

Análise da integração de ferramentas open-source no processo de desenvolvimento -

Poseidon & NetBeans

Paulo Trezentos
PFSI – MEIC / IST
Paulo.Trezentos@iscte.pt

ABSTRACT

Neste documento é analisada a possibilidade de utilização ferramentas *open-source* no processo de desenvolvimento. Concretamente, são analisadas duas ferramentas: o Poseidon[1] e NetBeans[2]. O primeiro é uma ferramenta de modelização e o segundo um ambiente de desenvolvimento (IDE). Nas conclusões, traça-se uma avaliação da qualidade das ferramentas, complementada com a análise da sua integração no processo de desenvolvimento.

Keywords

Poseidon, NetBeans, UML, modelização, Java

1. INTRODUÇÃO

O ciclo de vida de uma aplicação tem nas suas fases iniciais a necessidade de utilização de ferramentas de *software* que as suportem.

As ferramentas de apoio à fase de modelização e desenvolvimento são geralmente muito dispendiosas e pesam na estrutura de custos da empresa que pretende desenvolver a aplicação.

Assim sendo, é equacionado neste documento a possibilidade de utilização de ferramentas *open-source* ou de baixo custo para colmatar essas necessidades.

Tabela 1. Tabela com o ciclo de vida da aplicação e aplicações possíveis de serem utilizadas

Análise de Requisitos	Modelização	Desenvolvimento	Testes
RequisitePro	Rational Rose	Visual C++	TestDirector
Emacs / vi (?)	Poseidon	NetBeans	

A tabela 1 situa as duas aplicações no processo de desenvolvimento.

O motor à utilização destas duas ferramentas não se restringe ao seu baixo custo mas também à sua robustez e ao facto de serem

multi-plataforma. Ambas as ferramentas são desenvolvidas em Java.

2. POSEIDON

2.1 Contexto

O Poseidon é um software desenvolvido a partir do ArgoUML, pela mesma equipa de desenvolvimento.

O ArgoUML é uma ferramenta de modelização inicialmente desenvolvida por Jason Robins e que se encontra sobre uma licença de utilização e distribuição *open-source*. De forma a rentabilizar esse desenvolvimento foi constituída a empresa Gentleware que produz uma versão do ArgoUML (chamada Poseidon) que contém mais algumas funcionalidades e inovações.

Apesar do Poseidon ser comercial, existe uma versão com certas limitações disponível para a comunidade utilizar sem encargos. O Poseidon está disponível 4 versões: *Community*, *Standard*, *Professional* e *Enterprise* que serão de seguida analisadas.

Para a elaboração deste documento foram testadas as versões *Community 1.4* e *Professional 1.5*.

2.2 Versões

Como podemos observar pela tabela 2, consoante a versão adquirida temos acesso a funcionalidades diferentes da aplicação.

Figura 3. Desenho de diagramas de classes no Poseidon

No canto inferior esquerdo, temos uma visão geral de todo o diagrama.

Cada classe pode ser especificada em detalhe na parte inferior da janela. Nomeadamente, com a tradicional adição de atributos e métodos. O desenho das classes pode ser realizado recorrendo às funcionalidades de alinhamento e arrastamento (*drag & drop*).

O desenho dos vários elementos pode ser realizado através do recurso à barra de ferramentas apresentada na parte superior da janela.

2.3.2 Casos de utilização

A elaboração de um Diagrama de Utilização é realizada da mesma forma que é realizada noutras aplicações.

No caso da monitorização da Grid, foram aprofundados os vários casos de topo em casos de utilização mais detalhados.

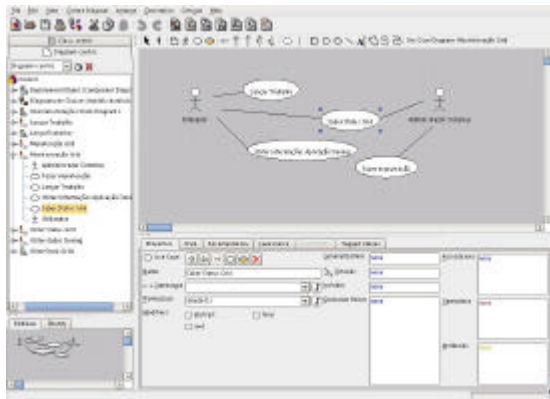


Figura 4. Desenho de diagramas de utilização no Poseidon

2.3.3 Diagrama de sequência

O desenho de diagramas de utilização no Poseidon tem algumas limitações. Concretamente, na impossibilidade de arrastamento de elementos na hierarquia presente no lado esquerdo da imagem directamente para o espaço do diagrama.

