

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

Leandro Ayala Cosme

**DESENVOLVIMENTO DE UM PLUG-IN DO ECLIPSE PARA GERAÇÃO DE
CÓDIGO JAVASERVER FACES**

Orientador:

Gleydson de A. F. Lima, MSc

NATAL/RN

2008

LEANDRO AYALA COSME

**DESENVOLVIMENTO DE UM PLUG-IN DO ECLIPSE PARA GERAÇÃO DE
CÓDIGO JAVASERVER FACES**

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte como requisito obrigatório para obtenção do grau de Especialista em Desenvolvimento de Sistemas Corporativos.

Orientador:

Gleydson de A. F. Lima, Msc.

NATAL/RN

2008



Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Cosme, Leandro Ayala

Desenvolvimento de um Plug-in do Eclipse para geração de código Javaserwer
Faces / Leandro Ayala Cosme. – Natal: FARN, 2007.
38 f.

Orientador: Gleydson de Azevedo Ferreira Lima (M. Sc.)

Monografia (Especialização em Sistemas Corporativos) – Faculdade
Natalense para o Desenvolvimento do Rio Grande do Norte.

1. Linguagem Java – Monografia. 2. Desenvolvimento de Linguagem de
Programação – Monografia. 3. Integrated Development Environment - IDE –
Monografia. I. Título.

RN/FARN/BC

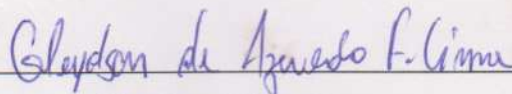
CDU 004.43

RB- 30.714
FARN
BIBLIOTECA
Nº Reg. 0000905
16, 04, 08
004.43
C 868d
2008
ex 01

LEANDRO AYALA COSME

**DESENVOLVIMENTO DE UM PLUG-IN DO ECLIPSE PARA GERAÇÃO DE
CÓDIGO JAVASERVER FACES**

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, aprovada em ____ de ____ de ____, pela banca examinadora composta pelos seguintes membros:



Gleydson Lima, Msc.

Orientador

Membro

Examinador

AGRADECIMENTOS

Primeiramente a DEUS pela oportunidade concedida de mais esta vitória profissional, por concedendo saúde, uma boa família, bons amigos e me ajudando nas horas difíceis.

Ao meu Orientador Prof. Msc. Gleydson de A. F. Lima, pelo incentivo e responsabilidade para me auxiliar em todos os momentos desta Monografia.

E, Finalmente, a todos os que cooperaram de forma indireta neste trabalho, orientando e auxiliando para que obtivesse um bom resultado no final desta Monografia.

A Deus por ter me dado condições para
terminar mais esse projeto.

A minha Família que me apoiou nessa jornada,
como em todas as fases da minha vida.

AGRADECIMENTOS

Primeiramente a DEUS pela oportunidade concedida de mais essa vitória profissional, me concedendo saúde, uma boa família, bons amigos e me ajudando nas horas difíceis.

Ao meu Orientador Prof. Msc. Gleydson de A. F. Lima, pelo incentivo e disponibilidade para me auxiliar em todos os momentos desta Monografia.

E, Finalmente, a todos os que cooperaram de forma indireta neste trabalho, orientando e auxiliando para que obtivesse um bom resultado no final desta Monografia.

RESUMO

O desenvolvimento com a tecnologia Java conta com varias opções de softwares para ambiente de desenvolvimento integrado (IDE - *integrated development environment*). Há algum tempo uma das ferramentas mais populares é o projeto open source Eclipse. Ele é uma plataforma extensível através de *plug-ins*, todas as suas ferramentas, incluindo também as partes básicas, são *plug-ins*. Por ser um ambiente de desenvolvimento aberto, o seu uso efetivo em desenvolvimento corporativo requer agregar *plug-ins* para tecnologias Java especializada, principalmente para a plataforma JEE (*Java Enterprise Edition*). O objetivo deste trabalho é o desenvolvimento de um *plug-in* gerador de código JSF, mostrando todos os conceitos relacionados à plataforma da IDE Eclipse, sua principal ferramenta para desenvolvimento de *plug-ins*, o PDE (*Plugin Development Enviroment*), estrutura e ciclo de vida de um *plug-in* e mostrar as dificuldades encontradas para desenvolvimento da ferramenta. A geração automática de código JSF (*JavaServer Faces*) tem como finalidade facilitar a produtividade no desenvolvimento de sistemas WEB utilizando-se dos recursos que a plataforma Java oferece. O *plug-in* gerador de código JSF e sua integração ao Eclipse disponibiliza mais um recurso que poderá ser utilizado para aumentar a produtividade no desenvolvimento de sistemas corporativos para ambiente WEB.

Palavras-chave: Java, *Integrated Development Environment*, *Open source*, *Plug-ins*, *Java Enterprise Edition*, *JavaServer Faces*, *Plugin Development Enviroment*, WEB, CRUD, Geração de código.

ABSTRACT

The development with Java technology has several options of software for integrated development environment (IDE). For some time one of the most popular tools is the open source Eclipse project. It is extensible platform through plug-ins, all its tools, including the parties basic, are plug-ins. As an open environment development, its effective use in corporate development requires adding plug-ins for specialist Java technology, mainly to the platform JEE (Java Enterprise Edition). The goal of this project is the development of a plug-in code generator for JSF, showing all the concepts related to the platform of the Eclipse IDE, its main tool for the development of plug-ins, the PDE (Plugin Development Environment), structure and cycle of life of a plug-in and show the difficulties encountered in developing the tool. The automatic generation of JSF (JavaServer Faces) code aims to facilitate the productivity in systems development using Web resource through Java platform. The plug-in code generator for JSF and its integration to Eclipse provides one more resource that could be used to increase productivity in the development of corporate systems for WEB environmental.

1. INTRODUÇÃO	16
1.1. O Eclipse	18
1.2. Estrutura de um plug-in	19
1.3. Ciclo de Vida de um plug-in	20
1.4. Exemplo de um simples Plug-in	21
2. O JAVASERVER FACES	24
2.1. Argumento	24
2.2. MVC (Model-View-Controller)	24
2.3. Padrão MVC segundo JSF	25
2.4. Ciclo de vida de uma aplicação JSF	25
2.5. Características e Vantagens do JSF	26
3. ENVOLVIMENTO DO PLUG-IN PARA O ECLIPSE	27
3.1. REFLECTION API	27
3.2. CLASSES	28
3.2.1. Imports: java	29
3.2.2. Parte Gerador java	29
3.2.3. Parte Gerador JSF java	32
3.3. IMPLEMENTAÇÃO	34
4. EMPACOTAMENTO	36
5. CONCLUSÃO	37
6. REFERÊNCIAS	38

SUMÁRIO

1 INTRODUÇÃO.....	8
2 REFERENCIAL TEÓRICO.....	10
2.1 A LINGUAGEM JAVA.....	10
2.1.1 Definição.....	10
2.1.2 Características.....	10
2.1.3 Plataformas.....	12
2.2 AMBIENTES DE DESENVOLVIMENTO INTEGRADO – IDES.....	14
2.2.1 Eclipse.....	14
2.2.2 NetBeans.....	14
2.2.3 JDeveloper (www.oracle.com/technology/products/jdev).....	14
2.2.5 IntelliJ IDEA (www.jetbrains.com/idea/).....	15
2.3 ECLIPSE IDE.....	15
2.3.1 Histórico.....	16
2.3.2 Arquitetura.....	16
2.4 PLUG-INS.....	18
2.4.1 Estrutura de um plug-in.....	19
2.4.2 Ciclo de Vida de um plug-in.....	20
2.4.4 Exemplo de um simples <i>Plug-in</i>	21
2.5 JSF (<i>JAVASERVER FACES</i>).....	24
2.5.1 Surgimento.....	24
2.5.2 MVC (Model-View-Controller).....	24
2.5.3 Padrão MVC segundo JSF.....	25
2.5.4 Ciclo de vida de uma aplicação JSF.....	25
2.5.5 Características e Vantagens do JSF.....	26
3 DESENVOLVIMENTO DO PLUG-IN PARA O ECLIPSE.....	27
3.1 REFLECTION API.....	27
3.2 CLASSES.....	28
3.2.1 Generator.java.....	29
3.2.2 FormGenerator.java.....	29
3.2.2 ListGeneratorJSF.java.....	32
3.4 IMPLEMENTAÇÃO.....	34
3.5 EMPACOTAMENTO.....	36
4. CONCLUSÃO.....	37
REFERÊNCIAS.....	38

1 INTRODUÇÃO

A evolução das linguagens de programação, a busca pelo aumento de produtividade no processo de desenvolvimento de software e pela melhoria contínua de qualidade, levou o largo uso de Ferramentas de Desenvolvimento. Especificamente no desenvolvimento Java as Ferramentas de Desenvolvimento Integrado (IDE do inglês *Integrated Development Enviroment*.) acompanharam essa evolução. A Sun, ao lançar a linguagem Java, pôs à disposição gratuitamente o pacote *Java Developer's Kit* (JDK), a primeira ferramenta que possibilitou o desenvolvimento da linguagem. Com o crescimento da linguagem Java no mercado, várias empresas, como a IBM, investiram na criação de IDEs. Atualmente existem diversas ferramentas no mercado, e uma das mais populares, há algum tempo, tem sido o projeto *open source* Eclipse (www.eclipse.org).

Por definição o Eclipse é uma estrutura de *software* para ambientes de desenvolvimento *Rich Client*. O *Rich Client* oferece um ambiente de execução no cliente para interfaces ricas com o usuário, assim como os *scripts* relacionados a elas.

Eclipse é um Software livre, ou seja, seu código fonte está disponível para que qualquer um possa visualizá-lo, modificá-lo e distribuí-lo (<http://www.eclipse.org/legal/epl-v10.html>). Mantido por uma fundação, uma comunidade de grandes empresas (incluindo, IBM, HP, Oracle e SAP), pequenas empresas e programadores independentes comprometidos em promover a estrutura e estender sua funcionalidade, o Eclipse ganhou popularidade e força por vários motivos. Um dos principais, que fazem com que se torne uma ferramenta poderosa, é possuir uma arquitetura de *software* aberta e extensível, permitindo que *plug-ins* sejam criados e facilmente integrados.

A criação de *plug-ins* visa aumentar o seu conjunto padrão de funcionalidades. Existe uma grande variedade de *plug-ins* disponíveis, entre gratuitos e comerciais, para adicionar e ampliar toda gama de características e recursos. A Tecnologia Java possibilita o desenvolvimento de aplicações nas diversas plataformas, como a WEB, os dispositivos móveis e ambientes distribuídos. *Plug-ins* bastante populares como o *Web Tools Platform* (<http://www.eclipse.org/webtools>), que desenvolve ferramentas para o desenvolvimento J2EE Web no Eclipse, é um exemplo para auxiliar no desenvolvimento dessas diversas áreas.

Este Projeto tem como objetivo mostrar todos os conceitos relacionados à criação de *plug-ins*: sua estrutura, funcionamento e vantagens da utilização na ferramenta Eclipse IDE.

