*.

Uma aplicação *J2EE* pode basear-se em três ou quatro camadas, como mostra a Figura 2.2, geralmente as aplicações *J2EE* multicamadas são consideradas como aplicações de três camadas, pois estão instalados em três diferentes locais, que são: as máquinas de banco de dados, as máquinas clientes e as máquinas do servidor *J2EE*, disponível no site http://www.java.sum.com/j2ee/tutorial/download.html. Veja processo da aplicação na Figura 2.2

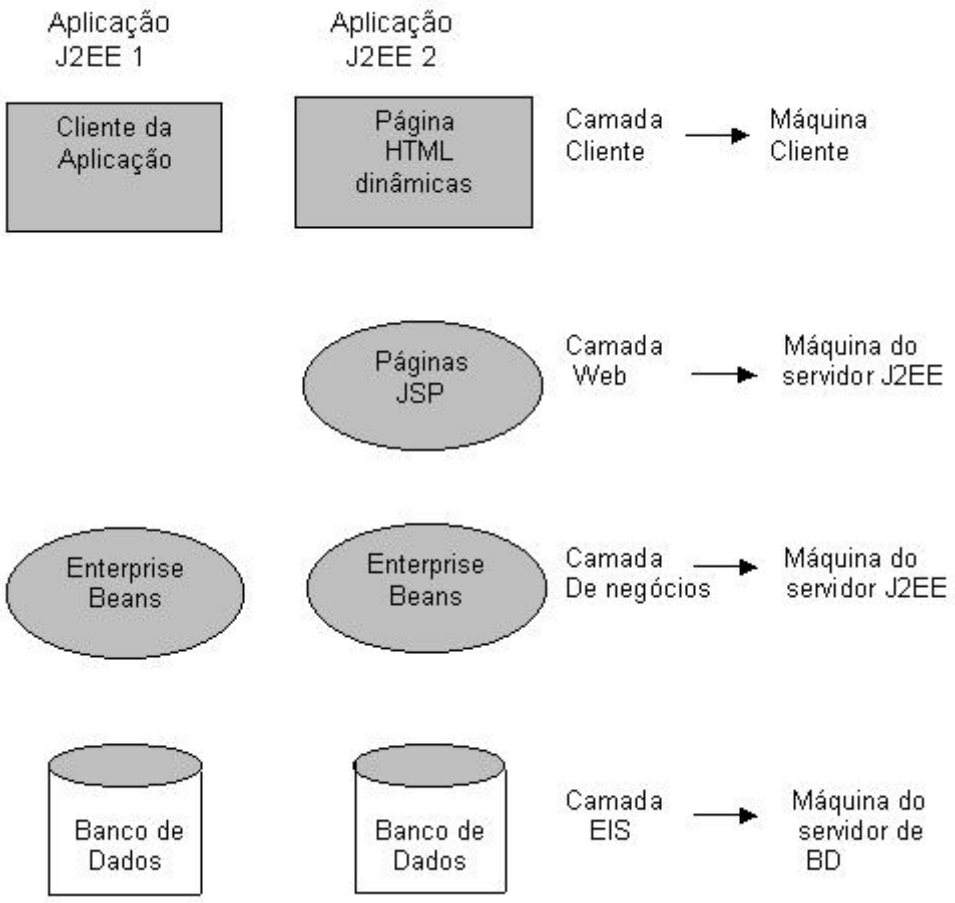


Figura 2.2 – Aplicação Multicamada [10]

2.3.1 Componentes *J2EE*

Aplicações *J2EE* são formadas por componentes. Um componente *J2EE* é uma unidade de software funcional. A especificação *J2EE* define os seguintes componentes:

- Clientes da aplicação e *applets* são componentes que são executados no cliente.
- Componentes da tecnologia Java *Servlet* e *Java Server Pages* são componentes Web que são executados no servidor.

- Componentes *Enterprise JavaBeans* são componentes de negócios que são executados no servidor.

Os componentes *J2EE* são escritos em *Java* e compilados como um outro programa na linguagem.

2.4 Arquitetura *J2EE*

O *J2EE* possui um modelo de programação em 3 camadas onde os clientes estão na camada 1, e solicitam serviços pela camada 2 onde está o *J2EE*, esta verifica, faz uma análise dos dados que estão na camada 3 e ele se encarrega em devolver aos clientes que estão na camada 1 os resultados. Maiores informações podem ser encontradas no site http://www.blaz.com.br/webdev/programacao/jsp. Funcionamento espe

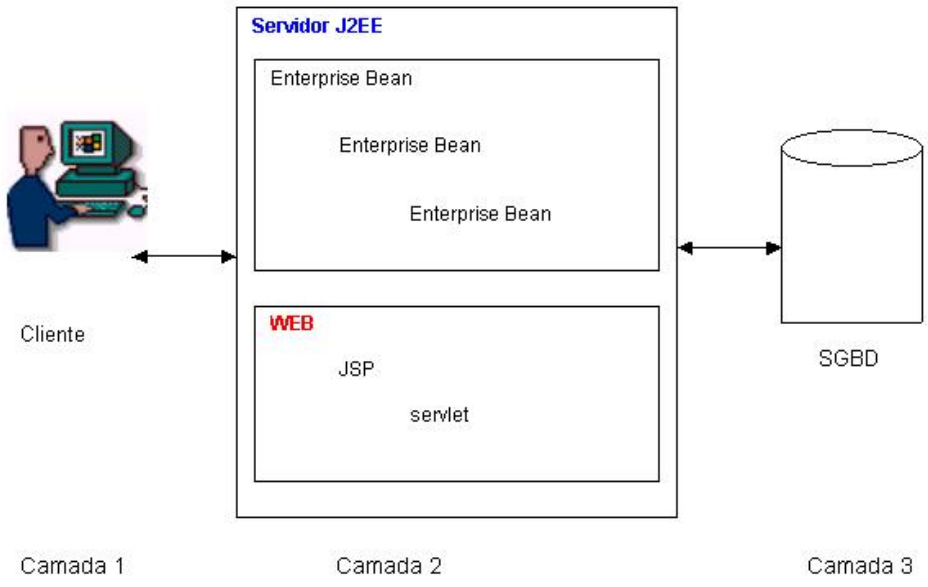


Figura 2.3 - Arquitetura do *J2EE* [03]

2.5 Tecnologias

A Plataforma *J2EE* é composta de:

- Especificação do *J2EE* – Que define os requisitos do *J2EE*;
- Conjunto de Testes de Compatibilidade – valida a compatibilidade dos sistemas desenvolvidos com a plataforma;
- Implementação de Referência (RI) – Valida a especificação do *J2EE*;
- Modelo de Programação de Aplicações (*Blueprints*) – Catálogo de padrões de projeto e orientações para a construção de aplicações *J2EE*.

2.6 Serviços da plataforma *J2EE*

Resumo das especificações *API´s* (interfaces de programação de aplicações) da plataforma *J2EE*.

2.6.1 *EJB* (*Enterprise JavaBeans*)

Define modelo de componentes servidor. O *Enterprise Bean* pode ser imaginado como uma porção de estrutura que pode ser usado sozinho ou com outros *Enterprise Beans* para executar a lógica de negócios no servidor *J2EE*.

2.6.2 RMI/IIOP (*Remote Method Invocation*)

Permite a comunicação remota entre objetos *Java*. Um programa *Java* pode chamar métodos de um programa *Java*

2.6.3 *Java IDL* (*Java Interface Definition Language*)

Integração entre objetos *Java* e outros objetos remotos, interfaces remotas para suportar *CORBA*.

2.6.4 *JNDI* (*Java Naming and Directory Interface*)

Proporciona funcionalidade de nomeação de diretório. Com *JNDI* uma aplicação *J2EE* pode recuperar qualquer objeto *Java* nomeado.

2.6.5 *Servlets*

Permite definir classes de *servlet* específicas para *HTTP*. Recebe um formulário de um página, verifica os dados e retorna uma página de resposta.

2.6.6 *JMS* (*Java Messaging Services*)

É um sistema de mensagens padrão este deixa que os componentes *J2EE* enviem, criem, recebam e leiam mensagens.

2.6.7 *JTA* (*Java Transaction API*)

Oferece uma interface padrão para demarcar as transações. Se existe uma aplicação que executa duas operações de acesso ao banco de dados o qual são separadas mas dependentes entre si, irá usar o API JTA para demarcar, inclusive as duas operações, começa, desfaz e confirma.

2.6.8 *JTS* (*Java Transaction Service*)

Define um serviço de gerenciamento de transações distribuídas, baseados no *OTS CORBA*.

2.6.9 *JDBC* (*Java DataBase Conectivity*)

A partir de métodos de linguagem de programação *Java* o *JDBC* autoriza a chamada de comando *SQL*. O *JDBC* pode ser usado a partir de uma página *JSP* ou *Servlet* para acessar o banco de dados sem passar pelo *Enterprise Bean*.

O *JDBC* contém duas partes:

- Interface em nível de aplicação para acessar um banco de dados.
- Interface de serviço para anexar em driver *JDBC* à plataforma *J2EE*.

Ou seja, *API JDBC* fornece um mecanismo independente de acesso a banco de dados.

