INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Plataforma para Jogos Multi-Canal

João Chaves Carneiro Ribeiro

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente:	Prof. Luís Eduardo Teixeira Rodrigues
Orientador:	Prof. José Manuel da Costa Alves Marques
Vogais:	Prof. António Manuel Ferreira Rito da Silva
	Eng. José Afonso Vasconcelos Pires
	Prof. Rui Filipe Fernandes Prada

Setembro de 2007

Agradecimentos

Quero agradecer à minha família por sempre me terem apoiado e garantido a minha educação.
Ao amor da minha vida por me aturar durante o desenvolvimento desta tese.

Resumo

A disponibilização de conteúdos em múltiplas plataformas tem sido um tema bastante debatido nos últimos anos. Alguns modelos foram propostos de forma a apoiar a ideia de criar o conteúdo uma vez e disponibilizá-lo em todo lado. Esta tese pretende aferir sobre a possibilidade de integrar este conceito numa plataforma para jogos web (multi-jogador e open-source) já existente, de forma a permitir uma maior ubiquidade aos jogos na sua interacção com o utilizador.

Será tomado a cabo um processo de reengenharia da plataforma de forma a permitir a interacção multi-canal, mas mantendo as actuais funcionalidades da mesma.

Para fazer face às restrições associadas aos diferentes dispositivos de visualização (PC, PDA, telemóvel...), serão também propostas alterações à actual interface web da plataforma e demonstrada a possibilidade de inclusão de novas interfaces, que funcionem em paralelo, através de uma interface para o canal SMS.

No final será possível compreender os compromissos assumidos durante o desenvolvimento da solução e quais as vantagens de uma plataforma deste tipo.

Palavras chave: Plataforma de Jogos, Interacção, Multi-Canal, Reengenharia.

Índice

Lista de Figuras

Lista de tabelas

Lista de abreviações

API Aplication Programming Interface
Berserk BusinEss logic Runtime & SEcurity Resources Kit
CSS Cascading Style Sheets
DXW DimensioneX World description file
JAXB Java Architecture for XML Binding
JAX-WS Java API for XML Web Services
HTML HyperText Markup Language
HTTP HyperText Markup Language
MVC Model-View-Controller
PC Personal Computer
PDA Portable Digital Assistant
SMS Short Message Service
UDP User Datagram Protocol
W3C World Wide Web Consortium
XHTML eXtensible HyperText Markup Language

Capitulo 1

Introdução

Os jogos web-based têm vindo a emergir do anonimato, cada vez mais cibernautas aderem a este tipo de jogos e gastam mais do seu tempo online neles. Este trabalho tem como objectivo aferir sobre a possibilidade de alterar uma plataforma para jogos web-based e open-source já existente, de forma a que lhe seja possível a interacção multi-canal. Para tal, serão propostas as alterações necessárias a realizar e será feita a sua respectiva implementação.

1.1 Enquadramento

O trabalho desenvolvido nesta dissertação teve como ponto de partida um anterior projecto de fim de curso[1], apresentando-se de certa forma como uma continuação do mesmo.
O projecto em causa, teve como principal objectivo a criação de um jogo distribuído e multi-canal recorrendo preferencialmente a ferramentas open-source. O seu desenvolvimento levou a uma análise de várias plataformas de jogos que permitissem as funcionalidades exigidas pelo jogo, e à respectiva escolha da plataforma que se apresentou como a melhor opção. Na plataforma escolhida, foram ainda efectuadas verificações ao nível da sua extensibilidade, persistência, segurança, concorrência e escalabilidade. Sendo complementadas as lacunas identificadas nesse processo.

1.1.1 DimensioneX

O DimensioneX[1] foi a plataforma de jogos seleccionada por esse projecto[1]. Esta plataforma (figura 1.1) mais não é do que uma aplicação web, escrita em Java, que funciona sobre o contentor de aplicações Web: Apache Tomcat[2]. A plataforma é composta por dois *servlets*: **multiplayer** e **maintenance**.

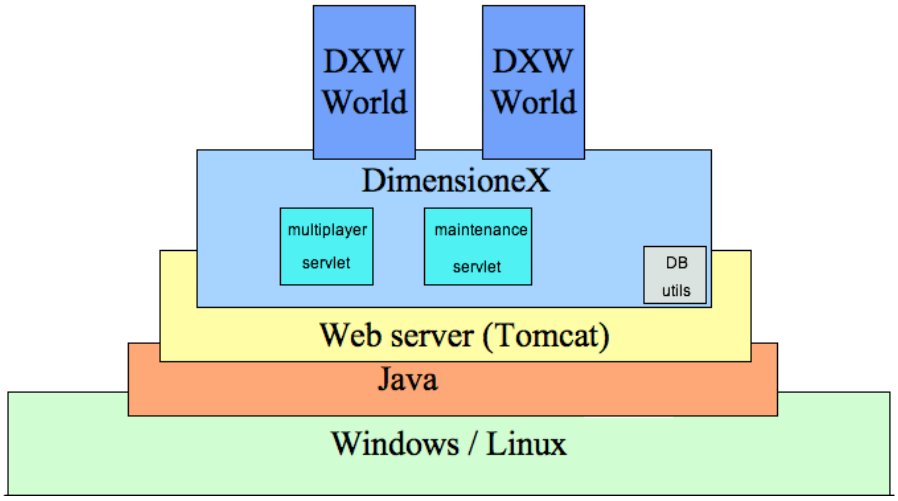

Figura 1.1 : Estrutura actual do DimensioneX.

[1] DimensioneX, Multiplayer Game Engine: http://www.dimensionex.net
[2] Apache Tomcat: http://tomcat.apache.org

Lógica do Jogo

A figura 1.2 contém as classes de objectos usadas pelo motor de jogo. Tendo a plataforma sido criada com a finalidade do desenvolvimento de jogos RPG, os objectos de jogo (AdvObject) existentes reflectem essa mesma intenção. No entanto, é também possível a criação de outro tipo de jogos (como jogos de estratégia por turnos ou aventuras gráficas). Um exemplo disto mesmo é a possibilidade de criação de um jogo de gestão de equipas recorrendo a esta plataforma[1].

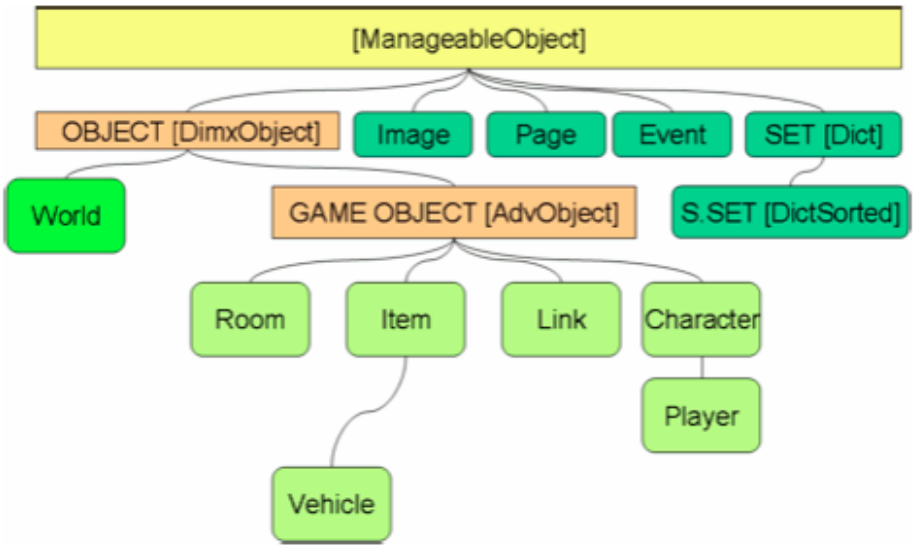

Figura 1.2 : Hierarquia das classes dos objectos do motor de jogo.

Um jogo no DimensioneX é chamado de **World** e, por si, é apenas uma descrição de salas, ligações, itens, personagens e eventos. Qualquer mudança no estado do jogo provém de uma interacção levada a cabo por um jogador (desencadeando uma cadeia de eventos). A única excepção a este facto é a invocação periódica do evento “onTick” (disparado a cada 20 segundos).

Um **Cluster** é um grupo de jogos interligados entre si, permitindo a movimentação de objectos entre eles. Este conceito mostra-se bastante interessante no desenvolvimento de jogos multi-jogador quando se pretende estender um determinado jogo, permitindo a inclusão faseada de novos cenários.

Características

A versão inicial da plataforma tinha as seguintes características:

- Suporte persistente de jogadores (através de um ficheiro XML por jogador).
- Suporte para jogos multi-jogador.
- Disponibilização de múltiplos jogos em simultâneo.
- Sistema de clusters: possibilidade de ligações entre jogos no mesmo cluster.
- Compatibilidade com qualquer browser.
- Suporte de elementos multimédia: imagens, ícones, sons e mensagens.

Após a finalização do projecto, as seguintes características foram incluídas na plataforma:

- Suporte Persistente de cada jogador através do uso de uma base de dados e respectiva cifra da informação pessoal do jogador.
- Painéis de registo, login e alteração da informação do utilizador incluídos na própria

plataforma (anteriormente teriam de ser definidos no código de cada jogo).

- Suporte multi-canal síncrono recorrendo a uma base de dados. Em cada onTick, são consultados os novos registos presentes numa tabela de interacções e executados os respectivos comandos.

1.2 Objectivos

As primeiras reuniões com os orientadores desta dissertação, permitiram estabelecer alguns dos objectivos a atingir. Chegou-se a um consenso em relação à fraca atractividade da presente interface do DimensioneX, surgindo como alternativa a criação de uma nova interface recorrendo à tecnologia *flash*[1] com o apoio da empresa beActive[2]. A expansão da plataforma para a utilização com outros canais de interacção foi também discutida em especial a interacção com dispositivos móveis (PDA e SMS).

Nas reuniões seguintes foram definidos os requisitos em concreto a atingir. A ideia principal seria separar a plataforma actual entre conteúdo (motor de jogo) e apresentação (interface web), cabendo a uma camada intermédia garantir o conjunto de operações passíveis de serem realizadas sobre o motor de jogo. Desta forma, passaria a ser possível o desenvolvimento de novas interfaces que interagissem paralelamente com o motor de jogo.

Como esta reestruturação da plataforma iria obrigar a adaptações à actual implementação da sua interface web, aproveitou-se também para repensar os dispositivos alvo desta interface. Como tal, foi decidido fazer a sua conversão para uma interface web mais abrangente, capaz de interagir também com dispositivos de pequenas dimensões.

Para demonstrar a capacidade de interacção multi-canal, decidiu-se criar uma interface SMS para a plataforma.

Seguidamente são apresentados os objectivos traçados para esta dissertação (reflectindo as decisões descritas anteriormente).

1.2.1 Reengenharia da plataforma

Sendo o alvo central desta dissertação, este processo pode ser descrito através dos seguintes objectivos:

- Suporte à interacção multi-canal;
- Facilidade de inclusão de novos canais de interacção;
- Garantia de todas as funcionalidades actuais da plataforma.
- Criação de uma interface web mais abrangente (PC, PDA, etc...).
- Inclusão de novas interfaces (SMS, Flash).

[1] Adobe Flash, http://www.adobe.com/products/flash
[2] beActive, http://www.beactive.pt

1.3 Organização e conteúdo

Para que melhor se compreenda a estrutura desta dissertação, será feita de seguida uma breve descrição do seu conteúdo.

O capitulo 2 fará referência ao trabalho de investigação realizado durante este projecto, sendo apresentadas as principais conclusões retiradas no final do mesmo.

Após uma fase inicial de análise ao DimensioneX (descrita no capitulo 3), chegou-se à conclusão das modificações a efectuar à actual estrutura da plataforma de forma a poder suportar a interacção multi-canal. Estas modificações consistem principalmente em efectuar uma separação entre a camada de apresentação e a camada de serviços (a parte lógica associada ao motor de jogo por si só). No entanto, será também necessário continuar a garantir o correcto funcionamento da plataforma especialmente em relação à concorrência das operações, segurança e persistência. A descrição de todo este processo é apresentado no capitulo 4 contendo a solução adoptada e um resumo da sua respectiva especificação[1].

Finalizada a fase de adaptação da plataforma, deverá ser aproveitada a possibilidade de criação de múltiplas interfaces recorrendo a diferentes canais de interacção. Para tal, no capitulo 5, é apresentada uma nova interface *web based* desenvolvida para esta plataforma capaz de interagir com qualquer browser de PC ou dispositivo móvel e seguidamente retratado o processo de inclusão de novas interfaces que interajam com outros canais de comunicação usando como exemplo o canal SMS. No capitulo 6 serão apresentados os resultados obtidos, enquanto que no capitulo 7 será apresentado trabalho futuro a realizar nesta plataforma (nomeadamente a inclusão de uma interface flash mais apelativa).

[1] Encontrando-se sua versão integral no Anexo A.

Capitulo 2

Estado de arte

No presente capitulo, será feita referência ao trabalho de investigação[2] realizado no desenvolvimento desta dissertação.