Os Objetivos Específicos do trabalho são:

- a) Mostrar as principais IDEs;
- b) Falar sobre o Eclipse IDE;
- c) Mostrar o ambiente de desenvolvimento de *plug-in*;
- d) Criação e testes de um *plug-in* gerador de código view.

Java é uma linguagem de programação orientada a objetos desenvolvida na empresa de Sun Microsystems com o objetivo de ser uma linguagem simples e segura. Desde seu lançamento, em maio de 1995, a plataforma Java foi adotada em uma variedade de, por qualquer outra linguagem de programação na história da computação (LACROIX, 2006). Em 2003 Java atingiu a marca de 4 milhões de desenvolvedores em todo mundo (www.javafree.com). Java continua crescendo e hoje é uma linguagem no mercado de desenvolvimento de *software*. Java tornou-se popular pelo seu uso em *aplicativos* e hoje possui seu ambiente de execução presente em *web browsers*, *cellular phones*, *PDAs*, *cellular phones*, *palmtops*, *cartões inteligentes*, entre outros.

Atualmente, Java é uma das linguagens mais utilizadas em diversas áreas em desenvolvimento de *software*, tais como:

- *Web e Data Distributed*;
- *Aplicação Científica*;
- *Aplicação e Desenvolvimento*;
- *Aplicativos Móveis*;
- *Desenvolvimento de Sistemas Corporativos*.

2.1 Características

O Java é orientado a objetos, isto significa que oferece muitas vantagens sobre as linguagens procedurais tradicionais. Como os objetos encapsulam dados e funções relacionadas em unidades coesas, é fácil localizar dependências de dados, isolar efeitos de alterações e realizar outras atividades de manutenção. Talvez o mais importante das linguagens Orientadas a Objeto é a reutilização de código. Não existem variáveis globais ou funções independentes. Toda variável ou função pertence a uma classe ou objeto e sempre que precisar ser invocada através deste meio.

O Java é distribuído, ou seja, há distribuição de informações para compartilhamento e trabalho conjunto, com a distribuição de carga de trabalho de processamento, é uma característica essencial dos aplicativos cliente/servidor. Felizmente para os programadores

2 REFERENCIAL TEÓRICO

2.1 A LINGUAGEM JAVA

2.1.1 Definição

Java é uma linguagem de programação orientada a objeto desenvolvida na metade da década de 90 pela *Sun Microsystems* com o objetivo de ser uma linguagem simples e eficiente. Desde seu lançamento, em maio de 1995, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação (LOZANO, 2006). Em 2003 Java atingiu a marca de 4 milhões de desenvolvedores em todo mundo (www.javafree.com). Java continuou crescendo e hoje é uma referência no mercado de desenvolvimento de *software*. Java tornou-se popular pelo seu uso na Internet e hoje possui seu ambiente de execução presente em *web browsers*, *mainframes*, SOs, celulares, *palmtops*, cartões inteligentes, entre outros.

Atualmente, Java é uma das linguagens mais utilizadas nas diversas áreas em crescimento na computação, tais como:

- Banco de Dados Distribuídos;
- Comercio Eletrônico;
- Jogos e Entretenimento;
- Dispositivos Móveis;
- Desenvolvimento de Sistemas Corporativos.

2.1.2 Características

O Java é orientado a objetos, isto significa que oferece muitas vantagens sobre as linguagens procedurais tradicionais. Como os objetos encapsulam dados e funções relacionados em unidades coesas, é fácil localizar dependências de dados, isolar efeitos de alterações e realizar outras atividades de manutenção. Talvez o mais importante das linguagens Orientadas a Objeto é a reutilização de código. Não existem variáveis globais ou funções independentes. Toda variável ou método pertence a uma classe ou objeto e conseqüentemente só pode ser invocada através deste meio.

O Java é distribuído, ou seja, há distribuição de informações para compartilhamento e trabalho conjunto, com a distribuição de carga de trabalho do processamento, é uma característica essencial dos aplicativos cliente/servidor. Felizmente para os programadores

Java, há uma biblioteca de procedimentos TCP/IP incluída na API. Isso facilita aos programadores o acesso remoto às informações, usando protocolos como HTTP e FTP.

O Java é compilado e independente de plataforma: Um programa escrito em Java precisa ser compilado antes de ser executado. O compilador traduz o código-fonte e gera arquivos objeto chamados arquivos de classe. Cada programa Java consiste da implementação de uma única classe. Depois de compilado, pode ser executado em qualquer plataforma onde exista um sistema de tempo de execução Java (*Runtime*). A *Java Virtual Machine* (JVM) é um programa que carrega e executa os aplicativos Java, convertendo os *bytecodes* em código executável de máquina. A JVM é responsável pelo gerenciamento dos aplicativos, à medida que são executados (<http://pt.wikipedia.org/wiki/JVM>).

O Java é multitarefa: Os objetos binários de códigos de *bytes* do Java são formados por seqüências de execução múltiplas e simultâneas. Essas seqüências são conhecidas como contextos de execução ou processos leves (*Threads*).

O Java é dinâmico: A linguagem Java foi projetada para se adaptar a um ambiente dinâmico, em constante evolução. Possui uma representação de tempo de execução que permite que o programa saiba a classe a que pertence um objeto, na hora em que o recebe, durante a execução. Isso permite a inclusão dinâmica de classes que podem estar em qualquer lugar da Internet. Java também suporta a integração com métodos nativos de outras linguagens (Santos, 2005).

O Java é robusto: Quanto mais robusto um aplicativo, mais confiável ele será. Isso é desejável tanto para os desenvolvedores de *software* quanto aos consumidores. A maioria das linguagens OO, como o C++ e Java, possuem tipos bastante fortes. Isso significa que a maior parte da verificação de tipos de dados é realizada em tempo de compilação, e não em tempo de execução. Isso evita muitos erros e condições aleatórias nos aplicativos (Santos, 2005).

O Java é seguro: Como o Java foi criado para ambientes de rede, os recursos de segurança receberam muita atenção. Por exemplo, se você executar um binário transferido por *download* da rede, o mesmo poderá estar infectado por vírus. Os aplicativos Java apresentam garantia de resistência contra vírus e de que não são infectados por vírus, pois não são capazes de acessar *heaps*, *stacks* ou memória do sistema. No Java, a autenticação do usuário é implementada com um método de chave pública de criptografia. Isso impede de maneira eficaz que *hakers* e *crakers* examinem informações protegidas como nomes e senhas de contas (Santos, 2005).

O Java é simples: Um dos principais objetivos do projeto do Java foi criar uma linguagem o mais próxima possível do C++, para garantir sua rápida aceitação no mundo do

desenvolvimento Orientado a Objeto. Outro objetivo do seu projeto foi eliminar os recursos obscuros e danosos do C++, que fugiam à compreensão e aumentavam a confusão que poderia ocorrer durante as fases de desenvolvimento, implementação e manutenção do *software*. O Java é simples porque é pequeno. O interpretador básico do Java ocupa aproximadamente 40k de RAM, excluindo-se o suporte a multitarefa e as bibliotecas padrão, que ocupam outros 175k. Mesmo a memória combinada de todos esses elementos é insignificante, se comparada a outras linguagens e ambientes de programação (Santos, 2005).

O Java oferece alto desempenho: Há muitas situações em que a interpretação de objetos de códigos de *bytes* proporciona desempenho aceitável. Mas outras circunstâncias exigem desempenhos mais altos. O Java concilia tudo isso oferecendo a tradução dos códigos de *bytes* para o código de máquina nativo em tempo de execução. O alto desempenho permite a implementação de seus aplicativos WEB em Java, na forma de programas pequenos e velozes, que podem ampliar significativamente os recursos tanto do cliente quanto do servidor (Santos, 2005).

2.1.3.3 JME (Java Micro Edition)

2.1.3 Plataformas

A tecnologia Java possui diferentes plataformas que atendem as diversas áreas do desenvolvimento de software.

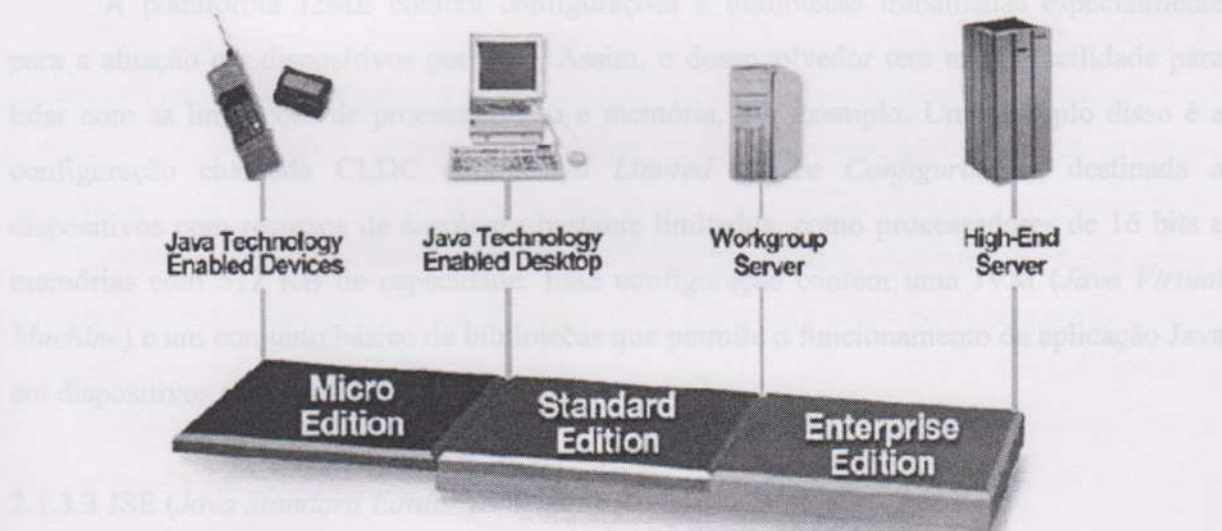


Figura 1 – Plataformas Java (www.javaman.com.br, 2007)

2.1.3.1 JEE (Java Enterprise Edition)

A Plataforma JEE tem como principal objetivo atender os requisitos de um ambiente corporativo. Os ambientes corporativos exigem robustez, alta disponibilidade, segurança,

confiabilidade e escalabilidade. A *Sun Microsystems* ao lançar a JEE criou uma plataforma que suportasse os rígidos requisitos dos ambientes corporativos atuais, através de uma arquitetura de aplicação multicamada, com enfoque no lado servidor e baseado em componentes. Na arquitetura multicamadas o *software* é construído através de componentes distribuídos em várias camadas, normalmente incluindo uma Camada Cliente, que é responsável pela interação com o usuário e é composta por componentes que interagem com as regras de negócio do ambiente, uma ou mais Camadas Intermediárias, também chamadas de camadas de negócio, e uma Camada de Dados que é responsável pelo gerenciamento dos dados e a persistência dos mesmos.