2.6.10 *JAVAMAIL*

Fornece um *framework* independente de protocolo para construir *email* e aplicações de mensagens. A plataforma *J2EE* inclui o com um provedor de serviços que permite que os componentes de aplicação enviem *email* pela internet.

2.6.11 *JSP* (*Java Server Pages*)

Uma página *JSP* é um documento baseado em texto, deixa colocar trechos de códigos do *servlet* direto em um documento texto.

2.7 *Contêineres J2EE*

Normalmente, aplicações multicamada são difíceis de escrever porque eles envolvem muitas linhas de código complicado para controlar e administrar o estado da transação, multi-encadeamento, recurso agrupando, e outros complexos detalhes de baixo nível. A arquitetura *J2EE* baseada em componentes e independente de plataforma torna as aplicações mais fáceis de escrever porque a lógica de negócios é organizada dentro do conceito de componentes reutilizáveis. Além, disso o servidor de *J2EE* provê serviços subjacentes na forma de um *contêiner* para todo tipo de componente. Com isso não precisa desenvolver estes serviços, podendo concentrar-se em resolver outros problemas. *Contêineres* são interfaces entre um componente *J2EE* e a plataforma onde se encontra a aplicação.

2.7.1 Componentes e *Contêiner*

Os componentes são módulos de uma aplicação, só podem ser criados dentro das regras padrões, classes e interfaces são executadas através de *Contêineres*. Um componente só influência com outro através de métodos e protocolo do *contêiner*.

Contêineres ficam localizados entre os componentes e clientes ele fica verificando serviços para os dois. Os *contêineres* autorizam que comportamentos dos componentes possam ser especificados no momento da implantação, ao invés de estarem definidos no código do programa. Veja na figura 2.4.

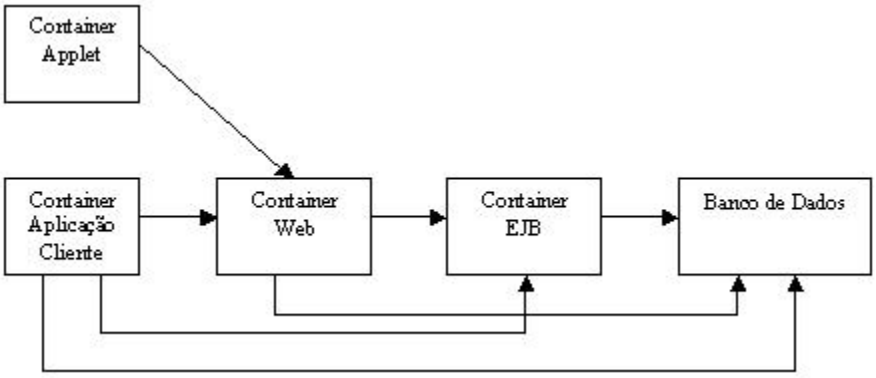


Figura 2.4 - *Contêineres* da Plataforma *J2EE*

2.7.2 Tipos de *Contêineres*

Existem os seguintes tipos de *contêineres* que executam no servidor *J2EE*:

- *Contêiner EJB* (*Enterprise JavaBeans*):

  Um *contêiner* para receber todas as *enterprise Beans* de uma aplicação ou apenas algumas, os *Enterprise Beans* e seus *contêineres* são executados no servidor *J2EE*.

- *Contêiner Web*:

  Um *contêiner* para receber todos os *servlets* *JSPs* e de uma aplicação ou apenas algumas, os Componentes Web e seus *contêineres* são executados no servidor *J2EE*.

- *Contêiner* da Aplicação Cliente:

  Gerencia a execução de componentes da aplicação. A aplicação cliente e seu *contêiner* rodam na máquina cliente.

- *Contêiner Applet*:

Gerencia a execução de *applets*. Consiste em uma camada Web, rodam na máquina cliente.

Os *servlets* e *JSP's* podem ser executados sem a existência "*J2EE server*" completo.

- Podem ser executados em um servidor Web com suporte sem possuir suporte a *EJB*.
- Como por exemplo: *Apache Tomcat*

Na figura, onde tem "*J2EE server*", na realidade, pode possuir máquinas diferentes. Maiores informações podem ser encontradas no site http://jacques.dsc.ufpb.br/cursos/j2ee/html/intro/intro.htm . Veja o funcionamento relacionado na figura 2.5

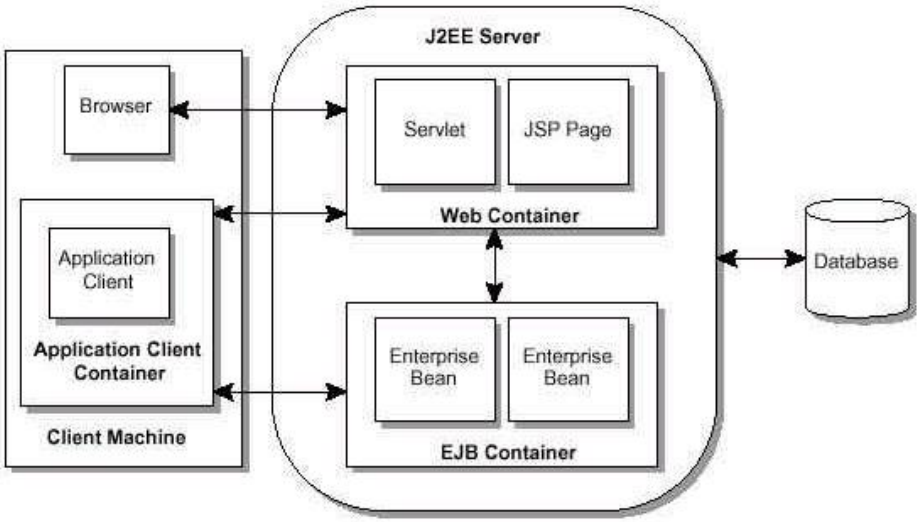


Figura 2.5 Servidor *J2EE e Contêineres* [12].

2.8 *Servlets*

Servlet é um programa feito em *Java*, faz parte da plataforma *J2EE* no qual roda no lado servidor de aplicação que cuida das mensagens entre um cliente e o servidor. Eles possuem acesso em todo o conjunto de

API´s (*Application Programming Interfaces,* interfaces de programação de aplicações) *Java*, como exemplo a *API JDBC*. Um *servlet* pode responder a solicitações feitas por um cliente pela construção de uma resposta que no qual é reenviado cliente. Um *servlet* pode executar várias solicitações concorrentes e podem também sincronizar solicitações. Eles podem repassar solicitações para outros servidores sou *servlets*.

2.8.1 Como funciona um *Servlet*

O servidor é que executa, carrega e gerencia os *servlets*, ele usa uma máquina virtual *java* para rodar os programas *java*. Como mostra na figura2.6, maiores informações podem ser encontras no site http://www.ic.unicamp.br/ .

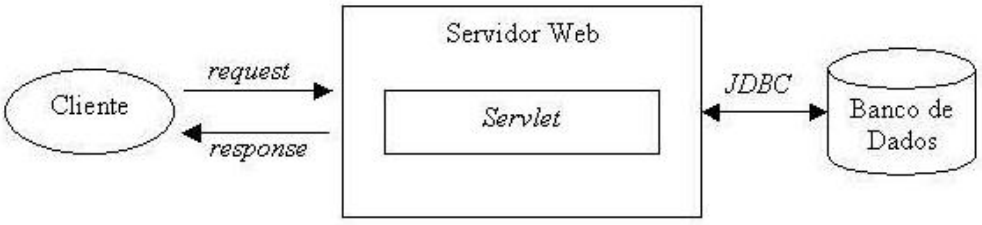


Figura 2.6 - Funcionamento de um *Servlet* [06].

O cliente envia uma solicitação ao servidor, este carrega o *servlet* e cria um *thread* para o *servlet* processar, o *servlet* fica carregado até que o servidor seja desligado. O servidor envia a informação de solicitação ao *servlet* e este cria uma resposta e a repassa ao servidor. O servidor envia a resposta ao cliente.

2.8.2 *API Servlet* do Java

Se refere a um conjunto que explica uma interface padrão entre um cliente e um *servlet Web*. A *API* encapsula solicitações de clientes como objetos e o servidor passa para o *servlet*, da mesma forma as respostas são encapsuladas e com isto o servidor repassa-as para o cliente.

2.8.3 *Applets* e *Servlets*

Se a aplicação visa atender um grande número de usuários os *applets* não é uma solução recomendada, pois vai necessitar que o usuário tenha um navegador com suporte em *java* na versão apropriada e contra partida os *servlets* do lado do servidor pode conter apenas páginas *HTML*, ou seja, o uso de *applets* não é aconselhado para ambientes com vários navegadores.

2.9 *Applets*

Uma página da *web* recebida da camada *web* pode compreender um *applet* embutido. Um *applet* é uma pequena aplicação *client* escrita na linguagem de programação de *Java* que executa no *Java* VM instalada no *web* browser. Porém, sistemas de *client* provavelmente precisarão de um Java *Plug-in* e possivelmente de um arquivo de segurança assim o *applet* podem executar com sucesso no *Web browser*. Componentes de *Web* são o API preferido por criar um programa *client* de Web onde não há necessidade de nenhum *Plug-in* e nem de arquivos de segurança nos

sistemas do cliente. Também, componentes de *Web* habilitam no projeto aplicações modulares porque eles provêem um modo para separar aplicações que programam do *design* da página Web. Isto significa que as pessoas envolvidas no *design* da página na *Web* não precisam entender da linguagem de programação *Java* para realizar suas respectivas tarefas. Se uma página que contém um *applet* é solicitado ao cliente a um servidor, o código executável é informado ao cliente e armazenado em uma área temporária e excluída depois da apresentação da página. Para incluir um *applet* a uma página é necessário inserir no *HTML* códigos necessários para execução do *applet* e no diretório da página o *applet* já deve ter sido compilado com extensão .class. *applets* não acessam recursos de máquinas locais e não faz conexão a outros servidores sem receber permissão.

