Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática


# INTEGRAÇÃO DE FERRAMENTAS DE DESENVOLVIMENTO NO ECLIPSE

projecto realizado na

EF Tecnologias de Software, SA

por

Miguel Rodrigues Valente

Mestrado em Engenharia Informática

2007

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática


# INTEGRAÇÃO DE FERRAMENTAS DE DESENVOLVIMENTO NO ECLIPSE

projecto realizado na

EF Tecnologias de Software, SA

por

Miguel Rodrigues Valente

Projecto orientado pela Prof.ª Dr.ª *Ana Paula Afonso*
e co-orientado pelo Eng.º *Nuno Costa*

Mestrado em Engenharia Informática

2007


FACULDADE · DE · CIÊNCIAS UNIVERSIDADE · DE · LISBOA

# Declaração

*Miguel Rodrigues Valente*, aluno nº 28799 da Faculdade de Ciências da Universidade de Lisboa, declara ceder os seus direitos de cópia sobre o seu Relatório de Projecto em Engenharia Informática, intitulado "Integração de Ferramentas de Desenvolvimento no Eclipse", realizado no ano lectivo de 2006/2007 à Faculdade de Ciências da Universidade de Lisboa para o efeito de arquivo e consulta nas suas bibliotecas e publicação do mesmo em formato electrónico na Internet.

FCUL, 1 de Junho de 2007

*Nuno Costa*, supervisor do projecto de *Miguel Rodrigues Valente*, aluno da Faculdade de Ciências da Universidade de Lisboa, declara concordar com a divulgação do Relatório do Projecto em Engenharia Informática, intitulado "Integração de Ferramentas de Desenvolvimento no Eclipse".

Lisboa, 1 de Junho de 2007

# Resumo

Nos tempos que correm, são cada vez mais utilizadas ferramentas que facilitam o trabalho a quem desenvolve software. Isto acontece por razões óbvias, sendo a mais determinante a de permitir ao programador preocupar-se o menos possível, com pormenores irrelevantes ao processo de criação de software, sejam eles inerentes à linguagem, o memorizar da totalidade das bibliotecas disponíveis ou o descobrir a localização de erros de sintaxe. O aparecimento deste tipo de ferramentas tem vindo a rentabilizar bastante o trabalho de quem as usa, não só em termos de velocidade mas, também de criatividade já que, menos interrupções ao raciocínio permitem ir mais longe em termos de abstracção e complexidade.

Neste contexto, a plataforma de desenvolvimento Eclipse tem tomado uma posição dianteira face a outras por várias razões. Destacam-se a facilidade de utilização, o excelente apoio à linguagem Java, e talvez a mais relevante para o contexto deste projecto, a possibilidade de alterar funcionalidades já existentes e até a de criar outras novas, integrando-as na plataforma. Esta mais valia tem proporcionado, nos últimos tempos, uma adaptação da plataforma original aos objectivos, gostos e preferências dos elementos de várias empresas sendo mesmo, muitas vezes criada uma nova versão do Eclipse à medida de cada um.

Este projecto tem como objectivo a integração de novas funcionalidades no Eclipse, entre as quais, um construtor específico para uma linguagem usada internamente, acompanhado de todas as funcionalidades que permitam e maximizem o seu uso. Tendo como objectivo principal o de rentabilizar e facilitar o desenvolvimento de software da empresa EF Tecnologias de Software, SA, este projecto foi elaborado no âmbito da disciplina de Projecto em Engenharia Informática, com uma duração de nove meses e com o objectivo de obter o grau de Mestre em Engenharia Informática.

PALAVRAS-CHAVE:
Eclipse, construtor, integração, desenvolvimento, software

# Abstract

Nowadays we are witnessing a growing use of software development tools. The main objective of these tools is to allow the programmer to waste, as little time as possible, with irrelevant details to the process of software creation. Like errors inherent to the language nature, memorizing all of the available libraries or discovering the exact lines where sintax errors occur. The growing use of this type of tools has improved dramatically the work of whom uses them, not only at the level of creativity but also the development speed. This advantage allows the developer to focus on the real issue, reaching higher levels of abstraction and complexity.

In this context, the development platform Eclipse has taken a lead position, among other platforms. It´s easy to use, it as excellent Java language support, and perhaps the most important feature for this project, the possibility to modify existing features and even to create new ones, integrating them in the platform. These features have alowed, in the last years, an evolution of the original platform to meet the users goals, tastes and preferences, sometimes even originating a new version of the Eclipse, built to meet specific needs.

This project's goal is the integration of new functionalities in the Eclipse platform. Among these tools, a builder for an internally used language, along with all the features that allow and maximize its use. To increase the productivity and ease up the work of the elements of the company EF Tecnologias de Software, SA is this application's main goal. It was developed in the scope of project of Engineering Computer Sciences, with a duration of nine months and the goal of achieving the degree of Master in Engineering Computer Sciences.

KEYWORDS:
Eclipse, builder, integration, development, software

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

O desenvolvimento de software tem, ao longo dos tempos, evoluído em vários aspectos, tendo contribuído para isso melhorias significativas a vários níveis. As novas ferramentas de desenvolvimento de software têm contribuído em larga medida para esta evolução. O seu papel é tornar transparente, para o utilizador, o processo de desenvolvimento de software. Transparente no sentido de, por um lado, automatizar o processo de compilação e a gestão de bibliotecas necessárias, e por outro, fornecer ferramentas auxiliares como editores adequados e assistentes de configuração. Utilizando este tipo de ferramentas, torna-se possível despender mais tempo e atenção, no processo criativo e na produção de soluções, em vez de, como muitas vezes acontece, gastar metade do tempo disponível, a resolver problemas inerentes à fase de compilação.

Este relatório descreve o processo de produção e integração, de um conjunto de ferramentas, na plataforma de desenvolvimento *Eclipse IDE,* que consiste num conjunto de ferramentas, integradas com o objectivo de fornecer tudo o que é necessário para desenvolver software. Cada grupo de utilizadores tem necessidades diferentes e por essa razão o *Eclipse* permite criar módulos de software para estender a sua funcionalidade. Tem mesmo um ambiente específico para o efeito, que será abordado na secção 1.1.1.

O conjunto de funcionalidades a desenvolver diz respeito ao *Domvs-PI,* uma plataforma de integração de serviços que engloba suporte a *web services,* serviço de autenticação, suporte a bases de dados, entre outros. Cada aplicação que é construída sobre o *Domvs-PI*, define a forma como o acesso a estas funcionalidades é feito. Esta definição é feita através de interfaces, que são produzidos a partir da interpretação de ficheiros, denominados ficheiros *Didl*, que contêm as regras para a geração destas interfaces.

Este conjunto é constituído por dois módulos: um construtor e um editor, que integram a plataforma *Domvs-PI* na plataforma de desenvolvimento *Eclipse*. Os módulos referidos, assim como este relatório, foram elaborados no âmbito do Projecto em Engenharia

Informática (PEI), com a duração de nove meses, tendo como data de início o dia onze de Setembro de 2006 e data final, o dia onze de Junho de 2006.

## 1.1 Motivação

Esta secção expõe uma descrição das duas plataformas sobre as quais o projecto incide, o *Eclipse IDE* e o *Domvs-PI*, e um breve enquadramento da instituição na qual foi realizado, a empresa EF Tecnologias de Software, SA. É também efectuada uma explicação das vantagens que os módulos de software criados com este projecto trazem para a empresa, já que são estas vantagens que constituem a motivação para todo este projecto.

### 1.1.1 Eclipse IDE

O *Eclipse IDE*[1] é um ambiente de desenvolvimento de software integrado, do inglês *Integrated Development Environnement (IDE)*. Como o próprio nome indica é um ambiente de trabalho, composto por várias ferramentas integradas, pensado e elaborado para desenvolver qualquer tipo de aplicação (ou parte de uma). O *Eclipse*, na sua forma mais simples, apenas reúne funcionalidades para gerir conceitos comuns a qualquer linguagem de programação, como a noção de projecto, de recurso dentro de um projecto, de visualização e edição do conteúdo desses recursos e a sua compilação. Esta ferramenta é particularmente utilizada como ambiente de desenvolvimento em Java (descrito na secção 1.3), na forma de módulos, todos eles independentes uns dos outros. Esta modularidade permite que se acrescentem novas funcionalidades, sobre qualquer uma das versões já existentes, sem fazer modificações nenhumas ao código já desenvolvido. Estas características tornaram o *Eclipse* num dos ambientes de desenvolvimento mais utilizados, já que permite com alguma facilidade, uma adaptação total da plataforma às necessidades de cada utilizador.

Uma das componentes do *Eclipse* é o *Plugin Development Environnement* ou *PDE*, que é focado na produção de *plug-ins*[1] e contêm um conjunto de ferramentas que auxiliam na configuração, gestão e desenvolvimento de *plug-ins*. Esta foi a principal componente utilizada no desenvolvimento deste projecto.

### 1.1.2 Domvs-PI

O *Domvs Internet Banking SOA* é uma ferramenta de desenvolvimento de software construída na *EF*, com o intuito de produzir e integrar serviços. É uma plataforma orientada

[1] Um *plug-in* ou *plugin* é um programa de computador (ou parte de um programa) que serve, normalmente, para adicionar funções a outro programa maior, fornecendo alguma funcionalidade especial ou muito específica.

a serviços e uma solução de integração multi-canal, que serve de base para construir e gerir um conjunto de serviços financeiros e de banca electrónica que sejam, ao mesmo tempo, flexíveis, escaláveis, extensíveis e de alta prestação. Consiste, pois, numa camada de *middleware* que expõe a sua lógica funcional e os seus componentes reutilizáveis de uma forma flexível. Esta segue uma perspectiva de integração a três níveis, separando a lógica de apresentação, da camada de suporte a negócios e dos serviços de integração de dados.

O *Domvs* utiliza uma linguagem de definição de serviços, expressa numa sintaxe bem definida, denominada *Didl*, a partir da qual é gerado um conjunto de interfaces *Java*, que são posteriormente implementados de acordo com as definições específicas de cada projecto. Esta sintaxe é escrita em ficheiros, conhecidos como ficheiros *Didl (Domvs Interface Definition Language)*, que têm essa mesma extensão.

O processo de geração destes interfaces, é feito através da interpretação das regras definidas nestes ficheiros. Esta interpretação recorre a *scripts*, que utilizam a ferramenta *Apache Ant*[2] [2], obrigando o programador a uma série de passos repetitivos por cada alteração que faz sobre um ficheiro *Didl*. Na figura 1.1 encontra-se representado, de uma forma simplificada, o típico ciclo de desenvolvimento, utilizando o *Domvs-PI*, sem o auxílio do software desenvolvido neste projecto. Este ciclo engloba três mudanças de contexto para cada alteração feita a um ficheiro *Didl*, o que implica perdas de rendimento, porque não só estas mudanças de contexto consomem tempo, como podem levar a perdas de concentração. Minorar este problema, tanto quanto possível é o objectivo central deste projecto.

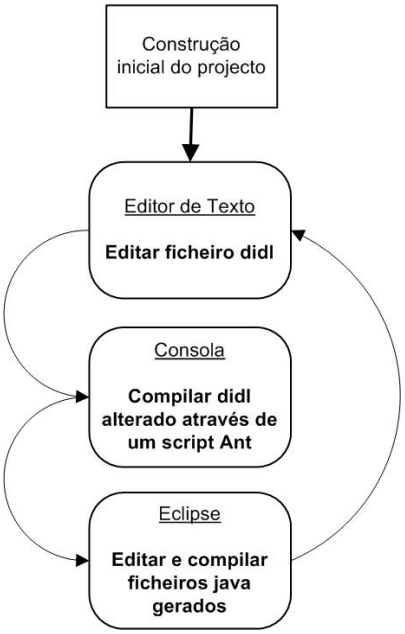


Figura 1.1: Ciclo Típico de Construção de Novas Funcionalidades

### 1.1.3 Enquadramento Institucional

A empresa *EF Tecnologias de Software, SA*[3], é uma empresa que fornece soluções profissionais na área das Tecnologias de Informação, focada principalmente na banca on-line e no sector financeiro, e que tem uma forte base tecnológica, embora se afirme principalmente como uma empresa que desenvolve soluções e que é orientada ao cliente.

Como plataforma central do desenvolvimento de software, a *EF* desenvolveu e utiliza a plataforma *Domvs-PI*, que não é uma aplicação *per se*, mas sim um conjunto de funcionalidades e serviços, com base no qual é desenvolvida uma grande parte do software criado na empresa.

[2] O *Apache* Ant é uma ferramenta utilizada para automatizar a construção de software. É semelhante ao *make* da linguagem C mas é escrito em linguagem *Java* e foi desenvolvido para ser utilizado em projectos desta linguagem.

Os colaboradores da empresa fazem uso do *Domvs-PI* em conjunto com a plataforma de desenvolvimento *Eclipse*. A primeira plataforma foi desenvolvida em *Java*, por essa razão é vantajoso integrar as duas plataformas, numa estrita colaboração, aproveitando a adaptabilidade do *Eclipse*. Assim, a ideia deste projecto terá surgido como uma solução, não só para minimizar os passos que cada colaborador tem que efectuar, ao utilizar o *Domvs-PI*, como para tirar maior partido do potencial da plataforma *Eclipse*.

## 1.2 Objectivos do Projecto

O objectivo central deste projecto, é integrar na plataforma *Eclipse,* um conjunto de ferramentas e funcionalidades, que permitam a utilização do *Domvs-PI,* em todo o seu potencial e com algum grau de automatização.

Após um estudo documental das plataformas, análise sobre *plug-ins* de *Eclipse* [4, 5, 6], do próprio *Domvs* e sua documentação, e das necessidades transmitidas como fundamentais, por parte dos colaboradores da empresa, foi realizada a seguinte lista de requisitos funcionais, a incorporar na plataforma *Eclipse*:

1. Definição de um construtor ou *builder*, como parte central da aplicação, responsável pela compilação de *Didls*, resolução de dependências, criação de pastas fonte e registo do estado actual da aplicação. Este construtor deve ser inserido na lista de construtores do *Eclipse* antes do construtor *Java*, de forma a que num só acto de construção, sejam avaliados os *Didls*, compilados os que são novos ou foram alterados e, de seguida, compilados os ficheiros *Java* gerados no passo anterior. Por uma questão de manutenção, serão apagados os ficheiros gerados por um *Didl* que já não exista ou que tenha sido alterado;

2. A definição da natureza *Domvs* associada ao novo construtor, de forma a que ao seleccionar um projecto, como sendo de natureza *Domvs,* passe a ser compilado com o novo construtor e que tenha disponíveis as respectivas funcionalidades;

3. Um editor de ficheiros *Didl* que reconheça a extensão destes ficheiros e que mostre uma sintaxe colorida, de acordo com as palavras-chave próprias da linguagem. Como parte deste editor deve ainda ser implementada uma janela conhecida como *outline* no *Eclipse,* que consiste numa vista hierárquica da estrutura do ficheiro;

4. Um sistema de visualização de erros que utilize o mecanismo de marcação de recursos do *Eclipse.* A partir de erros de sintaxe ou avisos recebidos do processo de compilação, deve ser possível marcar os ficheiros em que o erro ou aviso foi detectado, na respectiva linha;

5. Um assistente *(wizard)* de apoio à criação de novos ficheiros *Didl* que, para além da escolha do nome do ficheiro e pasta onde deve ser criado, deve permitir a escolha das opções específicas de cada *Didl*;

6. Uma página de propriedades de ficheiros *Didl* que permita, em qualquer momento, a alteração das opções específicas mencionadas no ponto anterior;

7. Uma página de preferências gerais do construtor, comuns a todos os projectos que o usem, como a escrita de um ficheiro de *log* e a visualização de uma consola de retorno ao utilizador sobre o processo de *build*, entre outras opções consideradas relevantes.