Tecnologias como a *Enterprise JavaBeans* (EJB), *JavaMail* API, *Java Server Faces* (JSF), JSP, *Servlets* e JDBC fazem parte da plataforma JEE dando suporte nas diversas camadas de desenvolvimento, tornando assim a plataforma mais completa da Tecnologia Java (JADI, 1999).

2.1.3.2 JME (*Java Micro Edition*)

O J2ME (*Java Micro Edition*) é o ambiente de desenvolvimento para dispositivos móveis ou portáteis, como telefones celulares e *palmtops*.

A plataforma J2ME contém configurações e bibliotecas trabalhadas especialmente para a atuação em dispositivos portáteis. Assim, o desenvolvedor tem maior facilidade para lidar com as limitações de processamento e memória, por exemplo. Um exemplo disso é a configuração chamada CLDC (*Connected Limited Device Configuration*), destinada a dispositivos com recursos de *hardware* bastante limitados, como processadores de 16 bits e memórias com 512 KB de capacidade. Essa configuração contém uma JVM (*Java Virtual Machine*) e um conjunto básico de bibliotecas que permite o funcionamento da aplicação Java em dispositivos com tais características.

2.1.3.3 JSE (*Java Standard Edition*)

O JSE (*Java Standard Edition*) é o ambiente de desenvolvimento mais utilizado. Isso porque seu uso é voltado a PCs e servidores, onde há bem mais necessidade de aplicações. Além disso, pode-se dizer que essa é a plataforma principal, já que, de uma forma ou de outra, o J2EE e o J2ME tem sua base aqui. Pode-se dizer que as outras plataformas são versões adaptadas/aprimoradas do J2SE para as aplicações a que se propõem.

2.2 AMBIENTES DE DESENVOLVIMENTO INTEGRADO – IDES

A linguagem Java tornou-se uma das linguagens mais utilizadas hoje no mundo por possuir uma área de abrangência de desenvolvimento em todas as áreas da tecnologia. Com isso várias empresas se interessaram em construir ferramentas de desenvolvimento, que vão das básicas as mais completas, das gratuitas as pagas (JADI, 1999). Aqui serão apresentadas algumas das ferramentas mais utilizadas hoje para o desenvolvimento Java.

2.2.1 Eclipse

Considerada como a ferramenta mais popular e uma das mais completas no mercado por ser livre, fazer parte de um projeto bastante sério, ativo e organizado, a IDE Eclipse é uma ferramenta bastante robusta criada para o desenvolvimento de aplicações complexas. Composto de três sub-projetos: Plataforma, JDT - Ferramentas de Desenvolvimento Java, Ambiente de Desenvolvimento de Plug-ins (PDE). Eclipse é uma espécie de plataforma de ferramenta universal - um IDE aberto extensível para tudo e nada em particular.

“a possibilidade de customização é sem dúvida, um dos pontos que devem ser considerados durante a escolha de uma ferramenta de desenvolvimento.” (ROCHA, 2006, p. 40)

2.2.2 NetBeans

NetBeans é um projeto *open-source*, consistindo em um IDE profissional cheio de recursos e uma plataforma sobre a qual você pode desenvolver qualquer tipo de aplicação. O NetBeans foi adquirido e colocado como open source pela Sun Microsystems, principal incentivadora do projeto e que o usa como base para o *Sun ONE Studio* (www.netbeans.org).

O IDE é modular, e existe uma enorme variedade de extensões gratuitas e comerciais para ele adicionando suporte a diversas tecnologias. Editor de código com destaque de sintaxe avançado; suporte a tecnologias JSP, XML, RMI, CORBA, JINI, JDBC e *Servlet*; suporte a Ant, CVS e outros sistemas de controle de versão; suporte conectável para compiladores, depuradores e serviços de execução; assistentes e ferramentas de geração de código e gerenciamento (netbeans.org).

2.2.3 JDeveloper (www.oracle.com/technology/products/jdev)

JDeveloper é um ambiente de desenvolvimento integrado gratuito (desde 2005) da Oracle que oferece funcionalidades para o desenvolvimento em Java, XML, SQL e PL/SQL,

HTML, JavaScript, BPEL e PHP. O JDeveloper cobre todo o ciclo de desenvolvimento desde a análise até a codificação, a manutenção, a otimização e a implantação (Wikipedia, 2007).

2.2.4 JBuilder (www.codegear.com/br/products/jbuilder)

O JBuilder é uma ferramenta para desenvolvimento Java, diferenciando-se pelo seu alto nível de maturidade (versão 12), robustez (uma das maiores aplicações Java do mundo) e um portfólio impressionante de usuários (tais como SAP AG, com 1.500 desenvolvedores).

Estrategicamente, o JBuilder busca a produtividade não apenas individual, mas da equipe como um todo. Prova disso é seu foco em garantia de performance, segurança de código-fonte, forte integração com ferramentas de Requisito e Gestão de Configuração e Mudança e auditoria de código, inclusive ferramentas de iniciativa *Open Source* com total integração na IDE, permite o uso de múltiplos Servidores de Aplicações e Sistemas Operacionais (Wikipedia, 2007).

2.2.5 IntelliJ IDEA (www.jetbrains.com/idea/)

O IntelliJ IDEA é um IDE comercial Java completo com alto nível de usabilidade e produtividade, através de recursos avançados sensacionais para edição e *refactoring*. Para Windows, Linux, MacOS X, Unix. Recursos como auto-completar, para Navegação, Busca & Substituição, Estilo de Codificação, JavaDoc, Controle de Versão (CVS, *SourceSafe*), Execução, Debug, Integração (Ant, JUnit etc.) entre outros (Wikipedia, 2007).

2.3 ECLIPSE IDE

O Eclipse é uma ferramenta extensível, aberta e portátil. Através de *plug-ins* o Eclipse torna-se uma IDE com vários papéis de desenvolvimento: responsável por testes, programador de componentes, integradora, *web designer* entre outras, servindo de suporte ao desenvolvimento de novas ferramentas. O próprio Eclipse é desenvolvido utilizando o Eclipse.

“Grande parte do sucesso do IDE Eclipse deve-se a sua arquitetura extensível (“uma IDE aberta e extensível para tudo e nada específico”)”. (ROCHA, 2006, p. 40)

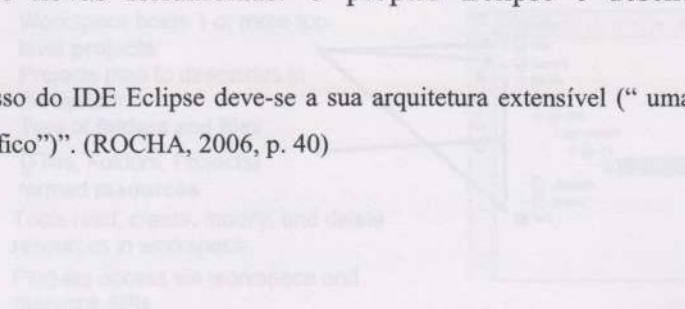


Figura 3 – Componentes Workspace (Java Magazine, 2006)

2.3.1 Histórico

A IBM possuía uma linha *VisualAge* que precisava de um reposicionamento comercial e uma refatoração da sua arquitetura. Essas necessidades deram origem ao Eclipse e sua estrutura de *plug-ins*, tornando-se a base da mais nova linha de IDEs da IBM.

Em 2005 o projeto Eclipse tornou-se independente da IBM quando a *Eclipse Foundation* tornou-se uma entidade jurídica, presidida por um executivo da Oracle. Este fato fez com que grandes empresas, como a SAP, a Borland, fizessem parte da fundação.

2.3.2 Arquitetura

O Eclipse possui uma arquitetura aberta, e expansível através do desenvolvimento de *plug-ins*. Todas as suas ferramentas são *plug-ins* exceto a sua plataforma *runtime* como mostra a figura 1.

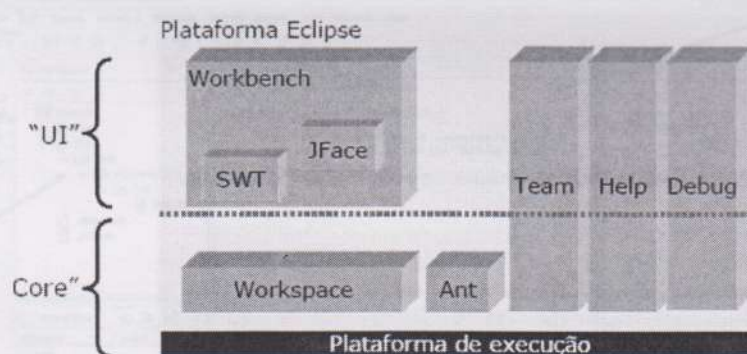


Figura 2 – Arquitetura do Eclipse (Java Magazine, 2006)

2.3.2.1 Workspace

Componente responsável pelo gerenciamento dos recursos (projetos, pastas e arquivos) ligados aos projetos dos usuários. O workspace mantém as informações dos projetos em um diretório comum, e cada projeto criado é mapeado para um subdiretório e possibilita trabalhar com vários projetos em paralelo.

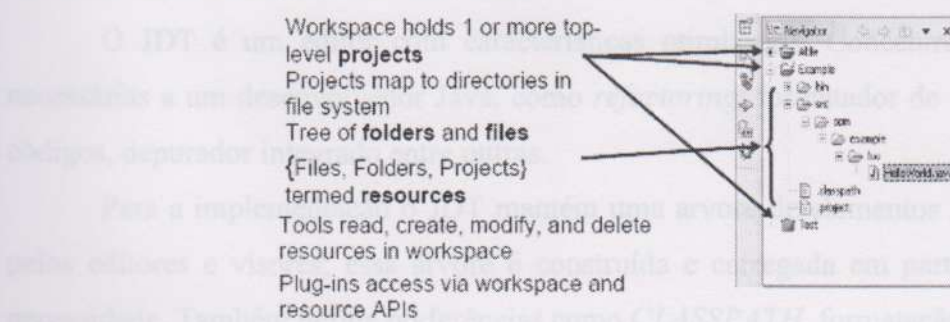


Figura 3 – Componente Workspace (Java Magazine, 2006)

2.3.2.2 Workbench

Componente responsável por fornecer uma estrutura para interação de ferramentas com os usuários. O *Workbench* é composto por editores que acrescentam ações aos menus e barras de ferramentas, visores que fornecem diversas informações ao usuário sobre os recursos que estão sendo editados. Todos esses componentes podendo ser acrescentados através de *plug-ins*. O SWT (*Standart Widget Toolkit*) oferece uma API gráfica portátil e integrada com o sistema operacional, o SWT utiliza JNI (*Java Native Interface*) para acessar as bibliotecas gráficas nativas do SO e renderizar os componentes gráficos utilizados na IDE (botões, imagens, *labels* etc.). O JFace (<http://wiki.eclipse.org/JFace>) é uma extensão de “alto nível” do SWT que oferece mecanismos para construção de diálogos, *wizards*, *actions* etc. O JFace Implementa a arquitetura MVC (*Model View Controller*).