2.10 *JSP* (*Java Server Pages*)

A Tecnologia *Java Server Pages(JSP)* é uma tecnologia para desenvolvimento de aplicações Web, no qual é parecido com *Hypertext Preprocessor (PHP)* e com *Microsoft Active Server Pages (ASP*). Permite criar páginas Web dinâmicas, tem a vantagem de poder ser executado em outros Sistemas Operacionais além da *Microsoft*. Permite aplicações que permitam acesso aos diversos Banco de Dados.

Java Server Pages utiliza *tags* igual às do *XML* e *scriptlets* escritas em java que gera o conteúdo dinâmico da página através do encapsulamento da lógica. Todas as *tags* (*XML* ou *HTML*) passam imediatamente para página de resposta. Facilitando para os desenvolvedores as páginas *JSP* são criadas usando páginas escritas em *HTML* e nela acrescentada as *tags XML* equivalentes. *J2EE servers* convertem JSP's para *servlets* quando solicitado.

2.10.1 Como Funciona

Através de um browser o cliente faz uma solicitação de uma página *JSP*, no qual será processada pelo servidor. Se for a primeira solicitação a páginas JSP é convertida em um programa *java* (*servlet*) no qual é compilado e gerado os *.class* gerando uma páginas *HTML* que será enviada a resposta ao browser do cliente. Na próxima vez que esta página for solicitada será verificado apenas se ocorreu mudanças ou não, se não constar nenhuma mudança o *.class* é carregado para gerar a página *HTML*.

2.11 Conclusão

Neste capítulo vimos definições de *J2EE*, *Servlet, applets* e *JSP*. Vimos o funcionamento do *J2EE*, suas aplicações e contêineres. Foi mostrado como as *applets* Java são incorporadas em páginas *Web*. No próximo capítulo será abordado *Enterpeise JavaBeans* com seus conceitos e benefícios.

3 *EJB* (*Enterprise Java Beans*)

A tecnologia *enterprise Java Beans* marca um modelo para desenvolver e disponibilizar componentes servidores que são usados pelo *java*. O *EJB* é uma ampliação do sistema de componentes de *Java Beans* para dedicar um suporte mais apropriado aos componentes servidores. Os componentes servidores são partes de uma aplicação que são executadas em um servidor de aplicações. A tecnologia *EJB* faz parte da plataforma da *Sun MicroSystem*, ela dedica suporte ao desenvolvimento de aplicações fundamentais em uma arquitetura de objetos distribuídos em várias camadas. Um componente *EJB* pode executar em qualquer plataforma e também representa pequeno volume em diferentes implementações de servidores de aplicações compatíveis com *EJB*. Por isto em uma infra-estrutura de serviços o ambiente *EJB* mapea automaticamente um componente. A infra-estrutura de EJB expõe uma série de interfaces padronizadas para que uma aplicação possa acessar: *RMI*, *JNDI CORBA*, *Servlets*, *JPS*, *JMS*, *JTS*, *JDBC*.

3.1 O que é *Enterprise Beans*

Enterprise Bean na linguagem de programação é um componente do lado do servidor que encapsula a lógica empresarial de uma aplicação. A

lógica de negócio é o código que cumpre o propósito da aplicação. Em um controle de inventário por exemplo aplicação de *Enterprise Bean* poderia implementar a lógica de negócio em métodos chamada e *orderProduct*. Invocando estes métodos, os clientes remotos podem ter acesso os serviços de inventário providos pela aplicação.

3.2 Benefícios do *Enterprise Beans*

Por várias razões, o *Enterprise Bean* simplifica o desenvolvimento de grandes, aplicações distribuídas. Primeiro, porque o *contêiner EJB* dispõe serviços de nível de sistemas do *Enterprise Beans*, o desenvolvedor do *Bean* pode concentrar em resolver o problema de negócio. O *contêiner* EJB não o desenvolvedor do *Bean* é responsável em nível de sistema, como gerenciamento de transações e autorização de segurança.

Segundo, porque como os *beans* e não os clientes contêm o negócio da aplicação lógica da aplicação, o desenvolvedor do cliente pode focalizar na apresentação do cliente. O desenvolvedor do cliente não tem que codificar as rotinas que implementam regras de negócios ou acesso a bancos de dados. Como resultado, um benefício que é particularmente, importante para clientes que executam em dispositivos pequenos.

Terceiro, porque os *Enterprise Beans* são componentes portáteis, o desenvolvedor da aplicação, pode construir aplicações novas de *Beans* a partir de *Beans* existentes. Estas aplicações podem ser executadas em

qualquer servidor de *J2EE* compatível.

3.3 Componentes *Enterprise Java Beans*

Enterprise Java Beans representa uma arquitetura de computação distribuída baseada em componentes. As características de um componente EJB:

- Em servidor compatível com a especificação *EJB*, podem ser distribuídos.
- Através de arquivos *xml*, pode ser configurado em tempo de implementação.
- As instâncias de *EJB* são gerenciadas e criadas em execução no *contêiner EJB*
- Fornece a visão da corporação a camada cliente.
- Controle transacional, atributos de segurança, e outros, são separados das classes e interfaces do componente EJB.

3.4 Tipos de *Enterprise Java Beans*

Os componentes *EJB* podem ser dos seguintes tipos: *Session Beans*, *Entity Beans*, *Message Driven Bean*.

3.4.1 *Session Beans*(*Beans* de sessão)

É um componente mais simples do *EJB*. Ele cria componente entre cliente e o servidor. Pode ou não possuir estado. Os *Beans* de Sessão são subdivididos em dois tipos: com informação de estado(*statefull*) e sem informação de estado(*stateless*).

Beans de sessão com informação de estado(*Beans Stateful*)

Na sessão *cliente-bean* o estado fica ali até que o cliente remove o *bean* e assim termina a sessão. Portanto representa uma sessão com um único cliente. Maiores informações podem ser encontradas no site http://berlin.inesc.pt/cadeiras/atsi/trabalhos . Veja como funciona na figura 3.1.

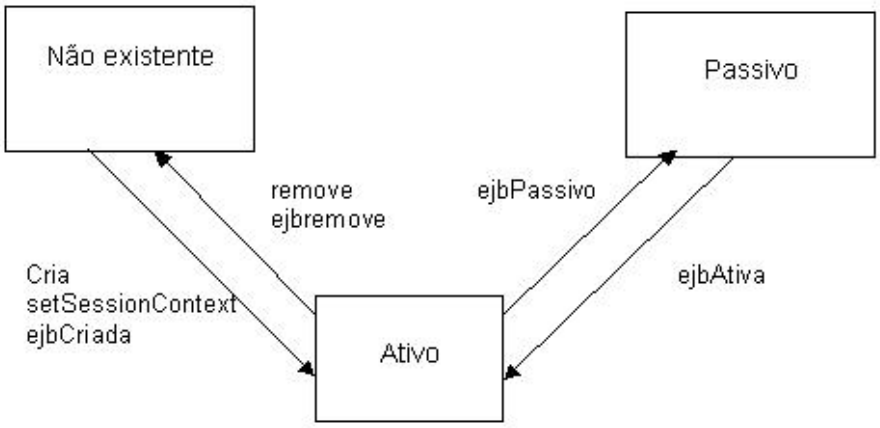


Figura 3.1 - Ciclo de vida de um *stateful* [13]

Beans de sessão sem informação de estado (*Beans stateless*)

Não mantêm um estado com o cliente em particular. Quando um cliente chama um método, as variáveis de instância podem conter um estado

somente durante o chamado do método. Quando o método termina este deixa de existir. Maiores informações podem ser encontradas no site http://berlin.inesc.pt/cadeiras/atsi/trabalhos . Veja como funciona na figura 3.2.

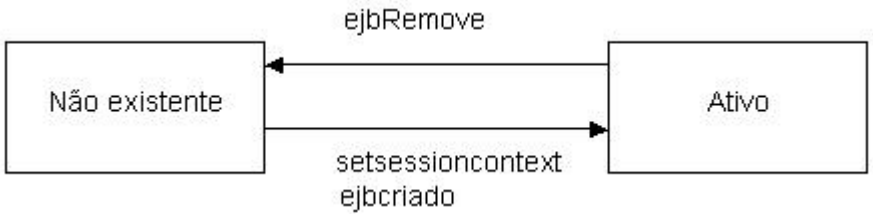


Figura 3.2 - Ciclo de vida de um *stateless* [13]

3.4.2 *Entity Beans* (Beans de Entidade)

É um componente um pouco parecido com o *session bean (Bean de sessão)*, mas suporta diversas sessões de clientes, e é necessariamente persistente. Um componente se acesso aos dados geralmente está associado a uma tabela de um banco de dados de forma geral fornece um mecanismo padronizado e orientado a objetos para acesso aos dados. O ciclo de vida deste tipo de *beans* é controlado pelo contentor e não pela aplicação. Maiores informações podem ser encontradas no site http://berlin.inesc.pt/cadeiras/atsi/trabalhos . Veja como funciona na figura 3.3.