Esta integração é obtida, através da reunião das funcionalidades e recursos necessários à sua execução, num ou mais *plug-ins,* disponibilizados aos colaboradores da empresa através de um *update site*[3], um espaço num dos servidores da empresa onde a aplicação é colocada e configurada de forma a que, introduzindo o *URL,* no gestor de *plug-ins* do *Eclipse,* a aplicação possa ser descarregada e actualizada automaticamente.

Para além desta lista inicial, surgiram, durante o processo de desenvolvimento, outros componentes que não faziam parte da listagem de objectivos iniciais: um gestor de alterações de projectos, um sistema de sugestões e a obtenção do *Domvs-PI* dinamicamente. Estes serão devidamente explicados nos capítulos 3 e 4.

## 1.3 Trabalho Relacionado

O ambiente de desenvolvimento em *Java* do *Eclipse,* exerceu, por vários motivos, a maior influência sobre a elaboração deste projecto. Este ambiente é composto por um construtor incremental[4] de ficheiros *Java* (que gera os ficheiros *bytecode*), um editor que mostra as palavras chave da linguagem *Java* coloridas, além de todas as ferramentas de apoio, que no caso deste construtor é um conjunto bastante completo. Para citar apenas as mais importantes:

- compilação instantânea, isto é, o código é compilado à medida que é introduzido, tornando mais fácil a elaboração de classes e métodos;

- editor de ficheiros *Java* com as palavras chave da linguagem coloridas de formas diferentes, para facilitar a leitura, e uma vista hierárquica do ficheiro;

[3] Um sítio na Internet ou na rede local, onde a versão mais recente de cada *plug-in*, é colocada, permitindo ao utilizador obtê-la ou actualizar a que já tem, para a última versão.

[4] Um construtor incremental permite que, a partir de uma construção total incial, apenas sejam recompilados os ficheiros que foram alterados (e os que dependem destes, se os houver), desde o último processo de construção, aumentando amplamente o desempenho da aplicação.

- marcação de ficheiros onde ocorrem erros de compilação, tipicamente com uma cruz a vermelho, visível no editor, na linha ou linhas onde o erro ocorre;
- sugestão de soluções e correcção automática para a maioria dos erros conhecidos e habituais;
- sugestões para completar palavras ou expressões comuns, que o utilizador apenas começou a escrever, ou mesmo antes de escrever qualquer coisa;
- *refactoring* que é um sistema que actualiza todas as referências para um método, classe ou atributo, quando o nome deste é alterado, mantendo a coerência do código.

As semelhanças na natureza e funcionalidades, entre o ambiente de desenvolvimento *Java* e o conjunto de *plug-ins* elaborado neste projecto, permitiram que aquele influenciasse as opções tomadas a nível conceptual.

## 1.4 Trabalho Desenvolvido

O construtor é considerado o módulo principal do trabalho realizado, foi desenvolvido na totalidade, principalmente pela especificidade do problema. Inicialmente foi utilizado um assistente, para produzir construtores e compiladores, incluído no *Eclipse*, no já mencionado *PDE*. Este gera um esqueleto inicial, que é equivalente para todos os tipos de construtores, como ponto de partida para a elaboração de um construtor incremental.

O editor foi construído a partir de um editor já existente, pois a linguagem a que se destina deriva de uma outra, bastante vulgar (o *XML*), bastando para o efeito expandir e personalizar um dos muitos editores disponíveis. Sobre este, foi apenas incluída informação sobre o que foi elaborado ou alterado, em relação ao original.

Além destes dois pontos centrais do projecto, existe um conjunto de ferramentas auxiliares e de adições gráficas feitas à plataforma que, quando consideradas relevantes, são descritas e/ou ilustradas.

## 1.5 Organização do Documento

Este documento está organizado da seguinte forma:

- No Capítulo 2 - Metodologia e Planeamento do Trabalho, é exposto o método utilizado para realizar este projecto, além de um planeamento do trabalho, respectivo mapa de *Gantt* e uma tabela de tarefas e respectivas datas. No final é efectuada uma comparação entre o trabalho que estava planeado e o que foi realizado.

- O Capítulo 3 - Construtor `DidlBuilder`, descreve a modelação e implementação, do módulo responsável pela construção de projectos *Domvs-PI* e das ferramentas produzidas, para possibilitar e maximizar o seu uso.
- No Capítulo 4 - Editor `DidlEditor`, é apresentado o editor de ficheiros *Didl* e os mecanismos utilizados para o implementar. São utilizadas algumas imagens da aplicação e é descrito o trabalho realizado para a sua implementação.
- O Capítulo 5 - Conclusão e Balanços faz um balanço sobre o trabalho realizado, além de algumas referências sobre trabalho a realizar no futuro sobre esta aplicação.

# Capítulo 2

# Metodologia e Planeamento do Trabalho

Neste capítulo é descrita a metodologia utilizada, ao longo do desenvolvimento deste projecto. Está também incluída, uma cronologia dividida semana a semana, que permite perceber o ritmo a que este foi realizado. Para cada etapa, encontra-se uma breve descrição dos seus pontos mais relevantes, além dos métodos aplicados para cumprir os objectivos, delineados no capítulo anterior.

## 2.1 Metodologia Utilizada

O método escolhido inicialmente, para a elaboração deste projecto, foi o método de desenvolvimento em cascata. Contudo, a existência de funcionalidades não previstas inicialmente, sugeriu que se adoptasse um modelo iterativo e incremental. A natureza modular desta aplicação, que mantém cada sector independente dos outros, permite regressar ao desenho ou mesmo aos requisitos, sem causar conflitos com os módulos já desenvolvidos. Na prática, as várias fases que compõem o modelo em cascata, foram finalizadas de forma incremental e através de mais do que uma iteração.

As fases de desenvolvimento foram as seguintes:

1. **Recolha de requisitos -** Foram recolhidos requisitos de duas formas distintas. No início do projecto, foi reunido pelo coordenador, um conjunto geral de requisitos, que a aplicação deveria cumprir, tendo alguns ficado em aberto. Numa fase posterior e quase até ao final da implementação da aplicação, surgiram outros requisitos, por causa de imprevistos ou por iniciativa de colaboradores da empresa, que foram acrescentados ao conjunto já existente;

2. **Análise de requisitos e desenho -** O modelo criado inicialmente foi alterado por dois motivos. Em primeiro lugar, a versão inicial do desenho encontrava-se ainda bastante incompleta e foi, a partir da primeira experiência com a implementação, constantemente actualizada, fruto do incremento permanente de requisitos. O segundo motivo foi uma limitação não prevista, o facto se não ser possível aceder à instância corrente de um construtor, o que obriga a guardar a informação de contexto noutro local. Todos os modelos presentes neste documento, foram elaborados utilizando a linguagem *UML*[1] através da plataforma *Microsoft Visio*[2];

3. **Implementação -** Esta foi a fase mais longa do projecto dada a sua complexidade e dimensão. Inicialmente, foi realizado o construtor, que foi dividido em dois módulos: 1º módulo - a construção total, com a compilação de todos os ficheiros *Didl*, guardar a informação relevante, sobre cada *Didl,* numa estrutura conveniente e a escrita para ficheiro de toda esta informação; 2º módulo - a construção incremental que, baseada na informação obtida na primeira, recompila apenas os ficheiros que tenham sido alterados ou adicionados. O construtor apenas ficou concluído no final da fase de implementação. O desenvolvimento de módulos e ferramentas adicionais foi efectuado em simultâneo, sendo as mais significativas o editor e toda a panóplia de módulos que o acompanham, o gestor de modificações, os assistentes de criação ou importação de *Didls* e os módulos de interface gráfico, com menus de propriedades e de preferências;

4. **Testes -** Cada módulo e ferramenta foi testada após a sua implementação. Para além destes testes unitários e por estes não cobrirem muitas vezes a totalidade das hipóteses de execução, foram elaborados, já numa fase final, um conjunto de testes exaustivos a executar sobre o sistema desenvolvido. Nestes testes o sistema foi submetido a um número elevado de circunstâncias possíveis, na tentativa de filtrar qualquer comportamento inesperado por parte da aplicação. Desempenharam também um papel fulcral, os testes efectuados por outros elementos da empresa, sendo que cada um usou a aplicação de forma diferente, em máquinas diferentes e sobre projectos diferentes, contribuindo para a fiabilidade e coerência do produto final;

5. **Produção -** A parte essencial deste projecto é disponibilizar a aplicação, para todos os utilizadores, de uma forma rápida e simples. Para atingir este objectivo, foi criado

[1] A *UML (Unified Modeling Language)* é uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objectos. Sintetiza os principais métodos existentes e é considerada uma das linguagens mais expressivas para modelagem de sistemas orientados a objectos. Por meio dos diagramas que produz, é possível representar sistemas de software, sob diversas perspectivas de visualização. Facilita a comunicação de todos os actores envolvidos no processo de desenvolvimento de um sistema (gerentes, coordenadores, analistas e programadores).

[2] *Microsoft Visio* é uma ferramenta simples que permite construir gráficos, fluxogramas e painéis técnicos, que permitem visualizar e comunicar ideias, informações e sistemas.

um *update site* num servidor da empresa, a partir do qual pode ser descarregada e instalada a aplicação, dentro do *Eclipse*. Foi elaborada alguma documentação sobre a aplicação (em *JavaDoc)* e o seu modo de funcionamento (ficheiros *leia-me* do editor e do construtor incluídos em anexo). A primeira, foi realizada para auxiliar qualquer colaborador da empresa que, no futuro, queira corrigir ou acrescentar alguma funcionalidade à aplicação. O segundo tipo de documentação foi produzido para ajudar o utilizador da aplicação, embora no caso específico deste software e pela sua natureza automática, não haja muita aprendizagem necessária à sua utilização.

## 2.2 Plano de Trabalho

De acordo com a metodologia descrita em 2.1, nesta secção apresenta-se um plano de tarefas com as respectivas datas de inicio e fim. A tabela 2.1 fornece uma visão descritiva do planeamento e na figura 2.1 apresenta-se um mapa de *Gantt*, que coloca em relação as várias tarefas.

| ID | Etapa | Início | Fim | Duração |
|---|---|---|---|---|
| 1 | Análise de Requisitos | 11-09-2006 | 27-11-2006 | 56d |
| 2 | Revisão de Requisitos | 22-01-2007 | 30-01-2007 | 7d |
| 3 | Desenho e Modelação | 02-10-2006 | 12-12-2006 | 52d |
| 4 | Redesenho Parcial | 01-02-2007 | 07-02-2007 | 5d |
| 5 | Implementação | 23-10-2006 | 02-05-2007 | 138d |
| 6 | Testes | 12-02-2007 | 31-05-2007 | 89d |
| 7 | Produção | 26-03-2007 | 02-05-2007 | 28d |
| 8 | Relatório Preliminar | 02-11-2006 | 24-11-2006 | 17d |
| 9 | Relatório Final | 26-03-2007 | 31-05-2007 | 56d |

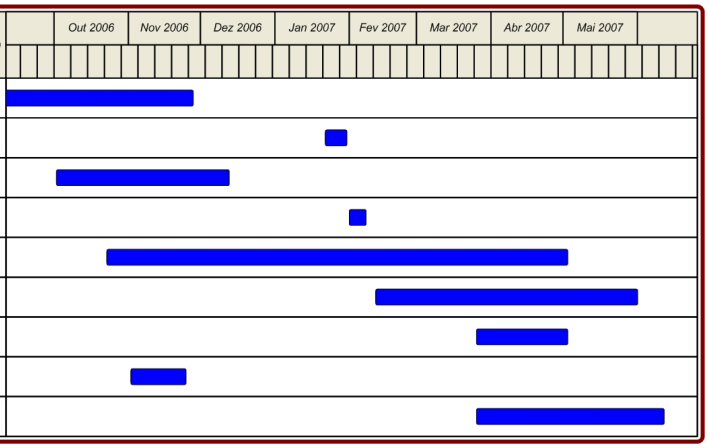


Figura 2.1: Mapa de Gantt

## 2.3 Planeado *versus* Concretizado

Comparando as tarefas planeadas com as que, de facto foram desenvolvidas, constata-se que o conjunto de funcionalidades proposto inicialmente foi implementado na totalidade.

| *Semana* | *Tarefas* |
|---|---|
| *de 11/9 até 22/9 2006* | - Adaptação à empresa: as pessoas, as infra-estruturas, o modo de trabalho;<br>- Instalação e configuração do software na máquina de trabalho;<br>**Recolha inicial e análise de requisitos:**<br>- Primeira aproximação ao problema a resolver: conhecer a plataforma *Domvs* sobre a qual o projecto será desenvolvido e saber quais os objectivos pretendidos pela empresa;<br>- Início do estudo sobre desenvolvimento de plug-ins para a plataforma *Eclipse*. |
| *de 25/9 até 6/10* | - Aproximação mais concreta à plataforma *Domvs* e ao seu modo funcionamento, experimentação do processo de construção, o uso do *Ant* e a geração de código através dos *Didls;*<br>- Estabelecimento de possibilidades e restrições sobre o desenvolvimento de *plug-ins* para a plataforma *Eclipse* face às inovações pretendidas. |
| *de 9/10 até 13/10* | **Desenho da aplicação:**<br>- Elaboração do esboço da aplicação a desenvolver: previsão do nível de automatização que se pode obter na construção, reconhecimento de um projecto *Domvs* pelo *Eclipse*, integração do acesso às novas funcionalidades através de alterações gráficas à plataforma *Eclipse*. |
| *de 16/10 até 27/10* | - Experimentação da integração dos métodos de construção no *Eclipse*;<br>- Correcção da primeira versão do desenho da aplicação e elaboração de uma segunda versão mais pormenorizada. |
| *de 30/10 até 1/12* | **Implementação preliminar:**<br>- Início do desenvolvimento da estrutura da aplicação, conjunto de classes, seus atributos e interfaces principais, implementação de um projecto simples para testes. |
| *de 4/12 até 15/12* | **Testes:**<br>- Testes iniciais à aplicação e ao seu uso em diferentes máquinas e software, regresso aos requisitos e sua consolidação;<br>- Elaboração de uma nova versão do desenho da aplicação. |
| *de 18/12 2006 até 2/3 2007* | **Implementação:**<br>- Desenvolvimento do protótipo da aplicação em toda a sua extensão, correcção de erros encontrados na fase de testes e alterações de acordo com a nova versão do desenho além do retorno obtido. |
| *de 5/3 até 23/3* | **Testes do protótipo:**<br>- Elaboração e execução de testes a fazer ao protótipo, possivelmente com a participação de outros elementos da empresa, para que sejam detectados erros ou o não cumprimento de algum ou alguns requisitos da aplicação. |
| *de 26/3 até 15/6* | - Elaboração do relatório final de projecto;<br>- Empacotamento da versão final da aplicação numa *feature* (conjunto de *plug-ins*) e configuração do *update site* para uma integração mais fácil da mesma;<br>- Correcção de alguns problemas detectados em produção. |

Tabela 2.1: Calendarização de Tarefas

Surge ainda um conjunto de funcionalidades extra, que não estavam incluídas no plano inicial e que foram realizadas. Assim, além da lista enumerada na secção 1.2, foram desenvolvidos os seguintes módulos:

1. Um gestor de modificações dos projectos, que permite adicionar registos de modificações aos projectos, além de gerir lançamentos de versões. Este módulo encontra-se descrito na secção 3.8;
2. Sistemas de sugestões estáticas e dinâmicas, para completar palavras no editor de *Didls*. Tanto para tipos primitivos, como para tipos definidos dentro do próprio ficheiro. Abordados na secção 4.1.2;
3. Estava inicialmente planeado incluir no *plug-in,* o *Domvs-PI* (necessário para efeitos de compilação), o que tornava o bom funcionamento deste software dependente da versão *Domvs-PI* que o acompanhasse. Para resolver esta questão, foi abordada uma solução mais complexa mas também mais eficiente. Esta solução consiste em obter em tempo de execução, um *Domvs-PI* (incluído no projecto ou mesmo descarregando da intranet da empresa), o que incrementa a compatibilidade e e eficiência da aplicação. Tema abordado na secção 3.2.1.