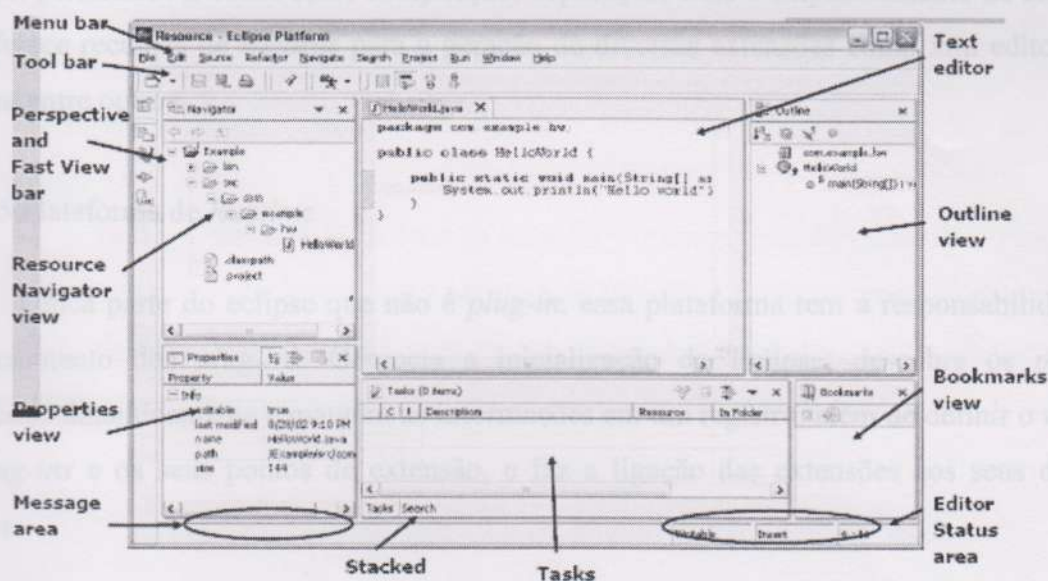


Figura 4 – Componente Workbench (Rocha, 2006)

2.3.2.3 JDT (*Java Developer Tooling*)

O JDT é um editor com características otimizadas. Concentra as funcionalidades necessárias a um desenvolvedor Java, como *refactoring*, formatador de código, *templates* de códigos, depurador integrado entre outras.

Para a implementação o JDT mantém uma árvore de elementos Java para navegação pelos editores e visores; essa árvore é construída e carregada em partes de acordo com a necessidade. Também define preferências como *CLASSPATH*, formatação do editor Java.

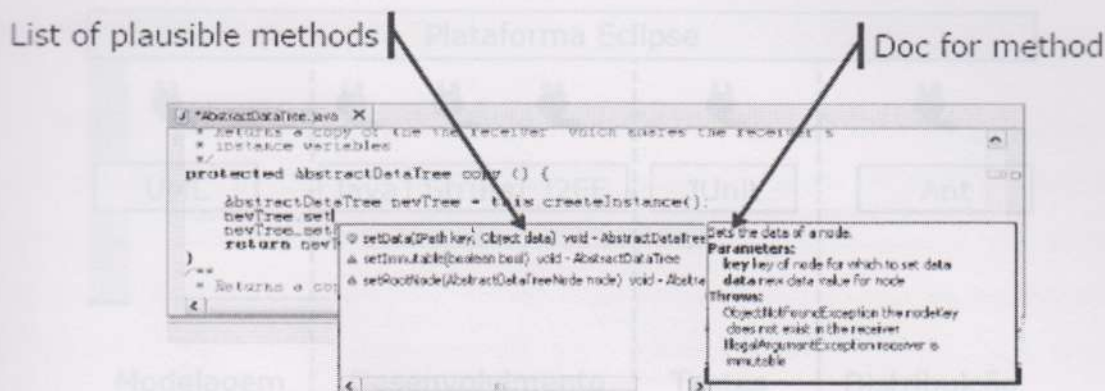


Figura 5 - Auto-Completar de código do JDT (Rocha, 2006)

2.3.2.4 PDE (Plug-in Development Environment)

O PDE é um ambiente de desenvolvimento de *plug-ins* dentro do workbench do Eclipse, permitindo a construção, compilação, depuração, teste e empacotamento de um *plug-in*. Oferece recursos de *wizards* para a geração de diversas extensões como xml editor, *help content* entre outros.

2.3.2.5 Plataforma de Runtime

Única parte do eclipse que não é *plug-in*, essa plataforma tem a responsabilidade de gerenciamento dos *plug-ins*. Gerencia a inicialização do Eclipse; descobre os *plug-ins* instalados dinamicamente e mantêm as informações em um registro, além de definir o modelo de *plug-ins* e os seus pontos de extensão, e faz a ligação das extensões aos seus devidos pontos.

2.4 PLUG-INS

Um *plug-in* é o menor bloco de construção do Eclipse, ou seja, é unidade mínima de funcionalidade. O Eclipse é composto por diversos *plug-ins* a ele agregados.

Uma aplicação complexa pode ser composta por vários *plug-ins* que também podem contribuir, através de pontos de extensão, com outras aplicações.



Figura 6 – Cenário de utilização de plug-ins (Java Magazine, 2006)

2.4.1 Estrutura de um plug-in

Um *plug-in* possui uma estrutura de diretórios que fica localizado no diretório principal do Eclipse; cada *plug-in* pertencendo ao seu próprio diretório de *plug-ins*.

O arquivo *plug-in.xml* contém as configurações de identificação do *plug-in*, pontos de extensões e relações com outros *plug-ins*.

O diretório Possui também as classes *.jar, arquivos de informação do *plug-in* (*about.html*) e imagens.

A partir da versão 3.0 do Eclipse os *plug-ins* passaram a ser empacotados em arquivos JAR (*Java Application Archive*), mas a estrutura de diretórios continua sendo válida.

Um *plug-in* pode contribuir com outros *plug-ins* através dos pontos de extensões e utilizar recursos de outros *plug-ins*.

```

C:\ECLIPSE
|
+---features
+---plugins
|   +---org.eclipse.ant.core_2.1.0
|   |   |   .options
|   |   |   about.html
|   |   |   antsupport.jar
|   |   |   plugin.properties
|   |   |   plugin.xml
|   |   +---lib
|   +---org.eclipse.compare_2.1.0
|   ...
+---workspace
  
```

Figura 7 – Estrutura de um plug-in (Rocha, 2006)

2.4.2 Ciclo de Vida de um plug-in

Durante sua inicialização, o Eclipse lê todos os arquivos de configurações (*plugin.xml*) implantados no diretório. As informações relativas a todos os *plug-ins* encontrados são armazenadas em um registro de *plug-ins* que fica disponível para todos os outros *plug-ins* em tempo de execução. Os *plug-ins* e suas respectivas classes só são carregados quando efetivamente são solicitados; isso permite que o carregamento do Eclipse seja constante independente do número de *plug-ins* instalados.

A partir da versão 3.0 do Eclipse a descoberta e ativação de *plug-ins* passaram a ser comandadas por um mecanismo baseado no *Open Services Gateway initiative* (OSGi). O OSGi tem como objetivo padronizar a maneira como os componentes devem ser registrados em uma aplicação, permitindo que sejam carregados ou descarregados dinamicamente. O uso dessa tecnologia melhorou a portabilidade da ferramenta entre diversos sistemas operacionais e fez com que os *plug-ins* passassem a ser implementados segundo um novo modelo (*OSGi bundles*).

2.4.3 Pontos de Extensões

Plug-ins são feitos para contribuir com outros *plug-ins*. Essas contribuições são feitas através dos pontos de extensão. Um *plug-in* pode estender os pontos de extensão de outros *plug-ins* como podem definir seus próprios pontos de extensão.

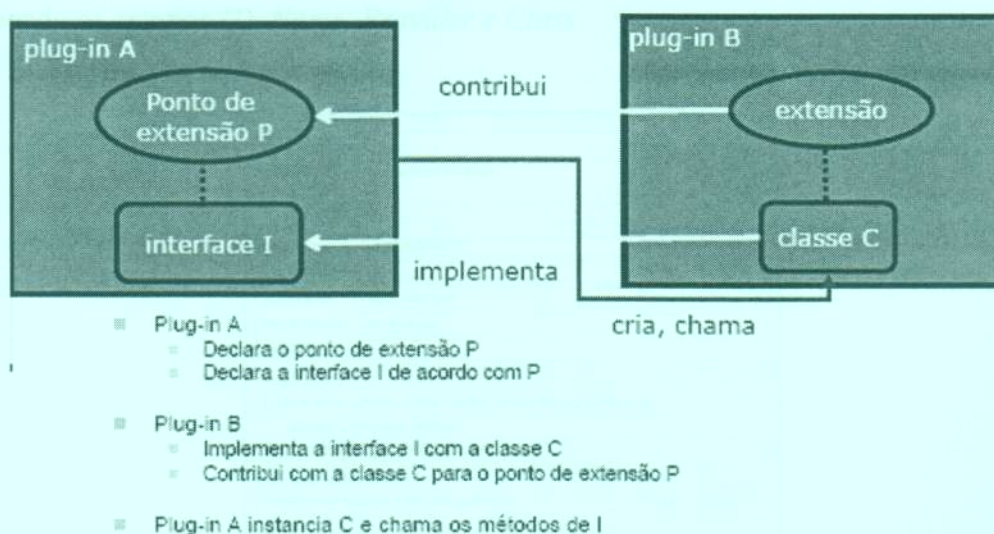


Figura 8 – Exemplo de Contribuição de Plug-ins (Java Magazine, 2006)

2.4.4 Exemplo de um simples *Plug-in*

Como visto anteriormente, o Eclipse possui uma ferramenta de auxílio na criação de *plug-ins*. O PDE auxilia na criação com editor gráfico para o *plug-in.xml*, assistente para geração de código do *plug-in*, e estrutura para a execução.

Passo 1: Criação de um *plug-in project*. Nessa tela devem ser informadas as características do projeto.

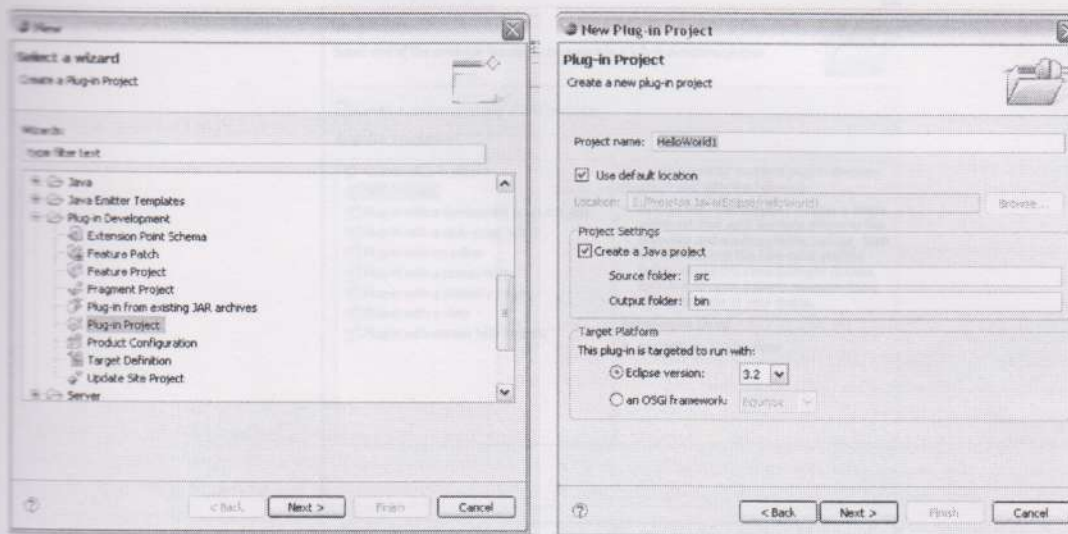


Figura 9 – Criação do Plug-in (Dados da Pesquisa, 2007)

Passo 2: Definindo dados básicos do plug-in, como identificador (*plug-in ID*), nome (*plug-in Name*) e classe de gerenciamento (*Class Name*). Devemos deixar as opções *default*, preenchendo os campos *ID*, *Name*, *Provider* e *Class*.