Encontrar plataformas para jogos multi-canal já existentes mostrou-se ser um trabalho infrutífero que apenas levou ao conhecimento de algumas ideias para jogos multi-canal[3]. Posto isto, optou-se por separar a investigação em duas áreas: a disponibilização de conteúdos através de múltiplos canais de apresentação; as características principais do DimensioneX e a sua contra-analise face a motores de jogos semelhantes. Os resultados obtidos nestas duas áreas foram posteriormente relacionados de forma a permitir estruturar um funcionamento do DimensioneX com interacção multi-canal.

2.1 Disponibilização de conteúdo em múltiplos canais

Para compreender as características inerentes a uma plataforma multi-canal, foram pesquisadas algumas abordagens para a disponibilização de conteúdos em múltiplos canais[4][5]. A ideia seria compreender o processo de pegar num determinado conteúdo[1] e faze-lo interagir com diferentes canais de apresentação, em especial perceber que tipo de restrições se encontravam associadas.

2.1.1 Heterogeneidade entre dispositivos (e respectivos canais)

O principal problema a resolver surge precisamente da existência de múltiplos canais de apresentação, assim como diferentes tipos de dispositivos a utilizar cada um desses canais. Se estas heterogeneidades já se faziam sentir ao nível do mesmo canal (p.e. na interoperabilidade entre browsers Web) mais importância apresentam quando se fala em múltiplos canais.

Para cada canal de comunicação, poderão variar os protocolos de comunicação suportados, a largura de banda disponível, o seu respectivo custo e até a sua disponibilidade. A estas condicionantes juntam-se as características próprias de cada dispositivo (mesmo entre dispositivos que usem o mesmo canal) a vários níveis: o seu poder computacional; a área de visualização suportada; as tecnologias adoptadas.

Para resolver este problema, foram estudadas duas abordagens: adaptação do conteúdo a cada canal no servidor; adaptação do conteúdo no cliente.

Adaptação do conteúdo no servidor (para cada canal)

Nesta abordagem, é realizada uma adaptação do conteúdo a disponibilizar tendo em conta as restrições de cada canal. Esta abordagem permite afastar o cliente de qualquer problema de apresentação, cabendo ao próprio servidor definir em avanço qual a apresentação do conteúdo a visualizar.

Apesar da transparência oferecida ao cliente na apresentação do conteúdo, esta abordagem também

[1] Não especificamente um jogo, mas sim qualquer aglomerado de informação

restringe a inclusão de novos canais de comunicação: o suporte a um novo canal de interacção fica dependente do servidor.

Adaptação pelo cliente

Nesta abordagem, o servidor apenas disponibiliza o conteúdo em causa em cada um dos canais de comunicação. O cliente é posteriormente o responsável por efectuar as devidas adaptações desse conteúdo ás suas próprias capacidades de apresentação ao utilizador.
O maior problema desta abordagem encontra-se no facto de ignorar a possível heterogeneidade entre os canais de comunicação. Antes de mais, e para que o conteúdo possa vir a ser adaptado em cada cliente, deve ter sido em conta a possibilidade de este poder vir a ser transmitido num determinado canal. Tal não é obrigatoriamente verdade quando existe uma grande discrepância entre as larguras de banda (ou mesmo os custos) dos canais a interagir com o servidor.

2.2 Análise de outras plataformas para jogos

Tal como durante o trabalho anterior[1], foram analisadas algumas plataformas de jogos[2]. Desta vez a ideia não seria escolher uma plataforma (já que o DimensioneX já teria sido escolhido), mas antes perceber quais as principais características deste tipo de plataformas.
A escolha das plataformas baseou-se no DimensioneX, permitindo todas elas a criação de jogos RPG ou baseados em turnos. Apesar desta semelhança, cada uma apresentava diferentes características (em relação ao DimensioneX) em pelo menos um dos seguintes aspectos: o processo de criação de jogos, a riqueza dos jogos (ao nível de grafismo e interacção) e o processo de disponibilização/distribuição dos jogos.
Estes aspectos permitiram compreender os compromissos que o DimensioneX teve de adoptar para que fosse possível disponibilizar os seus jogos ao mais vasto conjunto de utilizadores. O facto da sua apresentação ser feita recorrendo simplesmente a linguagens de marcação (e algum código javascript), torna qualquer browser Web um cliente da plataforma. No entanto, e apesar da vantagem ao nível de interoperabilidade, isto restringe o nível de grafismo suportado e também o tipo de interacção que um utilizador pode ter com a plataforma.

2.2.1 Interacção com o motor de jogo

Num jogo multi-jogador, o estado de cada utilizador não depende simplesmente das suas acções: outro jogador ou até a componente de inteligência artificial do próprio jogo poderão ser os responsáveis por tal alteração. Como tal, deve ser exigida à plataforma de disponibilização do jogo o suporte para a interacção assíncrona (da informação do estado de um utilizador). Sendo o DimensioneX uma aplicação Web que utiliza um protocolo sem estado (HTTP), foram estudados alguns dos actuais mecanismos de interacção assíncrona existentes[6]. A técnica que se mostrou mais relevante para este caso (e a qual é adoptada) baseia-se em *"polling"* permitindo que o resultado de um pedido apenas seja visualizado num pedido precedente. Esta técnica subsistiu sobre todas as outras já que se apresentou como a única que não impõe a utilização de software adicional por parte do cliente.

2.3 Definição do conteúdo a disponibilizar

Tendo em conta que o motor de jogo será acessível através de uma aplicação Web e de acordo com as restrições de interacção impostas em 2.2.1, o conteúdo a disponibilizar por esta plataforma foi definido como um estado do jogo. Este estado é representado recorrendo a um conjunto de elementos multimédia como sons, imagens e vídeos, devidamente relacionados entre si.

A interacção de um utilizador com o motor de jogo é feita recorrendo a um conjunto de acções, disponibilizadas pelo mesmo, sendo actualizado o seu estado após cada interacção. O novo estado do jogo é depois apresentado ao utilizador como resposta á sua acção.

2.3.1 Adaptações a cada canal

Das duas abordagens estudadas para a adaptação do conteúdo (na secção 2.1.1), foram adoptadas as adaptações do conteúdo realizadas no servidor. Para o efeito, foi definida uma representação genérica para o estado de um jogador num jogo. Esta representação é posteriormente adaptada no servidor de acordo com as características de cada um dos canais de interacção.

É também importante apontar que, apesar de terem sido escolhidas as adaptações ao nível do servidor, a inclusão desta representação genérica permite a extensão da plataforma a clientes que possibilitem a adaptação desta representação. Ou seja, a menos que o canal de comunicação ou o dispositivo em causa não o permitam, continua a ser possível criar clientes que se adaptem a esta representação genérica.

Capitulo 3

Análise da plataforma

Serve o presente capitulo para descrever o processo de análise realizado sobre o DimensioneX.
Sendo este um processo de engenharia reversa, foi exigido algum esforço na modelação do funcionamento actual do DimensioneX: apesar de existir uma extensa documentação ao nível da sua utilizaçao[7], tal não se pode dizer ao nível da sua estrutura conceptual propriamente dita. Isto levou a que o estudo do código fonte fosse praticamente a única fonte de documentação existente.

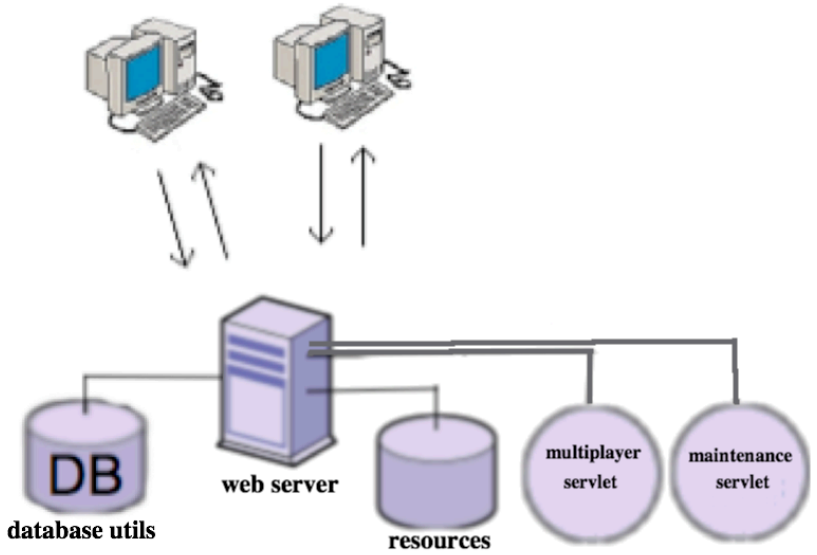

Figura 3.1 : Esquema do funcionamento do DimensioneX.
Os utilizadores interagem com a plataforma através de um browser. Os seus pedidos são reencaminhados pelo servidor para o respectivo *servlet* invocado. Os **resources** representam os ficheiros multimédia utilizados na definição dos jogos e Skins disponibilizadas pelo DimensioneX, a cada um destes ficheiros encontra-se associado um URL. O suporte persistente da informação de cada jogador é representado por **database utils**.

A figura 3.1 esquematiza o funcionamento actual da plataforma do ponto de vista das interacções com o utilizador. Tal como já foi referido, a plataforma é constituída por dois servlets: Um *servlet* **multiplayer**, responsável pelas interacções com o jogo tanto ao nível de um jogador como de um administrador; Um *servlet* **maintenance**, responsável por listar os jogos actualmente activos e permitir o acesso á sua interface de utilizador ou de administrador (redireccionando para tal o acesso para o servlet multiplayer).
O *servlet* **multiplayer** apresenta-se como a componente principal de todo o sistema sendo responsável por:

- Interagir com o utilizador: É através deste *servlet* que o browser de cada utilizador interage com o motor de jogo. São disponibilizadas operações de invocação de comandos específicos da plataforma (como registo, login ou alteração de jogo) assim como de invocação de comandos específicos de um jogo. As actualizações do estado do utilizador na plataforma são mantidas através da gestão das suas variáveis de sessão.

- Gerir clusters e jogos: Este *servlet* permite a existência de múltiplos jogos, distribuídos por diferentes clusters, sendo suportada a movimentação de objectos entre jogos do mesmo cluster. É também sua responsabilidade garantir a evolução do estado de cada jogo tendo em conta as várias interacções com os seus jogadores assim como a activação de eventos periódicos do respectivo jogo.
- Assegurar o suporte persistente dos jogadores: Para tal, a informação pessoal de cada jogador, assim como a sua posição actual, o conjunto de atributos e itens no jogo são mantidas numa base de dados.
- Administrar os vários jogos disponibilizados: Através de uma interface de administração, um administrador é capaz de executar um conjunto de comandos administrativos como reiniciar um jogo, incluir um script adicional ou recolher informação estatística sobre as invocações efectuadas no jogo.

O *servlet* **maintenance** é utilizado como uma interface de acesso a um jogo, permitindo a redirecção para a interface do respectivo jogo ou a interface de administração do mesmo. Tendo em conta a sua utilização limitada (sendo em ultima análise possível a criação de um documento HTML para a sua substituição), este não foi objecto de estudo desta dissertação.

Para facilitar a exposição da análise mais detalhada a esta plataforma, foi feita uma separação entre o que se considerou pertencer à sua interface e o que se considerou pertencer ao motor de jogo. Na secção 3.1 será descrita a análise realizada à interface do DimensioneX, é identificado o modo de apresentação de um jogo ao utilizador assim como o modo de interacção com o mesmo. Na secção 3.2 é retratado o funcionamento do motor de jogo, apresentadas as suas principais características e problemas encontrados.

3.1 Interface

A análise à actual interface do DimensioneX teve como principal objectivo a identificação de quais os elementos do motor de jogo usados na apresentação ao utilizador. Só desta forma será possível efectuar uma futura separação entre interface e motor de jogo.

3.1.1 As vistas do Jogo

Cada pedido do browser de um utilizador ao *servlet* **multiplayer** obriga à geração de um documento HTML como resposta. O conteúdo desta resposta depende do conteúdo da variável de URL **view**, sendo apresentados diferentes aspectos (vistas) do jogo consoante o seu valor.
De seguida serão apresentadas as várias vistas disponibilizadas pelo DimensioneX e os respectivos objectos do motor de jogo utilizados na sua exibição.

- **Vista de cena (view = scene)**
  Esta vista agrega um conjunto de atributos relativos à sala (Room) actual. A zona de animação é constituída pela imagem associada à sala e as imagens do respectivo conteúdo (itens, personagens, e ligações para outras salas). Nesta vista poderá ainda existir um painel

de navegação (NavPad) com mais ligações para outras salas, assim como uma lista de personagens, itens e/ou ligações que não tenham sido apresentadas na sala. Por baixo da zona de animação é apresentada (caso exista) a descrição da sala ou a mensagem de retorno da ultima acção efectuada.

Figura 3.3 : Exemplo da vista “scene”.

Objectos utilizados na apresentação:

- Player (o jogador)
- Room
- Link (para os vários caminhos possíveis)
- AdvObject (para o conteúdo da sala: itens e personagens)
- World

- **Vista de comandos (view = ctrls)**

  Esta vista contém os comandos disponíveis ao utilizador em cada instante. Durante um jogo, esta lista é composta pelo painel de comandos disponíveis do jogador e pelo painel de comandos disponíveis da actual sala. Antes do login no jogo, esta vista apresenta um painel específico do jogo de acordo com a acção a realizar (p.e. registo de jogador, painel de login ou recuperação de password).