Nos capítulos 3 e 4, encontram-se descrições sobre estes módulos, a sua implementação e o seu funcionamento.

## Sumário

Foi apresentado o planeamento do projecto e o método utilizado para o realizar. Foi efectuada uma comparação entre o trabalho planeado e o realizado, onde se verificou que foram implementadas mais algumas funcionalidades, para além das previstas.

# Capítulo 3

# Construtor `Didl Builder`

O construtor tem como função gerar, a partir de uma série de declarações, um conjunto de interfaces, essas declarações utilizam a sintaxe *XML* no contexto de *IDL*[1]. Este *IDL* é independente da linguagem com que as interfaces são geradas, permitindo por isso comunicação entre linguagens diferentes. Como já foi mencionado, os ficheiros usados para descrever estas interfaces têm o nome de ficheiros *Didl* (de *Domvs Interface Definition Language*) e é essa a origem do nome do módulo.

Para melhor compreender este módulo, não só para efeitos de desenvolvimento mas também para tornar possível a sua documentação, foi dividido em pacotes, consoante o tipo de função que desempenham. Esta divisão pode ser observada na figura 3.1. O pacote principal, que contém todos os outros chama-se `Didlbuilder` e contém a classe `DidlBuilderPlugin`, que é a classe responsável por activar o *plug-in* e pelo seu *ciclo de vida.*

Seguidamente encontram-se as descrições de cada um dos pacotes que implementam as várias funcionalidades enumeradas na secção 1.2 e as outras que surgiram posteriormente. Foi decidido, em alguns casos, acrescentar ao nome das secções, os nomes dos pacotes/classes que as representam, para facilitar a estruturação deste capítulo.

Nesta fase do desenvolvimento foi feito um estudo, primeiro sobre a plataforma *Eclipse*[4], o desenvolvimento de *plug-ins*[7, 8] e, mais concretamente, sobre o desenvolvimento de construtores a adicionar ao *Eclipse*[5]. Em segundo lugar, foi efectuado um estudo sobre o *Domvs*, a sua arquitectura e a forma como é usado. Este estudo foi complementado com experiências com código *Java,* não só no âmbito do *Domvs* e das tarefas *Ant* usadas para o compilar como no dos *plug-ins* do *Eclipse,* com o intuito de perceber até que

[1] *Interface Definition Language* - é uma linguagem de programação usada para descrever interfaces de componentes de software, que descreve estes interfaces de uma forma independente da linguagem, permite por exemplo, a comunicação entre módulos de software implementados com linguagens diferentes (com *Java* e *C++* por exemplo) é utilizada frequentemente em software de *RPC (Remote Procedure Call).*

ponto os objectivos do projecto eram realistas, e ter a noção de quanto tempo seria necessário para implementar cada funcionalidade. Estas experiências passaram por todas as vertentes do projecto: *plug-ins,* o processo de construção, os editores personalizados, as bibliotecas disponíveis e a disponibilidade de *wizards* do *Eclipse* para ajudar na criação destas ferramentas.

A linguagem utilizada para a implementação da aplicação, foi o *Java* já que as duas plataformas em torno das quais o projecto foi desenvolvido são escritas nesta linguagem, além de que o desenvolvimento de *plug-ins* para o *Eclipse* é forçosamente feito em *Java*.

## 3.1 Modelação e Desenho

Como foi referido na secção 1.4, a aplicação desenvolvida pode ser dividida em dois grupos distintos de ferramentas: o construtor `Didl Builder` e o editor `Didl Editor`. Ao contrário do editor, que foi construído a partir de um já existente, o construtor foi desenvolvido de raiz, pelo que são abordados neste capítulo a sua modelação e desenho.

Para revelar uma perspectiva geral e simples deste módulo, é utilizado um diagrama de classes simplificado, presente na figura 3.1. Este modelo contém apenas representados os pacotes de classes mais relevantes, sem qualquer atributo ou método representado. Esta visão simplista do construtor permite compreender a sua estrutura e a forma como os pacotes e as classes se interligam. Os pacotes a tracejado representam sistemas externos à aplicação e as classes neles contidas são as classes que foram desenvolvidas, no âmbito deste projecto.

As secções seguintes descrevem cada um dos pacotes presentes na figura 3.1, tornando possível verificar a forma como a aplicação está dividida e como os seus módulos internos comunicam através do desenho.

## 3.2 Núcleo (`Core`)

O conjunto `didlbuilder.core` constitui o núcleo da aplicação contendo a classe principal `DidlBuilder`, responsável por todo o processo de construção, a classe `DomvsNature`, que atribui a cada projecto a natureza *Domvs* activando o construtor e a classe `BuilderDefaults`, classe estática que guarda todos os valores por defeito usados no *plug-in,* nomes de ficheiros, opções booleanas por defeito, caminhos e pastas.

Segue-se uma explicação para cada classe e as opções de implementação tomadas.

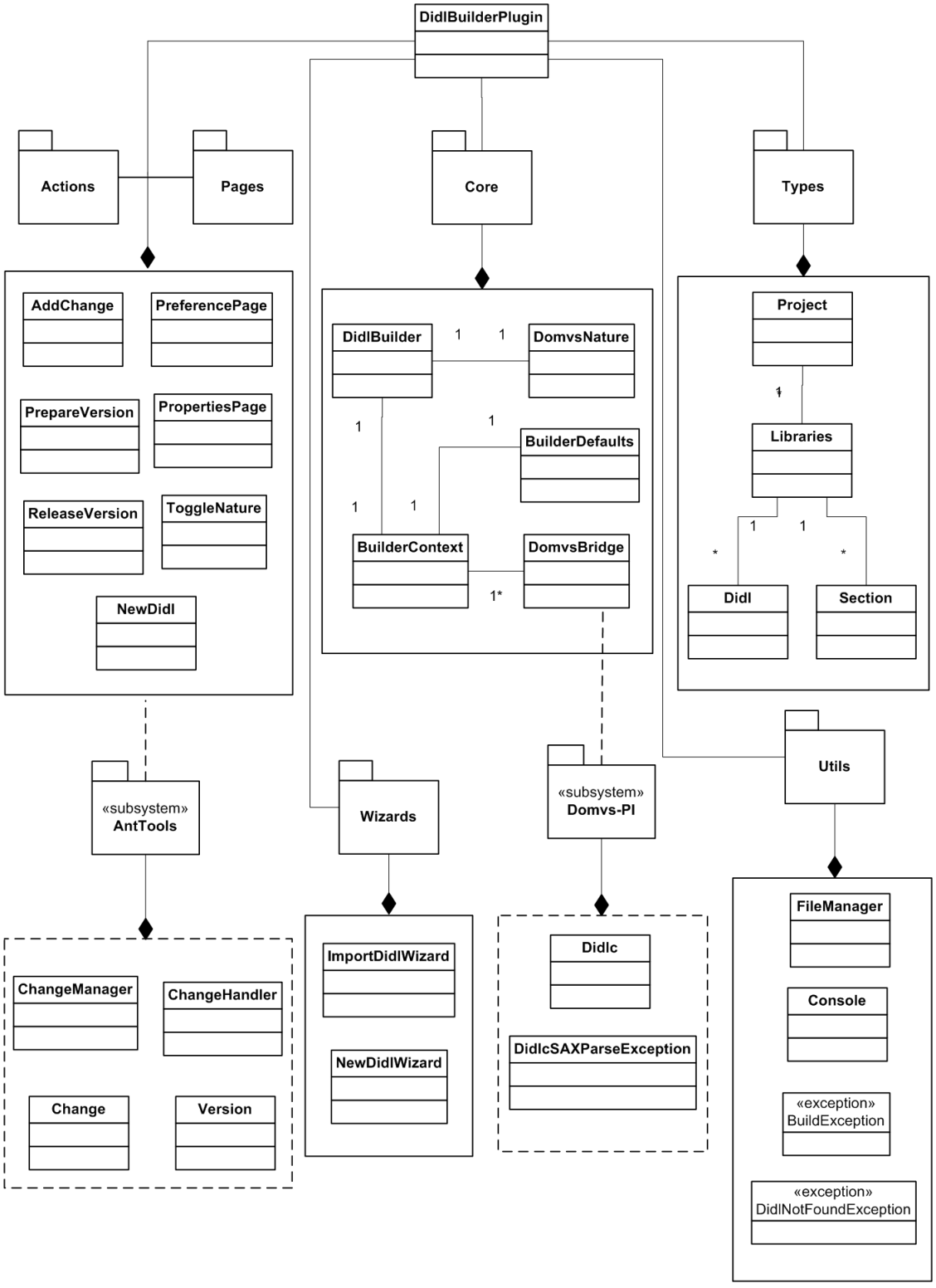


Figura 3.1: Diagrama de Classes Simplificado

### 3.2.1 Ligação ao Domvs (DomvsBridge)

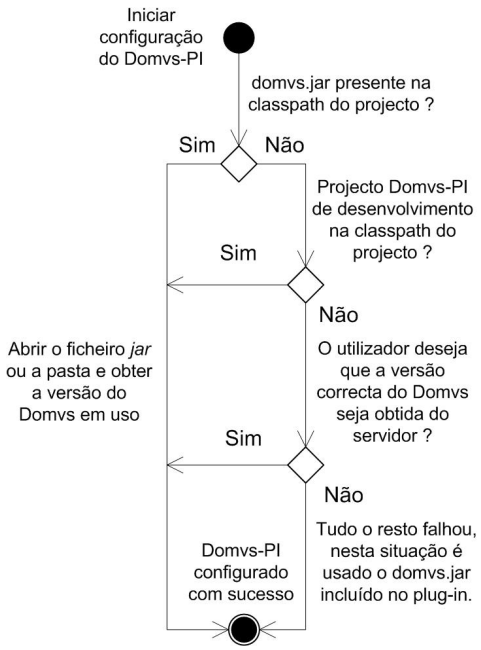


Figura 3.2: Pesquisa do *Domvs-PI*

Como o nome indica, é a classe responsável por fazer a ligação ao *Domvs-PI* do projecto. Embora numa fase inicial o *Domvs* fosse incluído no *plug-in*, esta opção tornou-se inaceitável. No mesmo *Eclipse* é possível ter dois projectos diferentes, que usem versões diferentes do *Domvs* (o que é altamente provável). Neste caso o *plug-in* apenas funcionaria bem com o projecto que tivesse uma versão de *Domvs* igual ou compatível com aquela que é utilizada pelo *plug-in*. Como solução obtém-se, em tempo de execução, o *Domvs* utilizado por cada projecto e compila-se com esse mesmo *Domvs* os ficheiros *Didl,* garantido assim total compatibilidade. Esta ligação é conseguida em dois passos diferentes. Numa primeira fase é a classe `BuilderContext` que configura o *Domvs* (na figura 3.2 é possível verificar as regras pelas quais este é obtido e configurado). Para compilar, é obtida por reflexão *Java*[2] a classe `Didlc`.

No pacote a tracejado *Domvs-PI,* está a classe `Didlc` que já se encontrava desenvolvida e constitui o compilador de *Didls,* que é invocado pelas tarefas *Ant* e também pelo `Didl Builder,` por meio da classe `DomvsBridge`. As alterações efectuadas sobre a classe `Didlc` resumiram-se a alterar o retorno de todos os métodos de geração, para passarem a retornar a lista de interfaces *Java* que geraram. Esta alteração permite apagar os ficheiros gerados a partir de um *Didl*, quando este foi alterado, mantendo as pastas de geração limpas. Esta informação é também escrita para ficheiro, para poder ser usada entre sessões de *Eclipse.*

### 3.2.2 Natureza Domvs (DomvsNature)

Responsável por atribuir a um dado projecto a natureza *Domvs*, activando assim o construtor de ficheiros *Didl* para esse projecto. A acção (secção 3.6) que a activa, é visível quando se chama o menu de contexto de um projecto (botão direito do rato, em cima do projecto) e chama-se *Enable Domvs Nature*, a sua acção resume-se a verificar se o construtor `DidlBuilder` já consta na lista de construtores do projecto e, caso não conste, incluí-lo.

[2]Reflexão ou *reflection* no original, é uma técnica da linguagem *Java,* que permite instanciar classes e invocar métodos através dos seus nomes, um conceito que possibilita a programação dinâmica. Classes, interfaces, métodos, atributos e construtores podem ser descobertos e usados em tempo de execução.

### 3.2.3 Construção (DidlBuilder)

Esta é a classe central de todo o plug-in e é responsável pelo ciclo de construção de todos os projectos. Deste a gestão do arranque, passando pela leitura dos ficheiros internos, que conservam a informação relevante sobre os *builds* entre sessões de *Eclipse*, até à escrita dos mesmos no final, passando pela compilação dos *Didls*. Todas as outras classes são auxiliares desta. Existem dois modos de construção, construção total/inicial ou construção incremental:

#### Construção Total/Inicial (*Full Build*)

É nesta primeira construção que é configurado o projecto, ou seja, é obtida toda a informação necessária para o compilar. Como passo inicial, é configurado o *Domvs* (como foi explicado, em pormenor, na secção 3.2.1). De seguida, são localizados os ficheiros de propriedades com as listas de *Didls* a compilar. O mecanismo de construção do *Eclipse,* na primeira construção de um projecto, fornece todos os recursos do projecto de forma recursiva, permitindo que sejam guardadas as localizações dos ficheiros que interessam. Este mecanismo, por ser recursivo, permite também filtrar à partida a análise de pastas que não são relevantes, como as pastas de geração. Na fase seguinte, os ficheiros *Didl* são compilados e caso as pastas para onde o conjunto de interfaces deve ser gerado, não existam, estas são criadas. Finalmente é gerado um mapa de dependências entre *Didls* e é armazenada a informação relevante em ficheiro.