Figura 10 – Dados do plugin (Dados da Pesquisa, 2007)

Passo 3: Nessa etapa o PDE permite a escolha de diversos *templates* de *plug-ins* disponíveis. A utilização de um *template* facilita o desenvolvimento, pois o PDE gera automaticamente o “esqueleto” de todas as classes envolvidas e configura adequadamente os pontos de extensão necessários. Dependendo da necessidade, podem ser feitas alterações necessárias no código para ajustar o comportamento do *plug-in*. Nesse caso, vamos selecionar o *template* “hello, world”.

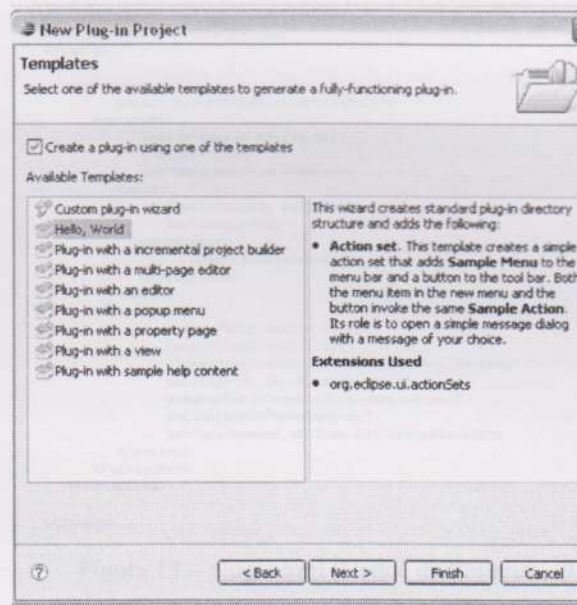


Figura 11 – Escolhendo um template para o plugin (Dados da Pesquisa, 2007)

Passo 4: será criado um menu item no menu principal do Eclipse. Nesse passo definimos o nome da classe que executa o *plug-in*. Nesse exemplo será exibida uma *Message Box* com o texto definido.

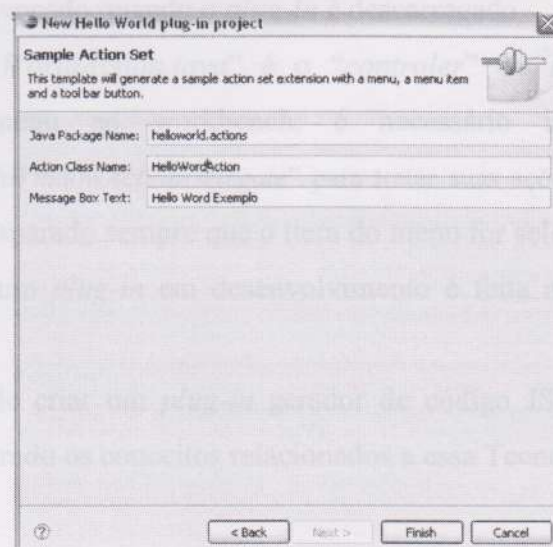


Figura 12 – Definição da classe Action (Dados da Pesquisa, 2007)

Após esses passos o *plug-in* já foi criado. Foram geradas pelo PDE as classes necessárias para a execução do *plug-in* e as extensões necessárias. No arquivo `plugin.xml` podemos identificar que ele estende do *plug-in* "org.eclipse.ui.actionSets", adiciona um menu ao workbench do Eclipse e identifica a classe que será chamada quando o menu for acionado como mostra a figura abaixo.

```
<?xml version="1.0" encoding="UTF-8"?>
<eclipse version="3.2"?>
<plugin>

  <extension
    point="org.eclipse.ui.actionSets">
    <actionSet
      label="Sample Action Set"
      visible="true"
      id="HelloWorld1.actionSet">
      <menu
        label="Sample <sampleMenu"
        id="sampleMenu">
        <separator
          name="sampleGroup">
        </separator>
      </menu>
      <action
        label="sample Sample Action"
        icon="icons/sample.gif"
        class="helloworld1.actions.HelloWorldAction"
        tooltip="Hello, Eclipse world"
        menubarPath="sampleMenu/sampleGroup"
        toolbarPath="sampleGroup"
        id="helloworld1.actions.HelloWorldAction">
      </action>
    </actionSet>
  </extension>
</plugin>
```

Figura 13 – `plugin.xml` (Dados da Pesquisa, 2007)

O PDE também gera automaticamente uma classe responsável por gerenciar o *plug-in*, nesse caso a classe "HWPlugin.java". como estamos implementando um *plug-in* visual, ela estende a classe "org.eclipse.ui.plugin.AbstractUIPlugin". Nessa classe dois métodos merecem destaques: `start()`, que é invocado quando algum código do *plug-in* é chamado pela primeira vez; e `stop()`, invocado quando o *plug-in* é descarregado.

A classe "HelloWorldAction.java" é o "controler" do *plug-in*. Como o *plug-in* adiciona um novo menu ao workbench, é necessário implementar a interface "org.eclipse.ui.IWorkbenchWindowActionDelegate" para tratar suas ações. O principal método da classe é o `run()`, que é disparado sempre que o item do menu for selecionado.

A Execução de um *plug-in* em desenvolvimento é feita abrindo outra instancia do Eclipse.

Como objetivo de criar um *plug-in* gerador de código JSF (JavaServer Faces), na próxima seção será mostrado os conceitos relacionados a essa Tecnologia.

2.5 JSF (JAVASERVER FACES)

O JSF é uma Tecnologia que incorpora características de um *framework* MVC para WEB e de um modelo de *interfaces* gráficas baseado em eventos.

2.5.1 Surgimento

As tecnologias voltadas para o desenvolvimento WEB têm mudado constantemente. Inicialmente os sites só possuíam conteúdos estáticos, evoluindo para sites mais dinâmicos ao longo do tempo. A primeira tecnologia voltada para construção de páginas dinâmicas foi a CGI (*Common Gateway Interface*). Os programas CGI podiam ser escritos em qualquer linguagem de programação, mais possuíam problemas de escalabilidade e portabilidade.

Em seguida vieram os *servlets*. Similarmente às CGIs, *servlets* são pequenos programas feitos em Java que encapsulam alguma funcionalidade inerente à sua aplicação WEB. Diferentemente das CGIs, *servlets* são objetos Java que não precisam ser executados em outro processo: o processamento é executado dentro de uma *thread* do processo do servidor. No entanto, eles ainda não resolvem o problema da separação das regras de negócio da visualização, dificultando a manutenção.

Em seguida das CGIs surgiram as páginas JSP (*JavaServer Pages*). Possuem conteúdos reutilizáveis e são facilmente codificados, porém, ainda não resolviam o problema de manutenção das aplicações.

O problema de manutenção foi resolvido com a utilização dos *designs patterns* no desenvolvimento de aplicações WEB. No desenvolvimento de aplicações WEB utilizando Java, o *design pattern* utilizado em MVC (*Model-View-Controller*).

2.5.2 MVC (Model-View-Controller)

A idéia do padrão MVC é dividir uma aplicação em três camadas: modelo, visualização e controle.

O modelo é responsável por representar objetos de negócio, manter o estado de aplicação e fornecer ao controlador o acesso aos dados.

A visualização representa a interface com o usuário, sendo responsável por definir a forma como os dados serão apresentados e encaminhar as ações dos usuários para o controlador.

O controlador é responsável pela ligação entre a visualização e o modelo. Interpreta as ações do usuário e as traduz para a operação sobre o modelo, onde são realizadas as mudanças e gerando a visualização apropriada.

2.5.3 Padrão MVC segundo JSF

No JSF o controle é composto por um servlet denominado *FacesServlet*, por arquivos de configuração e manipuladores. O *FacesServlet* faz o controle recebendo as requisições da WEB, redirecioná-las para o modelo e depois devolver uma resposta. Os arquivos de configuração realizam mapeamentos e associações de ações e definem também as regras de navegação. O modelo representa os objetos de negócio e executa uma lógica de negócio ao receber os dados vindos da camada de visualização. A visualização é composta por uma árvore de componentes (hierarquia de componentes UI), tornando possível unir um componente ao outro para formar *interfaces* mais complexas.

2.5.4 Ciclo de vida de uma aplicação JSF

No início do ciclo de vida, um pedido é feito pelo controlador do *FacesServlet*, que examina o pedido e extrai o ID da *view*, que é determinada através do nome da página JSP. A *view* contém todos os componentes da interface do usuário. O controle do JSF utiliza a ID da *view* para enxergar os seus componentes. Caso a *view* não exista, o JSF irá criá-la, mas caso exista o JSF *controller* criará.

O segundo passo os componentes precisam ser recuperados ou criados a partir de um objeto *FacesContext*, seguidos pelos seus valores. Os valores podem vir através da requisição de parâmetros como também podem vir através de *cookies* ou de cabeçalhos.

Após a fase de recebimento de valores entra a fase de validação. Nessa fase do ciclo de vida cada componente terá seu valor validado pelas regras de validação. As regras de validações podem ser pré-definidas, através do JSF, ou definidas pelo desenvolvedor. Se um valor recebido for inválido, uma mensagem de erro é adicionada ao *FacesContext* e o componente é marcado como inválido. No caso de um componente inválido o JSF parte para a resposta da requisição, que é exibida com a validação de erros da *view*. Não havendo erros, o JSF parte para a fase de atualização dos valores.

Na fase de atualização de valores, os valores do modelo são atualizados no servidor. Esses valores são atualizados através dos seus *backings beans* (também conhecidos como *beans* gerenciados).

Podemos dizer que a última fase é que a aplicação é invocada para realizar a submissão de formulário. Cada componente da aplicação será convertido, validado e aplicado para o modelo de objetos, podendo agora realizar a lógica de aplicação de negócios. Nessa fase deve ser definida a próxima sequência de *view* através da configuração das regras de navegação no arquivo *faces-config.xml*.