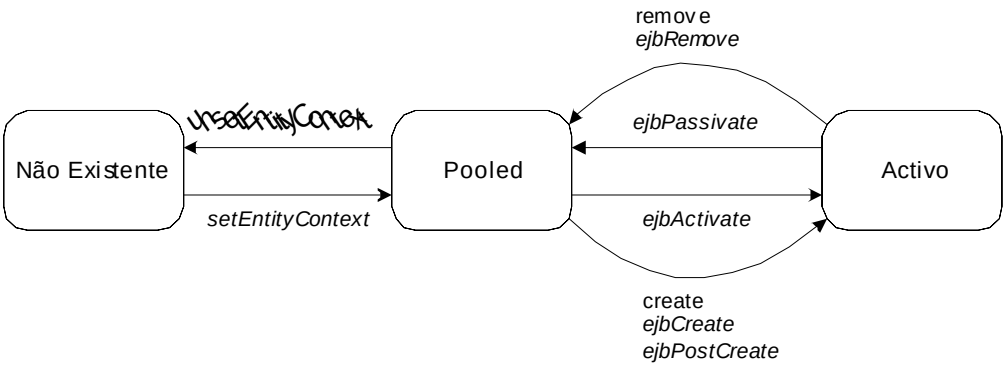


Figura 3.3 - Diagrama de estados de um *Entity Bean* [13]

3.4.3 *Message Driven Bean* (*Bean* Orientado a Mensagem)

Objeto de negócio consumidor de mensagens, permitem que aplicação *J2EE* processem mensagens assíncronas. Os *beans* orientados a mensagem processam apenas mensagem *JMS*, futuramente eles serão usados para processar outros tipos de mensagens. As mensagens podem ser enviadas por qualquer componente *J2EE*.

3.5 Arquitetura dos *Enterprise Java Beans*

O funcionamento dos *EJB* é diferente dos objetos que conhecemos, seu acesso é feito através de ambiente remoto, com isto deve existir uma camada de serviços que permita que os clientes utilizam e localizam os *EJBs* a partir da rede. Por isto o servidor EJB através do *contêiner* fornece estes serviços. E para isto é necessário das interfaces *home* e Remota.

3.6 Conclusão

Neste capítulo conhecemos um pouco do *Enterprise Bean* conceitos, seus benefícios, componentes seus tipos(sessão, entidade, mensagem). O *Enterprise Bean* requer uma arquitetura de objetos distribuídos pela *internet*, dispõe um ambiente para o gerenciamento de componentes. No próximo capítulo será abordado o estudo de caso.

4 Estudo de Caso

Neste capítulo é desenvolvido um programa de cadastro de endereço de *CEP*, criando os componentes necessários através da integração do *J2EE* com uma ferramenta de desenvolvimento chamada *Magic*. A seguir será descrito todos os procedimentos para criar um componente *EJB*. Este capítulo é baseado na documentação de referência do software *Magic eDeveloper*.

4.1 *J2EE* e *Magic eDeveloper*

Magic Application server e *J2EE* são integrados através de *EJBs*. Aplicações desenvolvidas com o *Magic* e*Developer* integram com os servidores de aplicação *J2EE* como o *Bea WebLogic*, *IBM WebSphere* e outros. O desenvolvedor define componentes desenvolvidos com o *Magic eDeveloper* como componente *EJB*, que exibe serviços baseados em aplicações *EJB*. Empresas que encontram dificuldades de entrar no mundo *J2EE*, devido aos obstáculos tecnológicos encontrados e a falta dos desenvolvedores *Java*, podem ter a vantagem de trabalhar com *Magic eDeveloper* para criar componentes *EJB* de forma rápida.

Magic possui a vantagem de poder trabalhar com vários bancos ao mesmo tempo. No *Magic* a uma funcionalidade na aplicação incluindo

estrutura de dados, regras de negócio, lógica de programação e apresentação.

4.2 *Magic eDeveloper*

Tecnologia desenvolvida pela *MAGIC Software Enterprise* uma conceituada e reconhecida empresa de desenvolvimento de *softwares* com mais de 15 anos de mercado com sede em Israel com atuação destacada nos maiores centro tecnológicos do mundo. *Magic* é uma ferramenta de desenvolvimento com repositório de regras, totalmente gráfica, voltada para trabalhar com banco de dados. A *Magic Software* do Brasil atua nas áreas de comercialização, treinamento, prestação de serviços profissionais e consultoria.

A *Magic Software* do Brasil utiliza única e exclusivamente a tecnologia *Magic*, fornecendo a seus clientes tecnologia de ponta em seus projetos de TI.

- Rápido desenvolvimento sem código;
- Desenvolvimento Web e C/S no mesmo paradigma;
- Independência de plataforma(*iSeries-OS/400*, *Windows2000*, *XP* entre outros;
- Particionamento de lógica/Servidor de aplicações/*Middleware*;
- *J2EE*(*Java*);
- *WebServices*;
- *Ebusiness.*

4.3 *Jboss Application Server*

JBoss é um servidor de aplicações *Java*, no começo se chamava *EJBoss* (*EJB Open Source Server).* É compatível com as especificações

J2EE. Devido aos custos, o *JBoss* ainda não conseguiu certificação J2EE da *Sun*, mas está trabalhando para adquirir.

O *JBoss* é uma implementação do *EJB* 1.1 (e parte de *EJB* 2.0), ou seja é um servidor *contêiner* de *Enterprise Java Beans*. O *JBoss* dispõe de um servidor de *EJB*, não inclui Web *contêiner* para *servlets*/JSP, embora há pacotes disponíveis incluindo *Tomcat* ou *Jetty*.

4.4 Funcionalidades da ferramenta *Magic eDeveloper*

A seguir serão descritas as funcionalidades da ferramenta *Magic eDeveloper* que foram utilizadas no estudo de caso.

4.4.1 *Component Builder*

Com o *Magic eDeveloper* pode-se criar componentes *EJB* usando programas existentes, pode-se definir objetos de aplicação como componentes e então usá-los em outras partes da aplicação. Um componente é um arquivo de aplicação *Magic* com uma interface. Usando a interface, outra aplicação *Magic* pode chamar um objeto dentro da primeira aplicação, e executar ou usar o objeto como se fosse parte da aplicação *host*. O arquivo (*MCI*) criado é chamado de uma Interface de *Component Magic*.

O C*omponent Builder* tem como funcionalidade, empacotar os módulos da aplicação, e os salva na forma de um arquivo *jar*. Este arquivo conterá todos os módulos exigidos inclusive arquivos *.class e .jar*.

Pode-se criar os seguintes componentes:

Magic Interface, COM Interface, Web Service Interface, EJB Interface, XML, JAVA. Serão Mostrados apenas os componentes referentes ao *Magic* com *EJB* (*Enterpiase Java Beans*).

4.4.2 Magic Interface Builder

O *Magic Interface Builder* permite criar um componente *Magic* (*MCI*). Como

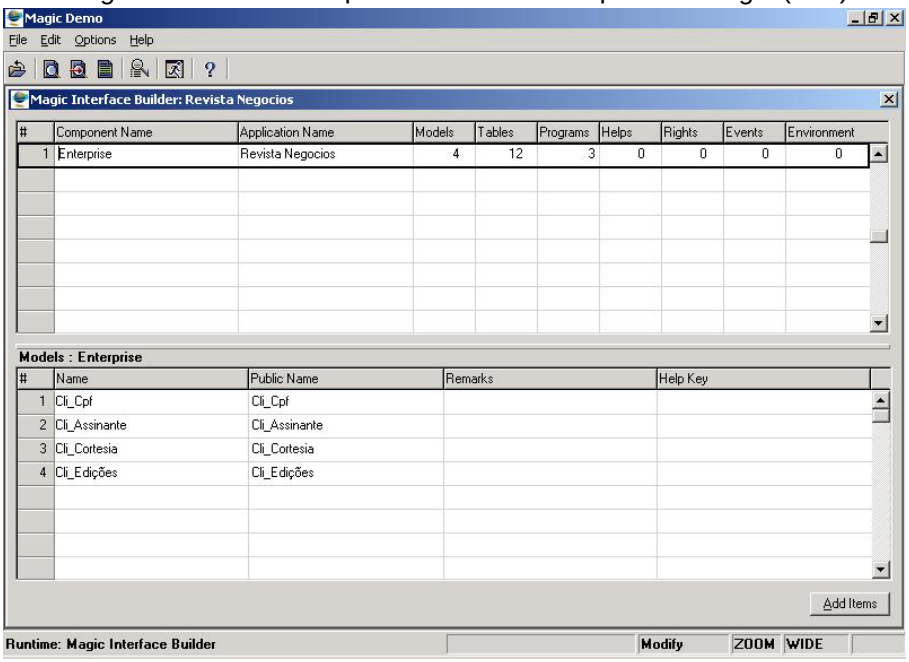


Figura 4.1 Magic Components Builder Repository

Ao criar um componente *Magic*, as colunas são preenchidas da seguinte forma:
Compoment Name: Nome do componente é obrigatório.
Application Name: o *Magic* automaticamente mostra o nome da aplicação atual.
Os tipos de itens que podem ser chamados em um arquivo de *MCI* são:
Models, Tables, Programs, Helps, Rights, Events, Environment Settings.