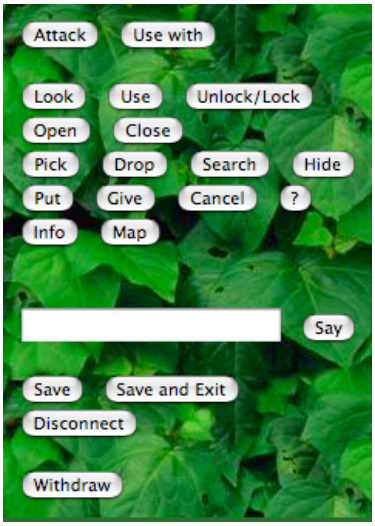

Figura 3.4 : Exemplo de vista “ctrls”.

Objectos utilizados:

- Player (contém identificadores dos painéis a visualizar nesse instante)
- World (aceder aos painéis concretos)
- Panel
- Ctrl
- Skin

- **Vista de mensagens (view = msgs)**

  Esta vista mostra todas as mensagens destinadas ao jogador, provenientes de conversas entre personagens do jogo (humanas ou não) assim como de outras interacções (como a descrição de um combate).

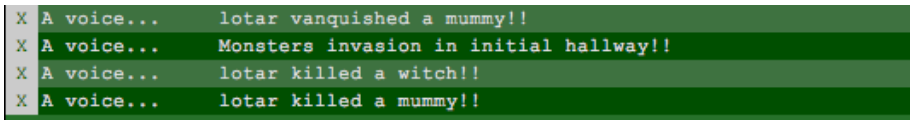

Figura 3.5 : Exemplo da vista “msgs”.

Objectos utilizados:

- Player
- Client (onde se encontram as mensagens)
- Message
- World
- Skin

- **Vista do mapa (view = map)**

  Esta vista apresenta, caso exista, o mapa descritivo do jogo e a posição actual do jogador no mesmo.

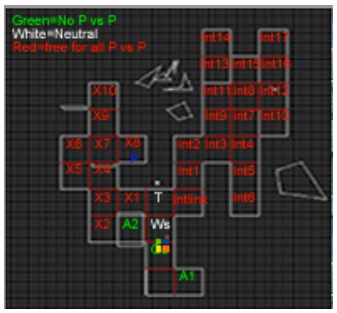

Figura 3.6 : Exemplo da vista “map”.

Objectos utilizados:

- Player (informação da posição actual)
- World (contém o URL do mapa)
- Skin

- **Vista dos jogadores (view = players)**

  Esta vista exibe o número de jogadores actualmente presentes no jogo. Caso seja incluída a variável de URL **format** com o valor “extended”, é também exibida a localização de cada jogador.

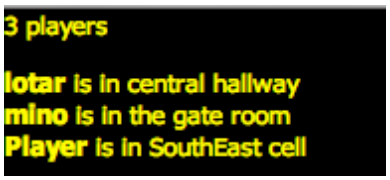

Figura 3.7 : Exemplo da vista “players”.

Objectos utilizados:

- World
- Skin

- **Vista de resultados (view = hof)**

  Esta vista mostra a lista dos jogadores que terminaram o jogo.

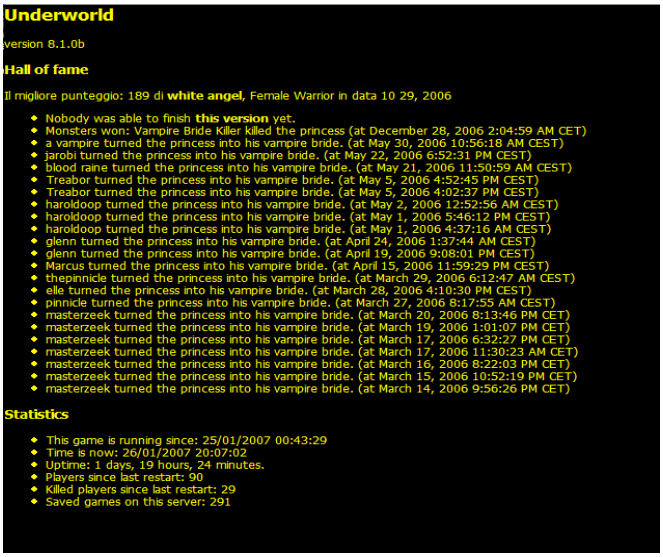

Figura 3.8 : Exemplo da vista “hof”.

Objectos utilizados:

- World
- Skin

- **Vista de administração (view = admin)**

  Esta vista exibe a interface de administração para o jogo seleccionado[1].

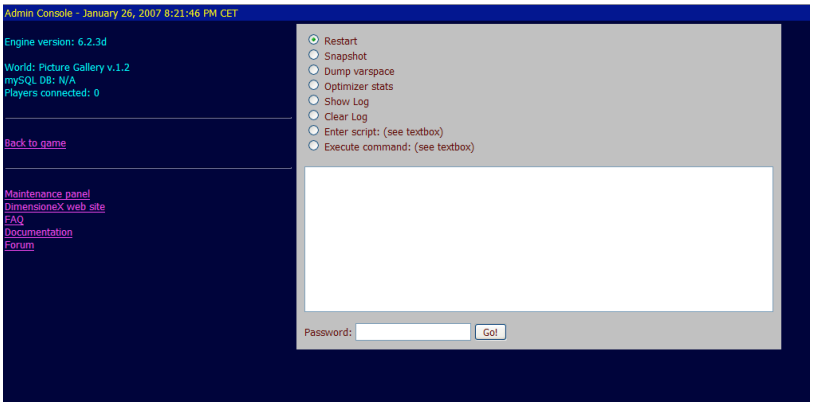

Figura 3.9 : Exemplo da vista “admin”.

[1] Developer’s Reference, Section 1.3.5

Objectos utilizados:

- World
- Skin

3.1.2 Configuração da interface

A interface do DimensioneX é composta por um conjunto das vistas anteriormente descritas. Como tal, é possível configurar quais as vistas a serem usadas, a sua disposição no browser e os mecanismos que as interligam.

Ao nível de cada jogo, existe um ficheiro de configuração onde é definido:

1. A estrutura[1] do documento HTML a ser apresentado ao utilizador (com um conjunto de frames de cada vista).
2. O nome do ficheiro com as mensagens do jogo[2], sendo possível adaptar o jogo a uma outra língua através da simples alteração deste campo.
3. As Skins[3] que podem ser escolhidas pelo utilizador.

Ao nível global, o ficheiro **client.script** define todo o código *Javascript* a ser incluído no documento HTML gerado por cada jogo. Este código tem como função fazer a interligação entre as *frames* e o documento principal, realizando operações de actualização de *frames* e invocação de comandos no *servlet* **multiplayer** quando necessário.

3.1.3 Funcionamento

Para interagir com o DimensioneX, um utilizador necessita apenas de um browser Web. Toda a comunicação é feita através do protocolo HTTP recorrendo aos métodos GET e POST.
A interacção de um utilizador com um jogo é iniciada através de um pedido ao *servlet* **multiplayer**. A identificação do jogo em causa é feita através da utilização da variável **game** incluída no URL[4]. Como resposta a este pedido , é devolvido um documento HTML gerado tendo em conta a configuração do respectivo jogo.

Para cada frame desse documento que aponte novamente para o *servlet,* é feito um novo pedido ao *servlet.* O conteúdo devolvido irá depender da variável de URL **view** que irá identificar o tipo de vista a exibir.

---

[1] Developer's Reference, Sec. 1.1.11
[2] Developer's Reference, Sec. 1.2.3
[3] Developer's Reference, Sec. 4
[4] Esta variável é guardada em sessão de forma a identificar o jogo durante os pedidos que se irão seguir relativos a cada frame.
[9] Mais uma vez recorrendo a código *Javascript*.

Os comandos a efectuar no jogo são invocados pela função sendCmd() (presente em **client.script**) a partir do cliente. Esta função efectua a submissão do formulário presente na *frame* com a vista dos comandos (ctrls) que contém (entre outros) 3 campos ocultos de nome **cmd**, **arg1** e **arg2.** Estes campos são utilizados para identificar o comando a ser invocado e até dois argumentos. O valor destes campos é também alterado através de funções disponíveis no ficheiro **client.script**.
O resultado da execução de cada comando será visualizado de imediato na frame "ctrls" e posteriormente nas restantes frames após actualização das mesmas[9].

3.2 Motor de Jogo

Seguidamente é descrita a análise feita ao DimensioneX do ponto de vista de motor de jogo. São identificadas as suas principais funções e revisto o seu funcionamento interno.

3.2.1 Principais funções

Interacção com o jogo

Tal como já referido no capitulo introdutório, cada jogo é definido por um conjunto de salas, ligações, itens, personagens e eventos. Para que existam alterações no estado de um jogo, é necessário que haja uma interacção por parte de um utilizador. É da responsabilidade do *servlet* **multiplayer** a manipulação interna do estado do jogo para reflectir essa mesma interacção.

Gestão dos jogos

Para que seja possível a disponibilização de múltiplos jogos, é guardada uma instância por cada jogo disponível pela plataforma, agrupados pelo seu respectivo cluster. O carregamento do conteúdo de cada jogo é feito durante a primeira interacção com o mesmo, sendo usados os seus respectivos ficheiros de configuração[1]. A movimentação de objectos entre jogos do mesmo cluster é feita, caso existam, no final de cada interacção com o utilizador.

Evento Periódico (onTick)

Um mecanismo curioso é o de activação do evento periódico onTick. Segundo o manual de utilizador, este evento é disparado de 20 em 20 segundos para cada jogo da plataforma, no entanto tal poderá não se verificar. Este evento é executado em cada jogo sempre que tenham passado, **pelo menos**, 20 segundos desde a sua anterior execução (nesse mesmo jogo), sendo esta verificação efectuada **sempre que existe uma interacção de um utilizador com esse jogo**.

Persistência da informação de cada jogador

A informação de cada jogador é guardada numa base de dados a partir do momento em que o mesmo se regista num determinado jogo. Esta informação é posteriormente actualizada em cada invocação ao evento onTick desse mesmo jogo.

Interacção multi-canal síncrona

[1] Ver Developer's Reference 2.2 .

Recorrendo a uma tabela na base de dados, o DimensioneX possibilita a interacção com outros canais de comunicação. A tabela da base de dados é usada para guardar as interacções a realizar na plataforma. Durante a invocação do evento onTick, as interacções presentes nessa tabela são executadas.
Apesar deste mecanismo se apresentar como uma solução para a interacção multi-canal, é extremamente limitativo não fornecendo o nível de interacção mínimo que um jogo exige. Por outro lado, este mecanismos também não possibilita a leitura do resultado das invocações.

3.2.2 Funcionamento interno

De seguida é feita uma análise ao funcionamento interno da plataforma. Como tal, são descritos os mecanismos usados para garantir o acesso concorrente aos jogos, o modo como a informação de cada jogador é guardado de forma persistente, alguns aspectos de segurança encontrados na plataforma e por ultimo quais as actuais hipóteses de extensão da plataforma.

Concorrência

Sendo a plataforma multi-jogador, encontram-se implementados mecanismos de exclusão mútua que garantem a correcta actualização dos jogos (worlds) e respectivos clusters após acessos concorrentes de vários jogadores. Apesar do uso de secções criticas ser imperativo (tendo em conta a natureza da plataforma), a implementação actualmente em vigor apresenta-se como extremamente restritiva e impossibilita a escalabilidade da plataforma.
Para garantir o sincronismo na leitura/escrita de dados em um determinado jogo (world), o *servlet* **multiplayer** apenas trata de uma interacção por jogo em cada instante. Ficando todas as outras em espera. De notar que não é feita qualquer verificação sobre o conteúdo da interacção a realizar (i.e. não são feitas distinções entre operações de leitura ou escrita ou mesmo uma operação inválida).
Tendo também em conta o sistema de *frames* utilizado pela interface, que obriga a várias interacções com o *servlet* **multiplayer** em cada operação do utilizador (como descrito na secção 3.1.3), ainda é mais visível o impacto que este sistema causa na degradação da performance de toda a plataforma.

Persistência

A plataforma utiliza uma base de dados para guardar informação sobre cada jogador. A informação guardada contém os dados pessoais do jogador, informação necessária à identificação de cada jogador (username/password/info de password) e o seu estado actual no jogo.
A implementação desta funcionalidade ainda se encontra num estado bastante experimental pelo que foram encontrados alguns problemas graves. Um dos problemas detectados deve-se ao facto de ter sido implementada com a ideia de garantir a persistência de apenas um jogo, o que não deveria acontecer tendo em conta que o DimensioneX suporta vários jogos em simultâneo. No que toca à estrutura das tabelas da base de dados, existe uma relação dupla entre a tabela de jogadores e a tabela de itens totalmente desnecessária que apenas exige espaço extra e obriga a um maior número de acessos à base de dados.