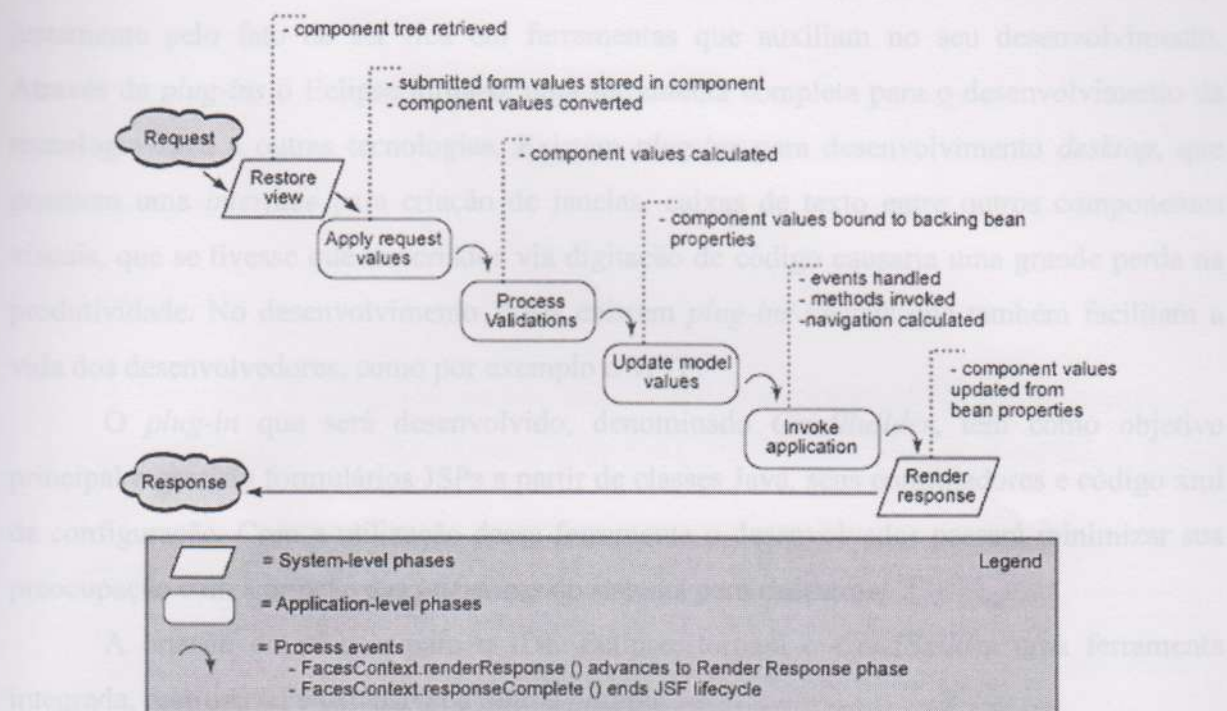


Figura 14 – Ciclo de vida de uma aplicação JSF (PITANGA, 2006)

2.5.5 Características e Vantagens do JSF

Java Server Faces é um bom *framework* para desenvolvimento WEB por ser um *framework* que possui um conjunto de componentes de *interface* já definidos, fornece um conjunto de *tags* JSP para acessar os componentes, reusar esses componentes de pagina, faz associação dos componentes do lado cliente com os manipuladores de eventos do lado servidor e fornece uma separação de funções que envolvem a construção de aplicações WEB.

JSF possui dois principais componentes: Java APIs para a representação de componentes UI e o gerenciamento de seus estados, manipulação/observação de eventos, validação de entrada, conversão de dados, internacionalização e acessibilidade; e *taglibs* JSP que expressam a *interface* JSF em uma página JSP e que realizam a conexão dos objetos no lado servidor.

3 DESENVOLVIMENTO DO PLUG-IN PARA O ECLIPSE

A Busca por ferramentas que possibilitem maior produtividade no desenvolvimento de Sistemas Corporativos vem crescendo e fazendo com que a cada dia novas ferramentas sejam criadas visando suprir essa necessidade. Java é uma das linguagens mais utilizadas hoje justamente pelo fato de ser rica em ferramentas que auxiliam no seu desenvolvimento. Através de *plug-ins* o Eclipse torna-se uma ferramenta completa para o desenvolvimento da tecnologia Java e outras tecnologias. Existem *plug-ins* para desenvolvimento *desktop*, que possuem uma *interface* para criação de janelas, caixas de texto entre outros componentes visuais, que se tivesse que ser criados via digitação de código causaria uma grande perda na produtividade. No desenvolvimento WEB existem *plug-ins* visuais que também facilitam a vida dos desenvolvedores, como por exemplo o WTP.

O *plug-in* que será desenvolvido, denominado *CrudBuilder*, tem como objetivo principal a geração formulários JSPs a partir de classes Java, seus controladores e código xml de configuração. Com a utilização dessa ferramenta o desenvolvedor passará minimizar sua preocupação com a criação das *interfaces* do sistema para cadastros.

A criação do *plug-in* para o IDE Eclipse, tornará o *CrudBuilder* uma ferramenta integrada, reutilizável e extensível a outros *plugins*.

3.1 REFLECTION API

A API *Reflection* permite que seja possível obter, em tempo de execução, informações sobre classes, métodos e atributos. Com essa informação, é possível criar uma instância de uma determinada classe que não se conhece em tempo de compilação, ou mesmo executar um método dessa classe possuindo somente seu nome e os tipos de seus parâmetros. Essas funcionalidades são utilizadas em diversos *frameworks* conhecidos para eliminar trabalhos considerados braçais. Essa API é utilizada, por exemplo, pelo *WebWork* e o *Struts* para colocar os parâmetros recebidos pelo *request* dentro das propriedades de um *JavaBean* e pelo *Hibernate* para popular os objetos com informações de uma base de dados.

A API *Reflection* pode ser também utilizada para adaptar um determinado sistema dinamicamente à diferentes situações. Considere, por exemplo, uma aplicação que use uma classe X para comunicar-se com algum serviço. Agora, suponha que essa aplicação precise comunicar-se com um serviço diferente, usando a classe Y, que tem nomes de métodos diferentes. Sem reflexão, a aplicação teria de ser modificada e recompilada. Mas, se a reflexão

for usada, isso pode ser evitado. A aplicação poderia conhecer os métodos da classe X e essa classe lhe diria que método era usado para que propósito. Assim, quando um novo serviço for usado, via classe Y, a aplicação também procuraria pelos métodos necessários e os utilizaria. Nenhuma modificação no código seria necessária. Nem mesmo o nome da nova classe deveria ser recompilado com a aplicação, uma vez que ele poderia estar armazenado em algum arquivo de configuração, ser verificado e ter a sua classe carregada em tempo de execução. (JavaFree, 2007)

O *Crudbuilder* utiliza a API *Reflection* para obter informações, em tempo de execução, sobre os campos, métodos e construtores. Com esses dados é realizada a geração dos componentes JSF.

3.2 CLASSES

A ferramenta *CrudBuilder* possui varias classes com responsabilidades diferentes para a geração das interfaces. As classes principais serão mostradas e detalhadas o seu funcionamento.

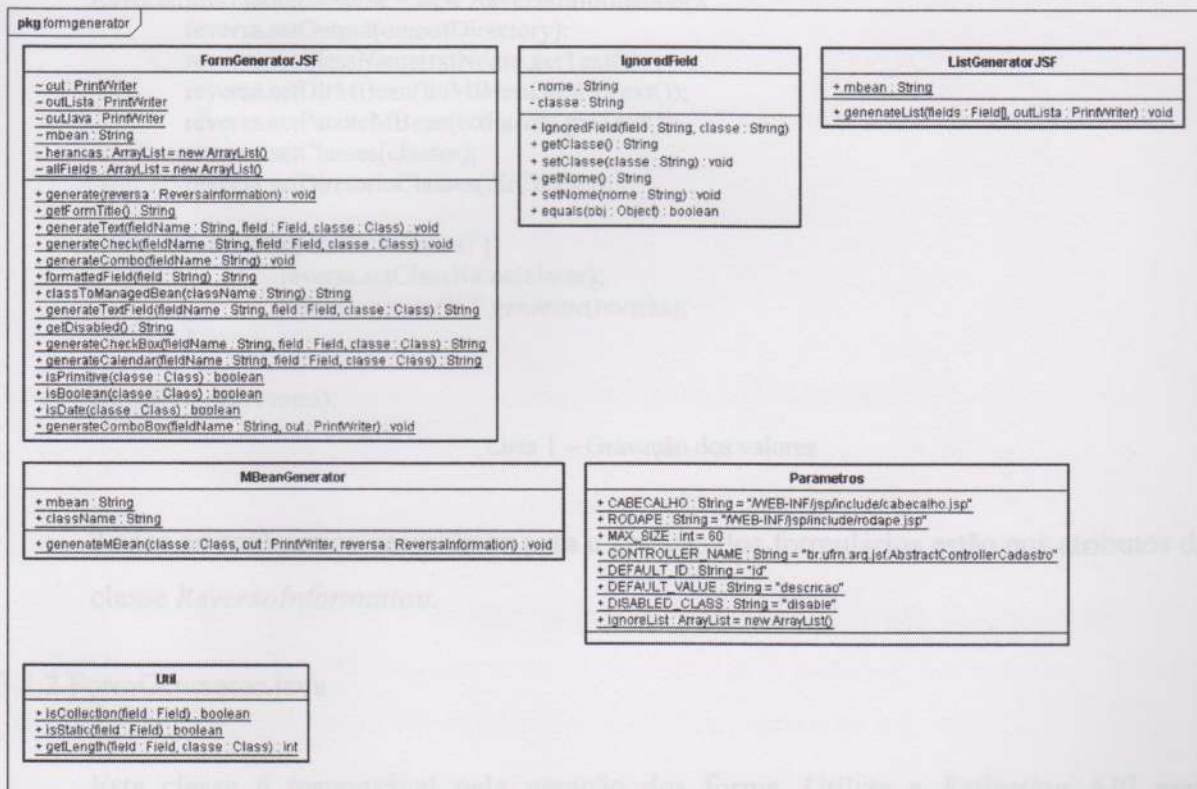


Figura 15 – Diagrama de classes do CrudBuilder

3.2.1 Generator.java

Classe Principal. Sua função é abrir uma *interface* gráfica solicitando os parâmetros necessários para a geração das JSPs. Os parâmetros solicitados nessa *interface* são os seguintes:

- Diretório de Saída: Diretório no qual serão geradas as JSPs.
- Diretório de MBeans: Diretório onde serão gerados os controladores.
- Pacote MBean: Pacote base para os controladores. Neste pacote ficarão todos os Mbeans gerados pela ferramenta.
- Classes: Conjunto de classes anotadas com JPA.

