

**Faculdade de Engenharia da Universidade do Porto**



**ESOF 2016/2017 - Mestrado Integrado em Engenharia Informática e Computação**

**Autores:**

Andreia Rodrigues - [up201404691@fe.up.pt](mailto:up201404691@fe.up.pt)

Eduardo Leite - [gei12068@fe.up.pt](mailto:gei12068@fe.up.pt)

Gonçalo Leão - [up201406036@fe.up.pt](mailto:up201406036@fe.up.pt)

Francisco Queirós - [up201404326@fe.up.pt](mailto:up201404326@fe.up.pt)

ESOF - Relatório 3

WorldEdit

Arquitetura de software

A arquitetura de *software* é a organização fundamental do sistema, e a interação entre os componentes que o compõem e o ambiente externo. O output do processo de *design* arquitetural é um modelo que representa a partição do sistema em subsistemas que podem ser desenvolvidos em paralelo e subsequentemente integrados no produto final. Esta fase de *design* corresponde ao passo que sucede ao de engenharia de requisitos na maior parte dos processos de *software*.

O *design* e documentação da arquitetura de *software* traz várias vantagens. Em primeiro lugar, permite uma melhor comunicação entre *stakeholders* pois a arquitetura apresenta o sistema de um ponto de vista de mais alto nível. Em segundo lugar, dado que o *design* fornece uma visão mais clara da estrutura da solução, torna-se mais fácil analisar se o sistema cumpre ou não certos requisitos não-funcionais como a sua fiabilidade e desempenho. Por fim, elaborar um modelo do sistema, pode permitir a reutilização de *software* em larga escala, caso seja possível aproximar este modelo com o de outro sistema previamente desenvolvido e com requisitos semelhantes. Com efeito, esta modelação do sistema permite identificar semelhanças e diferenças entre diferentes sistemas.

Vamos agora focar-nos em dois principais pontos.

Para começar, vamos apresentar o modelo arquitetural 4 + 1 e as suas vistas para o projeto em estudo.

De seguida, vamos explicar o que são padrões arquiteturais, e identificar alguns dos que foram usados no caso do WorldEdit.

Modelo arquitetural 4+1

Não é possível usar um único modelo para representar toda a informação relevante do sistema, visto que cada modelo apresenta-o de um ponto de vista específico. Exemplos de perspectivas de um sistema incluem a sua decomposição em módulos e a interação entre processos em tempo de execução. Assim, dado que as diferentes vistas sobre o sistema têm a sua importância em diferentes fases do seu desenvolvimento, torna-se importante desenvolver vários modelos para o representar.

O modelo arquitetural 4+1, apresentado por Philippe Krutchen em 1995, propõe que sejam usadas quatro vistas para definir o sistema: vista lógica, vista de implementação, vista de processo e vista de *deployment*. A estes quatro modelos, é adicionado um modelo de casos de uso (que já foi apresentado no segundo relatório, relativo à engenharia de requisitos), que indica os problemas que a aplicação pretende resolver.

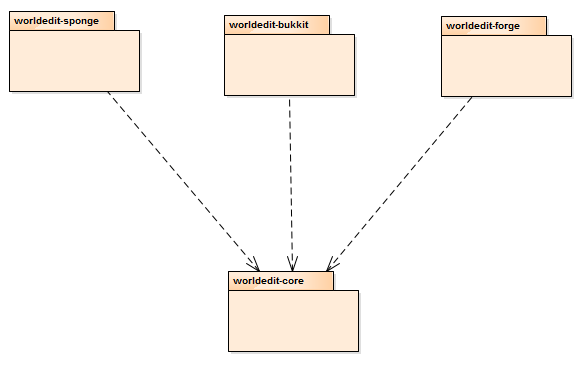
Cada uma destas vistas pode ser apresentada sob a forma de um diagrama UML, que será a abordagem adotada para apresentar o modelo arquitetural para o caso do WorldEdit.