Por outro lado, a representação de um objecto Actor é realizada sobre a forma gráfica de uma classe (como apresentado na fig.5), ao contrário de outras ferramentas que permitem ter representações mais intuitivas.

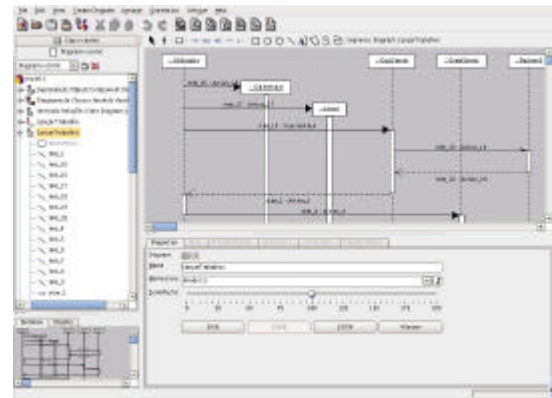


Figura 5. Desenho de diagramas de sequência no Poseidon

A fig.6 introduz -nos um diagrama produzido no Poseidon.

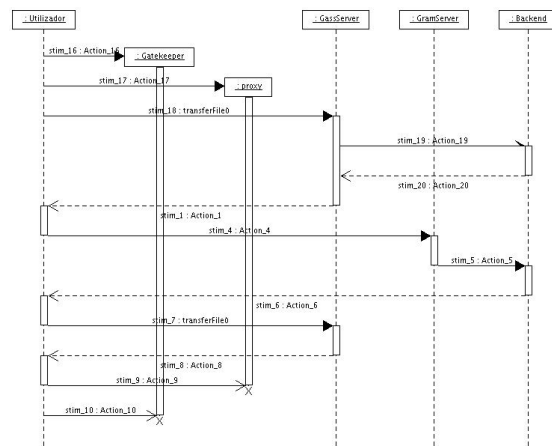


Figura 6. Diagramas de utilização Lançar Trabalho

2.4 Análise da Versão 1.5 Professional

A versão utilizada nas secções anteriores foi a disponibilizada gratuitamente à comunidade (*Community Edition*).

Contudo, para avaliar melhor o desempenho da ferramenta em características avançadas, foi instalada uma versão de teste (*trial*) da versão Professional.

De seguida, foram instalados os plugins *RoundTrip*, *UMLdoc*, *JarImport* e *MDLimport* que, de seguida, serão desenvolvidos.

2.4.1 RoundTrip

O RoundTrip é uma característica do software de modelização que nos permite actualizando iterativamente o modelos de classes através do código-fonte e vice-versa, mantendo assim um sincronismo entre a modelização e o desenvolvimento.

Para testar o RoundTrip foram executados os seguintes passos:

1. Gerado código-fonte Java a partir da classe CPU (fig.8)

2. Alterado o código fonte Java com o NetBeans
3. Refrescado o modelo de classes a partir do novo código
4. Produzir uma nova alteração no modelo de classes e assistir ao impacto sobre o código.

A figura 7 apresenta a janela do Poseidon que permite a geração de código-fonte a partir das classes.

Para além da directoria para onde queremos fazer a geração, podemos incluir outras configurações relacionadas com o código gerado.

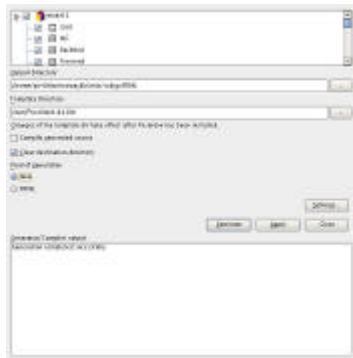


Figura 7. Geração de código Java

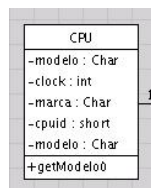


Figura 8. Classe CPU

Como foi dito, o primeiro passo foi a geração do código fonte a partir da classe CPU.