Os valores informados na *interface* serão “setados” na classe *ReversaInformation* como mostrado na lista 01. A classe *ReversaInformation* passa a possuir os valores que serão utilizados na classe *FormGenerator* para a geração dos Forms.

```
ReversaInformation reversa = new ReversaInformation();
reversa.setOutput(outputDirectory);
reversa.setClassName(txtNome.getText());
reversa.setDirMBean(lblMBeanDir.getText());
reversa.setPacoteMBean(txtPacote.getText());
reversa.setClasses(classes);
reversa.setDiretorioClasses(dirClasses);

for (String classe : classes) {
    reversa.setClassName(classe);
    FormGeneratorJSF.generate(reversa);
}

gravaReversa(reversa);
```

Lista 1 – Gravação dos valores

Todos os parâmetros necessários para a geração dos formulários estão nos atributos da classe *ReversaInformation*.

3.2.2 FormGenerator.java

Esta classe é responsável pela geração dos forms. Utiliza a *Reflection* API para recuperar os atributos das classes envolvidas.

A partir da classe *ReverseInformation* é instanciada uma classe a partir do atributo *className* como mostrado na lista 2 abaixo.

```
Class c = null;
```

```
try {
    c = Class.forName(reversa.getClassName());
} catch (Exception e) {
    JOptionPane.showMessageDialog(null, e.getMessage());
    return;
}
```

Lista 2 – Instância de classe

A etapa seguinte é a criação do diretório onde serão salvos os arquivos JSPs e o MBean de controle. O nome do diretório e o local foram informados na tela de entrada (Generator.java) e instanciados na classe ReversaInformation.java.

```
mbean = classToManagedBean(c.getName().substring(
    c.getName().lastIndexOf(".") + 1));

String directory = c.getName().substring(
    c.getName().lastIndexOf(".") + 1);

File dirSaida = new File(reversa.getOutput() + "/" + directory);
dirSaida.mkdir();

String formPath = reversa.getOutput() + "/" + directory + "/form.jsp";
String listPath = reversa.getOutput() + "/" + directory + "/lista.jsp";
String javaMBean = reversa.getDirMBean() + "/" + directory
    + "MBean.java";

try {
    out = new PrintWriter(formPath);
    outLista = new PrintWriter(listPath);
    outJava = new PrintWriter(javaMBean);

} catch (FileNotFoundException e) {
    JOptionPane.showMessageDialog(null, e.getMessage());
    return;
}
```

Lista 3 – Criação dos diretórios de saída dos arquivos gerados

O formulário é gerado utilizando os recursos da API *Reflection*. É possível identificar o tipo dos atributos e gerar os componentes do formulário relativos aos mesmos. Todos os atributos das classes superiores são recuperados e adicionados suas heranças.

Na lista 4 mostrada abaixo, é mostrado o trecho de código no qual é gerado o formulário JSP.

```
out.println("<%@include file=\"" + Parametros.CABECALHO + "\"%>");
out.println("<f:view>");
out.println("<h:form>");
```

```

out.println("<table class=\"formulario\">");
out.println(getFormTitle());

Class classe = c;

herancas.add(classe);

while (!classe.getSuperclass().equals(Object.class)) {
    herancas.add(classe.getSuperclass());
    classe = classe.getSuperclass();
}

for (int a = herancas.size() - 1; a >= 0; a--) {
    c = herancas.get(a);
    Field[] fields = c.getDeclaredFields();
    allFields.addAll(Arrays.asList(fields));
    for (int i = 0; i < fields.length; i++) {
        Field field = fields[i];
        IgnoredField test = new IgnoredField(field.getName(), c
        .getName());
        if (Parametros.ignoreList.indexOf(test) == -1) {
            if (!field.getName().equals("id")
            && !Util.isCollection(fields[i])
            && !Util.isStatic(fields[i])) {
                if (isPrimitive(field.getType())) {
                    if (isBoolean(field.getType())) {
                        generateCheck(field.getName(), field, c);
                    } else if (isDate(field.getType())) {
                        generateCalendar(field.getName(), field, c);
                    } else {
                        generateText(field.getName(), field, c);
                    }
                } else {
                    generateCombo(field.getName());
                }
            }
        }
    }
}

out.println("<tfoot><tr><td colspan=2>");
out.println("<h:commandButton " + "value=\"" + mbean
    + ".confirmButton}\" " + "action=\"" + mbean
    + ".cadastrar}\" /> <h:commandButton "
    + "value=\"" + "Cancelar\" action=\"" + mbean
    + ".cancelar}\" /></td>");
out.println("</tr></tfoot>");

out.println("</table>");
out.println("</h:form>");
out.println("</f:view>");
out.println("<%@include file=\"" + Parametros.RODAPE + "\"%>");

out.close();

```

Lista 4 – Geração do form.jsp

A geração de uma lista para a alteração e exclusão dos dados já adicionados é feita no passo seguinte. O processo de geração da Lista é feito pela classe `ListGeneratorJSF.java`. Essa classe realiza a geração de uma tabela com os dados dos atributos gerados no Formulário JSP.

```
ListGeneratorJSF.mbean = mbean;
Field[] fields = new Field[allFields.size()];
System.arraycopy(allFields.toArray(), 0, fields, 0, allFields.size());
ListGeneratorJSF.genenateList(fields, outLista);
```

Lista 5 – Geração da Lista

3.2.2 ListGeneratorJSF.java

A Classe `ListGeneratorJSF.java` também utiliza a API *Reflection* para recuperar o tipo dos atributos passados e gera uma tabela com os campos passados para a classe permitindo a exclusão e alteração dos dados.

```
outLista.println("<%@include file=\"" + Parametros.CABECALHO + "\"%>");
outLista.println("<f:view>");
outLista.println("<h2>" + mbean + "</h2><br>");

outLista.println("<h:outputText value=\"" + mbean + ".create\"/>");
outLista.println("<table class=\"listagem\"/>");
outLista.print("<caption class=\"listagem\"> Lista de " + mbean
+ "s</caption><tr>");
for (int i = 0; i < fields.length; i++) {
    if (!fields[i].getName().equals("id")
        && !Util.isCollection(fields[i]) && !Util.isStatic(fields[i])) {
        outLista.println("<td>" + fields[i].getName() + "</td>");
    }
}
outLista.print("<td></td><td></td>");
outLista.println("</tr>");
outLista.println("<c:forEach items=\"" + mbean
+ ".all\" var=\"item\"/>");
outLista.println("<tr>");

for (int i = 0; i < fields.length; i++) {
    if (!fields[i].getName().equals("id")
        && !Util.isCollection(fields[i]) && !Util.isStatic(fields[i])) {
        {
            outLista.println("<td>${item." + fields[i].getName()
+ "}</td>");
        }
    }
}
outLista.println("<h:form>");
outLista.println("<td width=20>");
outLista
    .println("<input type=\"hidden\" value=\"" + mbean + ".id\" name=\"id\"/>");
outLista
    .println("<h:commandButton image=\"/img/alterar.gif\" value=\"Alterar\"
action=\"" + mbean + ".atualizar\"/>");
```

```

outLista.println("</td>");
outLista.println("</h:form>");

outLista.println("<h:form>");

outLista.println("<td width=25>");
outLista
    .println("<input type=\"hidden\" value=\"${item.id}\" name=\"id\"/>");
outLista
    .println("<h:commandButton image=\"/img/delete.gif\" alt=\"Remover\"
action=\"#{"
                                + mbean + ".preRemover}\"/>");
outLista.println("</td>");
outLista.println("</h:form>");
outLista.println("</tr>");
outLista.println("</c:forEach>");
outLista.println("</table>");
outLista.println("</f:view>");

outLista.println("<%@include file=\"" + Parametros.RODAPE + "\"%>");

```

Lista 6 – Classe ListGeneratorJSF

As classes principais para a geração dos formulários foram apresentadas acima. O funcionamento do FormGenerator está concentrado especificamente nessas classes.

3.3 ESTRUTURA DO PLUGIN

O *Plug-in* possui uma estrutura padrão gerada pelo PDE (*Plug-in Development Environment*). Possuindo um arquivo de configuração XML, uma classe que controla o ciclo de vida do *plug-in* e uma classe *action* para execução das suas tarefas.

No arquivo `plugin.xml` estão as configurações do *plug-in* para seu funcionamento no Eclipse. Este arquivo é fundamental para o seu funcionamento, pois é nele que estão todas as funcionalidades a serem adicionadas. Quando o Eclipse é carregado em memória ele lê todas as informações contidas neste arquivo, e a partir disto é possível que novas funcionalidades sejam acrescentadas ao seu ambiente de desenvolvimento. O `plugin.xml` possui um editor gráfico, mas também é possível editar manualmente o fonte do arquivo XML. Definimos um ponto de extensão no nosso *plug-in* para ser criado um *wizard* baseado nos pontos de extensão do Eclipse. Existem vários pontos de extensão para cada funcionalidade do Eclipse, como por exemplo *wizards*, *views* e *editors*. No *Plug-in* desenvolvido aqui utilizamos o ponto de extensão `org.eclipse.ui.newWizards`.

```

<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>

<extension
id="wizard"
name="FormGenerator"

```

```
point="org.eclipse.ui.newWizards">
<category
name="Form Generator"
id="wizard.category1">
</category>
<wizard
name="Form Generator"
icon="icons/wizard.gif"
category="wizard.category1"
class="br.farn.formgenerator.FGWizard"
id="wizard.wizard1">
<description>
Geração de Formulários WEB
</description>
</wizard>
</extension>

</plugin>
```

Plugin.xml

3.4 IMPLEMENTAÇÃO

Para a implementação do *plug-in* foi utilizado o PDE que auxilia na criação dos arquivos de configuração e toda a sua estrutura, escondendo todos os detalhes da infraestrutura, possibilitando se preocupar somente com a implementação da “lógica de negócios” do *plug-in*, aproveitando os recursos dessa ferramenta bastante valiosa.

Utilizei o ponto de extensão `org.eclipse.ui.newWizard` que possibilita a criação de um *wizard* para auxiliar na geração dos formulários Web, apresentando uma tela para preenchimento dos dados necessários para a geração. Ao finalizar o preenchimento dos dados as classes responsáveis pela geração dos formulários serão invocadas.

O arquivo `plugin.xml` possui as configurações de pontos de extensões utilizados para criação do *plug-in* e identificação.

```
<extension
id="wizard"
name="FormGenerator"
point="org.eclipse.ui.newWizards">
<category
name="Form Generator"
id="wizard.category1">
</category>
<wizard
name="Form Generator"
icon="icons/wizard.gif"
category="wizard.category1"
class="br.farn.formgenerator.FGWizard"
id="wizard.wizard1">
<description>
Geração de Formulários WEB
</description></wizard>
</extension>
```

Utilizamos o ponto de extensão “newWizards” que possibilita a integração no eclipse. A execução do *plug-in* ficará disponível na área de criação de projetos, classes e outros, mostrado na figura 14 e figura 15.