Vista lógica

A vista lógica apresenta as abstrações mais importantes do sistema sob a forma de objetos ou classes de objetos.

**Pacotes principais**

No diagrama abaixo, pode-se ver o diagrama UML dos principais *packages* do projeto. As setas representam as dependências entre módulos.

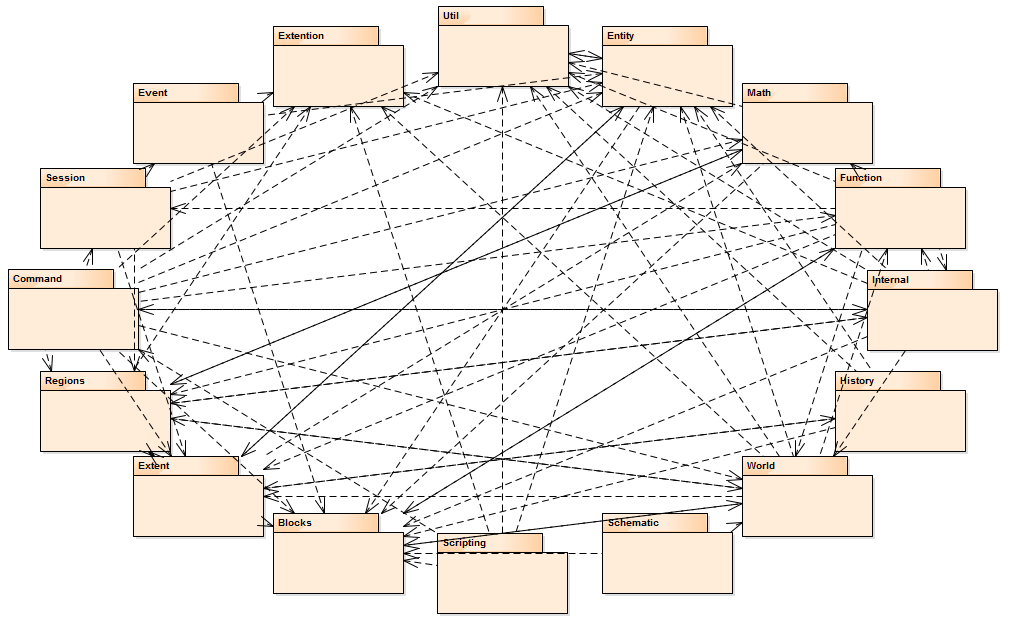


*Figura 1: Diagrama UML de “packages” dos principais pacotes do WorldEdit*

O pacote “worldedit-core” é o elemento central do projeto, visto que contém a camada lógica do WorldEdit. Trata-se de um módulo independente da plataforma (Bukkit, Forge, …) utilizada para estender o Minecraft.

Os pacotes “worldedit-sponge”, “worldedit-bukkit” e “worldedit-forge” contém as classes necessárias para adaptar as funcionalidades-base do WorldEdit à plataforma Sponge, Bukkit e Forge, respetivamente. Assim, são módulos dependentes da plataforma.

Na figura abaixo, são representados os pacotes do núcleo lógico do WorldEdit, dentro do pacote worldedit-core, onde ocorre a maior parte do processamento do plugin.



*Figura 2: Diagrama UML dos principais “packages” em worldedit-core*

A tabela seguinte descreve a responsabilidade de cada módulo.