4.4.3 *Magic Component Builder Properties*

Pode especificar as seguintes propriedades:

Description: Descrição do arquivo *MCI*;

Revision: O número de revisão do arquivo *MCI*;

MFF: Selecionado serve para especificar que o componente é uma aplicação *flat-file Magic*. Quando uma aplicação de *flat-file* é especificada, o banco de dados da aplicação fica desativado, pois a persistência passa a ser efetuada em arquivo;
Application File Name: Arquivo da aplicação atual;
Application Database: Banco de Dados do arquivo *MCI*;
Help File Name: Caminho do arquivo *Help*;
Help Key: Combinação da chave que abre o arquivo de ajuda do componente.

4.4.4 Item Type Repository

O Item *Type Repository* mostra os itens chamados ao tipo selecionado, os detalhes são mostrados abaixo.

Item Name: Mostra os itens selecionados na lista de itens;
Puclic Name: Exibe automaticamente o nome público do programa selecionado;
Remarks: Pode adicionar informações referentes ao item;
Help Key: Pode colocar um número que é fixo para abrir uma página de *Help* do arquivo especificado no *component properties*.

4.4.5 *Environment Repository*

No *Environment Repository,* escolhe-se seguintes categorias para visualizar a lista de itens que pode-se acrescentar em um arquivo *MCI*.

Environment Settings
Servers
Services
Database
Logical Names

4.4.6 Gerando o arquivo *Magic Component Interface*

Depois de inserir os itens de *MCI* e propriedades, clicar em *Options* / *Build Interface File* para criar o arquivo *MCI*, irá mostrar a tela com o nome do arquivo (*component name*), como mostra a figura 4.2.

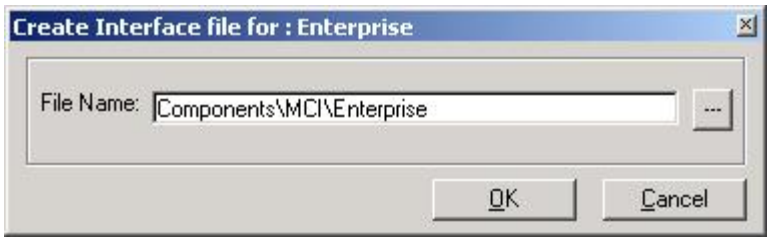


Figura 4.2 Arquivo *MCI*

Se o diretório para o arquivo *MCI* não existir o *component Builder* criará.
Se o arquivo *MCI* já existir no caminho especificado irá surgir uma mensagem perguntando se deseja sobrepô-lo.

4.4.7 Integrando Componente para a Aplicação

Quando abre-se a lista de seleção em uma nova aplicação a *View list* no topo da janela permite definir os objetos para seleção, como mostra abaixo:

All: Todos os objetos estão disponíveis na janela de seleção da aplicação atual e do componente;
Internal: Estão disponíveis somente objetos da aplicação atual;
Component Name: Todos os nomes de componentes para *components repository* aparecem. Podendo selecionar para ver um componente específico.

4.4.8 Carregando um Componente

Pode-se usar componentes para operações de aplicações específicas, e podem ser usadas por outras aplicações. Pode usar componentes prontos a partir de exportação de componentes. Podendo compartilhar objetos carregando o componente em sua aplicação.

1. Abra a aplicação que deseja usar o componente;
2. No menu clicar em *workspace* seleciona *Components*;
3. Pressione F4 para criar uma nova linha;
4. No menu clicar em *options* selecione *Load/Reload*;
5. Seleciona o componente que deseja clica em abrir.

Com isto o componente estará aberto e pode-se usar em sua aplicação.

4.4.9 Selecionando um Componente para Integração

Abaixo mostra os procedimentos para selecionar uma tabela de componente:

1. Abre o programa *repository* e pressione F4;
2 . Pressione Ctrl+G para ativar *Automatic o Program Generator* (*APG*);
3. Em *main Table* pressione F5 para abrir a janela com as tabelas da aplicação atual e os componentes.

4. Selecione a tabela exigida para criar um programa baseado em componente ao invés de uma tabela *repository* da tabela da aplicação atual. Figura 4.3 mostra a tela do *Program Generator.*

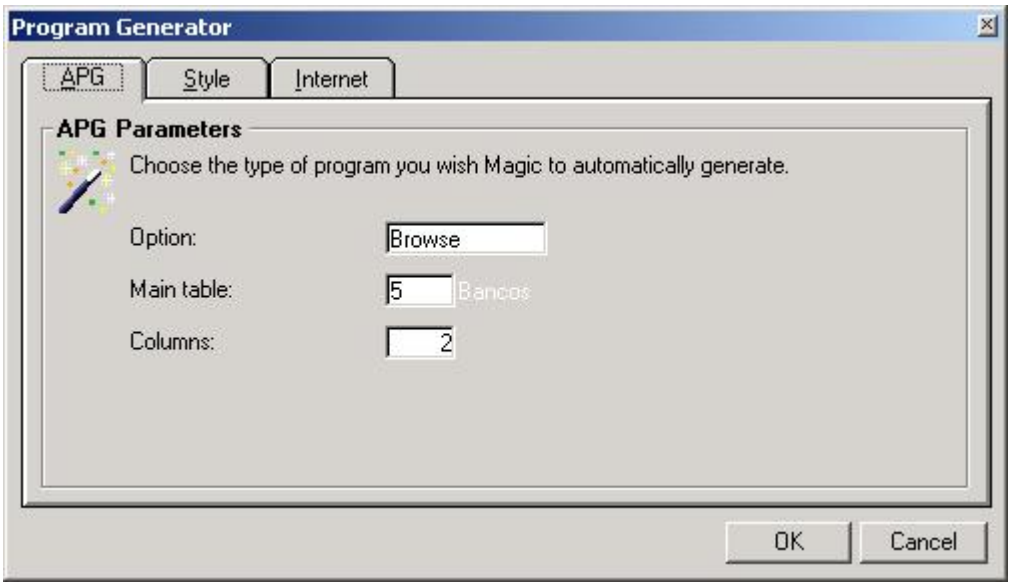
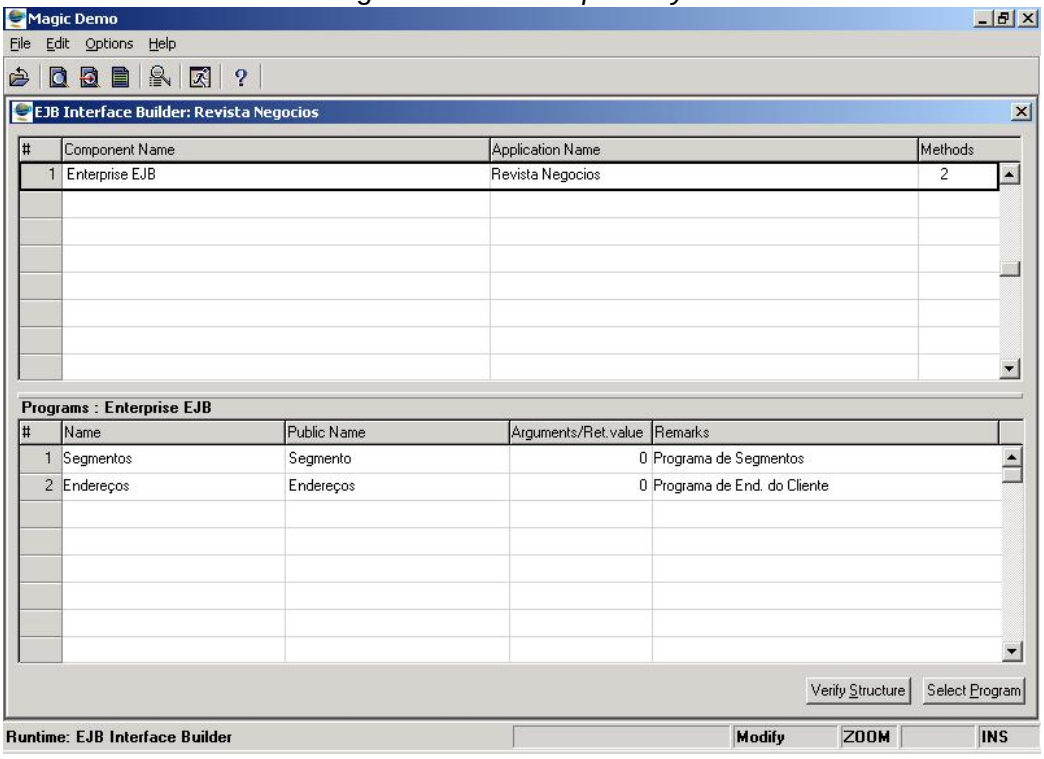


Figura 4.3 *Program Generator*

4.4.10 Enterprise JavaBean Interface Builder

Este permite criar componentes *EJB*. O *EJB Interface Builder* habilita uma aplicação *Magic* a ser exposta como um *EJB* em um *J2EE enterprise server*. No menu, clicar em *component / EJB Interfac*. O *EJB repository* abrirá, como mostra a figura 4.4.