O resultado foi o apresentado na listagem 1, que como podemos verificar reflecte bem a classe criada. A associação à classe Nó é estabelecida correctamente.

```

/** Java class "CPU.java" generated from Poseidon for UML.
 * Poseidon for UML is developed by .
 * Generated with template engine.
 */
import java.util.*;
/**
 * * @author prrt *
 */ public class CPU {
    // attributes /** *
    * Represents ... *
    */
    private Char model
o;
    private int clock;
    private Char marca;
  
```

```

private short cpuid;
    // associations
    public
    Nó Nó
    // operations
    public void getModelo() {
        // your code here }
// end getModelo }
// end CPU
  
```

Listagem 1 - Código-fonte original produzido pelo Poseidon

De seguida, foi alterado o código, incluído um novo atributo (**cacheSize**) uma nova associação (à classe **Memória**) e um novo método (**returnCacheSize()**).

```

// attributes /** *
 * Represents ... *
 */
private Char modelo;
private int clock;
private Char marca;
private short cpuid;
private short cacheSize;
// associations
public Nó Nó ;
public Memória Memória;

// operations
public void getModelo() {
// your code here }
// end getModelo }

public short returnCacheSize() {
returnCacheSize;
}
  
```

Listagem 2 - Código-fonte editado no NetBeans

Todas as alterações realizadas ao nível do código foram correctamente integradas pelo Poseidon na classe CPU.

Apesar deste sucesso, foram realizados outros testes e em algum deles o Poseidon não conseguiu adicionar novos métodos à classe ou estabelecer novas associações. Tal deveu-se a certas formatações mal interpretadas.

Após a actualização da classe, voltámos a realizar uma alteração na classe que resultou na produção correcta do código-fonte (listagem 3).

A actualização / sincronização pode ser realizada automaticamente, devendo o utilizador definir qual o período que deseja para essa actualização (por exemplo, 60 segundos).

```

// attributes /** *
 * Represents ... *
 */
private Char modelo;
private int clock;
private Char marca;
private short cpuid;
  
```

```

private short cacheSize;
private short cacheModel;

// associations
public Nó Nó ;
public Memória Memória;

// operations
public void getModelo() {
// your code here }
// end getModelo }

public short returnCacheSize() {
returnCacheSize;
}

```

Listagem 3 - Código-fonte gerado a partir das classes

2.4.2 UMLdoc

O módulo (plugin) para a produção de UMLdocs apenas está disponível na versão *Professional*.

Contudo todas as versões disponibilizam a produção de documentação elementar em HTML, com as classes desenhadas.

Um exemplo dessa documentação elementar é apresentada na fig.

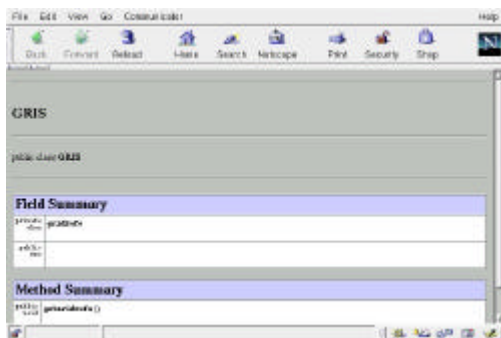


Figura 9. Documentação elementar produzido pelo Poseidon

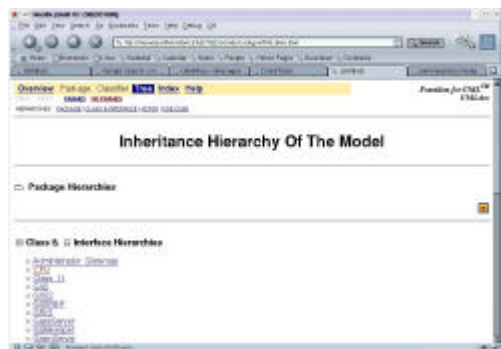


Figura 10. Documentação UMLdoc

Por outro lado, para uma documentação mais sofisticada, a versão *Professional* permite utilizar um módulo chamado UMLdoc que produz documentação mais extensiva em HTML.