|  |  |
| --- | --- |
| **Módulo** | **Descrição** |
| Blocks | Módulo usado para representar os blocos no contexto do WorldEdit. |
| Command | Módulo usado para a interpretação e execução de comandos. |
| Entity | Módulo usado para representar entidades do jogo. Uma entidade é um objeto dinâmico no Minecraft, por oposição a blocos (estáticos). Exemplos incluem: personagens, inimigos e *minecarts*. |
| Event | Módulo de classes usadas para representar os diferentes tipos de eventos do jogo (por exemplo, o clique com o botão esquerdo ou direito do rato). |
| Extension | Módulo de ponto de entrada no package worldedit-core. As classes deste pacote são usadas por worldedit-bukkit/forge/sponge para aceder à camada da lógica do WorldEdit. |
| Extent | Módulo utilizado para fazer a ponte entre o mundo e as entidades que ele contém. |
| Function | Módulo usado para aplicar funções sobre os vários elementos do mundo (blocos, personagens…). Estas funções correspondem a comandos “parsed” pelo módulo command. Exemplos de funções incluem a troca blocos ou que conta blocos, etc. |
| History | Módulo utilizado para representar o histórico, útil para operações de copy paste. |
| Internal | Módulo que guarda constantes, eventos e as estruturas de dados necessárias ao processamento do Console UI. |
| Math | Módulo usado pelos outros módulos para efectuar as operações matemáticas necessárias ao seu funcionamento. |
| Regions | Módulo utilizado para representar uma região do mundo no contexto do WorldEdit. |
| Schematic | Módulo utilizado para processar as schematics, uma funcionalidade do plugin que consiste em carregar uma região de um ficheiro para o jogo. |
| Scripting | Módulo utilizado para a funcionalidade de scripting to plugin, que permite combinar funcionalidades de forma complexa em scripts, que pode ser corridos no jogo utilizando comandos do plugin. |
| Session | Módulo utilizado para representar a sessão do utilizador, por sessão entende-se o período de tempo desde o momento do login até ao momento do logoff. |
| Util | Módulo com classes utilitárias. |
| World | Módulo utilizado para representar o mundo no contexto do WorldEdit. |

*Tabela 1 - Responsabilidade de cada módulo da aplicação*

Vista de implementação

A vista de implementação mostra como é que o *software* pode ser decomposto em diferentes componentes que podem ser desenvolvidas em paralelo por um programador ou uma equipa de desenvolvimento.

Na figura abaixo, apresenta-se o diagrama UML dos principais componentes do WorldEdit.

*Figura 3: Diagrama UML de “components” do WorldEdit*

O componente EventHandler é responsável por todos os eventos gerados pelos utilizadores, tais como cliques com o rato sobre objetos do mundo ou a escrita de um comando pela linha de comandos do jogo.

O componente Command é usado pelo EventHandler para fazer *parsing* dos comandos escritos a partir da linha de comandos (representado pelo componente MinecraftCLI). Certos componentes fornecem ao EventHandler uma interface para que este possa comandos de um certo tipo.

O componente History permite o processamento de comandos associados ao histórico da sessão.

O componente SchematicManager permite processar comandos do tipo schematic, que lidam com um formato de ficheiro específico para representar construções (como casas, castelos…) no Minecraft. O módulo SchematicManager fornece também ao Clipboard uma interface para que este possa carregar ficheiros.

O módulo ScriptManager permite ao EventHandler executar *craftscripts*, que se tratam de ficheiros que executam uma tarefa complexa de edição do mundo do *Minecraft*, como a geração aleatória de um labirinto.

O componente Clipboard permite executar comandos associados ao clipboard da sessão, o que possibilita a manipulação de regiões (como como copy, paste e rotate) com a forma de um paralelipípedo. Este componente fornece também ao Schematic uma interface para que este possa manipular os objectos que carregou.

O módulo WorldEditor fornece ao componente EventHandler uma interface que lhes possibilita fazer alterações ao mundo. É o componente responsável por fazer a ponte entre o mundo no minecraft (representado pelo componente MinecraftWorld) e a sua representação no plugin.

Vista de *deployment*

A vista de *deployment* apresenta o *hardware* do sistema e como os componentes de *software* estão distribuídos pelos seus processadores.

A figura abaixo apresenta o diagrama UML de *deployment* para o caso do WorldEdit.

Deployment Diagram.png

*Figura 4: Diagrama UML de “deployment” do programa*

Os artefactos marcados a cinzento tratam-se de elementos externos ao próprio WorldEdit (não foram obtidos por compilação do seu código).