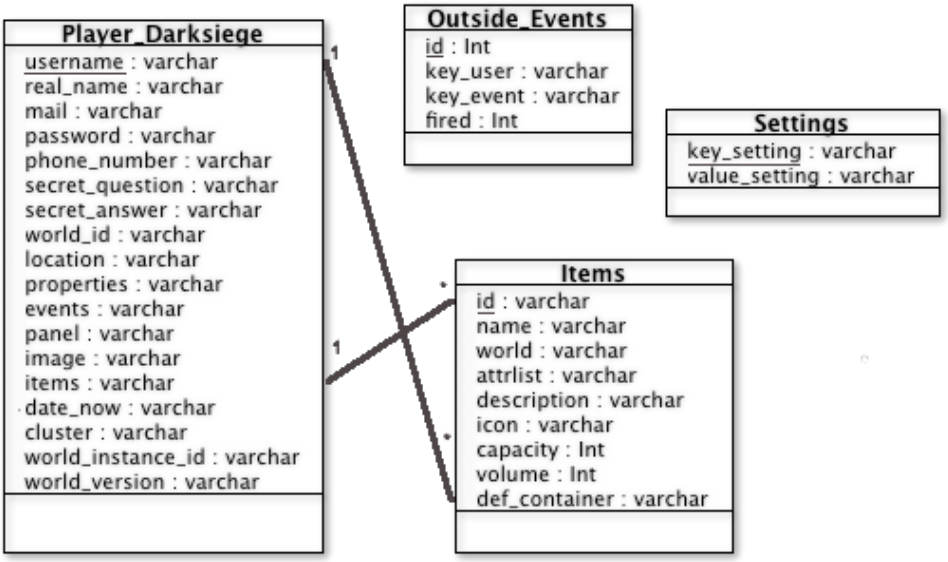

Figura 3.10 : Tabelas usadas para suporte persistente.

De certa forma também relacionado com esta estrutura, notou-se alguma ineficiência na utilização de operações de guardar e restaurar informação da base de dados. Durante o evento onTick(), para cada jogador activo, é efectuada uma consulta e uma escrita para cada item que esse jogador possua[1]. Esta operação sofre ainda um peso adicional já que se encontra dentro da secção critica descrita anteriormente.

O tratamento de excepções nas funções com ligação à base de dados também possuem algumas falhas, apanhando por vezes erros desnecessários (reportando os mesmos nos ficheiros de log).

Segurança

A segurança desta plataforma é um tema bastante relativo. Actualmente, o DimensioneX ainda se apresenta como um projecto académico com pouca incidência no mundo comercial. Sobre esta perspectiva, os problemas apontados e resolvidos pelo projecto anterior[1] foram suficientes para completar essas lacunas.

De qualquer forma, existem alguns aspectos de implementação que deveriam ser tomados em consideração:

Estando a plataforma assente sobre uma aplicação web, deve existir um uso rigoroso das variáveis de sessão/pedido. Isto é causado em parte pela estrutura do código do DimensioneX, mas também por vezes por alguma imponderação: é por vezes possível criar cenários em que se consegue alterar o fluxo de execução esperado de um pedido.

O uso de variáveis de instância num *servlet* Java é também algo que deve ser usado com alguma prudência. Mesmo tendo em conta que cada pedido é tratado sequencialmente pelo *servlet*, é preciso ter em atenção que o valor de tais variáveis poderá mudar entre dois pedidos, o que nem sempre é tomado em consideração.

[1] Se tomarmos como exemplo um jogo com 10 jogadores activos com uma média de 3 items por jogador, serão efectuadas 30 operações de consulta e mais 30 operações de escrita em cada tick.

A informação de cada utilizador guardada na base de dados encontra-se cifrada com uso ao algoritmo DES. Para cada utilizador é usada uma chave diferente gerada a partir da função de resumo MD5 e o seu nome de utilizador. Adicionalmente, em vez de ser guardada a palavra-chave cifrada, é antes guardada o valor da sua função MD5. O nome de utilizador é por sua vez guardado na base de dados sem que lhe seja aplicada qualquer cifra. Isto permite a qualquer atacante, que consiga obter acesso à base de dados, extrair toda a informação pessoal que se julgava ilegível. Esta informação poderá ser utilizada, por exemplo, para posteriormente alterar a palavra-chave de um utilizador (recorrendo à funcionalidade de recuperação da palavra-chave).

3.2.3 Extensibilidade

Para facilitar a extensão de jogos, o DimensioneX possui o mecanismos de clusters e o sistema de configuração de mensagens. O mecanismos de clusters permite a inclusão de extensões a um jogo com um número de alterações ao jogo inicial mínimo, tornando-se apenas necessária a codificação da ligação entre o jogo inicial e a respectiva expansão criada. O sistema de configuração de mensagens a partir de ficheiros de texto permite estender um jogo a outros idiomas sendo para isso apenas necessário a inclusão do texto traduzido num ficheiro de texto, reconfigurar o jogo para utilizar o ficheiro em questão e seguidamente reiniciar o jogo.

Quanto à extensão da interface da plataforma, a utilização de *templates* no código HTML a ser gerado permite alguma flexibilidade na alteração do aspecto da interface do DimensioneX, sendo possível reestruturar a informação apresentada ao utilizador através de um conjunto de primitivas. No entanto, o nível de detalhe dessas primitivas é bastante baixo, não permitindo grande manipulação dos elementos da interface (como itens, jogadores ou mensagens) mas apenas ao nível das vistas anteriormente descritas (na secção 3.1.1).

Apesar da extensão a outros canais de interacção ser possibilitada através do recurso à base de dados, esta ficará sempre dependente da frequência de activação do evento onTick o que limita o tipo de interacção realizado a um funcionamento síncrono. Assim, a adaptação desta plataforma a um funcionamento assíncrono e multi-canal apresenta-se possível, apenas, através da modificação dos ficheiros fonte da plataforma.

3.3 Definição dos requisitos

Depois de analisada a interface e o motor de jogo do DimensioneX, é agora possível passar à definição dos requisitos a atingir com este projecto (de forma a cumprir os objectivos).

Motor de jogo

1. Interacção multi-canal
    a. Encapsulamento do motor de jogo
        i. Definição dos tipos de invocações realizáveis sobre o motor de jogo (comandos, tipos de argumentos e tipos de retorno).
        ii. Criação de vistas para os objectos de domínio do motor de jogo usados na definição dos tipos de invocações.

b. Inclusão de ficheiros multimédia de alta e baixa qualidade adaptáveis à largura de banda do canal de interacção.

2. Garantir o funcionamento actual da plataforma.
    a. Criação de um mecanismos de gestão dos jogos.
    b. Actualização da camada persistente.
    c. Gestão de acessos concorrentes.

Interface

1. Interface web genérica
    a. Disponibilização da informação mais relevante sempre tendo em conta nível de informação suportado por cada dispositivo.
    b. Facilidade de configuração da sua estrutura.
    c. Garantir que sejam mantidas todas as funcionalidades da anterior interface web.
2. Interface SMS
    a. Operações de registo, *login* e *logout* de um jogador.
    b. Informação sobre o estado actual do jogador.
    c. Possibilidade de invocação de comandos de um jogo.

Capitulo 4

Reestruturação da plataforma

Feita a análise do funcionamento actual do DimensioneX, chega agora a altura de descrever o trabalho realizado sobre a mesma para atingir os requisitos na secção 3.3.

Tal com já referido nos objectivos, foi decidido realizar uma separação entre a apresentação do conteúdo (neste caso, a apresentação do estado de um jogo aos seus jogadores) da lógica por detrás da geração desse mesmo conteúdo (o funcionamento do motor de jogo). Esta separação serviu também para permitir a inclusão de novos canais de interacção (e respectivas interfaces) na plataforma. Para o efeito, foi adoptado o modelo MVC (Modelo – Vista - Controlador) sendo usado um controlador, responsável por fazer a correspondência entre invocações de acções (serviços) de várias interfaces (de diferentes canais) com o motor de jogo. A estrutura adoptada encontra-se exemplificada na figura 4.1.

Para a definição deste controlador, recorreu-se à definição de uma API dos serviços disponibilizados pelo motor de jogo e do respectivo conteúdo devolvido pelos mesmos.

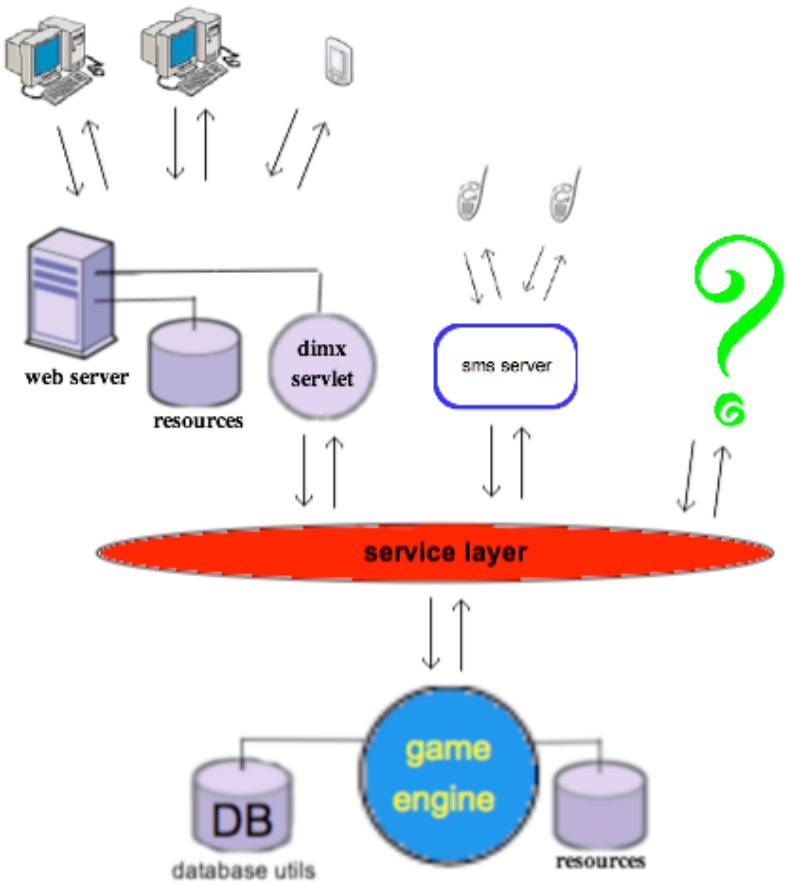

Figura 4.1 : Nova estrutura do DimensioneX.

Nesta nova estrutura, existe uma camada de abstracção (service layer) que separa o motor de jogo das suas várias interfaces. Desta forma, o acesso ao motor de jogo é feito recorrendo a esta camada, permitindo ao mesmo receber invocações de qualquer interface que suporte a comunicação com essa camada.

Na secção 4.1 será descrito o processo de alteração da plataforma para inclusão do referido

controlador, a especificação dos serviços fornecidos, assim como o conteúdo devolvido pelo mesmo. Seguidamente na secção 4.2 é apresentado o novo funcionamento do motor de jogo e também as alterações efectuadas na resolução dos problemas internos encontrados anteriormente.

4.1 Controlador

O acesso ao motor de jogo passou a ser feito através de um conjunto de serviços (i.e. tipo de interacções) disponibilizados, assim como o respectivo conteúdo devolvido por cada um desses serviços. Desta forma, o motor de jogo passa a ser acedido simplesmente através deste conjunto de serviços independentemente da interface que esteja a ser usada para o efeito.

Para que o acesso ao motor de jogo seja independente da plataforma adoptada, foi criado um *web service* recorrendo à tecnologia JAX-WS[1]. Desta forma, a comunicação entre cliente (interface) e servidor (motor de jogo) é feita apenas recorrendo a *standards* da industria (HTTP, SOAP e WSDL)[2].
A disponibilização do motor de jogo através de um *web service* permite também deixar de limitar o numero de interfaces disponíveis a cada instante: com esta abordagem, uma nova interface poderá iniciar (ou terminar) de interagir com o motor de jogo sem que seja interrompido o funcionamento do mesmo.

Sendo toda a especificação deste controlador um pouco extensa, esta foi incluída no Anexo A, sendo de seguida feito um resumo do seu conteúdo.

4.1.1 Definição das vistas

Identificados (na secção 3.1.1) os elementos de cada objecto do motor de jogo utilizados pela interface, procedeu-se à criação de versões desses objectos apenas com esse conteúdo. Os dados serviram para posteriormente serem utilizados na definição dos parâmetros e tipos de retorno dos serviços a invocar sobre o motor de jogo. Desta forma, o motor de jogo passa a ser visto como uma "caixa negra", cabendo ao invocador de um serviço conhecer a respectiva API que o descreve. Esta API conta não só com a especificação de todas as vistas dos objectos do motor de jogo criadas, mas também com alguns valores de domínio necessários á correcta compreensão dessas vistas (p.e. a representação textual dos tipos de AdvObjInterface[3]).

4.1.2 Serviços

A definição do conjunto de serviços a disponibilizar pelo motor de jogo, e através dos quais qualquer interface poderá interagir com o mesmo, encontram-se também especificados no Anexo A. São indicados os tipos de parâmetros de cada serviço e respectivo tipo de retorno assim como as condições necessárias á sua execução.

---

[1] Java Api for XML Web Services, https://jax-ws.dev.java.net .
[2] Standards definidos pelo World Wide Web Consortium, http://www.w3c.org .
[3] ver anexo A.

4.1.3 Compromissos estabelecidos

Com a separação da plataforma, surgiram algumas decisões a tomar: que tipo de segurança deverá existir entre interfaces e motor de jogo? Como serão os ficheiros multimédia acedidos por cada interface?