O resultado do UMLdoc é muito semelhante ao produzido pelo JavaDoc.

Atente-se que, apesar da documentação extensiva, o UMLdoc não tem mapas de imagem (*image maps*) que permitam uma navegação intuitiva entre os vários níveis de uma hierarquia, à semelhança do Rational Rose. Contudo, podemos organizar as classes hierarquicamente e *navegar* nos seus métodos e atributos.

2.4.3 Importação de JAR

Outro módulo interessante testado foi a importação de *bytecode* Java, ou seja, ficheiros com extensão *.jar*.

Para o teste ser real, foi importado o *bytecode* do ArgoUml.

Dado a dimensão do ArgoUML, esta tarefa revelou-se demorada por duas vias.

Primeiro, porque a ferramenta pode demorar 1 a 2 horas a produzir as classes correspondentes a um *.jar* complexo¹.

Em segundo lugar, porque dado o elevado número de ficheiros, é necessário fazer manualmente a operação de importação.

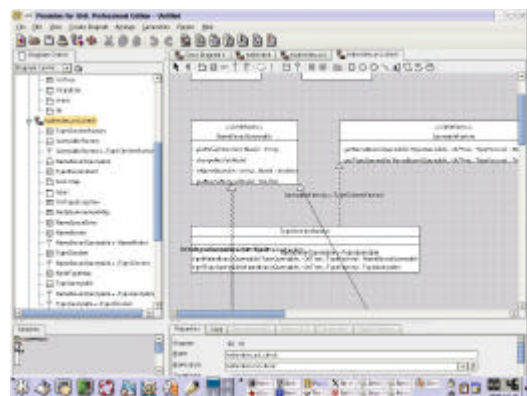


Figura 11. Diagrama de classes importado do Jar file do ArgoUML

Foi interessante verificar as associações correctamente estabelecidas entre as várias classes.

Esta ferramenta de *reverse engineering* verificou-se eficaz.

Outra das funcionalidades base do Poseidon é a importação de código Java e o correspondente desenho do diagrama de classes a partir do mesmo.

¹ Num computador P4 a 2Ghz com 512 MB de RAM

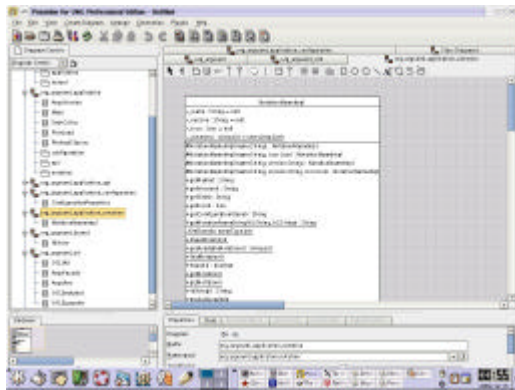


Figura 12. Diagrama de classes importado do código fonte do ArgoUML

A figura 12 mostra a classe produzida directamente do código fonte.

Para manter a consistência, voltámos a usar o ArgoUML como modelo de teste.

Assim, foi curioso verificar a semelhança em entre os modelos produzidos a partir do bytecode e a partir do código fonte. Apesar de teoricamente deverem ser iguais, algumas diferenças foram verificadas.

Não foi possível quantificar o grau das diferenças dado a extensão do número de classes e de não haver um mapeamento directo no nome dos ficheiros de *bytecode* e a respectiva classe.

2.4.4 Importação de projectos do Rational Rose (MDL)

Dado o quota de mercado do *software* Rational Rose, é importante que uma ferramenta de modelização permita a importação de projectos MDL (extensão do Rational).

Apesar de gravar os ficheiros em XML, ou mais concretamente em XMI, o Poseidon permite na versão *Professional*, a importação de projectos do Rational Rose.



Figura 13. Importação de projecto do Rational Rose

A figura 13 apresenta precisamente a importação de um desses projectos.