#### Construção Incremental (*Incremental Build*)

Este é o modo utilizado para construir um projecto, a partir do momento em que este foi construído pelo menos uma vez. Como o próprio nome indica, a construção feita neste modo é parcial, ou seja, são apenas construídos ou reconstruídos os *Didls* que tenham sido criados ou alterados desde o último processo de construção. São também considerados os ficheiros de propriedades onde consta a lista de *Didls*. Se algum destes ficheiros tiver sido alterado, são recompilados todos os *Didls* que nele constam. A recompilação implica a remoção de todos os ficheiros *Java* que foram gerados a partir desse *Didl* e este ser compilado de novo. Esta ordem de eventos encontra-se ilustrada na figura 3.3.

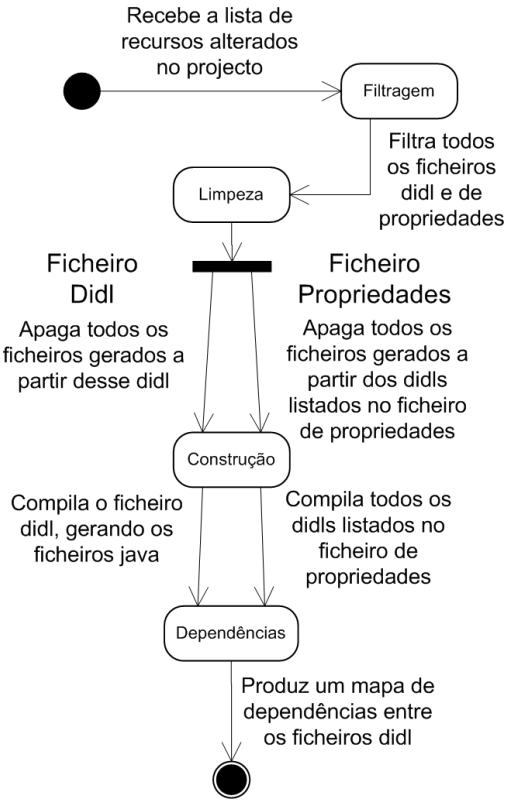


Figura 3.3: Construção Incremental

O mapa de dependências produzido na fase final deste ciclo, permite encontrar (através dos *imports* declarados), para cada *Didl*, quais os *Didls* que dependem dele. Isto permite que, ao recompilar um ficheiro, se saiba quais os que dependem dele, para serem recompilados.

### 3.2.4 Estado e Contexto (`BuilderContext`)

Já na fase de desenho surgiu um problema relacionado com o facto de não se conseguir obter a instância corrente do construtor `DidlBuilder`. Por ser o *Eclipse* a instanciá-lo não é possível tornar o construtor privado, condição necessária para garantir uma classe *singleton*[3], nem obter a instância corrente. Para contornar esta questão, foi criada a classe *singleton* `BuilderContext` para onde foi migrada toda a informação de contexto do processo de construção, ou seja, a informação relativa aos projectos e *Didls*, que deve estar ao alcance de qualquer classe do *plug-in*. Assim, a partir do momento em que qualquer classe do *plug-in* é invocada, esta classe é instanciada e configura, através dos ficheiros internos de projecto, todos os projectos com natureza *Domvs*.

[3] *Singleton* é nome dado a um conhecido padrão de desenho, que consiste em restringir a instanciação de uma classe a apenas um objecto, utilizado normalmente para coordenar dados e/ou acções através de um sistema, garantindo coerência já que cada classe acede à única instância existente.

## 3.3 Tipos (Types)

Este é o conjunto de classes que representam pastas ou ficheiros do sistema de ficheiros, com as suas interligações e lista de atributos.

- `Project` - Representa um projecto presente no espaço de trabalho do *Eclipse* com a natureza *Domvs* ligada e tem como atributos a lista de bibliotecas (`Libraries`), as listas de recursos *Eclipse*[4], a ponte para o *Domvs-PI* (`DomvsBridge`, na secção 3.2.1) e o gestor de modificações do projecto, além de outras informações funcionais como o caminho para o projecto, a versão de *Domvs-PI* que usa, entre outros.

- `Library` - são as bibliotecas do projecto que contêm *Didls*. Estas são configuradas como *source folders* ou pastas fonte no *Eclipse* e contêm listas dos vários ficheiros `Didl` e `Section`, o caminho desde a pasta do projecto e os caminhos fonte e de geração de ficheiros.

- `Didl` - Classe nas instâncias da qual se encontra armazenada toda a informação sobre cada *Didl*, a sua localização, as suas propriedades e a lista de ficheiros gerados a partir dele, além da lista de *Didls* que dele depende.

- `Section` - Tipo especial de *Didl*, que usa o mesmo formato e serve para dividir um ficheiro *Didl* em vários. Por questões de organização. Durante a execução deste projecto, a extensão destes ficheiros, que era *didl* passou a ser *didls,* por ser essencial no *Eclipse* saber distinguir entre este tipo de ficheiro e o anterior.

## 3.4 Utilitários (Utils)

Do grupo das classes utilitárias são de referir:

1. A classe `FileManager` onde acontece toda a escrita e leitura de ficheiros. Existem três tipos de ficheiros geridos pela classe `FileManager`:

   - O ficheiro interno de cada projecto *didlc.info*, contem a informação sobre a última construção, as bibliotecas que existem, os ficheiros *Didl* que contêm e as suas propriedades, as interfaces *Java* gerados por cada *Didl* e o mapa de interdependências;

[4]Ficheiros e pastas do ponto de vista do *Eclipse,* com um conjunto de atributos particulares, que são armazenados, para mais tarde utilizar.

- Nos ficheiros de propriedades *didlc.properties*, *apigen.properties* e *changes.properties* constam, respectivamente, a lista de ficheiros *Didl* a compilar e as suas propriedades, a lista de classes a compilar no final e as propriedades relacionadas com *Changes* e *Releases*;
- Os ficheiros *changes.xml*, que guardam toda a informação sobre *Changes* e *Releases* relativa a cada projecto. No final deste documento, em anexo, encontra-se um trecho deste ficheiro para consulta.

2. A consola de retorno `Console`, que permite ao utilizador receber algum retorno sobre o processo de construção;
3. As excepções utilizadas pelo construtor, `BuildException` e `DidlcSAXParseException`.

As duas excepções implementadas cumprem duas funções distintas. A `BuildException` é uma forma simples de capturar problemas relacionados com o processo de construção e caracterizá-los, com informação relevante sobre o problema. Estas excepções e sua informação são escritas para um ficheiro de log, tornando mais fácil o depuramento de erros quando estes ocorrem noutros computadores que não o de desenvolvimento. A `DidlcSAXParseException` intercepta as excepções retornadas pelo compilador do *Domvs* (o `Didlc` descrito na secção 1.1.2) e acrescenta informação sobre o erro. Neste caso é acrescentado o número de linha e de coluna da ocorrência do erro, tornando possível marcar no editor a linha onde se deu o erro (figura 3.4).

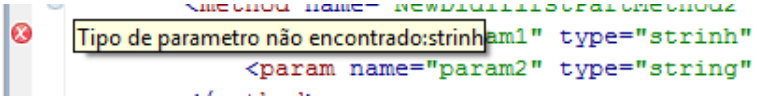


Figura 3.4: Marcação de Erros

## 3.5 Assistentes (`Wizards`)

Para facilitar algumas tarefas auxiliares, relacionadas com a natureza destes projectos, foram elaborados dois assistentes. Um para importar recursos para o projecto e outro para criar recursos específicos.

A criação de um novo *Didl* é feita através do assistente `NewDomvsFileWizard`, que foi implementado e incluído na lista de *wizards* do *Eclipse*. Este assistente aparece no menu *New* do *Eclipse* e na barra de ferramentas principal (figura *3.5*). No ecrã de assistente de criação de novos *Didls* (ver figura 3.6) o utilizador pode seleccionar as propriedades do novo *Didl*, o seu nome e a pasta onde o quer guardar. Após esta definição existe um módulo responsável por escrever esta informação para o ficheiro de propriedades correspondente, que está na raiz da biblioteca em que o *Didl* é criado (*Source folder* para o *Eclipse*). No final, é aberto um editor com o novo *Didl*, que será o editor descrito na secção 4, caso este esteja instalado.

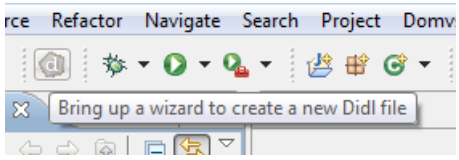


Figura 3.5: Atalho para Criação de *Didls*

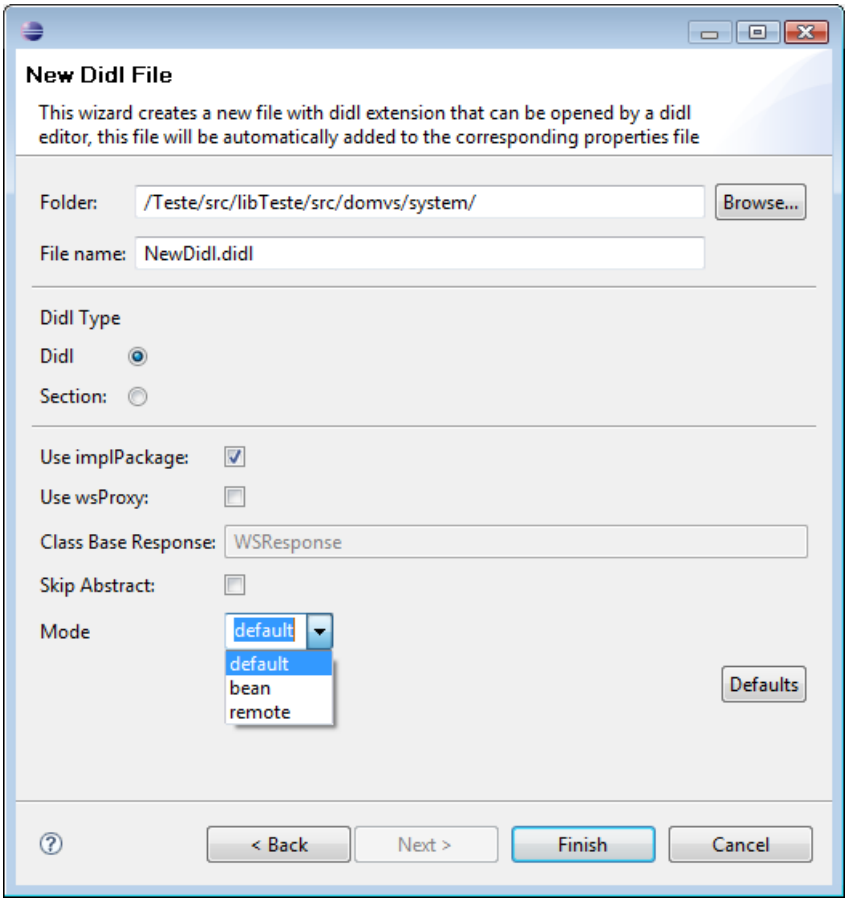


Figura 3.6: Assistente de Criação de Novos *Didls*

Para importar um ficheiro *Didl* deverá ser utilizado o assistente `ImportWizard`, constituído por dois ecrãs. O primeiro permite seleccionar um ficheiro com extensão *Didl* e escolher a pasta de destino dentro do projecto (figura 3.7). O segundo ecrã permite a selecção das propriedades, nome e pasta do *Didl*, através de um menu gráfico e escreve esta informação para o ficheiro de propriedades correcto.

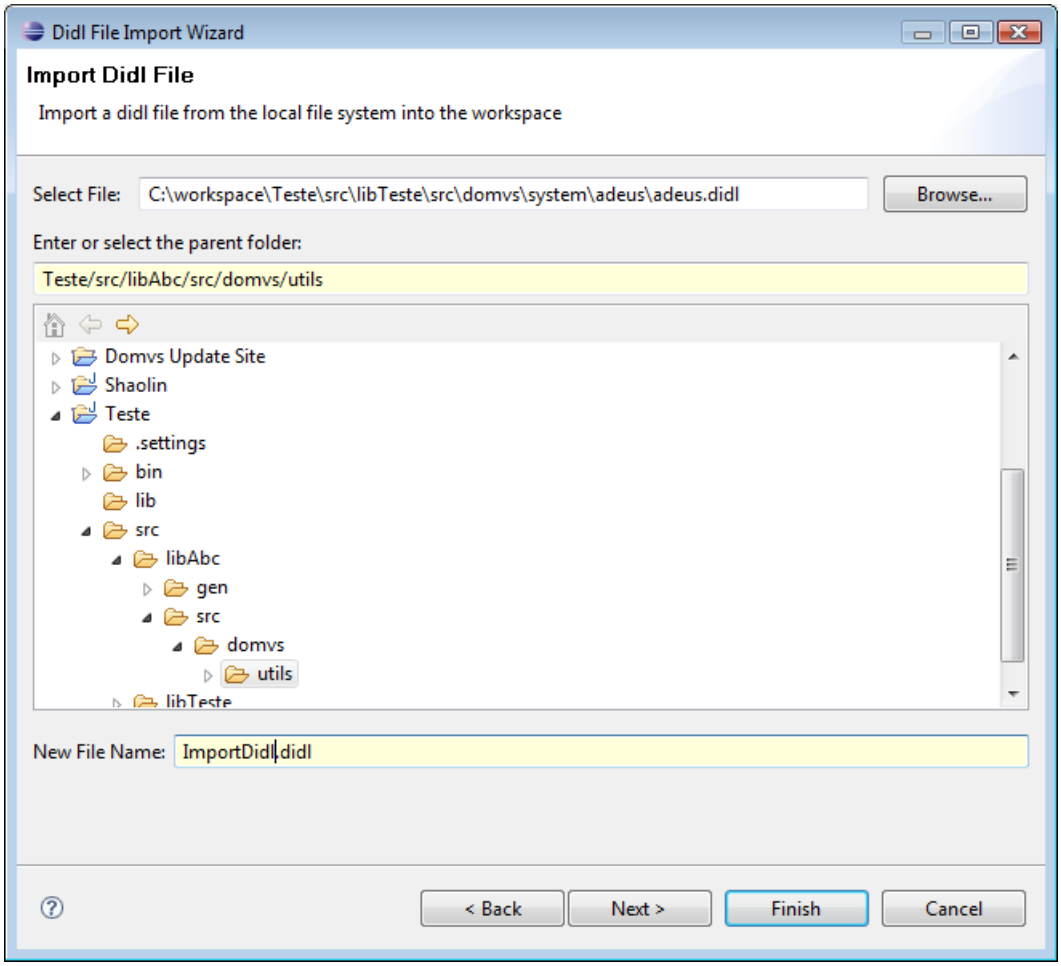


Figura 3.7: Assistente de Importação *Didls*

## 3.6 Acções e Páginas (`Actions` e `Pages`)

Na plataforma *Eclipse*, cada botão de uma barra de ferramentas ou de um menu corresponde a uma acção, o que, na prática, faz corresponder, a cada botão, uma classe que