Apesar do funcionamento do DimensioneX passar, com esta nova estrutura, a ser distribuído, foi decidido continuar a definir a plataforma como o conjunto de todas as suas componentes. Assim, é considerado que tanto a rede de interligação dos seus vários componentes, assim como os próprios, são seguros. Como tal, será feito qualquer controlo de acesso ás invocações dos serviços realizadas pelas interfaces (fora as verificações normais associadas a cada serviço[1]).

Para resolver o problema de acesso aos ficheiros multimédia do motor de jogo, duas abordagens foram pensadas: Uma das abordagens seria o de criar um ponto de acesso comum a todas as interfaces, desta forma qualquer alteração a estes ficheiros seria prontamente propagado para todas as outras; A segunda abordagem seria o de replicar os ficheiros do motor de jogo para cada uma das interfaces, facilitando o escalamento com a desvantagem da sincronização dos repositórios.
Como já à partida o DimensioneX suporta a definição de ficheiros multimédia remotos nos seus jogos, optou-se por usar a segunda abordagem. Desta forma, é deixado ao critério do administrador da plataforma qual dos dois sistemas a adoptar.

4.2 Motor de Jogo

Depois de apresentada a camada de serviços, chega agora o momento de estruturar o funcionamento do motor de jogo. Este trabalho incidiu sobretudo nos mecanismos de actualização e gestão dos jogos, tendo os objectos específicos do jogo sido praticamente inalterados (à excepção da remoção de algumas ligações á interface e a inclusão de novas funcionalidades).

4.2.1 Gestor de jogos

O grande responsável por todo funcionamento do motor de jogo passou a ser o objecto WorldManager. Este objecto agrega o conjunto de Clusters disponíveis e os seus respectivos jogos (Worlds), sendo apenas através dele que se torna possível o acesso a qualquer jogo.

[1] p.e. a verificação da palavra-chave de administrador na execução de um commando de administrador

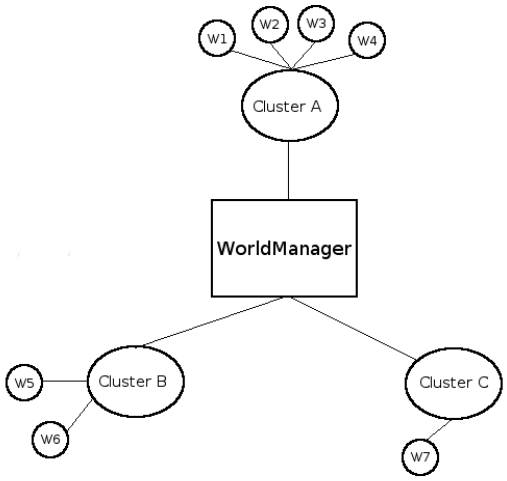

Figura 4.2 : Exemplo da estrutura de um WorldManager.

Na figura 4.2 pode ser visto um exemplo do conteúdo de um gestor de jogos. A adicionar ao mecanismo de acesso a cada jogo, é também da responsabilidade deste objecto a (re-)inicialização de um jogo, a movimentação consistente de objectos entre jogos do mesmo cluster e a inicialização do suporte persistente.

4.2.2 Serviços disponibilizados

Tal como já referido, encontra-se especificado no Anexo A o conjunto de serviços disponibilizados pelo motor de jogo. Tendo já sido descritas as características de cada serviço, será agora explicado o seu conteúdo.

A criação de cada um destes serviços teve como base a análise ao anterior *servlet* **multiplayer**, tendo sido identificadas as várias acções que eram realizadas e criado um serviço para cada uma delas.

As figuras 4.3 e 4.4 mostram os diagramas hierárquicos das classes de serviços criados. A adopção desta estrutura permitiu uma mais facilitada extensão da plataforma a novo serviços, algo que não se apresentava como trivial anteriormente.

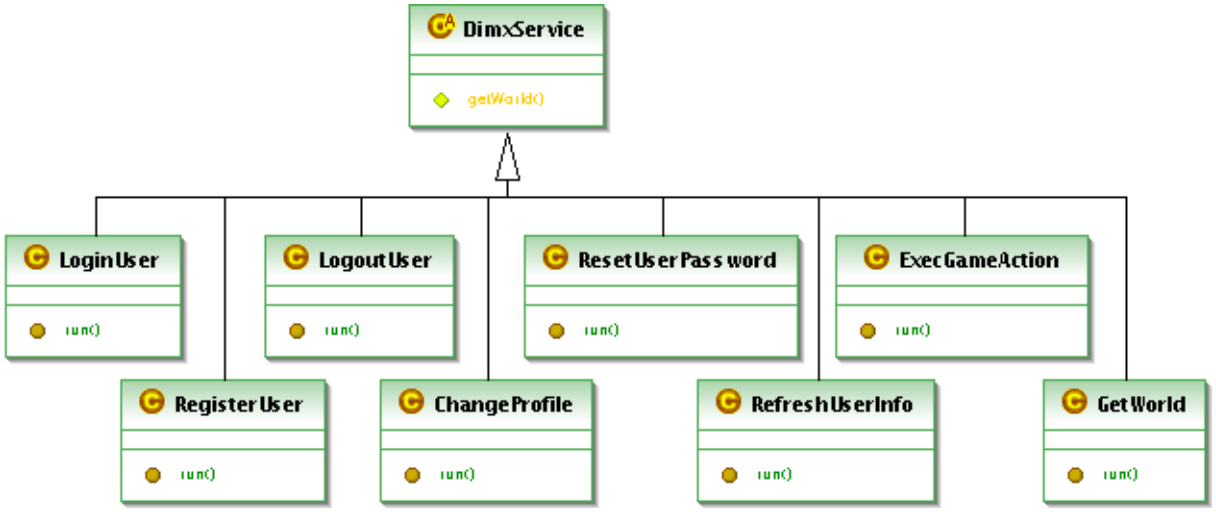

Figura 4.3 : Diagrama de classes UML dos serviços normais e de utilizador do motor de jogo.

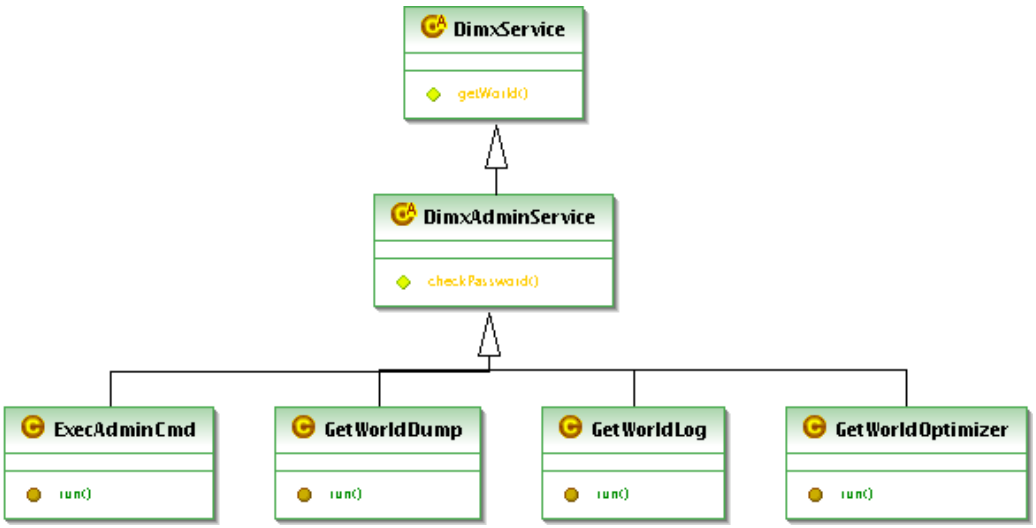

Figura 4.4 : Diagrama de classes UML dos serviços de administrador do motor de jogo.

Durante cada invocação à camada de serviços, é criado um objecto, da classe do tipo da invocação, e executado o seu método **run()** com os respectivos argumentos. Tendo em conta a possibilidade de poderem existir várias invocações a serem tratadas concorrentemente, é da responsabilidade de cada serviço a gestão dos acessos concorrentes e respectivas secções criticas.

4.2.3 Alterações internas

Seguidamente serão descritas as alterações internas realizadas ao motor de jogo. As alterações reflectem os requisitos impostos na secção 3.3.

Mecanismos de acesso concorrente

Foram incluídos métodos no objecto World (responsável por cada jogo) que manipulam um *lock* de leitura e outro de escrita. Desta forma, os serviços poderão utilizar este mecanismo para definir secções criticas tanto de leitura como escrita e com isto melhorar a performance da invocação de cada serviço: quando dois serviços são executados concorrentemente (por duas threads distintas), apenas poderá haver o bloqueio de uma delas caso algum desses serviços adquirir um *lock* de escrita.

A figura 4.5 mostra o exemplo de uma sequência de invocações possíveis. Neste caso, são demonstrados dois possíveis tipos de bloqueio: uma thread bloquear ao tentar adquirir o *lock* de escrita enquanto não forem libertados todos os *locks* de leitura; e uma thread bloquear ao tentar adquirir o lock de leitura enquanto não for libertado o lock de escrita.

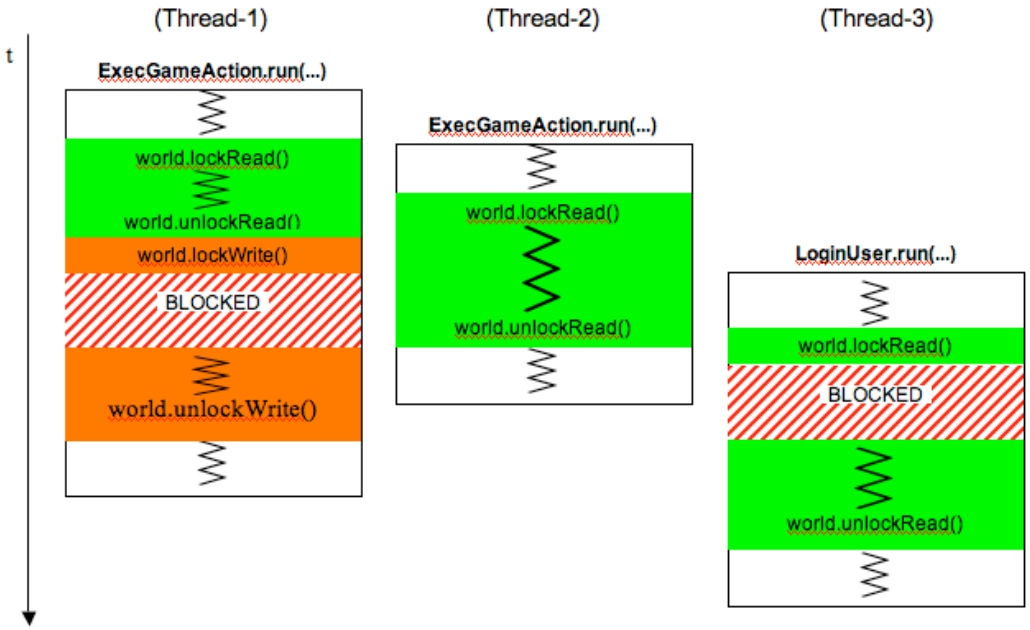

Figura 4.5 : Exemplo de acessos concorrentes a um jogo (world).
A zona a verde representa uma secção critica de leitura enquanto que a zona a vermelho representa uma secção critica de escrita.
1) A thread-1 fica bloqueada quando tenta adquirir o lock de escrita, ficando nesse estado até que todas as threads actualmente com o lock de leitura (apenas a thread-2 neste caso) o libertem.
2) A thread-3 fica bloqueada quando tenta adquirir o lock de leitura, ficando nesse estado até que todas as threads actualmente com o lock de escrita (apenas a thread-1 neste caso) o libertem.

Suporte de eventos periódicos

Para que o evento onTick passasse a ser invocado periodicamente, foi criado um objecto de nome WorldTick que representa uma thread cuja única função é a execução deste evento. A cada objecto World passou a estar adicionado um WorldTick responsável pela activação periódica deste evento, deixando de estar dependente de qualquer interacção com o utilizador. Tendo em conta o suporte persistente garantido durante este evento, a sua execução é protegida por um *lock* de leitura.

Novas funcionalidades

Tendo em conta a possibilidade de interacções com o motor de jogo de canais com menor largura de banda, foi modificado o objecto Image do motor de jogo para permitir a inclusão de um elemento alternativo, de menores dimensões. Esta alteração obrigou também a que fosse mudado o *parsing* da definição IMAGE[1] na linguagem de descrição de um jogo (DXW), de forma a que seja possível ao programador de um jogo acrescentar dois, em vez de um, URLs. O URL adicional é opcional e, quando incluído, refere-se ao objecto a ser usado para apresentação em dispositivos de menor dimensão.

[1] Ver Developers Reference Secção 2.10 .

Resolução de problemas

Foi realizada uma reestruturação do funcionamento do mecanismo de guardar e recuperar o estado de cada jogador. Quanto à estrutura da base de dados, para cada Cluster passaram a ser criadas 3 tabelas: uma de registo da informação de cada jogador , uma relativa aos itens de cada jogo que pertençam a algum jogador e uma com informação adicional relativa a cada jogo.