Figura 4.4 *EJB Repository*



Ao criar *EJB Interfac Builder* as colunas são preenchidas da seguinte forma:

Compoment Name: Nome do componente é obrigatório.
Application Name: o *Magic* Automaticamente mostra o nome da aplicação atual.
Methods: Exibe automaticamente o número de programas que chamou o arquivo EJB Interface.

4.4.11 EJB Programs Repository

O *EJB Programs Repository* as colunas são preenchidas como mostra abaixo:

Program Name: Mostra na tela o programa de acordo com que foi especificado na lista de programas. O programa tem que ser uma tarefa Batch;
Public Name: Exibe automaticamente o nome do programa selecionado, este nome não pode ser mudado;
Arguments and Returned Values: Exibe o número total de argumentos. Para seleciona-los pressione F5;
Remarks: Pode adicionar informações referentes ao programa.

Clicar no botão *verify Structure*, para começar a checagem do programa. Se não há nenhum erro na estrutura do programa o *Magic* mostra uma mensagem de confirmação de OK da estrutura. Se há erros o *Magic* mostra uma advertência.

4.4.12 *EJB Settings*

No menu clicar em *Options* / *Settings*, para selecionar um dos servidores de aplicação, *J2EE Enterprise Servers*.

- *BEA WebLogic Version 5 or 6*
- *Sun Reference Version 1.3*
- *JBOSS*
- *Oracle Enterprise Server Version 9i*
- *Sun ONE Enterprise Server Version 7*
- *Fujitsu Interstage Enterprise Server Version 5*
- *WebSphere*

Como mostra a figura 4.5:

Figura 4.5 *J2EE Aplication Server*

4.4.13 Gerando o arquivo Componente *EJB*

Depois de especificar os programas *EJB* e *Settings*, clicar em *Options* / *Build Jar File* para criar o arquivo *EJB*.

Se o diretório para o arquivo *EJB* não existir o *EJB Interface Builder* criará. Se o arquivo *EJB* já existir no caminho especificado irá surgir uma mensagem perguntando se deseja sobrepô-lo.

Quando o processo de criação do arquivo *JAR* estiver completo o *Component Builder* mostra que o arquivo foi criado com sucesso mostrando uma mensagem com a seguinte informação:

Bean XYZ packed as into path\XYZ.jar,successfully

Como mostra a figura 4.6.

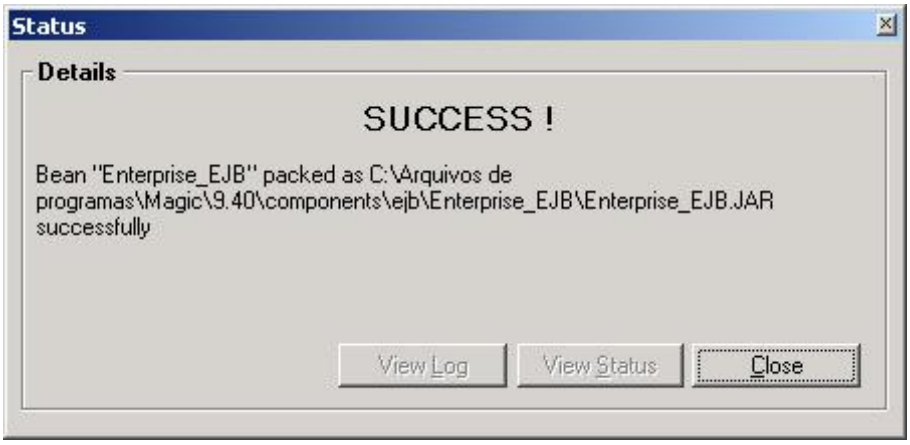


Figura 4.6 Resultado do arquivo *.jar*

Se o *Magic* não gerar o arquivo, irá mostrar detalhes de erro, podendo ver o arquivo de *View Log* e *View* status.

OBS: Para criar o arquivo *JAR* tem que ter o *JDK* versão1.3 ou superior instalado na máquina. Se estiver criando o arquivo *JAR* para um destes servidores (*BEA WebLogic Version 5 or 6, Sun Reference Version 1.3, JBOSS, Oracle Enterprise Server Version 9i, Sun ONE Enterprise Server Version 7, Fujitsu Interstage Enterprise Server Version 5, WebSphere*), também precisa estar instalado.

Os arquivos são salvos na pasta *EJB* do diretório principal do *eDeveloper*. Para facilitar os testes o componente *builder* gera uma aplicação teste chamada <*Bean name*>*_test_app.ear*. Esta aplicação não é parte integrante do *EJB* gerado, mas muito envolvida com aplicação

cliente simples que pode ser ativado e modificado se precisar testar o *EJB*.

Para fazer o *deploy* de um EJB criado precisa executar os seguintes passos:

- Montar um recurso de *URL* para o servidor *J2EE*;
- *Startar* o servidor *J2EE*;
- *Startar o Deploy Tool*

4.4.14 Configuração do *EJB* (*Enterprise Java Bean*)

O *Magic Component Builder* cria um *Deployment Descriptor* (*DD*) arquivo que descreve o *EJB* recentemente criado em pacotes, isto no arquivo *JAR*. Este arquivo contém as informações que o *Bean* precisa para operar. Os dois tipos de cenários são ambientes e recursos.

4.4.15 Additional Generated Jar Files

Arquivos adicionais são criados dependendo da seleção *J2EE Application Server*. Um destes arquivos é chamado *ClientTest*. O *ClientTest* é uma pequena aplicação Java provê uma maneira fácil de conferir uma conexão entre uma aplicação Java e uma aplicação *Magic*.

4.4.16..... Compartilhando um Evento Entre Aplicações

Um Evento *Handler* de uma aplicação define como um Evento Global no Programa Principal, pode ser pego em um componente quando o evento é disparado de um outro componente, ou na própria aplicação principal, a aplicação *host*.

Um componente que inclui um Evento *Handler* no componente programa principal que você define como uma *Sub-Tree* também pode ser pego no componente quando o programa que disparou o evento também for parte do componente.

A procura para o caminho *Event Handler* é como segue:

- O ponto inicial é a tarefa onde o evento foi disparado do *Runtime* para o Programa Principal da aplicação *Host*.
- Depois disto o Evento *Handlers* é procurado nos componentes carregados e no programa principal, mas somente para os *handlers* definidos como Global.
- Aplicação *host* evento *handlers* definidos como Global, serão tratados como *handles* de *Sub-Task*.

Os componentes evento *handlers* que estão definidos como *Sub-Task* disparam de um programa na mesma aplicação de componente, será tratado como *handlers* Globais.

4.4.17.....Mantendo a Aplicação de Componente Carregada

Quando cria um componente, deve incluir informação sobre a ajuda, cores e arquivos de fontes que são usados naquele componente específico.

Uma vez em *Runtime*, o componente está carregado, algumas das propriedades de ambiente da aplicação de componente, como *Database*, *DBMS*, *Server* e *Services Properties* e *Logical Names*, são acrescentados às seções de *Magic.ini* dentro da aplicação *host*.
OBS: Ao apagar um componente da aplicação *host*, as propriedades acrescentadas não são apagadas automaticamente do arquivo Magic.ini.

Pode fixar ambiente especial de valores quando criar um componente novo no *Component Interfac Builder*. Estes valores são armazenados dentro do arquivo de *MCI*. Quando carregar o componente da aplicação *host* estes valores entram no ambiente *host*.
Nas propriedades de componente pode fixar um local e nomeá-lo para arquivo de *help*. O arquivo novo é usado para o objeto de componente ao invés de arquivo *host*. Use os arquivos de aplicação *host* para os objetos de componentes se não definir um local para este arquivo.

4.5 Comparando *Magic* e *Java*

Na tabela abaixo mostra os tipos de *Magic* e os referentes tipos de *Java*:

Tabela 4.1 Comparação de *Magic* e *Java*

Magic	*Java*
Alfa	*String / stringBuffer*
Blob	*String / stringBuffer / Byte*
Numeric	*Byte / short / long / float / double*
Logical	*Boolean*
Data / Time	*String*

Pode escolher tipos de Java para os parâmetros e pode retornar valores que são *Blod* ou campos Alfa, ou expressões que podem retornar *Blod* avaliam tal como *FILE2BLB* ou campos de *Alfa*(*STR*).

4.6 Descrição detalhada do desenvolvimento do estudo de caso.

Foi desenvolvido um programa de cadastro de endereços de CEP utilizando o banco *Pervasive* (Banco utilizado pelo *Magic*) contendo os cadastros de CEP, logradouro, Bairro, Cidade e UF. A tabela criada se chama CEP.

O programa de cadastro é constituído de um componente *EJB* com nome *Enterprise_EJB*, que o *Magic* cria automaticamente em um diretório com este nome no diretório principal do *developer,* gerando um arquivo .*JAR* contendo todos os módulos necessários.

Para acesso a este componente, um outro software *client* de *J2EE* chama este *EJB* (.jar) acessando os métodos *Magic* através de interfaces *J2EE* para executar o programa contido neste componente. Para isto é preciso configurar o *Magic Broker* na máquina onde o *Magic* está instalado.