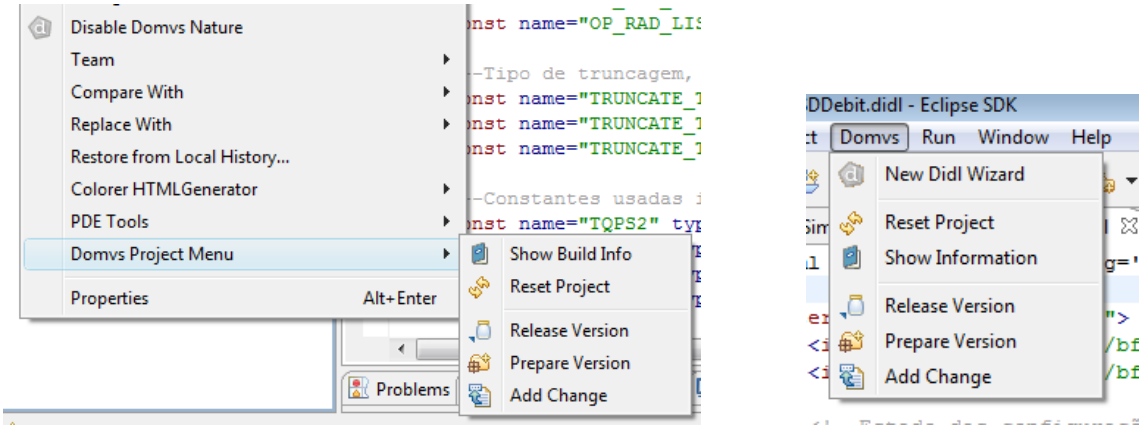


Figura 3.8: Menu *Domvs*
No *pop-up* de um projecto à esquerda e na barra de ferramentas do *Eclipse*, à direita.

implementa a acção que o botão deve executar. Por outro lado, as páginas correspondem aos ecrãs, desde a mais simples notificação, com apenas um botão de *OK,* até ao mais complexo menu de preferências, todos são vistos como "páginas". Assim a uma acção corresponde uma página, pelo que ambos os conceitos encontram-se descritos, em conjunto, nesta secção.

Os pacotes `didlbuilder.actions` e `didlbuilder.pages` contêm quase todas as classes que fornecem a interacção, entre o construtor e o ambiente gráfico do *Eclipse*, fornecendo o retorno e os menus de propriedades ou preferências.

### 3.6.1 Menu Domvs-PI

Este menu surge na barra de ferramentas principal do *Eclipse* e no menu de contexto de um projecto com a natureza *Domvs* (figura 3.8). Disponibiliza as acções e correspondentes páginas para:

- Chamar o assistente de criação de um novo *Didl;*
- Reiniciar o projecto seleccionado, apagando toda a informação de contexto sobre o projecto;
- Mostrar informação sobre o processo de construção do projecto;
- As três acções relativas ao Gestor de Modificações:
  - Adicionar uma modificação;
  - Preparar uma versão;

– Lançar uma versão.

O grupo das três últimas funcionalidades é detalhado na secção 3.8.

Este menu tem a particularidade de desactivar os botões (através de cor acinzentada) quando não é um projecto que está seleccionado ou quando este não tem a natureza *Domvs* ligada. Neste caso apenas o botão do assistente para criar um novo *Didl* se mantém sempre activo e todos os outros dependem da selecção.

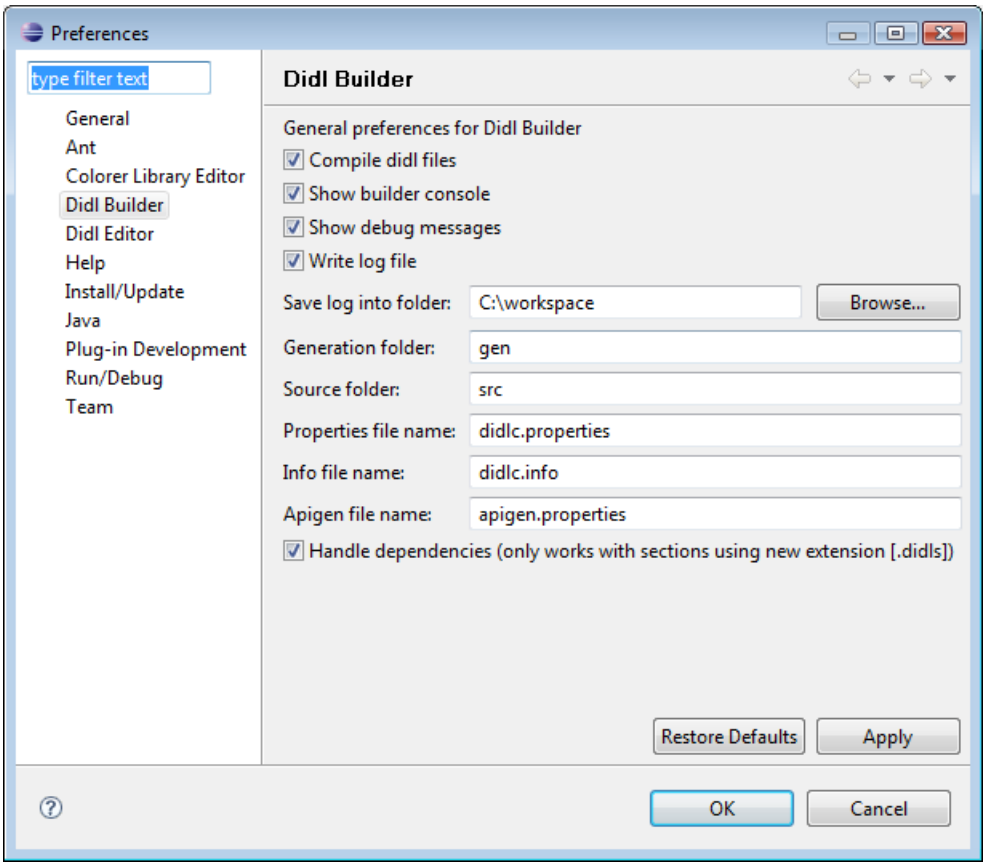


Figura 3.9: Menu de Preferências do Construtor

### 3.6.2 Menus de Propriedades

Este conjunto de ecrãs menu de preferências gerais do construtor (fig. 3.9) e o menu de propriedades dos ficheiros *Didl* (fig. 3.10). Nas preferências gerais permite seleccionar opções globais, como o nome dos ficheiros de configuração, das pastas fonte e de geração, entre outras. No menu de propriedades de um ficheiro *Didl* permite escolher opções que influenciam a forma como o ficheiro é compilado e o que é gerado a partir dele.

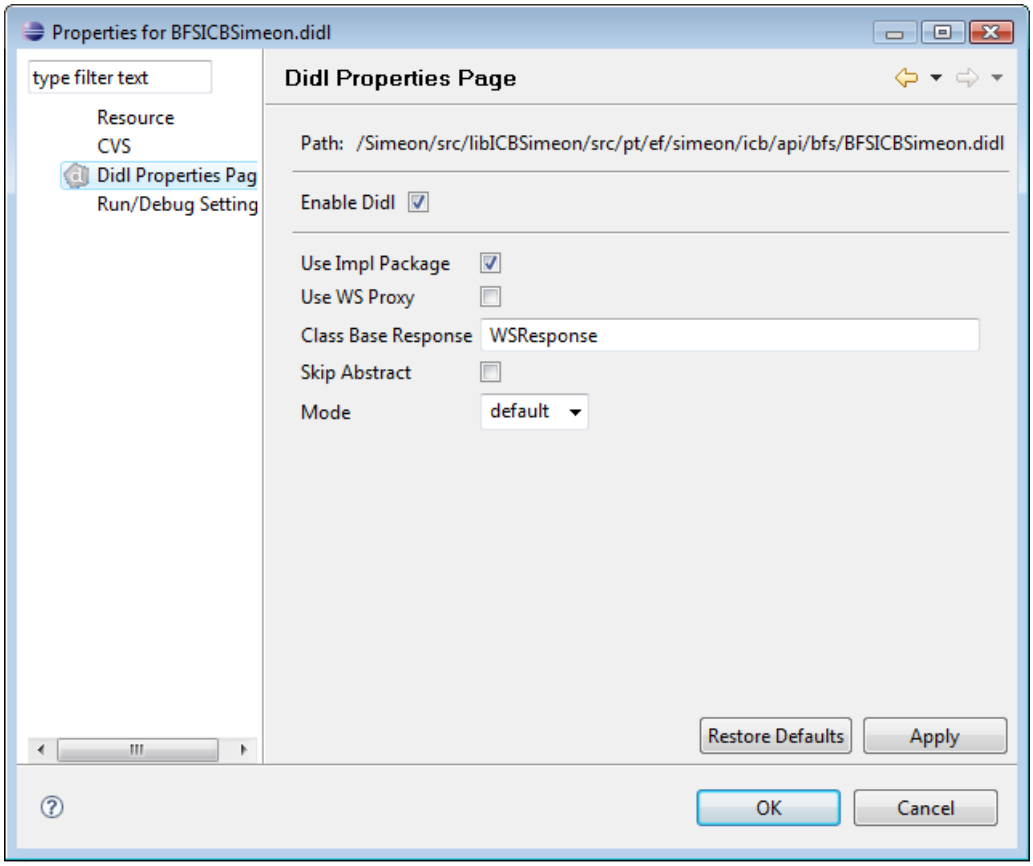


Figura 3.10: Menu de Propriedades dos *Didls*

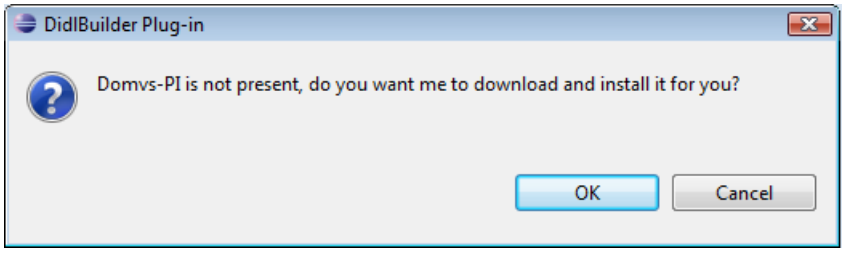


Figura 3.11: Exemplo de um *Pop-up*

Outras mensagens de informação ou perguntas realizadas ao utilizador utilizam também classes deste tipo, embora em versões mais simples. Um exemplo deste género de páginas está ilustrado na figura 3.11.

## 3.7 Coerência no Estado da Construção

Uma das maiores dificuldades sentidas no desenvolvimento do construtor, foi manter a coerência do estado do construtor, entre sessões de *Eclipse.* Quando um utilizador inicia o *Eclipse*, depois de já ter construído projectos noutras sessões, deve poder utilizar qualquer funcionalidade fornecida pela aplicação, incluindo a construção incremental. Para isto ser possível e já que algumas funcionalidades necessitam de informação sobre a última construção do projecto, é necessário que toda essa informação seja escrita para ficheiro e lida, quando o *plug-in* arranca, ou seja, quando uma das suas classes for chamada. Manter este estado, sem utilizar demasiados recursos, constituiu um trabalho de alguma complexidade. É necessário haver um compromisso entre a quantidade de informação guardada e o tempo despendido para mantê-la coerente, por forma a não tornar a aplicação demasiado "pesada".

Como já foi discutido neste capítulo, não é possível obter a instância corrente do construtor. Por esta razão foi criada uma classe para gerir toda a informação referente à construção (a classe `BuilderContext`). Esta classe apenas pode ser instanciada uma vez, logo no arranque do *plug-in*, como foi explicado na secção 3.2.4. A partir desse momento todas as classes que necessitem de informação relativa à construção consultam essa instância, mantendo a coerência de estado entre sessões de *Eclipse*. Esta manutenção do estado corresponde a actualizar e a guardar em ficheiro, toda a informação relativa a ficheiros *Didl*, como o caminho para a sua localização dentro do projecto, as suas propriedades, a lista de outros *Didls* de que depende e a lista de ficheiros *Java* gerados a partir dele, após cada processo de construção.

## 3.8 Gestor de Alterações

O pacote de funcionalidades *AntTools*, cuja relação com os restantes pacotes pode ser visualizado na figura 3.1, é um conjunto de ferramentas já se encontrava desenvolvido. É um conjunto de tarefas *Ant,* que cumprem funções auxiliares e cuja função não se considera relevante para este documento. O módulo da gestão de alterações foi incluída neste pacote de software, para permitir a sua utilização a partir da consola. Como este conjunto de ferramentas (o *AntTools*) é um *jar* independente do *Domvs-PI,* com dimensões reduzidas, foi decidido inclui-lo no *plug-in*, contendo o Gestor de Alterações, em vez de inserir este gestor directamente no *plug-in*, como inicialmente foi planeado.

Este módulo não estava inicialmente previsto, tendo surgido já numa fase avançada do projecto. É completamente independente do resto do software, pelo que não foi necessário alterar nada nos módulos já existentes, para incorporá-lo. A razão para incluir este con-

junto de ferramentas neste projecto, passa pelas vantagens de haver um interface gráfico para utilizá-lo e por ser aplicável aos mesmos projectos que os restantes módulos.

### Contexto

Sempre que algum colaborador da empresa realiza alterações ou inserções num projecto, regista a alteração num ficheiro de texto. Quando já existem algumas alterações pendentes ou surge uma alteração que o justifique, é lançada uma nova versão com o conjunto de alterações pendentes. Este processo tem alguns problemas. Um deles é a possibilidade de cada colaborador registar as alterações à sua maneira. Outro problema é a geração de um documento que informe o cliente das alterações e novidades, incluídas na nova versão. Para que este documento seja gerado automaticamente, o ficheiro de alterações tem que respeitar um formato rígido. Não é o caso de um ficheiro de texto.

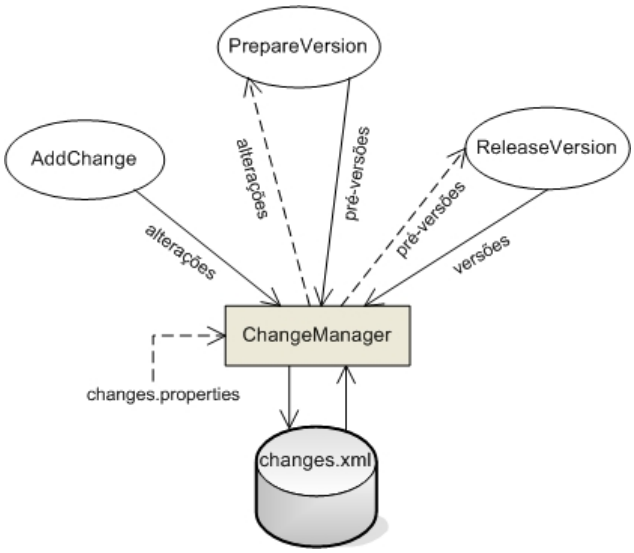


Figura 3.12: Funcionamento do Gestor de Alterações

Para melhorar todo este sistema, implementou-se um gestor com interface gráfico, que facilita a gestão e exige que o ficheiro de registo tenha um formato rígido. Foi implementado um conjunto com três menus, um para cada passo do processo. Esta funcionalidade foi incorporada no já existente projecto *AntTools*, para permitir que, para além de ser incorporado no *plug-in* possa ser utilizado na linha de comandos.