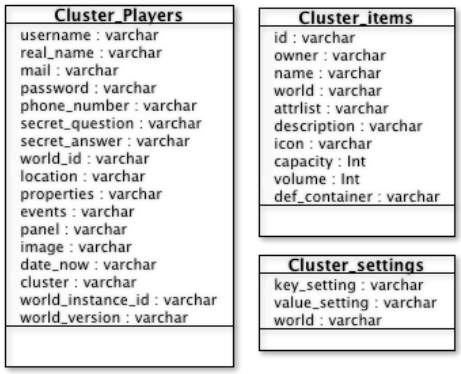

Figura 4.6 : Tabelas utilizadas por cada Cluster na base de dados.

A confidencialidade dos dados pessoais de cada utilizador continuam a depender dos níveis de segurança da base de dados, se esta for comprometida assim serão os dados do utilizador lá guardados. No entanto, e para evitar que alguém com acesso à base de dados consiga ter informação suficiente para alterar a password de um utilizador, o campo **secret_answer** da base de dados passou a guardar o seu *hash* MD5 em vez do seu valor real.

Capitulo 5

Interfaces

Feito o encapsulamento do motor de jogo e respectiva adaptação à interacção multi-canal, passou-se à fase de desenvolvimento das suas interfaces. Na secção 5.1 será descrito o trabalho realizado ao nível da interface antiga da plataforma, é apresentada uma nova interface web e as suas (novas) funcionalidades. Na secção 5.2, é feita referência à agora possível adição de novas interfaces que interajam com o motor de jogo, recorrendo a outros canais de comunicação. Ainda nessa secção, e a nível de exemplo, é apresentado um simulador de uma interface do DimensioneX que recorre a um canal de SMS.

5.1 Interface Web

Tal como referido em 1.2.1, definiu-se o compromisso de criar uma interface web com novas funcionalidades. Numa primeira abordagem, foram realizados esforços no sentido de converter a interface actual do DimensioneX para suportar a nova estrutura da plataforma. No entanto, o modo como era gerado o código HTML a devolver ao jogador, assim como a complexidade necessária para a inclusão de novos elementos na interface, originou uma mudança no planeamento sendo abandonado o processo de adaptação. Em seu lugar, foi definida uma nova interface web-based para o DimensioneX capaz de manter as mesmas funcionalidades da anterior mas também com uma estrutura mais configurável e capaz de gerar código XHTML mais limpo.
Seguidamente será apresentada a estrutura da nova interface *web* e posteriormente as novas funcionalidades trazidas pela mesma.

5.1.1 Estrutura

A estrutura interna da interface continua a ser suportada por um *servlet* em linguagem Java, no entanto, a geração de código que primeiro era feita recorrendo a simples escritas no *buffer* de saída (dos pedidos HTTP de clientes), agora é feita utilizando uma ferramenta *open-source* de geração de texto através da utilização de *templates*[1]. Como tal, foram criados *templates* para as possíveis cenas a devolver ao browser do cliente, cada um destes *templates* é capaz de transformar os vários elementos internos da interface num único documento a devolver ao cliente. Desta forma a reedição dos *templates* torna-se possível mesmo por um *web developer* com pouca ou nenhuma experiência em Java, precisando apenas do conhecimento dos elementos internos usados.
A informação incluída nestes elementos, a usar nos templates, é depende do estado do jogador no jogo, sendo actualizada sempre que existe uma invocação ao motor de jogo proveniente do respectivo utilizador.

[1] Freemarker, http://freemarker.sourceforge.net/

5.1.2 Generalização do canal web utilizado

A nova interface implementada passou a gerar código em XHTML, seguindo as recomendações W3C. Desta forma, o código gerado passou a utilizar uma linguagem de descrição mais abrangente, capaz de ser correctamente renderizada não só nos tradicionais browsers de um computador pessoal, mas também em browsers de dispositivos móveis como PDA ou alguns telemóveis.
Durante o seu desenvolvimento, e também a pensar na multiplicidade de dispositivos alvo desta interface, foi também repensada a forma como a informação deveria ser apresentada ao utilizador. Como tal, foram incluídas as seguintes características:

Detecção de *small screen*: Utilizando código Javascript no cliente, foi introduzido um sistema de detecção automática do tamanho do ecrã. Esta informação permite a posterior escolha de qual o tipo de imagens a mostrar ao utilizador – a imagem normal ou a imagem destinada a dispositivos de dimensão reduzida[1].

Caixas de tamanho fixo com *scroll*: Foi criada uma secção de tamanho fixo para apresentar a informação de resposta a uma interacção e outra para a caixa de mensagens do utilizador. Isto permite manter o mesmo tipo de apresentação usado através do sistema de frames anterior.

Botões de minimizar/expandir: Foram incluídos botões deste tipo em todas as listagens de objectos. Desta forma, torna-se possível ocultar informação que não seja tão relevante para o utilizador enquanto aumenta o campo visual de outro tipo de informação mais pertinente. Estas listagens fazem parte da apresentação de cada sala (Room) e o seu conteúdo varia entre informação dos atributos do jogador, itens no quarto, jogadores no quarto e ligações para outros quartos.
Estes botões foram também incluídos em cada mensagem da caixa de mensagens do utilizador.

Agrupamento dos comandos por estado: A anterior interface apresentava ao utilizador todos os comandos possíveis de serem executados. No entanto, alguns desses comandos apenas poderiam ser executados depois de seleccionados os seus respectivos argumentos. Para prevenir esse excesso de informação, os comandos foram agrupados tendo em conta o seguinte critério: no caso de nenhum objecto estar seleccionado, são apresentados os comandos sem argumentos; no caso de estar um objecto seleccionado, são apresentados os comandos de um e dois argumentos[2].

Apresentação selectiva dos elementos do quarto (Room): O facto de estar, ou não, um objecto seleccionado influencia a apresentação dos elementos presentes na sala. Os objectos apenas se encontram visíveis no caso em que ainda nenhum foi seleccionado ou então no caso de se estar a seleccionar um segundo objecto (neste caso, o primeiro objecto seleccionado já não se encontra visível).

5.1.3 Templates das cenas

A geração do documento (XHTML) a devolver ao utilizador não depende apenas da actualização dos

[1] A inclusão destes dois tipos de imagem encontra-se descrita na secção 4.2.3 .
[2] a selecção de um comando deste tipo é precedido por um cena para escolha do segundo argumento.

elementos internos da interface (como o conteúdo do jogo, da sala ou até do jogador) mas também de qual o *template* usado. Seguidamente serão descritos os *templates* criados agrupados pelo contexto da sua utilização.

Cenas da plataforma. Para apresentar os menus de jogo *offline* da plataforma (como o login, registo de utilizador ou actualização do perfil) foi criado um único *template* (offlineScreen.ftl) que mostra o painel de comandos do menu em causa e opcionalmente o logótipo do jogo em causa. Para a apresentação de erros graves na plataforma é usado um outro *template* (erroScreen.ftl) e um outro ainda para a detecção do tamanho do ecrã do utilizador (detectClientScreen.ftl).

Cenas de jogo. Durante um jogo, uma cena pode se encontrar em 3 estados: o utilizador não tem nenhum objecto seleccionado na cena; o utilizador tem um objecto seleccionado; o utilizador vai seleccionar um segundo objecto. Os dois primeiros estados possíveis são gerados recorrendo ao mesmo *template* (onlineScreen.ftl) enquanto que a cena de selecção do segundo objecto é gerada recorrendo a um outro *template* (chooseArg2Screen.ftl).

Cenas de administração. Para apresentar o menu de comandos de administrador, foi criado um outro *template* (worldAdminScreen.ftl). A informação devolvida por alguns desses comandos também exigiu a criação de mais cenas sendo elas: cena de optimização (worldOptimizerScreen.ftl); cena de log (worldLogScreen.ftl); cena de estado do jogo (worldSnapshotScreen.ftl); cena de listagem das variáveis de jogo (worldDumpScreen.ftl);

5.2 Demonstração de outras interfaces

5.2.1 Desenvolvimento de novas interfaces

Face à nova estrutura do DimensioneX, torna-se possível a existência de múltiplas interfaces a interagir com um único motor de jogo. A esta característica, junta-se o facto de ser possível a inclusão/remoção de uma interface sem condicionar o funcionamento do motor de jogo.
O processo de criação de uma interface poderá ser estruturado da seguinte forma:

Processo de criação de uma nova interface

1. Escolher canal de interacção com o utilizador.
2. Escolher serviços a usar do motor de jogo.
3. Definir grau de detalhe da informação, devolvida por cada serviço escolhido, a disponibilizar.
4. Codificar a correspondência entre interacções com o utilizador e respectivas invocações de serviços no motor de jogo.

Durante este processo, as decisões impostas ao programador da nova interface todas têm a ver com a forma de disponibilização da informação. Tendo em conta o canal de interacção com o utilizador e o respectivo dispositivo utilizado, o programador deverá escolher os serviços do motor de jogo a disponibilizar assim como seleccionar a informação relevante de toda a informação devolvida por cada um desses serviços. O programador é depois livre de escolher o modo como essa informação

será disponibilizada ao utilizador.

5.2.2 Exemplo – Interface SMS

Para demonstrar a interacção do motor de jogo com novas interfaces, foi desenvolvido um pequeno servlet que simula um telemóvel a comunicar por SMS.

Figura 5.1 : Screenshot do simulador de SMS.

Face aos custos de cada mensagem e o limite de 160 caracteres por mensagem, a utilização desta interface deverá ser bastante limitada.

Os serviços a disponibilizar seriam os de registo, login e logout de um utilizador, um serviço de informação do estado do jogador no jogo e um serviço de execução de um comando no jogo. A ideia é que a interface disponibilize um sistema de registo de um jogador, assim como um serviço de subscrição a cada jogo e também um conjunto de comandos a realizar no mesmo.

Para que a especificação desta interface esteja concluída, resta apenas descrever as relações entre as interacções dos utilizadores com os serviços disponibilizados pelo motor de jogo. A tabela seguinte faz essa relação.

Serviço da Interface SMS	Serviço do Motor de jogo
Registo de um jogador	RegisterUser
Iniciar Subscrição / Terminar Subscrição	LoginUser / LogoutUser
Estado do jogador no jogo	RefreshUserInfo
Interacções do jogador no jogo	ExecGameAction

Tabela 3.1 : Relação entre interface SMS e motor de jogo

Resta apenas referir que a informação dada ao jogador sobre o estado do jogo será mínima, correspondendo à sua localização e eventos que lhe tenham sido activados. A lista de comandos passíveis de serem utilizadas pelo jogador poderá ser disponibilizada pela interface e requer que o

utilizador já possua alguns conhecimentos sobre a sua correcta utilização (através da leitura do respectivo manual do jogo).

Capitulo 6

Resultados

6.1 Desenvolvimento do motor de jogo

A fase de isolamento do motor de jogo, e respectiva criação dos serviços e vistas, tornou-se na fase com maior tempo despendido durante este projecto. Dos maiores responsáveis pelo tempo gasto, podem ser destacados a análise ao funcionamento da plataforma e as alterações internas feitas no motor de jogo.

Durante a fase de análise à plataforma, deparei-me com uma interface de geração de código HTML codificada em Java composta por várias funções (para cada tipo de cena a mostrar). A geração do *output*, a devolver por cada interacção, recorria apenas a funções de escrita, compostas por *tags* HTML e variáveis pertencentes a elementos de domínio do motor de jogo. Devido à complexidade do código, decidi que deveria tentar numa primeira abordagem identificar a informação (proveniente do motor de jogo) mais utilizada na interface e criar um protótipo do novo motor de jogo que disponibilizaria essa informação. A partir desse protótipo, fui modificando a informação de cada vista dos elementos do motor de jogo a devolver para a interface.

6.1.1 Berserk vs JAX-WS

O primeiro protótipo do novo motor de jogo, referida anteriormente, foi construído com base numa *framework* de nome Berserk[1]. A adopção desta framework foi justificada pelo facto de já ter tido contacto com a mesma em projectos passados, dispensando assim o tempo necessário de ambientação.

Esta versão (DimensioneX com Berserk) continuava a possuir a mesma estrutura da figura 3.1, mas agora o *servlet* **multiplayer** passava a utilizar o invocador de serviços do *Berserk* nas suas interacções com o motor de jogo. Este por seu lado encontrava-se presente numa classe aparte, estática e acessível apenas através dos serviços criados.

Do ponto de vista conceptual, estas alterações permitiram facilitar a inclusão de novos modos de interacção com o motor de jogo. Seria agora possível, por exemplo, incluir novos *servlets* capazes de interagir com o mesmo motor de jogo. No entanto, este sistema implicaria a partilha do mesmo contexto pelas diferentes interfaces assim como deixaria o seu programador dependente da linguagem Java para o efeito. Assim, decidi realmente separar o DimensioneX em duas componentes distintas: um servlet Java que serviria de interface Web e uma aplicação Web, usando JAX-WS, que disponibilize os serviços oferecidos pelo motor de jogo, sendo esta a versão final da plataforma desenvolvida.

A implementação da versão final do motor de jogo teve como principais obstáculos a adaptação à nova tecnologia adoptada (JAX-WS) e a as alterações internas realizadas (descritas na secção 4.2.3).