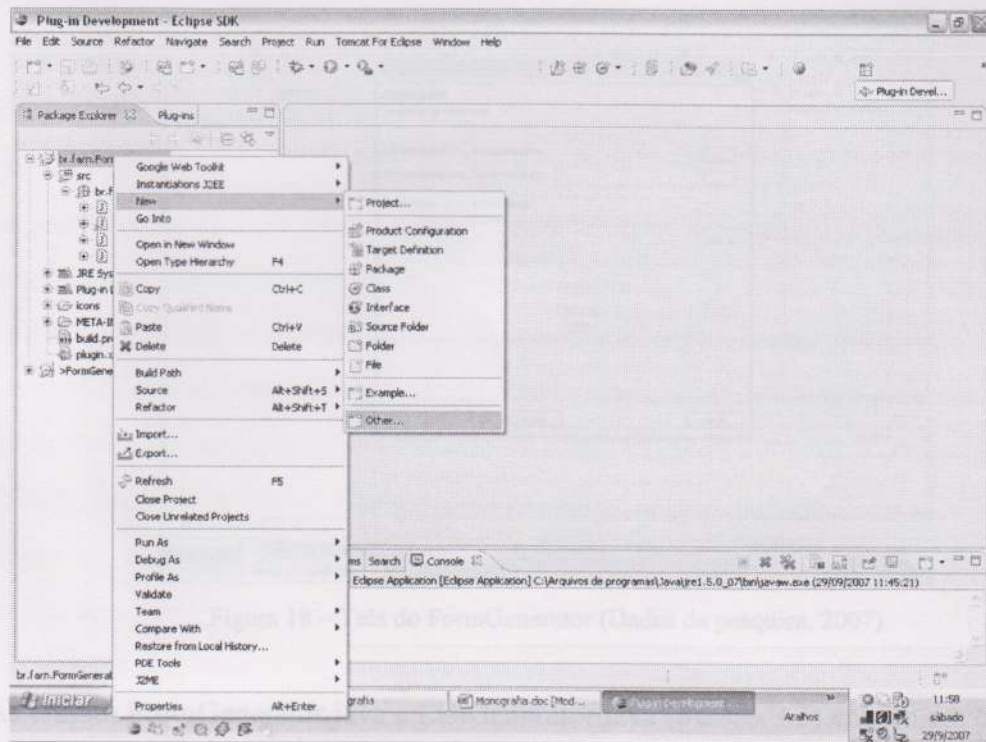


Figura 16 – Ponto de extensão do Plugin (Dados da pesquisa, 2007)

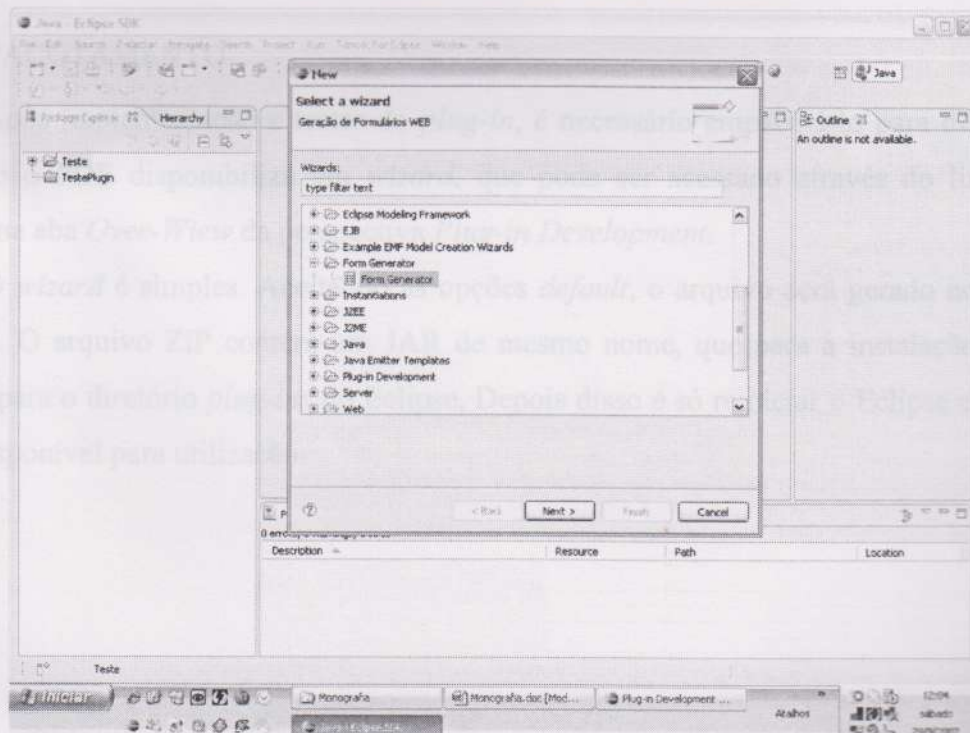


Figura 17 – Wizard do FormGenerator (Dados da Pesquisa, 2007)

Para a criação das páginas do *wizard* foram utilizados os padrões SWT e Jface. Com essas duas classes temos os componentes necessários para a criação da *interface*. A página do *wizard* possui todas as entradas de dados para a execução da geração dos formulários como mostrado na figura 16.

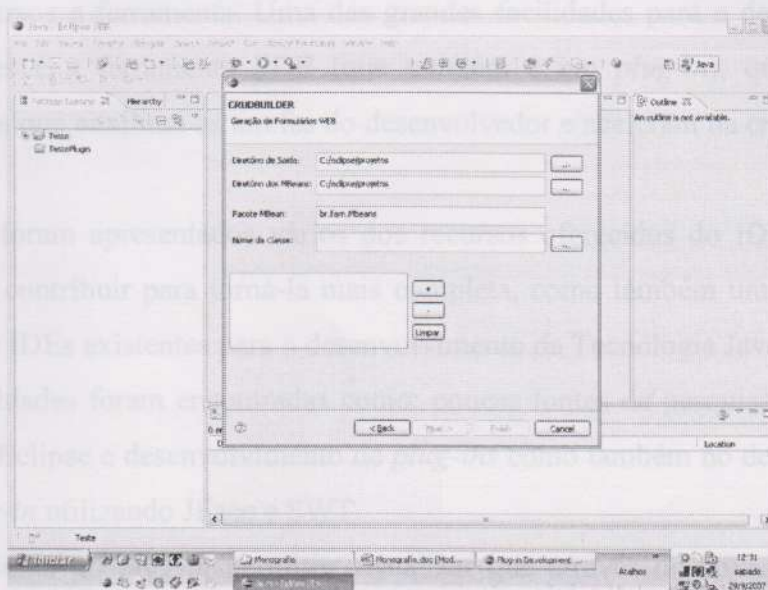


Figura 18 – Tela do FormGenerator (Dados da pesquisa, 2007)

As classes `FormGenerator.java` e `ListGenerator.java` (Sessão 3.2) entram em cena para a geração dos pacotes, MBeans e dos formulários JSPs.

3.5 EMPACOTAMENTO

Após implementação e testes do *plug-in*, é necessário empacotá-lo para distribuição. Para isso o PDE disponibiliza um *wizard*, que pode ser acessado através do link *Export Wizard*, na aba *Over-View* da perspectiva *Plug-in Development*.

O *wizard* é simples. Aceitando as opções *default*, o arquivo será gerado no diretório indicado. O arquivo ZIP conterá um JAR de mesmo nome, que para a instalação deve ser copiado para o diretório *plug-ins* do Eclipse. Depois disso é só reiniciar o Eclipse e o *plug-in* estará disponível para utilização.

4. CONCLUSÃO

O Eclipse IDE possui uma grande capacidade de extensão, e sua arquitetura foi pensada para permitir que diversas contribuições (*Plug-ins*) possam trabalhar em conjunto para oferecer novos recursos a ferramenta. Uma das grandes facilidades para o desenvolvimento dessas contribuições é a ferramenta PDE (que também é um *plug-in*), que contém um conjunto de recursos que auxiliam as tarefas do desenvolvedor e aceleram na criação de novas extensões.

Neste trabalho foram apresentados vários dos recursos oferecidos do IDE Eclipse, sua arquitetura e como contribuir para torná-la mais completa, como também uma apresentação resumida das outras IDEs existentes para o desenvolvimento da Tecnologia Java.

Algumas dificuldades foram encontradas como: poucas fontes de pesquisa em português sobre a plataforma Eclipse e desenvolvimento de *plug-ins* como também no desenvolvimento da interface do *plug-in* utilizando JFace e SWT.

O Desenvolvimento do *plug-in* permitiu visualizar que através de novas contribuições podemos ter cada vez mais uma ferramenta completa, facilitando no desenvolvimento de Sistemas Corporativos em todas as áreas na qual o mesmo será desenvolvido.

ROCHA, A. D. (2005). Criando Plug-ins para o Eclipse. *Java Magazine*, 40-48.

SANTOS, L. (2 de Abril de 2005). Introdução a Plataforma Eclipse. Rio de Janeiro, Rio de Janeiro, Brasil.

TCBITA, R. (15 de Fevereiro de 2007). Desenvolvimento de Plug-ins para o Eclipse e tecnologias relacionadas. São Paulo, São Paulo, Brasil.

Wikipédia (04 de 07 de 2007). IDEs Java. Acesso em 23 de 08 de 2007, disponível em Wikipédia.

http://www.javafree.org/w/index.php?title=Java_%28linguagem_de_programa%29&start=13

JavaFree (11 de 08 de 2007). Reflection API. Acesso em 11 de 08 de 2007, disponível em JavaFree: www.javafree.org

REFERÊNCIAS

ECLIPSE CONTRIBUTORS. (21 de Agosto de 2007). **Eclipse.org**. disponível em: <www.eclipse.org/plugins>. Acesso em: 1 de Agosto de 2007,

Eclipse. (08 de 10 de 2007). *Plugin Central*. Acesso em 08 de 10 de 2007, disponível em Eclipse: <http://www.eclipseplugincentral.com>

FREIRE, A. (20 de Janeiro de 2003). **Além do Eclipse** - Contribuindo com seus próprios Plug-ins. Local: São Paulo, 2007.

JANDI, P. (01 de 01 de 1999). **Introdução ao Java**. São Paulo, São Paulo, Brasil.

LOZANO, F. (3 de Maio de 2004). **NetBeans x Eclipse - A evolução dos IDEs livres para Java**. São Paulo, São Paulo, Brasil.

PITANGA, T. (01 de 08 de 2006). JSF: **A mais nova tecnologia Java para desenvolvimento WEB**. Rio Grande do Norte, Natal, Brasil.

ROCHA, A. D. (2006). **Criando Plug-ins para o Eclipse**. *Java Magazine* , 40-48.

SANTOS, I. (2 de Abril de 2005). **Introdução a Plataforma Eclipse**. Rio de Janeiro, Rio de Janeiro, Brasil.

TOMITA, R. (15 de Fevereiro de 2007). **Desenvolvimento de Plug-ins para o Eclipse e tecnologias relacionadas**. São Paulo, São Paulo, Brasil.

Wikipedia. (01 de 07 de 2007). *IDEs Java*. Acesso em 23 de 08 de 2007, disponível em Wikipedia:
http://pt.wikipedia.org/w/index.php?title=Java_%28linguagem_de_programa%C3%A7%C3%A3o%29&action=edit§ion=13

JavaFree. (11 de 08 de 2007). *Reflection API*. Acesso em 11 de 08 de 2007, disponível em JavaFree: www.javafree.org