### Funcionamento

Este módulo começa por carregar as propriedades específicas de cada projecto que existe no espaço de trabalho do *Eclipse*. Se o projecto não contém este ficheiro (*changes.properties*), é copiada uma versão *standard* do mesmo, para a raiz do projecto. O conteúdo deste ficheiro especifica as opções que aparecem no menu para acrescentar alterações. No final deste documento, em anexo, encontra-se o conteúdo do ficheiro *standard*, para consulta. Em paralelo, é carregado o conteúdo do ficheiro *changes.xml*, que contém todas as alterações, pré-versões e versões do projecto. Em anexo, encontra-se um exemplo do conteúdo deste ficheiro, para verificar a estrutura do mesmo.

As três acções que podem ser executadas com este gestor são: a criação de uma nova alteração *(AddChange)*; a preparação de uma versão (*PrepareVersion);* e o lançamento de versões (*ReleaseVersion*). O acesso a estas funções encontra-se no menu *Domvs* (figura 3.8), disponível quando um projecto de natureza *Domvs* está seleccionado. Um esquema deste gestor pode ser consultado na figura 3.12.

### 3.8.1 Adicionar uma Alteração

Sempre que um colaborador da empresa faz uma alteração a um projecto, deve, através do menu `AddChange`, registar a alteração. Para registar essa alteração tem que cumprir algumas restrições, ou seja, se algum campo não for preenchido com o formato adequado, ou não estiver preenchido, sendo obrigatório, a alteração não pode ser adicionada. Na figura 3.13 podemos ver o menu em questão, preenchido com os dados de uma alteração fictícia, a título de exemplo. O utilizador deverá introduzir o tipo e a origem da alteração, a partir de um um conjunto pré-estabelecido, uma referência que apenas é pedida em casos específicos (configurado no, já mencionado, ficheiro de propriedades) e o nome de quem fez o pedido da alteração e a data respectiva. Existe uma caixa de texto livre para introduzir a descrição da alteração e, no final, um campo opcional para notas, que são habitualmente um aviso sobre a necessidade de algum tipo de actualização. Por exemplo, escolhendo a nota *JDK*, surge uma nota no texto da alteração a mencionar que a versão do *Java* deve ser actualizada (também estas opções e o seu texto correspondente são configuráveis a partir do ficheiro de propriedades *changes.properties*).

### 3.8.2 Preparar uma Versão

Anteriormente quando uma versão era lançada, eram simplesmente agregadas a essa versão, todas as modificações pendentes (ainda não incluídas em nenhuma versão). O conceito de preparar uma versão surge, com a finalidade de melhor gerir as modificações, tornando possível escolher quais as modificações que devem ser incluídas numa dada versão e quais as que devem continuar pendentes, além de escolher o sub-projecto (se o houver) em questão. Na figura 3.14 pode ser visto este menu, com as várias alterações pendentes. A opção de *clean* para remover a alteração da lista de pendentes, o sub-projecto alvo e um pequeno sistema de pesquisa, para facilitar a busca quando existem várias alterações em espera.

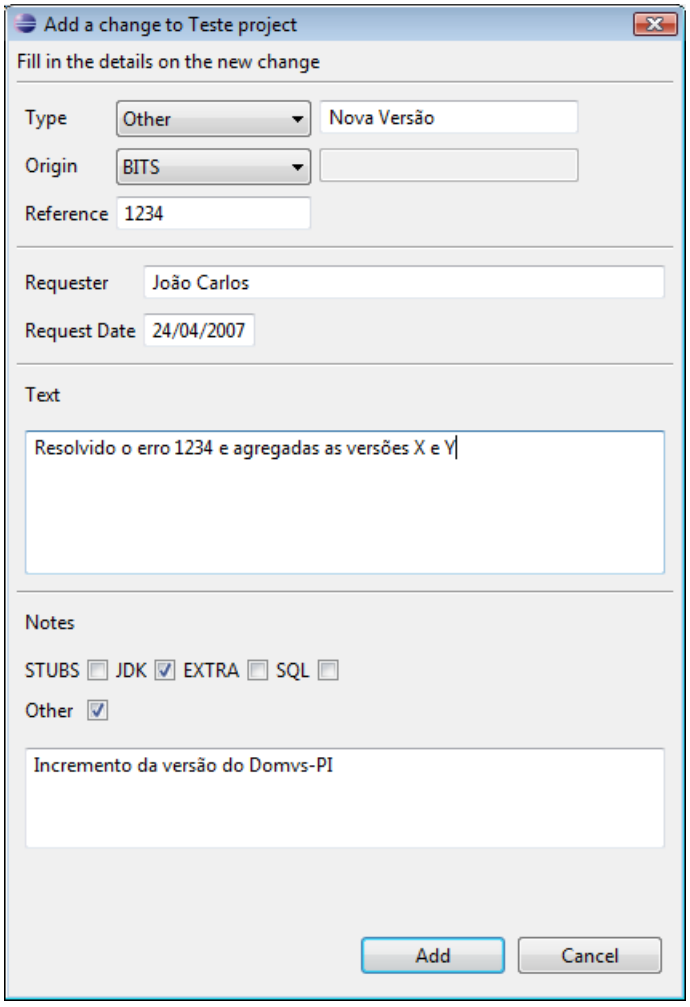


Figura 3.13: Criar uma Nova Alteração

### 3.8.3 Lançar uma Versão

Lançar uma versão é a última fase deste processo. Para tal basta escolher entre as pré-versões disponíveis e quais os seus números de versão. Assim, uma nova versão é registada no ficheiro *changes.xml*, as pré-versões desaparecem e as respectivas alterações marcadas com *clean pending* são retiradas da lista de pendentes. Na figura 3.15, pode-se observar um exemplo do lançamento de uma versão. Os números de versão que são sugeridos são calculados a partir de um ficheiro, que define qual a principal biblioteca do projecto em questão, cuja localização deve ser anotada no ficheiro *changes.properties* do projecto. Este número de versão pode ser editado, já que o número que surge é apenas um incremento à micro versão corrente (no caso do *Teste - Particulares* a versão corrente seria 1.0.2).

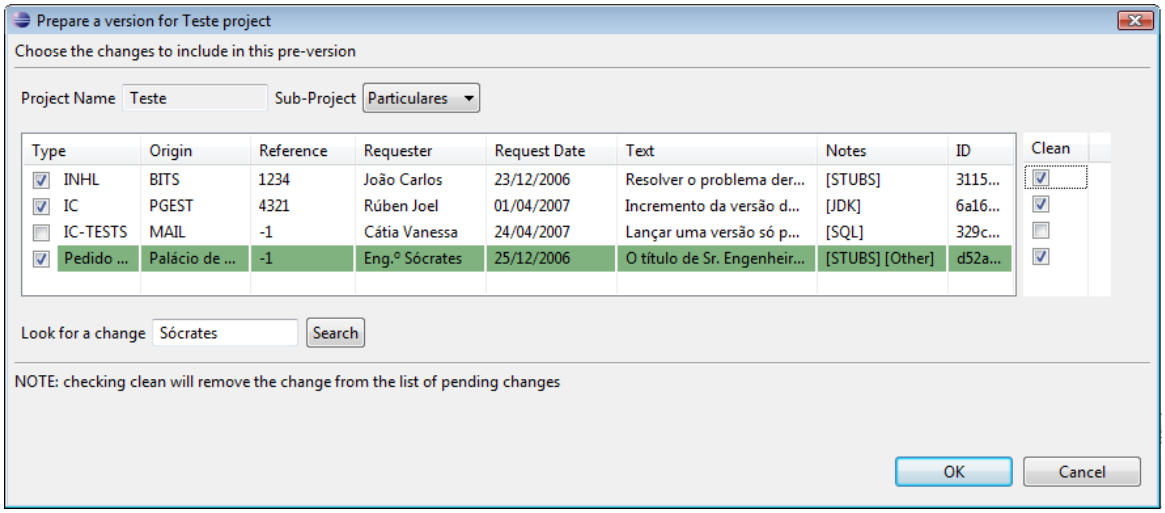


Figura 3.14: Preparar uma Versão

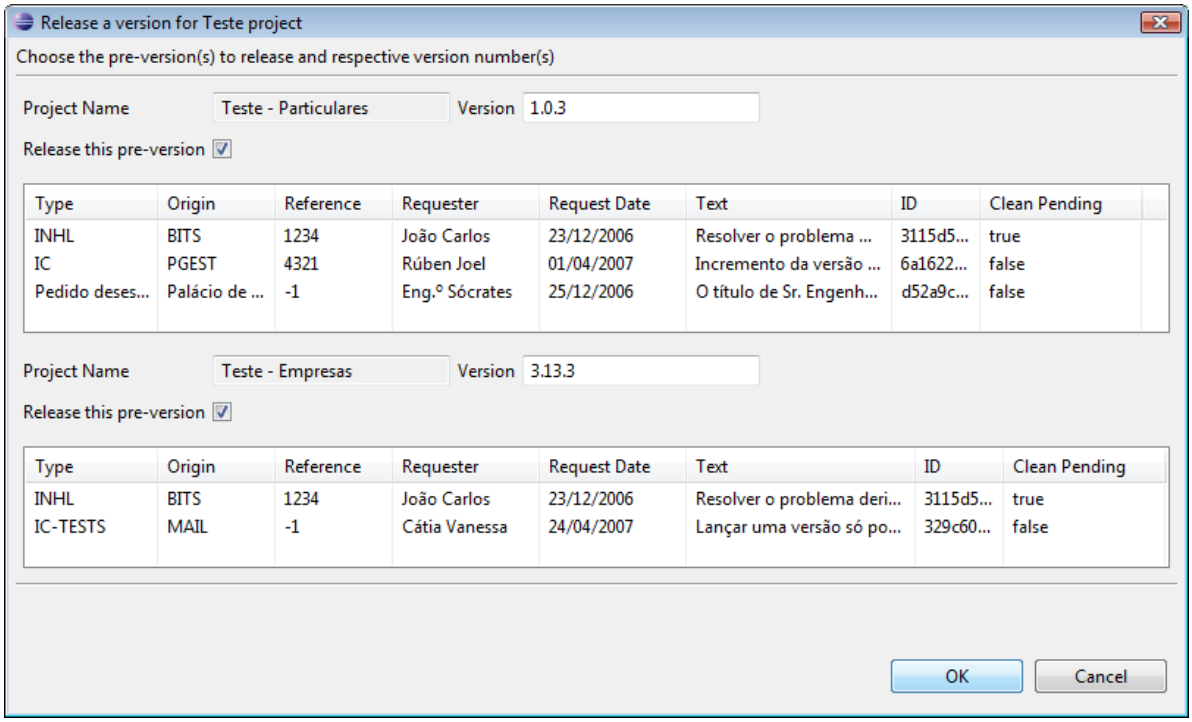


Figura 3.15: Lançar uma Versão

### Utilização

O gestor foi incluído no pacote *AntTools*, onde já constava uma tarefa de *ant* para lançar versões, que foi modificada para utilizar este gestor. Este pacote existe em todos os projectos de natureza *Domvs*, e foi agora incluído no *plug-in* `Didl Builder`, pelo que pode ser utilizado de duas formas. A utilização completa só é possível através do *plug-*

*in,* tendo acesso aos três menus apresentados. O lançamento de versões pode também ser executado numa consola, basta que existam alterações pendentes ou pré-versões associadas ao projecto em questão.

## Sumário

Neste capítulo foi descrita a primeira abordagem ao problema e a maneira como foi elaborado o desenho de uma solução, além de revelar as divisões entre os principais módulos que foram desenvolvidos. Foi também apresentado um diagrama de classes simplificado do construtor, com os pacotes que o compõem e as relações entre as principais classes. Foram utilizados alguns esquemas para simplificar o entendimento dos fluxos envolvidos e algumas imagens de menus do *Eclipse* elaborados. Foi descrito o núcleo da aplicação, que coordena todo o processo de construção desde o arranque do *Eclipse,* os tipos definidos para gerir os recursos de cada projecto, a ligação ao *Domvs-PI* obtida dinamicamente, os assistentes que permitem incluir ou criar recursos específicos e no final, foi apresentado o gestor de alterações.

# Capítulo 4

# Editor `Didl Editor`

De modo a facilitar e apoiar a edição dos ficheiros *Didl* foi construído um editor próprio. Após um período de experimentação de vários editores XML foi escolhido o editor *Amateras[9].* De facto este editor possui um conjunto de características que se revelaram importantes na escolha, nomeadamente: os menus de preferências, o *outline* e o código fonte estar disponível no *SourceForge[10].*

Inicialmente o editor foi implementado como sendo um módulo extra do *Amateras*, uma vez que este permite editar, entre outros, ficheiros *html* e *jsp.* De facto sendo estes tipos de ficheiros usados na maioria dos projectos, fazia sentido aproveitar. Por motivos de compatibilidade com os outros editores e *plug-ins*, esta opção acabou por não ser considerada, tendo ficado apenas o editor de *Didls*, implementado sobre o mecanismo já existente para ficheiros *XML,* não sendo aproveitadas as restantes funcionalidades do *Amateras.*

Este editor é constituído por dois componentes principais ou módulos: o editor de conteúdo propriamente dito, onde se edita o ficheiro, que para além da sintaxe colorida, apresenta a funcionalidade de sugestões automáticas e a vista estrutural, onde se tem uma vista mais global e hierárquica do conteúdo do ficheiro, e que é actualizada sempre que o ficheiro é alterado. Para além destes dois módulos, foi incluído um menu de preferências, uma adaptação do menu original do *Amateras,* tal como pode ser observado na figura 4.1.