[1] BusinEss logic Runtime & SEcurity Resources Kit, http://berserk.sourceforge.net .

As adaptações feitas ao primeiro protótipo, para que passasse a usar JAX-WS, exigiram que fossem revistas as variáveis de instância de cada uma das classes dos objectos usados na definição dos serviços: para que a sua descrição em XML fosse gerada automaticamente (recorrendo ao JAXB), foi necessário adaptar o uso de classes não suportadas (neste caso a classe Map) e resolver algumas relações cíclicas entre objectos.
De qualquer forma, a adopção desta tecnologia tornou o motor de jogo ainda mais portável já que a sua comunicação com qualquer interface (criada recorrendo a qualquer linguagem de programação) se encontra bem definida recorrendo a standards da indústria.

6.1.2 Alterações internas

Das alterações internas ao motor de jogo realizadas, são de salientar as incorrecções encontradas aquando da escrita/leitura de informação na base de dados e também o facto de todo este mecanismo ter sido desenvolvido apenas para funcionar para um jogo. Apesar de estas alterações não terem sido planeadas inicialmente, tornou-se necessário alterar todo o seu funcionamento com o risco da plataforma não funcionar caso tal não tivesse sido feito. Em termos de perfomance, a redução do numero de acessos à base de dados, a redução das secções críticas do motor de jogo e respectiva utilização de trincos de leitura/escrita, julgo terem permitido tornar toda a plataforma mais escalável.

6.2 Desenvolvimento da interface Web

A adopção da linguagem de marcação XHTML teve como principal objectivo garantir uma apresentação menos ambígua e que obrigasse a uma codificação mais rigorosa. Na verdade, o passo mais importante nesse sentido foi dado através de validações W3C sobre o documento gerado, coisa que, ao contar com o numero de erros encontrados, nunca teria sido feito na anterior interface. Desta forma, passa a haver a garantia que a apresentação do documento XHTML nos vários *browsers* dependerá apenas dos mesmos e não de algum erro de marcação durante a geração do documento.

6.2.1 Comparação com a interface antiga

Tal como referido na secção 5.1.2, foi feita uma reorganização da maneira como a informação é disponibilizada ao utilizador. Esta reorganização, aliada também á separação entre motor de jogo e interface, permitiu diminuir o numero de interacções médios realizados tanto ao nível do *servlet*, como ao motor de jogo.
Seguidamente será feita uma comparação entre a antiga interface do DimensioneX e a nova interface (desenvolvida durante esta dissertação). O factor de comparação usado foi o numero de pedidos feitos por um utilizador à plataforma durante a execução de uma acção. Foram escolhidas para o efeito as acções que se apresentam como as mais comuns: Iniciar um jogo, mover entre salas e a execução dos vários tipos de comandos.

Acção: Iniciar um jogo	
Antiga interface	**Nova interface**
Carregamento do URL com o identificador do jogo	Carregamento do URL com o identificador do jogo
Carregamento da frame scene	
Carregamento da frame ctrls	
Carregamento da frame msgs	
Actualização da frame ctrls (resultado da submissão da informação de login)	Actualização da cena (resultado da submissão da informação de login) para a cena do jogo
Actualização da frame scene	
Actualização da frame msgs	

Acção: Mover para uma sala	
Antiga Interface	**Nova Interface**
Actualização da frame ctrls (resultado da submissão do pedido de mover)	Actualização da cena (resultado da submissão do pedido de mover) com a nova sala
Actualização da frame scene	
Actualização da frame msgs	

Acção: Executar comando sem argumentos	
Antiga Interface	**Nova Interface**
Actualização da frame ctrls (resultado da submissão do comando)	Actualização da cena (resultado da submissão do comando) após execução do comando
Actualização da frame scene	
Actualização da frame msgs	

Acção: Executar comando de um argumento	
Antiga Interface	**Nova Interface**
Actualização da frame ctrls (resultado da submissão do comando e respectivo argumento)	Mudança de cena (resultado da selecção do argumento) com um objecto seleccionado
Actualização da frame scene	Mudança de cena (resultado da submissão do comando e respectivo argumento) após execução do comando
Actualização da frame msgs	

Acção: Executar um comando de dois argumentos	
Antiga Interface	**Nova Interface**
Actualização da frame ctrls (resultado da submissão do comando e respectivos argumentos)	Mudança de cena (resultado da selecção do argumento) com um objecto seleccionado
Actualização da frame scene	Mudança de cena (resultado da escolha de um comando de dois argumentos) para a escolha do segundo argumento

Tabela 5.1 : Comparativo de acessos às interfaces por um utilizador.

A cada campo da tabela corresponde uma interacção do utilizador com a respectiva interface. Também de notar é o facto de que em todas as acções, na nova interface, apenas é realizado um acesso ao motor de jogo propriamente dito. Tendo em conta que a nova interface do DimensioneX não necessita de qualquer mecanismo de controlo de acesso concorrente, isto significa que a probabilidade de existir um bloqueio é sempre a mesma (e sempre inferior à probabilidade tida na anterior interface).

Caso se queira comparar a performance das duas interfaces (na óptica do utilizador), penso que a nova interface Web estaria sempre em vantagem. Mesmo estimando que o conjuntos de interacções de cada uma das acções (na antiga e na nova interface) originasse a mesma quantidade de dados, estas continuariam a ser disponibilizadas mais rapidamente na nova interface Web graças ao uso do sistema de expansão/minimização da informação de cada tabela (os elementos “escondidos” apenas são carregados durante a expansão da tabela).

6.2.2 Extensibilidade da nova interface

Tal como aconteceu no desenvolvimento de praticamente todo este projecto, foi sempre dada importância à sua possibilidade de extensão a novas funcionalidades a todos os níveis. A interface Web não foi excepção: a utilização da framework open-source Freemarker é uma mais valia desta nova interface, tornando a estruturação da interface ainda mais flexível e versátil (para quem a configura). É certo que esta estruturação era também possivel anteriormente, no entanto obrigava ao seu autor o conhecimento da linguagem Java e de todo o funcionamento interno da interface.

Adicionalmente, a interface continua a suportar o sistema de codificação de estilos (CSS) que já se encontrava presente na plataforma. As primitivas usadas neste sistema foram estendidas para incluir os novos elementos usados (caixa de mensagens com scroll, botões de maximização / minimização e respectivas caixas de items / jogadores / estado / caminhos). Este sistema continua a ser configurável através das skins definidas no motor de jogo, isto porque deverá ser da responsabilidade do programador de um jogo a definição deste tipo de apresentação.

6.2.3 Abrangência a dispositivos de pequenas dimensões

Para testar o correcto funcionamento desta nova interface com dispositivos de pequenas dimensões, foi usado um simulador de PDA e um browser com a capacidade de simular a sua utilização num

dispositivo móvel. Infelizmente não é possível afirmar que qualquer dispositivo móvel será capaz de apresentar correctamente esta interface sem que para tal sejam feitos testes físicos a esse dispositivo, no entanto a nível teórico é possível garantir que qualquer dispositivo com um browser capaz de suportar *Javascript* e CSS nível 2 será capaz de interagir com o DimensioneX.

6.3 Interface SMS

Apesar de apenas simular a interacção SMS, esta interface permitiu demonstrar a capacidade de interacção multi-canal desta plataforma.

Devido ao grau de informação capaz de ser transmitido por este canal de comunicação, teve de ser feita uma selecção dos comandos da plataforma passíveis de serem executados por esta interface assim como resumida a informação a devolver. Tal como já referido, a utilização deste serviço exige à partida que o seu utilizador tenha conhecimento dos comandos (e seus resultados) de cada jogo disponível. Isto deve-se ao facto de cada jogo conter um numero dinâmico de comandos possíveis de serem executados a cada momento. De notar que o conhecimento dos comandos específicos da plataforma (login/logout e registo) não é obrigatório, já que para estes é possível incluir uma descrição dos mesmos na própria interface (já que estes são estáticos em cada jogo).

Capitulo 7

Trabalho Futuro

Seguidamente serão propostos novos projectos a realizar no DimensioneX que poderão vir a enriquecer o mesmo.

7.1 Novas Interfaces

Uma das propostas que surgiram durante o desenvolvimento desta tese, foi o da criação de uma interface flash mais apelativa. Apesar deste projecto não ter sido concretizado, continua a ser uma possibilidade de extensão nesta plataforma.

A ideia proposta passa pela utilização de um middleware que estabeleça a correspondência entre as interacções de uma interface criada em flash com o web service do motor de jogo. Para tal, este middleware trataria de pedidos GET/POST descritivos das interacções a realizar no motor de jogo, e devolveria uma representação em XML do conteúdo devolvido pelo mesmo. Este documento XML seria posteriormente carregado na interface flash que procederia á devida apresentação ao utilizador.

7.2 Aplicações de Suporte ao Desenvolvimento de Jogos

O maior problema presente na plataforma neste momento não se encontra directamente relacionada com o funcionamento da plataforma. Na verdade, o que parece estar a limitar a sua evolução e utilização é a falta de conteúdos (jogos) para a mesma.

Ao contrário do que acontece noutras plataformas para jogos[1], o DimensioneX não fornece nenhuma ferramenta de apoio ao desenvolvimento dos jogos. Apesar da existência de um manual bastante detalhado de auxilio ao programador, o seu trabalho exige por vezes um elevado nível de concentração durante a codificação da estrutura do jogo (salas, itens, personagens) já que todo o trabalho tem de ser realizado sem o auxilio a meios visuais[2].

O que proponho é a criação de um IDE de auxílio ao desenvolvimento de jogos no DimensioneX que possua, preferencialmente as seguintes características:

- Sistema visual de apoio á criação da estrutura de um jogo. A ideia será, por exemplo, explorar a utilização do “drag-and-drop” na definição das imagens de cada objecto, assim como na colocação de itens e personagens em cada sala e nas ligações entre salas.
- Anexação de eventos a objectos através de menus de contexto. A edição do código de cada evento seria posteriormente assistida com as funcionalidades básicas de edição de texto, como “syntax highlighting” e “autocompletion”.
- Sistema de execução e debug do jogo, através de integração automática com o web service do motor de jogo.

[1] Por exemplo as analisadas em [1] e [2].
[2] A visualização pode ser feita apenas através do carregamento do jogo, o que não permite uma abordagem de desenvolvimento bottom-up.

7.3 Criação de jogos baseados no funcionamento multi-canal

Tendo este projecto se focado mais ao nível conceptual da plataforma, seria interessante agora perceber até onde se poderia inovar na criação de novos jogos.

A possibilidade de interacção com diferentes canais de comunicação permite uma maior ubiquidade a um jogo, criando uma nova funcionalidade capaz de ser explorada no desenvolvimento de jogos.

O exemplo apontado em [3] apresenta-se como um possível uso para esta funcionalidade, no entanto, com o nível certo de criatividade, certamente que outros poderão vir a ser desenvolvidos.

Capitulo 8

Conclusões

Esta dissertação ajudou a compreender alguns dos compromissos por detrás de, tanto as plataformas para jogos, como a interacção multi-canal.

Hoje em dia, a interacção multi-canal apresenta-se como uma característica bastante relevante de uma qualquer plataforma de disponibilização de conteúdo. A, cada vez maior, heterogeneidade entre dispositivos capazes de interagir com a Internet obriga a que sejam repensadas as formas de apresentação de conteúdo e, principalmente, que seja feita uma clara distinção entre a forma como o conteúdo é acedido e a forma como o conteúdo é apresentado ao utilizador. Através deste trabalho, foi possível apresentar uma abordagem para permitir este tipo de interacção e também identificar os compromissos que tal mecanismo impõe.

O DimensioneX sofre uma evolução bastante positiva ao nível da transparência da sua execução assim como ao nível das suas capacidades. O facto de passar a suportar a inclusão de múltiplas interfaces foi sem duvida uma mais valia para este projecto, no entanto a sua re-estruturação interna e também a sua nova interface web é também um importante factor para a sua adopção a nível comercial.

A utilização apenas de ferramentas open-source e tecnologias compostas a partir de standards da industria serve também como um convite à extensão da plataforma no futuro. Tendo em conta que a própria plataforma é um projecto open-source, é também esperada que esta sua re-organização cause algum impacto ao nível da sua evolução: apesar desta dissertação assumir um caminho diferente do desenvolvimento actual da plataforma, pode sempre servir de exemplo ao trabalho que deve ser feito ao nível interno para facilitar a sua aprendizagem e a inclusão de novas de contribuições.

8.1.1 Exemplo do modelo Acção-Percepção: Arianne

Já depois da análise em [2], tomei conhecimento de uma plataforma de jogos que poderá servir para exemplificar o tipo de funcionamento adoptado para o motor de jogo do DimensioneX. O Arianne[1] já tem alguns anos de existência (fundado em 1999) e apresenta-se como uma plataforma robusta de criação de jogos em tempo real.

A plataforma é apresentada como seguindo uma filosofia de desenho Acção – Percepção. Uma percepção mais não é do que um conjunto de elementos que descrevem um instante do jogo a um utilizador. Sempre que um utilizador realiza uma acção sobre o jogo, este actualiza a sua percepção do jogo.