4.6.1 Configurações

Antes de abrir a aplicação, clicar em *Settings* / *Servers* no campo *name* digite um nome para o *broker,* em *Server Type* selecione *Magic Requests Broker* .

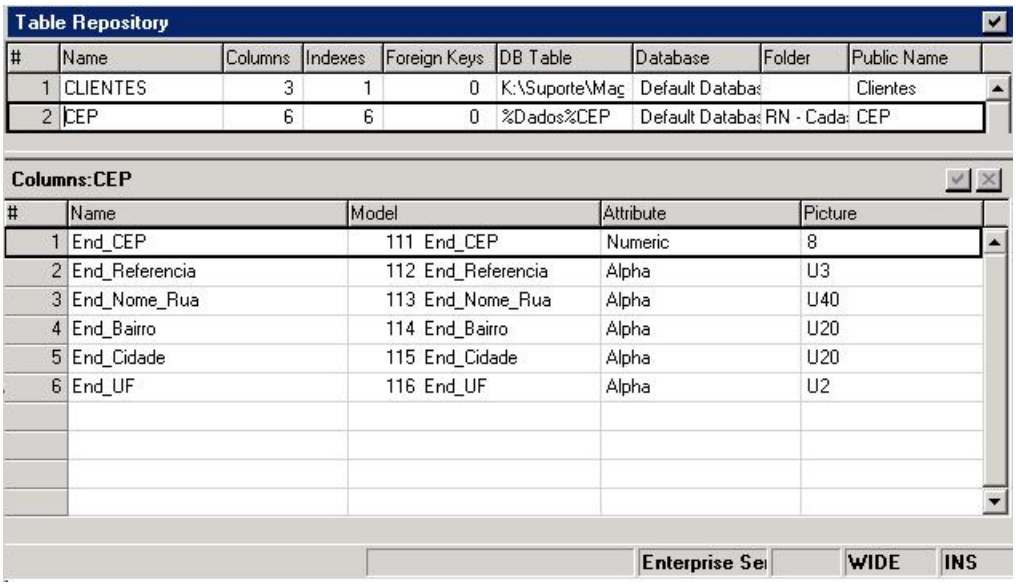
Definindo o ambiente:

Clicar em *Settings / Environment* clicar na aba *Server*, fazer as seguintes configurações:

- Em *Activate as Application server* selecionar *Yes*.
- Em *Messaging Server* selecionar o *Broker* criado.
- Em http Requester digitar */Magic93Scripts/MGRQISPI93.dll*.
- Em *Web Document Alias* digitar */Magic93Scripts*.

4.6.2 Criando a Tabela

Para criar uma tabela, clique em *Tables*, e crie uma nova linha e pressione F5 e depois F4 para criar os campos da tabela, como mostra a figura 4.7



4.7 Campos da tabela CEP

Para criar mais um campo nesta tabela basta criar uma nova linha e inserir o campo desejado.

Na figura 4.8 é mostrada os índices da tabela CEP.

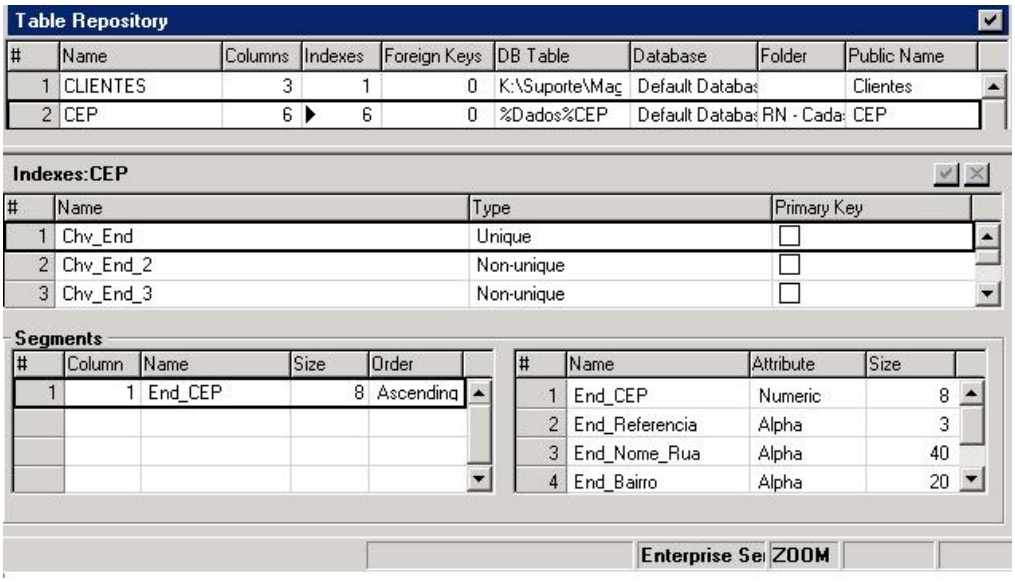


Figura 4.8 Indices da tabela CEP

Para criar mais um índice nesta tabela, pressione F5, crie uma nova linha inserindo o índice que deseja.

4.6.3 Criando um Programa

Há mais de uma maneira para criar um novo programa:

- Clicar em *programs*, criar uma nova linha, pressionar *Ctrl*+*G*, em *Main table,* pressionar F5 para inserir a tabela que será utilizada e em *Option* selecionar o tipo de programa.

- Em tabelas selecione a tabela em que deseja criar um programa, pressione *Ctrl+G*. Em *Mode* selecione o tipo que deseja, em *Display* selecione o tipo de tela que deseja, clicar ok.

Depois do programa criado, pode-se fazer alterações no mesmo bastando clicar em *programs*, selecionar o programa e pressionar F5, que será n

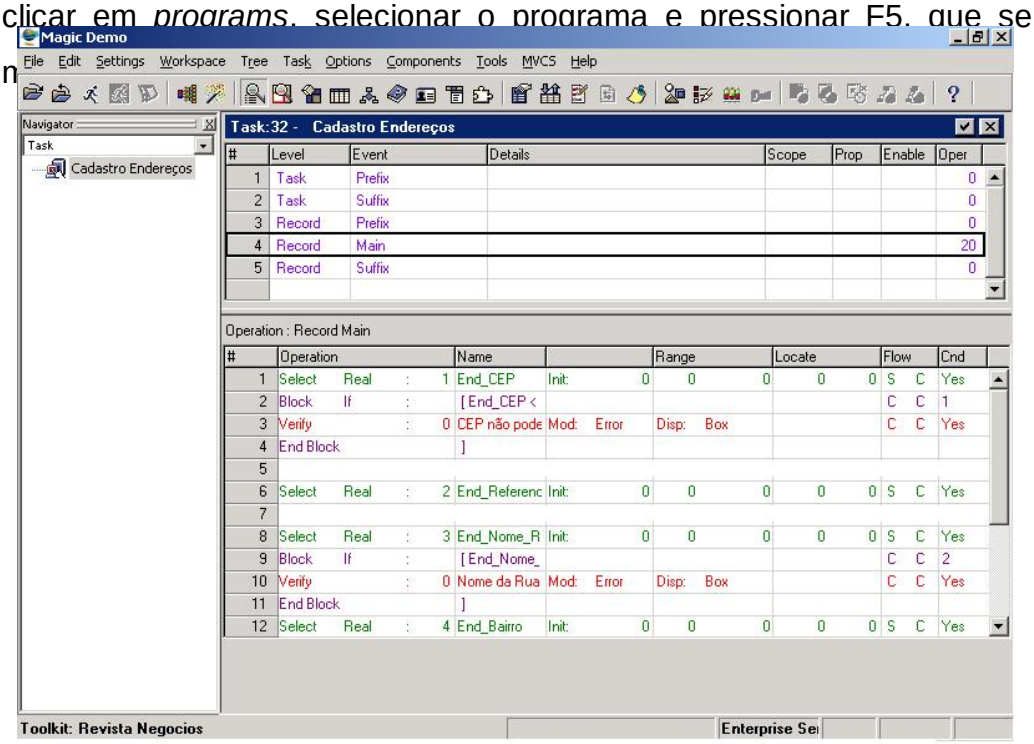


Figura 4.9 Tela para alteração no programa

a

4.