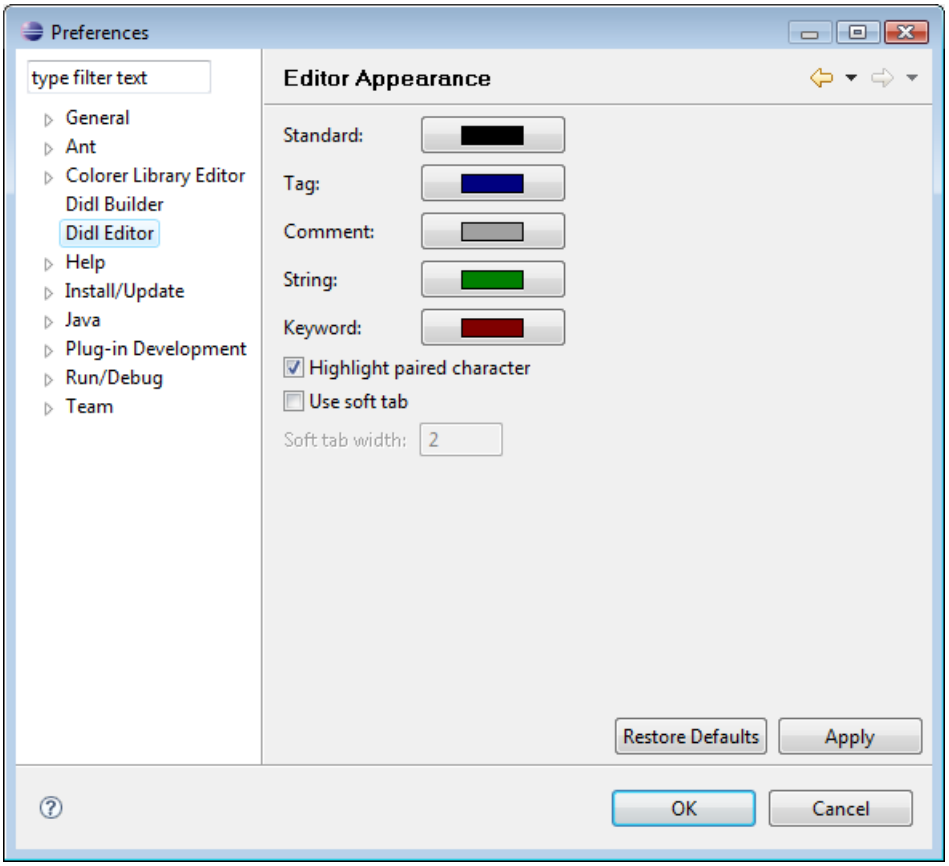


Figura 4.1: Menu de Preferências do Editor

## 4.1 Editor de Conteúdo

O editor de conteúdo foi construído a partir do editor *Amateras* acrescentando as seguintes funcionalidades:

- Coloração de palavras chave;
- Menu de preferências.

Relativamente à coloração foi acrescentada uma nova cor ao conjunto já disponível, para colorir o conjunto de palavras chave da linguagem *Didl,* ou seja, as palavras *interface, method, structure*, entre outras. De seguida foi adicionado ao menu de preferências do editor, o campo correspondente, visível na figura 3.10, em *keyword.* A figura 4.2 mostra o efeito global deste editor.

Didl.didl

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<interface name="Didl">

        <method name="DidlfirstPartMethod1" result="void">
             <param name="param1" type="string" optional="false"/>
             <param name="param2" type="string" optional="false"/>
         </method>

         <method name="DidlfirstPartMethod2" result="void">
             <param name="param1" type="string" optional="false"/>
             <param name="param2" type="string" optional="false"/>
         </method>

         <method name="DidlfirstPartMethod3" result="DidlFirstPartStruct1">
             <!-- param 1 -->
             <param name="param1" type="string" optional="false"/>
                         <param name="param3" type="string" optional="false"/>
             <!-- param 2 -->
             <param name="param2" type="string" optional="false"/>
             <!-- Isto é uma exception -->
             <exception type="Didlexception1" />
         <!-- Teste 1 2 3 -->
         </method>

         <structure name="DidlFirstPartStruct1">
             <!-- Constante -->
             <const name="const1" type="int">0</const>
             <!-- Variável 1 -->
             <var name="var1" type="int"/>
             <var name="var2" type="int"/>
             <var name="var3" type="int"/>
         </structure>

         <exception name="Didlexception1" />

</interface>
```

Figura 4.2: Editor de *Didls*

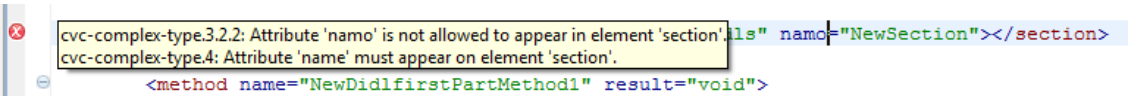


Figura 4.3: Verificação Sintáctica com *XSD*

Figura 4.4: Sugestões para Elementos Base

Foi aproveitado um mecanismo existente no editor *Amateras* para gerar um ficheiro *XSD*[11][1], com o conjunto de regras que define um ficheiro *Didl* válido, este ficheiro foi gerado a partir de um conjunto grande de *Didls,* onde são usadas todas as possibilidades e variações da linguagem. Depois de criado este ficheiro, foi incluído no *plug-in*, com duas finalidades diferentes, correcção sintáctica anterior ao processo de construção e sugestões para completar elementos estáticos, funções que serão abordadas de seguida.

### 4.1.1 Correcção Sintáctica

Com o conjunto de regras definido no ficheiro *xsd* descrito, foi utilizado o mecanismo que verifica a sintaxe *xml* logo no instante em que o documento é modificado e guardado. Este mecanismo permite alertar o utilizador caso este cometa algum erro, no instante em que guarda o documento e não após o processo de construção, que pode ser demorado. Alguma informação sobre o erro é mostrada, quando o utilizador faz o ponteiro do rato pairar sobre a marca do erro, visível na figura 4.3.

Existe um conjunto de erros que são detectados apenas pelo compilador. Os erros de natureza semântica surgem apenas após a compilação e neste caso, embora a maneira como são apresentados seja semelhante à dos anteriores, a sua implementação é diferente. No caso anterior, o mecanismo já acompanhava o editor *Amateras*, bastando o ficheiro *xsd* ser gerado correctamente e fazer breves alterações ao código. No caso de erros semânticos foi necessário implementar um mecanismo que se encontra incorporado no construtor (ver secção 3.4).

### 4.1.2 Sugestões Automáticas

Qualquer utilizador habitual do *Eclipse* (assim como de outros *IDE's*), está familiarizado com uma das suas principais funcionalidades, as sugestões para completar palavras. Es-

[1] *XSD* ou *XML Schema* é uma linguagem baseada no formato *XML,* para definição de regras de validação ou esquemas, em documentos no formato *XML*. Foi a primeira linguagem de esquema para *XML* a obter o estatuto de recomendado por parte do W3C. Esta linguagem é uma alternativa ao *DTD (Document Type Definition)*, cuja sintaxe não é baseada no formato *XML*.

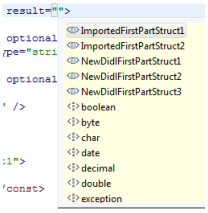


Figura 4.6: Sugestões para Tipos

tes mecanismos funcionam sugerindo, ou palavras chave que são estáticas e fazem parte de um conjunto estipulado à partida, ou dinamicamente guardando tipos e/ou métodos definidos pelo utilizador. Inicialmente é sugerido todo o conjunto de palavras disponíveis, restringindo esse conjunto à medida que o utilizador digita caracteres, verificando de entre as palavras disponíveis quais as que começam com os caracteres já escritos.

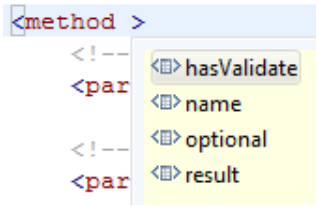


Figura 4.5: Sugestão de Atributos

As sugestões estáticas utilizam a informação contida no ficheiro *xsd* para escolher as palavras a sugerir. Assim, através das teclas que accionam as sugestões (*Ctrl + Tab* ou '<'), o utilizador pode ver a lista de todos os elementos que se enquadram no presente contexto, em relação à posição do cursor. Com a tecla '<' surgem os vários elementos disponíveis, como se pode constatar pela figura 4.4, se escolher *const*, por exemplo, o resultado é o dito elemento com os seus atributos obrigatórios, prontos a preencher. Se a sugestão for pedida com um elemento aberto, surge o conjunto de atributos que esse elemento pode conter, figura 4.5.

As sugestões dinâmicas são semelhantes às estáticas, no sentido em que são accionadas da mesma forma e surgem da mesma forma, diferem na maneira como são obtidas. Embora exista um conjunto estático de tipos, chamados de tipos primitivos, outros são obtidos dinamicamente. Quando o ficheiro é aberto, é produzida uma lista de estruturas definidas dentro do ficheiro e nos ficheiros importados ou secções, através de uma análise do conteúdo do ficheiro, que é guardada em *cache.* Sempre que o utilizador acrescenta uma estrutura ao ficheiro e o guarda, esta é acrescentada à lista que já existe, em *cache,* e surge quando o utilizador voltar a utilizar o sistema de sugestões. Na figura 4.6 está um exemplo onde surgem os tipos primitivos e as estruturas disponíveis.

## 4.2 Vista da Estrutura *(Outline)*

Esta ferramenta, que acompanha a maior parte dos editores, é de grande utilidade, especialmente quando o ficheiro editado é grande, pois disponibiliza uma vista global da

estrutura do ficheiro. Se a linguagem em questão é de alguma forma hierárquica, essa hierarquia é também visível, permitindo navegar com mais facilidade, como se de um índice se tratasse.

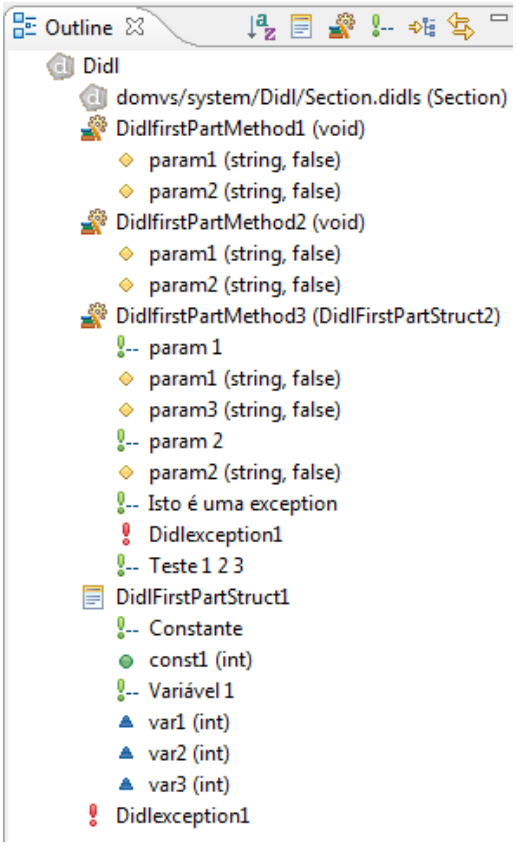


Figura 4.7: Vista de Estrutura

Embora o editor *Amateras* já disponibilize uma vista deste tipo para ficheiros *xml*, foram necessárias muitas alterações para obter a vista que se encontra na figura 4.7. Foi aproveitado o mecanismo existente, que já geria um conjunto de nós pais e filhos, numa árvore hierárquica, mas foi alterado todo o visual e acrescentadas algumas funcionalidades. Através da implementação de uma barra de ferramentas, visível no topo da figura 4.7, foi disponibilizado o seguinte conjunto de funcionalidades:

1. Utilização de um conjunto de ícones novos, mais adequados para este contexto;
2. Sincronização entre o editor e a vista hierárquica e vice-versa, ou seja, seleccionando um elemento na vista, é automaticamente seleccionada a zona do editor correspondente e vice-versa, o primeiro ícone a partir da direita liga e desliga esta função;

3. Expansão da árvore na sua totalidade. Por defeito apenas um nível é expandido (segundo botão a partir da direita);
4. Visualização dos comentários na vista hierárquica (terceiro ícone a partir da direita);
5. Filtragem de elementos, que permite que se visualize apenas métodos ou apenas estruturas, ou ambos (quarto e quinto botões a partir da direita, respectivamente);
6. Ordenação alfabética dos itens da vista, para facilitar a visualização de ficheiros mais complexos (último ícone).

## Sumário

Neste capítulo foi descrito o trabalho executado, no âmbito do editor de ficheiros. Não foram consideradas algumas características interessantes do editor original *Amateras*, por estas já estarem implementadas no editor original. Assim, após uma breve introdução ao editor que serviu de ponto de partida, foram descritas as alterações feitas no menu de preferências, no editor de conteúdo e na vista hierárquica e o sistema de sugestões dinâmicas.

# Capítulo 5

# Sumário e Conclusões

Este projecto foi realizado no âmbito da disciplina de Projecto em Engenharia Informática (PEI), na empresa *EF Tecnologias de Software, SA* com o propósito de obter o grau de Mestre em Engenharia Informática (MEI). Permitiu travar conhecimento com o universo profissional, na área da engenharia informática, concretamente com o desenvolvimento de software e todas as fases envolvidas neste processo.

Constituiu uma experiência enriquecedora e aliciante, não só porque apresentou a oportunidade de percorrer todas as fases de desenvolvimento de uma aplicação, com alguma liberdade de concepção, como também por ser uma plataforma de desenvolvimento muito utilizada, o *Eclipse IDE*. Foi também adquirida uma perspectiva mais real das relações hierárquicas dentro de uma empresa e do cumprimento de horários e prazos.

De seguida encontra-se um sucinto sumário do trabalho realizado, algumas menções sobre trabalho futuro a realizar sobre a aplicação e no final, uma breve conclusão.

## 5.1 Sumário do Trabalho Realizado

Os objectivos inicialmente propostos e planeados foram totalmente concretizados e foram ainda efectuadas algumas tarefas inicialmente não previstas.

Realizei o que estava planeado e ainda alguns componentes e funcionalidades extra, que não estavam incluídas no plano inicial. Foi implementado um construtor incremental, com editor e um conjunto de ferramentas auxiliares que tornam o seu uso mais rápido e agradável. Foi utilizado como modelo, o construtor de *Java* do *Eclipse,* não só porque é bastante intuitivo como pelo hábito que já existe, entre informáticos, com as suas funcionalidades. Foram executados testes sobre a aplicação, em conjunto com alguns elementos da empresa, ao longo das últimas semanas, o que contribuiu para uma ferramenta mais completa e fiável, fácil de obter e de configurar.

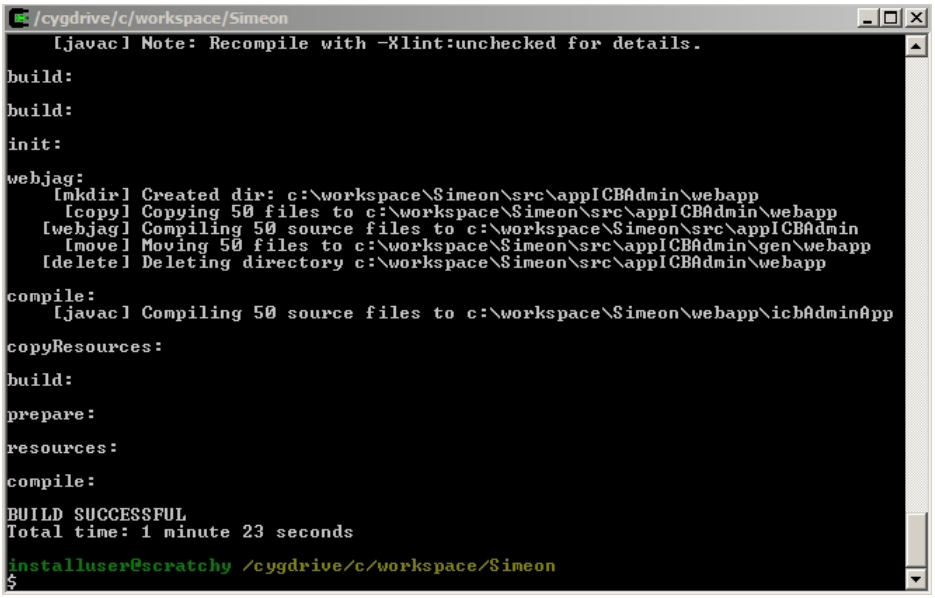


Figura 5.1: Construção Incremental com o *Ant*

## 5.2 Impacto da Aplicação

Na fase final do desenvolvimento, alguns colaboradores da empresa instalaram a aplicação nos seus computadores. A partir das experiências que efectuaram, surgiram pequenos problemas a resolver e sugestões de funcionalidades a inserir ou alterar. Os problemas que surgiram ajudaram a depurar mais rapidamente a aplicação, contribuindo para a estabilidade e fiabilidade da versão final. Algumas sugestões originaram botões na vista de estrutura do editor (figura 4.7), outras alteraram o modo como o construtor procura o *Domvs-PI* (figura *3.2).* Estas sugestões, por partirem de um ponto de vista de utilizador, tornaram a aplicação mais completa e utilizável.

Uma forma de fazer uma comparação directa entre, o modo de construção usado anteriormente e o executado por esta aplicação, é alterar um ficheiro *Didl* e construir o projecto utilizando os dois métodos, verificando o tempo ocupado por cada um. Nas figuras 5.1 e 5.2 podem ser observados os tempos levados por cada um dos métodos, o teste foi executado sobre o mesmo projecto, alterando o mesmo ficheiro *Didl.* A construção feita dentro do *Eclipse* é bastante mais rápida, aliando a construção incremental do `DidlBuilder` à construção incremental de *Java* do *Eclipse.*

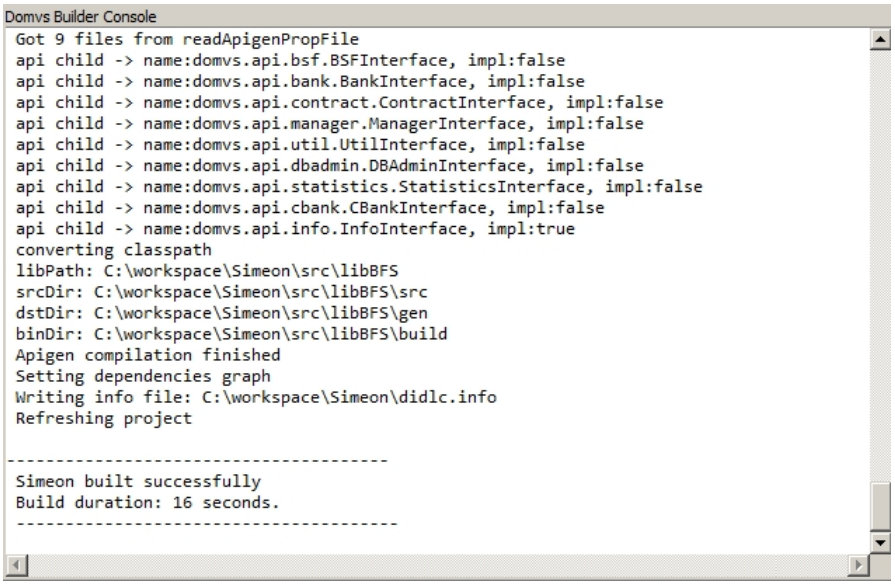


Figura 5.2: Construção Incremental com o `DidlBuilder`

## 5.3 Trabalho Futuro

Como trabalho de continuidade, seria interessante realizar mais algumas funcionalidades. Ao observar o construtor de *Java* para o *Eclipse*, podem ser enumeradas algumas funcionalidades que faz sentido implementar, para completar esta aplicação, nomeadamente:

- um mecanismo de sugestões para resolver erros conhecidos e encontrados na compilação;
- gerador de blocos de código que são sempre definidos da mesma forma;
- formatar o código automaticamente, seguindo padrões pré-definidos;
- um *refactor*, que permita a actualização de todas as referências a uma estrutura, método ou classe, quando é alterado o seu nome.

Uma modificação a fazer no futuro, seria migrar o código para o *Java 6*, lançado recentemente e que incluí bibliotecas para leitura e escrita de ficheiros *XML*, mais eficientes do que as usadas actualmente. Outro motivo para a migração é a constatação efectuada pela *Sun Microsystems*, de que um programa *Java* compilado em *Java 6*, fica imediatamente mais rápido a executar. Não fazia sentido, neste momento ter-se implementado a aplicação com *Java 6,* pois, por razões de compatibilidade, a maior parte dos colaboradores da empresa utiliza o *Java 5*, existindo mesmo alguns que utilizam, pela mesma razão, o *Java 1.4*.

## 5.4 Conclusão

Um projecto como este que torna mais rápido, fácil e agradável o trabalho dos elementos da EF, constitui sem dúvida uma mais-valia para a empresa e uma ferramenta de trabalho sólida, extensível e determinante para o trabalho futuro.

Fazendo um balanço destes nove meses, deve ser mencionado que este projecto é foi sem dúvida aliciante e requereu o uso de todos os conhecimentos adquiridos na faculdade, ao nível do planeamento, do desenho de aplicações, de programação, da resolução de problemas como da capacidade de abstracção, tanto pela sua extensão como pela sua complexidade. Em suma, tratou-se de um projecto que consolidou as capacidades e conhecimentos adquiridos ao longo do curso de Engenharia Informática, e foi um estágio construtivo e profissionalizante.

# Bibliografia


[1] Eclipse.org Home - http://www.eclipse.org/, 2007.

[2] The Apache Ant Project - http://ant.apache.org/, 2007.

[3] EF Tecnologias de Software - http://www.ef.pt/, 2007.

[4] John Arthorne and Chris Laffra. *Official Eclipse 3.0 FAQs,* Addison Wesley Professional, Jun 2004.

[5] Project Builders and Natures - http://www.eclipse.org/articles/Article-Builders/builders.html, 2004.

[6] Plugin Architecture - http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html, 2003.

[7] Eclipse SDK Help and API - http://help.eclipse.org/, 2007.

[8] Eclipse IDE Corner Articles - http://www.eclipse.org/articles/, 2006.

[9] Project Amateras - http://amateras.sourceforge.jp/, 2006.

[10] SourceForge: Create, Participate, Evaluate - http://sourceforge.net/, 2007.

[11] W3C XML Schema - http://www.w3.org/XML/Schema, 2007

# Índice

# Anexos

## Ficheiro *leia-me* do construtor `DidlBuilder`