Tal como fora apresentado no primeiro relatório, o WorldEdit é um *plugin* para Minecraft. Por outras palavras, trata-se de uma extensão do servidor que adiciona novas funcionalidades ao jogo ou modifica as já existentes. Assim, do lado do servidor, é necessário um *game server* que seja compatível com *plugins* (mais especificamente, com o WorldEdit) pois o *server* oficial do Minecraft não tem este tipo de suporte. Logo, além do executável de Minecraft, referido por Minecraft.jar no diagrama, também é necessário, do lado do servidor, um executável de um game server, MinecraftServerX.jar, e a respetiva versão do WorldEdit, WorldEditX.jar.

Em termos de plataformas suportadas pelo WorldEdit, o projeto suporta o Bukkit, o Forge e o Sponge (e, consequentemente, outras plataformas que derivam destas, como o Spigot). Além disso, o MinecraftEdu (uma plataforma para fins pedagógicos) vem com uma versão antiga *built-in* do WorldEdit.

O WorldEdit suporta também, de forma não-oficial, as plataformas LiteLoader e Canary.

Opcionalmente, podem ser fornecidos ficheiros do tipo .schematic ou .js, do lado do servidor, para que os utilizadores possam utilizar os respetivos *schematics* e *craftscripts* para editar o mundo.

Ainda sobre o diagrama UML apresentado acima, convém referir que a máquina do servidor pode também funcionar como uma das máquinas *client*, passando a ser denominada de *host*.

Vista de processo

A vista de processo apresenta o sistema como um conjunto de processos que interagem entre si em tempo de execução. Esta vista é útil para averiguar se certos requisitos não-funcionais (como o desempenho e a disponibilidade) estão a ser implementados corretamente.

Activity Diagram.png

*Figura 5: Diagrama UML de atividades do programa*

A partir do diagrama UML de atividades da figura acima, podemos ver qual é a interação entre o *input* do utilizador e o *plugin*. O diagrama é simples pois reflecte a falta de camadas extensas entre o utilizador e o WorldEdit. Após o comando ser recebido (*Command Reception*), este é interpretado de forma sintática (*Command Parsing*) para determinar se é um comando existente, se está corretamente formatado, se os seus argumentos são válidos e se o utilizador tem permissão de executar esse comando. Se o comando for aceite, é invocado a função que lhe está associada e o “mundo” é alterado (*World State Change*).

Padrões arquiteturais

O objetivo de um padrão é representar, partilhar e reutilizar conhecimento.

Um padrão arquitetural é uma descrição abstrata e estilizada de uma boa prática de *design* arquitetural que já foi testada em diferentes ambientes e sistemas. Assim, um padrão descreve uma solução para um problema recorrente num certo contexto, e à qual está associada um certo grau de credibilidade. Estes padrões são parecidos com os padrões de desenho com a diferença fundamental que os padrões arquiteturais abrangem todo o sistema em vez de estarem relacionados com apenas alguns dos seus componentes.

Vamos agora analisar alguns dos padrões arquiteturais mais conhecidos e ver se estes são usados no projeto WorldEdit. Convém relembrar que, num contexto prático, mesmo se forem seguidos os princípios gerais de um determinado padrão, o mapeamento entre o padrão e a arquitetura de um programa em concreto nem sempre é perfeito, podendo haver algumas diferenças ao nível do *design* da arquitetura.

Model-View-Controller

O padrão arquitetural Model-View-Controller separa o sistema em três principais componentes.

A componente Model encapsula todo o estado interno do programa e notifica o View de mudanças deste seu estado.

A componente View é responsável por apresentar ao utilizador a informação contida em Model (através de um *command line interface* ou *graphical user interface*, por exemplo). É também responsável por interrogar a componente Model para averiguar se este atualizou o seu estado e de enviar eventos gerados pelo utilizador à componente Controller.

A componente Controller mapeia as ações do utilizador em atualizações do estado interno do Model. Além disso, é responsável por selecionar a vista a ser usado no interface com o utilizador.