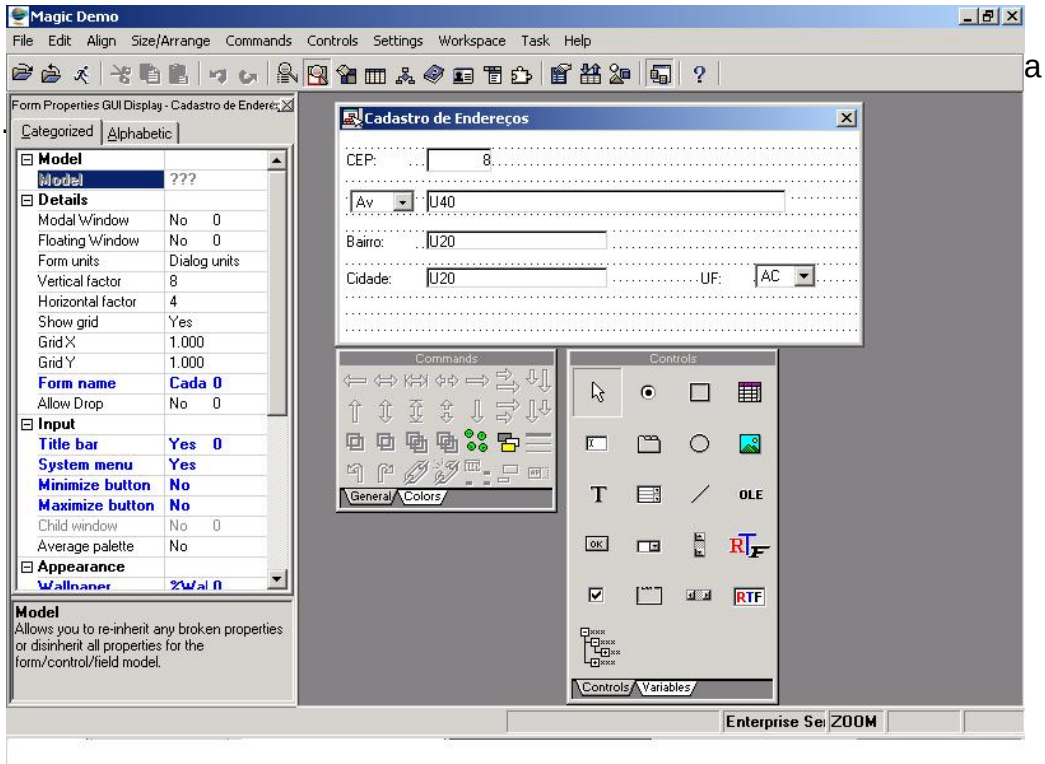


Figura 4.10 Formulário do programa

4.6.4 Criando Cliente *Web*

Criado a partir da tabela CEP existente. Localizada a tabela pressione *Ctrl+G.* Em *Mode* selecione *Generate* em *Option* selecione *Browser client* e em *HTML file name* pode alterar o nome ou deixar o nome que ele gerou (deixei o nome gerado), clicar ok. Para executá-lo abri[illegible] mo[illegible]

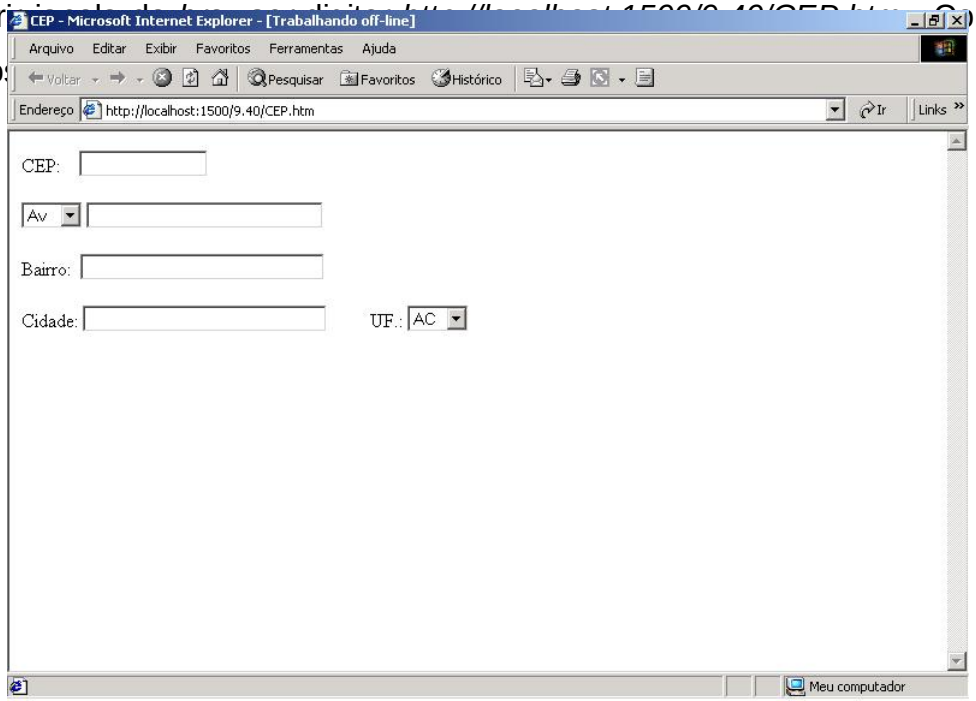


Figura 4.11 Cliente *Web*

4.6.5 Tipo de *Enterprise JavaBean* usado no *Magic*

O *Magic* usa as informações fornecidas pelo programa através da tela *Table Repository* e gera um *Beans* de Sessão (*Stateless*) sem informação de estado. O *deployment* (distribuição) deste componente é feito automaticamente pelo *Magic*. A partir de uma requisição de serviço

ainda de um cliente *J2EE* ou Cliente *Web*. O Container seleciona ou cria se necessário uma instância do componente. Porém um cliente que usa o componente será atendido pela mesma instância, então nem sempre há um estado entre *contêiner* e chamadas de métodos. Portanto nenhum contexto poderá ser mantido dentro do *contêiner*. Um novo contexto será criado para a duração de cada solicitação.

4.7 Conclusão

Neste capítulo conhecemos a integração do *J2EE* com a ferramenta de desenvolvimento chamada *Magic*, foi demonstrado os procedimentos a ser executado para fazer este integração e vimos também uma visão do funcionamento desta ferramenta. No próximo capítulo será abordado a conclusão de todo o trabalho.

5 Conclusão

A plataforma *J2EE* está orientada para a computação empresarial. Baseia-se em componentes reutilizáveis (*Enterprise Java Beans, Java Server Pages, servlets*), permitindo um desenvolvimento simples, rápido e eficiente. Inclui elementos para simplificar a interoperabilidade, tais como o *JDBC* para bases de dados, serviços para sistemas o *JNDI*, *JMS*, *JTS*, *EBJ*, *JSP* e outros.

Definindo um padrão para desenvolvimento de aplicações em multicamadas tornando mais fácil o desenvolvimento de aplicações corporativas pois promove uma padronização baseada em componentes através do fornecimento de um conjunto complexo de serviços para estes componentes e da manipulação automática ao ambiente corporativo, como segurança gerência de recursos, escalabilidade.

Esta plataforma tem sido adotada para implementação de soluções para Web, como no desenvolvimento de pacotes customizáveis voltados para as mais diferentes áreas de atuação. O número de empresas que adotam esta plataforma vem aumentando a cada dia e dentre elas destacamos *Oracle, Bea, Sun*, *IBM*.

A plataforma *J2EE* tem por finalidade reduzir a complexidade de desenvolvimento de aplicações corporativas onde os benefícios são melhores aplicativos, menor custo, menor tempo de desenvolvimento, e maior competitividade no mercado.

Foi apresentado como estudo de caso uma aplicação integrando o *J2EE* com uma ferramenta *RAD - Rapid Application Development* de desenvolvimento rápido *MAGIC* (*RAD*), os procedimentos apresentados demonstram a praticidade na criação componentes *EJB* visando agilidade no desenvolvimento de Projetos.

A ferramenta *Magic* oferece uma ampla produtividade para o desenvolvimento e execução de soluções *Business*. Tem como principal característica de oferecer um ambiente de desenvolvimento totalmente orientado ao preenchimento de tabelas. Com este ambiente possibilita a rápida criação e customização de sistemas de grande porte e complexas aplicações distribuídas.

Referências Bibliográficas

[01] SENAC, graduação, Faculdade São Paulo Disponível: http://www.cei.sp.senac.br/graduacao. Acesso 03/06/2002.

[02] Novo Milenio, Jornal Eletrônico Novo Milenio Disponível: http://www.novomilenio.inf.br/. Acesso 03/06/2002.

[03] Blaz, O Dilema de aprender JSP Disponível: http://www.blaz.com.br/webdev/programacao/jsp. Acesso 14/03/2002.

[04] OLIVEIRA Alcione de Paiva, Apostila *Servlet*/JSP Universidade Federal de Viçosa. Disponível: htttp:// www.dpi.ufv.br/~alcione/proginter2001/apostilaservletjsp.pdf. Acesso 18/04/2002.

[05] ABERIUM soluções Disponível: htttp:// www.aberium.com. Acesso 21/09/2002.

[06] Instituto de Computação, Universidade Estadual de Campinas Disponível: http://www.ic.unicamp.br/ Acesso: 29/09/2002.

[07] Instituto Superior Técnico Disponível: http://berlin.inesc.pt/cadeiras Acesso: 29/09/2002.

[08] Macromédia, Produtos Macromédia Disponível: http://www.macromedia.com/br/software Acesso: 03/10/2002.

[09] Programa de Pós-Graduação em Computação Pesquisa Disponível http://www.inf.ufrgs.br/~pasin/ Acesso 09/11/2002.

[10] SUN MicrosoftSystems.TheJ2ee Tutorial. Disponível no site http://www.java.sum.com/j2ee/tutorial/download.html Acesso 09/11/2002.

[11] JBOSS. Disponível no site http://www.Jboss.org Acesso 07/11/2002.

[12] Introdução e motivação: Arquiteturas em n Camadas http://jacques.dsc.ufpb.br/cursos/j2ee/html/intro/intro.htm . Acesso 21/10/2002

[13] Licenciatura em Engª Informática e Computadores, Instituto Super Técnico. http://berlin.inesc.pt/cadeiras/atsi/trabalhos. Acesso 21/10/2002