Mais uma vez, verificou-se que este é um processo moroso que pode levar muitos minutos.

As funcionalidades mais avançadas do Rational Rose (scripts, estereótipos, etc...) não são importáveis para o Poseidon.

2.5 Compatibilidade com ArgoUML

Dado descender do ArgoUML e utilizar o formato XMI standard para armazenar os seus projectos, a interoperabilidade entre o Poseidon e o ArgoUML revelou-se excelente.

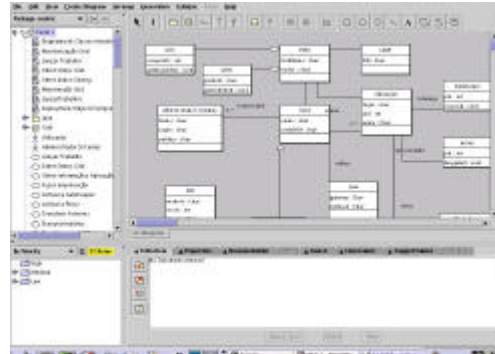


Figura 14. Importação no ArgoUML de um projecto do Poseidon

Como podemos verificar na fig. 14, o projecto de monitorização de Grids realizado no Poseidon foi correctamente importado pelo ArgoUML.

3. NETBEANS

O NetBeans [2] é um ambiente de desenvolvimento de aplicações (IDE).

Orientado para a linguagem Java permite o desenvolvimento rápido de aplicações com *Graphical User Interface* (GUI) ou não.

3.1 Contexto

O NetBeans está disponível sob a licença GPL pelo que pode ser utilizado e distribuído livremente sem existir a necessidade de pagamento de qualquer tipo de pagamento.

Para além do IDE, o projecto do NetBeans comporta ainda uma plataforma que pode ser utilizada para o desenvolvimento de aplicações.

Curiosamente, o Poseidon utiliza a plataforma NetBeans para o desenvolvimento do seu GUI.

3.2 Características

Como podemos verificar pela fig. 15, o NetBeans tem um interface muito semelhante a outros IDEs.

Concretamente, podemos na janela principal editar ficheiros de Java com o código colorido segundo a sintaxe.

Na parte esquerda do écran temos o navegador pelas várias classes / ficheiros.

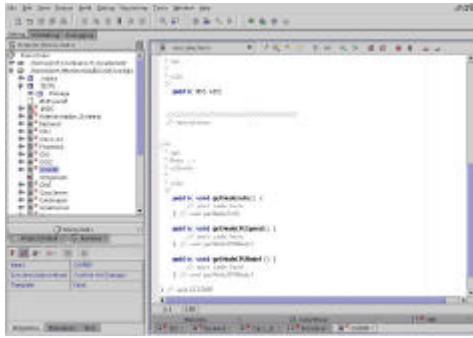


Figura 15. Aplicação NetBeans

No mesmo ambiente, é ainda possível fazer a compilação e depuração (*debugging*) do código Java.

As funcionalidades de *debugging* são semelhantes às que encontramos noutras plataformas, sendo bastante completas.

Outra característica muito interessante é o facto de todo o interface com um servidor de controlo de versões (como o CVS) ser suportado dentro do NetBeans. Pode-se assim, participar num esforço colaborativo no site SourceForge sem nunca sair do NetBeans.

O NetBeans tem um sistema de Wizards que permite criar rapidamente aplicações do zero. Os wizards são extensivos para diferentes tipos de aplicações Java: RMI, applets, java beans, ...

4. INTEGRAÇÃO DO POSEIDON COM NETBEANS

Depois da análise da duas aplicações, coloca-se a questão da sua integração.

Estando o Poseidon a montante do NetBeans, como poderemos que as classes modelizadas ajudem ao arranque do desenvolvimento, mantendo ao longo deste uma sincronia com o diagrama de classes.

4.1 Aplicações Independentes

O primeiro cenário é a utilização como aplicações independentes.

O esquema da fig. 16 exemplifica um processo típico.