No caso do WorldEdit, a arquitetura MVC não está presente no WolrdEdit em si. Contudo, é possível identificar uma arquitetura próxima deste modelo se tivermos em conta os componentes externos fornecidos pela plataforma usada (Bukkit, Sponge…). Vamos usar o diagrama UML de componentes apresentado anteriormente (figura 3) para mapear (de forma aproximada) os três principais blocos do padrão MVC.

Em primeiro lugar, a componente Model é constituída por MinecraftWorld, History, ScriptManager, SchematicManager e Clipboard. Estes componentes têm a responsabilidade de armanzenar informações acerca do mundo de jogo, do histórico das ações do utilizador, dos ficheiros do tipo *craftscript* e *schematic* e do *clipboard*.

Em segundo lugar, a componente View é representado pelos componentes externos MinecraftWorld e MinecraftCLI, que têm como responsabilidades respetivas apresentar o mundo do jogo (estado interno) ao jogador e ler os seus comandos através de uma *interface* por linha de comandos.

Por fim, a componente Controller é composta por EventHandler, Command e WorldEditor, que têm como objetivo receber o *input* do utilizador, interpretá-lo e atualizar a informação contida no componente Model. Trata-se da sequência de passos apresentados no diagrama de sequência da figura 5.

Arquitetura em camadas

O padrão arquitetural da arquitetura em camadas define que o sistema esteja organizado num conjunto de camadas. Cada camada tem o seu conjunto de funcionalidades e fornece serviços à camada acima.

No caso do WorldEdit, é possível identificar a estruturação do código em camadas observando o diagrama UML de pacotes da figura 1.

Os módulos para cada variante de software de servidor acedem ao módulo “core” de forma a que o core se pareça como uma espécie de serviço para o módulo do servidor.

Isto permite com que haja um reuso de código porque o núcleo da lógica no módulo core é comum aos três casos de variantes de servidor.

Conclusões e análise crítica

É do entender de todos os membros do grupo que as vistas apresentadas permitem uma boa compreensão da arquitetura do WorldEdit.

A vista lógica, expressa usando diagramas de *packages*, apresenta os blocos lógicos do programa e as suas interdependências. Enquanto que, a mais alto nível , os principais pacotes da aplicação estão bem organizados (figura 1), a mais baixo nível, existe uma forte interdependência entre blocos lógicos (figura 2), o que dificultou bastante a análise da arquitetura do programa ao nível do pacote *worldedit-core*.

Com a vista de implementação, apresentada com um diagrama de componentes, foi possível identificar uma boa modularização do código.

Com vista de *deployment*, foi possível apresentar a distribuição dos artefactos pelos componentes de *hardware*. A organização do programa a este nível é, na opinião da equipa, apropriado.

Para a vista de processo, escolheu-se de apresentar a sequência de passos desde o *input* por parte dos jogadores até à atualização do mundo, sendo estas as principais atividades da aplicação.

Para concluir, apesar de ter havido algumas dificuldades iniciais na análise da arquitetura, após uma análise mais minuciosa do código e do repositório, a organização do sistema do WorldEdit foi bem entendida e adequa-se bem aos requisitos (funcionais e não-funcionais) que foram identificados no segundo relatório.

Bibliografia

Software Engineering, Ian Sommerville, 9th Edition, capítulos 4, 5 e glossário.

Glossário do IEEE para a terminologia de engenharia de software, disponível em: <http://www.mit.jyu.fi/ope/kurssit/TIES462/Materiaalit/IEEE_SoftwareEngGlossary.pdf>

<https://github.com/sk89q/WorldEdit/tree/master/worldedit-core/src/main/java/com/sk89q/worldedit>

<http://docs.sk89q.com/worldedit/apidocs/overview-summary.html>

Website oficial do MinecraftEdu: <http://services.minecraftedu.com/wiki/What_is_MinecraftEdu>

Página *about* da *wiki* oficial do Spigot:

<https://www.spigotmc.org/wiki/about-spigot/>

<http://www.enginehub.org/worldedit/faq#explain-modding>