```
----------------------------

DidlBuilder - versão 1.3.0

1/5/2007

----------------------------

Builder de eclipse para compilar ficheiros didl, não só o build inicial mas também incremen-
tal, permitindo compilar apenas os ficheiros que foram alterados desde o último build.  Gere
as dependências entre ficheiros didls, obrigando ficheiros que dependem de um didl que foi al-
terado a ser recompilados.

-------------------------------------------------------------------------------

Instruções

-------------------------------------------------------------------------------

- para saber quais os didls a compilar, o builder usa os ficheiros didlc.properties que de-
vem estar presentes em cada biblioteca ao lado do build.xml;

- exemplos dos ficheiros de propriedades estão incluidos junto a este readme;

- para activar o builder para um dado projecto, clicar com o botão direito no projecto, es-
colher "Enable Domvs Nature";

-------------------------------------------------------------------------------

Funcionalidades

-------------------------------------------------------------------------------

- wizard para criar novos didls, disponível em New->Others->DidlWizards ou na toolbar prin-
cipal no icon D, ou ainda no menu Domvs;

- wizard para importar didls do sistema de ficheiros para o interior do workspace, no menu Import-
>Others->Domvs Wizards;

- menu de preferências gerais do builder em Window->Preferences->DidlBuilder;

- independente da versão Domvs-PI do projecto, este é carregado em tempo de execução, ou da
pasta lib do projecto, ou do projecto Domvs-PI de desenvol- vimento que exista no workspace,
```

senão no caso do utilzador aceder, faz o download da versão correcta (manifest do projecto), em último caso utiliza o domvs.jar que é incluído no plugin;

- para fazer reset ao builder e forçá-lo a um full build, existe a opção Reset Builder no menu de opções, obtido quando é clicado o botão direito em cima de um projecto e quando este tem a natureza Domvs ligada;

- o "clean" apaga todos os ficheiros e pastas geradas durante o processo de build;

- ficheiros de propriedades e didls:

* as opções por default são as mesmas que eram, sempre que um didl tem a opção default não é necessário escrevâ-la no ficheiro de props;

* se um didl fôr gerado através do wizard, o ficheiro de propriedades é automaticamente actualizado ou criado se não existir;

* se um ficheiro fôr importado, o respectivo ficheiro de propriedades é actualizado ou criado se não existir;

* se as propriedades de um didl forem alteradas o ficheiro de proprie- dades correspondente é actualizado;

* se um ficheiro de propriedades fôr alterado, esteja o eclipse ligado ou não, no próximo build são apagados os ficheiros gerados a partir da biblioteca a que corresponde esse ficheiro de propriedades e o seu conteúdo é compilado outra vez com as novas definições;

* se um didl existir dentro de um projecto mas não constar no ficheiro de propriedades devido, aparece como disabled, se abrirmos o seu menu de propriedades e não é compilado, ao ser enabled é acrescentado ao respectivo ficheiro. Por outro lado se existe um didl no ficheiro de props que não existe fisicamente é gerado um erro e o ficheiro de props fica marcado.

- gestão de changes, com menu de opções e texto aberto para criação de uma nova change, menu de escolha de changes a incluir numa pré-versão, lançamento de uma ou mais pré-versões, escrita e leitura de ficheiro xml entre sessões de eclipse, campos extra para changes geridos num ficheiro changes.properties presente na raíz de cada projecto.

---

Miguel Valente

## Ficheiro *leia-me* do editor `DidlEditor`

---

DidlEditor - versão 3.2.0

1/5/2007

---

Editor para ficheiros .didl e .didls com sintaxe colorida, sugestões para completar palavras chave, outline baseado na hierarquia dos elementos e menu de preferências

---

Funcionalidades

---

- sintaxe colorida com diferenciação entre elementos, atributos, strings e as palavras chave da linguagem didl;

- detecta erros de sintaxe e de estrutura logo no momento do save, a partir de ficheiros xsd com as directrizes de construção dos didls;

- sugestões para completar palavras baseadas nos ficheiros .xsd fornecidos, ou seja, palavras chave da linguagem e atributos obrigatórios de cada elemento, além das estruturas já definidas no próprio ficheiro e em imports ou secções, carregadas dinâmicamente;

- sugestões para completar os paths dos imports

- outline hierárquico com sincronia entre o editor e o outline e vice-versa, filtros que permitem ver apenas estruturas e/ou métodos, filtro para incluir no outline os comentários presentes no ficheiro, organização por ordem alfa- bética;

- links implementados no outline que permitem, através de um duplo-clique num import ou secção, que o ficheiro correspondente seja aberto no editor.

---

Miguel Valente

## Exemplo de ficheiro de propriedades *didlc.properties*

```
didl.0.file=domvs/utils/Didl1.Didl
didl.0.implPackage=false
didl.0.wsProxy=true
didl.1.file=domvs/system/Didl2.Didl
didl.1.implPackage=false
didl.1.skipAbstract=true
didl.2.file=domvs/utils/ImportedDidl.Didl
didl.2.implPackage=true
```

## Exemplo de ficheiro de alterações *changes.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<changes>
   <pendingVersion project="Teste" subProject="Empresas" pkgPath="src/libAbc/abc.pkg">
      <change id="329c60d9505dc7eba1cdb1dca72592ddc8aff03a" cleanPending="false"/>
   </pendingVersion>
   <pendingChanges>
      <change id="329c60d9505dc7eba1cdb1dca72592ddc8aff03a"/>
   </pendingChanges>
   <allChanges>
```

```
      <change id="329c60d9505dc7eba1cdb1dca72592ddc8aff03a"
         date="24/04/2007 - 17:26:43"
         type="IC-TESTS"
         origin="MAIL"
         taskId="-1"
         requester="Cátia Vanessa"
         requestDate="24/04/2007">
         <text>Lançar uma versão só porque sim.</text>
         <notes type="[SQL] ">É necessário alterar a base de dados ocorreram alterações de SQL;
</notes>
      </change>
   </allChanges>
</changes>
```

## Ficheiro de propriedades *changes.properties* por defeito

```
# tipos de changes que devem estar disponíveis
types.list=INHL IC IC-TESTS Other
# origens possíveis para as changes e a necessidade ou não de referência
origin.0.name=BITS
origin.0.reference=true
origin.1.name=PGEST
origin.1.reference=true
origin.2.name=MAIL
origin.2.reference=false
origin.3.name=TEL
origin.3.reference=false
origin.4.name=EF
origin.4.reference=false
origin.5.name=Other
origin.5.reference=false
# tipos de notas e o texto que deve ser agregado quando são escolhidas
note.SQL=É necessário alterar a base de dados ocorreram alterações de SQL
note.JDK=Foi utilizada uma nova versão de Java nesta alteração
note.STUBS=Esta alteração requer a re-geração dos stubs
note.EXTRA=coisas extra
# caminho para o ficheiro .pkg com a versão do projecto
path=src/libTeste/test.pkg
# prefixo para passar ao SetJarVersion
tagPrefix=TestTag
# sub-projectos possíveis, ou seja, variantes do projecto
subproject.0.name=Particulares
subproject.0.path=src/libTeste/test.pkg
subproject.1.name=Empresas
subproject.1.path=src/libAbc/abc.pkg
```

## Excerto do ficheiro *XSD* com a definição do elemento *method*

```
<xs:element name="method">
   <xs:complexType mixed="true">
      <xs:sequence minOccurs="0">
         <xs:element minOccurs="0" maxOccurs="unbounded" name="param">
            <xs:complexType>
               <xs:attribute name="name" type="xs:string" use="required" />
               <xs:attribute name="type" type="xs:string" use="required" />
               <xs:attribute name="optional" type="xs:boolean" use="optional" />
               <xs:attribute name="log" type="xs:boolean" use="optional" />
            </xs:complexType>
         </xs:element>
         <xs:element minOccurs="0" maxOccurs="unbounded" name="exception">
            <xs:complexType>
               <xs:attribute name="type" type="xs:string" use="required" />
            </xs:complexType>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:attribute name="result" type="xs:string" use="optional" />
      <xs:attribute name="optional" type="xs:boolean" use="optional" />
      <xs:attribute name="hasValidate" type="xs:boolean" use="optional" />
   </xs:complexType>
</xs:element>
```