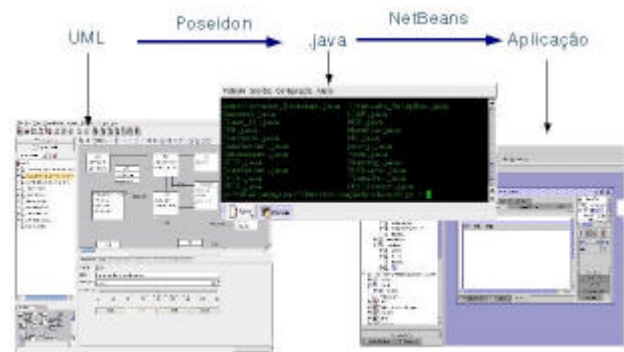


Figura 16. Aplicação NetBeans

O modelo de classes é produzido no Poseidon, sendo produzido o código Java a partir da mesma aplicação e sendo este código posteriormente importado pelo NetBeans.

4.2 Poseidon com módulo do NetBeans

Por outro lado, é possível que no próprio interface do NetBeans seja integrado o Poseidon.

A fig.17 demonstra como esse integração é realizada.

No NetBeans aparece uma nova pasta (“Modeling”) que dentro do seu contexto tem todas as operações de modelização.

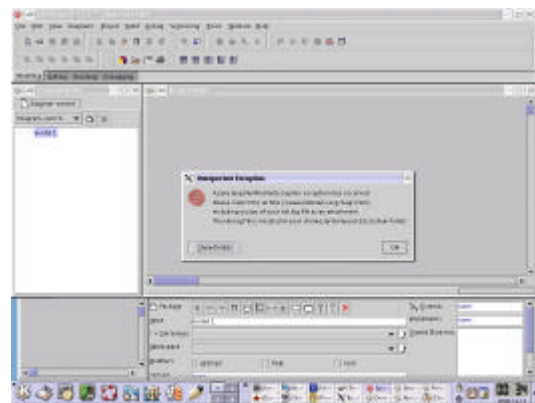


Figura 17. Integração do Poseidon no NetBeans

Conseguiria-se assim uma integração perfeita se não fosse a instabilidade.

Como podemos verificar pela própria imagem, a integração torna o NetBeans extremamente instável.

Acrescente-se ainda que para poder integrar o Poseidon, temos de possuir a versão 3.2 ou 3.3 do NetBeans.

A versão NetBeans 3.4 não permite a integração, pelo que para os testes realizados tivemos de fazer um *downgrade*.

5. CONCLUSÕES

O Poseidon revelou-se uma ferramenta estável e completa. Os plugins disponíveis cobrem as necessidades dos utilizadores mais exigentes.

Contudo, tem um preço que deverá ser suportado e que, apesar de tudo, é comportável e não se aproxima de outras soluções mais caras.

O NetBeans é igualmente completo, com a vantagem adicional de ser software livre. A desvantagem é estar muito direccionada para Java e não ser facilmente utilizável para desenvolver noutras linguagens. A sua curva de aprendizagem é algo difícil dado a sua complexidade.

Para concluir, pensamos que a utilização de ambas as ferramentas – em separado ou não - justifica-se plenamente no âmbito de desenvolvimento aplicacional. Mesmo numa vertente empresarial.

Tal deve-se à sua robustez, fiabilidade e preço. Contudo, é necessário salvaguardar que os computadores utilizados têm capacidade de CPU para os executar.

Desaconselha-se a sua utilização integrada dado a instabilidade apresentada (ver secção 4.2).

6. CRÉDITOS

O autor agradece as várias sugestões e críticas que foram feitas no âmbito das apresentações do seminário de PFSI do Mestrado em Eng. Informática e de Computadores (IST) e que serviram para o melhoramento deste documento.

7. REFERENCES

- [1] Sítio do Poseidon, <http://www.gentleware.com>
- [2] Sítio do NetBeans, <http://www.netbeans.org>
- [3] Sítio do ArgoUML, <http://argouml.tigris.org/>
- [4] NetBeans: the definitive guide (O'Reilly)
- [5] *Evaluating two Next-Generation UML tools*, Roger Smith
- [6] *Blueprint to a Grid Infrastructure*, Kesselman Editors, Ian Foster