Para suportar este modelo, a plataforma dispõe de um servidor com o motor de jogo que comunica por UDP com os vários clientes presentes no jogo. Um jogo é criado no servidor através da definição do conjunto de regras que define a lógica do jogo. A interface do jogo é codificada em python

[1] Arianne – A multiplayer online rpg framework, http://arianne.sourceforge.net/ .

funcionando como uma applet num browser, o que permite uma interacção assíncrona entre motor de jogo e interface (mas forçando à presença da maquina virtual Java).
Apesar de não ter sido graças ao conhecimento desta plataforma que se procedeu à definição da adaptação multi-canal genérica descrita no estudo[2] (nem à adaptação ao DimensioneX descrita nesta dissertação), foi através da adopção de um modelo deste tipo que se chegou à solução proposta.

8.1.2 Objectivos alcançados

O planeamento inicialmente realizado sofreu algumas alterações ao longo do tempo. Para tal deveram-se maioritariamente dois factores: o primeiro foi uma má estimativa do trabalho requerido pelos outros compromissos académicos, o segundo foi a crescente diminuição do contacto com a empresa beActive a quem deveria ter dado apoio durante a implementação de uma interface flash. Como tal, o período de reestruturação da plataforma prolongou-se e a interface Web foi estendida de forma a suportar dispositivos de ecrã reduzido (maioritariamente PDAs).
Mesmo tendo em conta as alterações ao planeamento, e com mais ou menos sobressaltos, foi possível atingir todos os objectivos a que esta dissertação se propunha.

Referências Bibliográficas

[1] G. Carreiro e P. Marques: Trabalho Final de Curso - Jogo Distribuído e Multicanal de Gestão de Equipas de Futebol. Instituto Superior Técnico, Outubro 2006 .

[2] João Ribeiro: Introdução á Investigação - Plataforma Para Jogos Multi-Canal. Instituto Superior Técnico, Janeiro 2007.

[3] S. Boll, J. Krosche, C. Wegener: Paper Chase Revisited - a Real World Game Meets Hypermedia, Proc. Of the 14th ACM conference on Hypertext and hypermedia.

[4] M. Adorni, F. Arcelli, L. Baresi, C. Bantini, A. Bianchi, D. Bianchini, A. Caforio, C. Cappiello, V. De Antonellis, C. Franza, G. Giunta, A. Limonta, P. Losi, A. Maurino, M. Melideo, S. Modafferi, C. Pandolfo, B. Pernici, P. Plebani, C. Raibulet, F. Tisato, E. Valle, A. Zilli: MAIS Reference Model. MAIS Project Report 1.3.2, Novembro 2003.

[5] S. Ceri, F. Daniel, M. Matera: Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. IEEE International Conference on Web Services (2004).

[6] M. Brambilla, S. Ceri, M. Passamani, A. Riccion: Managing Asynchronous Web Services Interactions. IEEE International Conference on Web Services (2004)

[7] C. Leoni: DimensioneX version 6.3.3 Developer's Reference, Agosto 2007 http://www.dimensionex.net/en/docs/default.htm .

Anexos

Anexo A – Especificação da camada de serviços presente no motor de jogo

1.1 Definição dos serviços disponibilizados

1.1.1 Serviços genéricos

- **GetWorld**
  - **Entradas**: String gameSlot
  - **Saída**: WorldInterface
  - **Descrição**: Este serviço devolve a vista (WorldInterface) do jogo presente em gameSlot no motor de jogo.
  - **Pré-Condições**: -
  - **Pós-Condições**: -
- **RegisterUser**
  - **Entradas**: String username, String password, String realName, String mail, String phone, String secretQuestion, String secretAnswer, String gameSlot
  - **Saída**: -
  - **Descrição**: Este serviço efectua o registo de um novo jogador no jogo especificado por gameSlot.
  - **Pré-Condições**: -
  - **Pós-Condições**: -
- **ResetUserPassword**
  - **Entradas**: String username, String mail, String secretQuestion, String secretAnswer, String newPassword, WorldInterface world
  - **Saída**: -
  - **Descrição**: Este serviço altera a password do utilizador identificado por username para newPassword caso sejam validados os campos mail, secretQuestion e secretAnswer. O jogo associado a este utilizador é identificado por gameSlot.
  - **Pré-Condições**: -
  - **Pós-Condições**: -
- **ChangeProfile**
  - **Entradas**:String username, String pass, String realname, String mail, String phone, String secretQuestion, String secretAnswer, String gameSlot
  - **Saída**: -
  - **Descrição**: Este serviço altera os campos do perfil de um determinado utilizador.

- **Pré-Condições**: Invocador deverá ser um utilizador previamente validado.
- **Pós-Condições**: -

- **LoginUser**
  - **Entradas**: String username, String password, String gameslot, String remoteAddr, String skinId
  - **Saída**: PlayerInterface
  - **Descrição**: Este serviço efectua a entrada de um utilizador no jogo identificado por gameslot. Para tal deverá ser validado o seu username e respectiva password. O utilizador poderá optar por uma Skin identificada por skinId. Este serviço devolve a vista do utilizador (PlayerInterface) logo depois de entrar no jogo.
  - **Pré-Condições**: -
  - **Pós-Condições**: -

1.1.2 Serviços de um utilizador

- **LogoutUser**
  - **Entradas**: PlayerInterface utilizador
  - **Saída**: -
  - **Descrição**: Este serviço efectua a saída do utilizador do jogo associado ao mesmo. O utilizador (PlayerInterface) é removido do jogo e o seu estado guardado na base de dados.
  - **Pré-Condições**: O invocador deverá ser um utilizador previamente validado.
  - **Pós-Condições**: -

- **ExecGameAction**
  - **Entradas**: PlayerInterface jogador, String cmd, String arg1, String arg2, List<StringMap> input.
  - **Saída**: PlayerInterface
  - **Descrição**: Este serviço efectua a execução de um comando no motor de jogo. Excluindo o identificador do comando, poderão ainda existir até dois argumentos e uma lista de pares id-valor de outros valores recebidos.
  - **Pré-Condições**: O invocador deverá ser um utilizador previamente validado.
  - **Pós-Condições**: -

- **RefreshUserInfo**
  - **Entradas**: PlayerInterface jogador
  - **Saída**: PlayerInterface
  - **Descrição**: Este serviço é um caso especial da execução de um comando no motor de jogo. Em vez de qualquer acção ser realizada pelo jogador, é apenas devolvido o novo estado do jogador (caso tenha sido alterado por outra entidade que não o próprio jogador).

- **Pré-condições**: O invocador deverá ser um utilizador previamente validado.
- **Pós-Condições**: -

1.1.3 Serviços de um administrador

- **ExecAdminCmd**
  - **Entradas**: String adminPassword, String cmd, String arg, String gameslot
  - **Saída**: String
  - **Descrição**: Este serviço executa um comando de administrador no motor de jogo. O comando identificado poderá ter até um argumento, mais um identificador do jogo em questão.
  - **Pré-Condições**: -
  - **Pós-Condições**: -
- **GetWorldDump**
  - **Entradas**: String adminPassword, String gameSlot
  - **Saída**: WorldDump
  - **Descrição**: Este serviço devolve uma descrição detalhada de todas as características do jogo identificado por gameSlot.
  - **Pré-Condições**: -
  - **Pós-Condições**: -
- **GetWorldLog**
  - **Entradas**: String adminPassword, String gameSlot
  - **Saída**: WorldLog log
  - **Descrição**: Este serviço devolve o ficheiro de log associado ao jogo identificado por gameSlot.
  - **Pré-Condições**: -
  - **Pós-Condições**: -
- **GetWorldOptimizer**
  - **Entradas**: String adminPassword, String gameSlot
  - **Saída**: WorldOptimizer
  - **Descrição**: Este serviço devolve informações sobre a performance do jogo identificado por gameSlot.
  - **Pré-Condições**: -
  - **Pós-Condições**: -

1.2 Definição da informação a disponibilizar

De seguida, serão especificados os tipos dos objectos usados pelo serviços (tanto como parâmetros de entrada como de saída).

A análise da interface actual do DimensioneX permitiu dissecar toda informação proveniente do motor de jogo. Como tal, foram definidos um conjunto de vistas para os objectos do motor de jogo apenas com a informação necessária para a sua apresentação. Nalguns casos foram criadas novos objectos quando tal pareceu necessário.
O diagrama UML de cada uma das classes de objectos descritos de seguida poderá ser visualizado no Anexo B.

1.2.1 Adaptors

A conversão das vistas na sua representação XML foi feita através da tecnologia JAXB. Desta feita, foi necessário tomar algum cuidado com o tipo das variáveis de instância de cada vista, isto porque o seu mapeamento em XML pode não ser suportado automaticamente. Neste caso, o problema apenas surgiu com a utilização da interface *Map* de Java.
Para ultrapassar este problema, foram criadas classes de adaptação de cada tipo de *Map* de forma a poderem ser correctamente mapeados em XML.
As classes de adaptação criadas foram as seguintes:

- **StringMap** - Esta classe representa um elemento de um Map em que tanto a chave como o seu valor são representados através da classe String.
- **LinkInterfaceMap** - Esta classe representa um elemento de um Map em que a sua chave é um identificador e o seu valor um LinkInterface (definido mais á frente).

Depois de definidas estas classes, foi feita uma substituição de todos os Map usados em vistas por uma lista de uma das classes de adaptação (de acordo com o tipo de cada Map respectivamente).

1.2.2 Vistas de utilizador

De seguida são apresentadas as vistas dos objectos do motor de jogo criadas para a disponibilização da informação do estado de um jogo ao utilizador.

WorldInterface

Vista representativa do objecto World, a informação nela contida é utilizada para identificar o jogo, fornecer informação adicional sobre o jogo (autor, versão e logótipo) e principalmente para disponibilizar atributos necessários ao jogo (painéis de comandos, mapa e identificação da instância do jogo actual).

getAuthor()	Criador do jogo.
getAuthorEmail()	Mail do criador do jogo.
getBGColor()	Cor de fundo de um Room (ver Dev. Ref. Sec. 2.8).
getDefaultPanel()	Devolve o menu (PanelInterface) de jogo predefinido.
getDefaultClient()	Devolve o ClientInterface predefinido do jogo. Este apenas é utilizado antes de um jogador se identificar no jogo.
getFrontPageText()	Devolve a mensagem a mostrar ao utilizador no ecrã inicial (login).
getGameSlot()	Numero do slot ocupado pelo jogo

getHiscoreText()	Devolve a mensagem a ser devolvida ao jogador com a melhor pontuação no jogo.
getHiscore()	Devolve a melhor pontuação obtida no jogo.
getHiscorer()	Devolve o nome do jogador actualmente com a melhor pontuação no jogo.
getHiscoreDate()	Devolve a data de registo da melhor pontuação do jogo.
getHiscoreFile()	Devolve um URL para o ficheiro que contém as melhores pontuações do jogo.
getIconSize()	Devolve o tamanho definido para um ícone no jogo.
getId()	Identificador do jogo
getLogoURL()	URL do logótipo associado ao jogo.
getMap()	URL com o mapa associado ao jogo.
getMsgs()	Devolve a WorldMessagesInterface associada ao jogo.
getName()	Devolve o nome do jogo.
getPanels()	Devolve a lista de todos os menus disponíveis no jogo.
getPlayers()	Devolve uma lista com todos os jogadores actualmente no jogo.
getPlayersCounter()	Devolve o numero de jogadores registados no jogo.
getKilledCounter()	Devolve o numero de jogadores que se encontram mortos no jogo.
getSceneLookType()	Modo de visualização do jogo. 0 – First Person View 1 – Third Person View
getSite()	Devolve o URL do site associado ao jogo.
getSkins()	Devolve a lista de todas as skins (SkinInterface) disponíveis pelo jogo.
getStartDate()	Devolve a data de inicio de execução do jogo.
getVersion()	Versão do jogo.
getWinnersCount()	Devolve o numero de jogadores que já terminaram o jogo.
getWorldInstanceId()	Identifica uma instância de um jogo
isCompass()	Presença constante da bússola se Verdadeiro.
isSimpifyNavigation()	Informa se a navegação em cada sala deve incluir ícones para outras direcções da mesma (false) ou não (true).

Tabela A.1 : Informação disponibilizada pela vista WorldInterface.

AdvObjInterface

Vista do objecto AdvObject usado pelo motor de jogo. Esta vista é usualmente usada quando se pretende listar o conteúdo de um Room, não sendo necessária mais do que a informação inerente a cada objecto do motor de jogo (sendo ela o seu identificador e modo de apresentação). O tipo real deste objecto é a única informação não genérica contida nesta vista.

getDescription()	Descrição deste objecto
getFacing()	Devolve a actual orientação do objecto (N, S, E ou O)

getIcon()	URL com o ícone que representa este objecto
getId()	Identificador do objecto
getImage()	Imagem associada ao objecto tendo em conta a actual orientação num Room.
getMapX()	Coordenada X da posição do objecto no mapa do World
getMapY()	Coordenada Y da posição do objecto no mapa do World
getName()	Nome associado ao objecto
getProperties()	StringMap com as propriedades do objecto
getShowFor()	(Ver Dev. Ref. 2.9.3)
getShowMode()	(Ver Dev. Ref. 2.8.1)
getShowX()	(Ver Dev. Ref. 2.8.2)
getShowY()	(Ver Dev. Ref. 2.8.2)
getTypePrefix()	Identificador do tipo de objecto associado. (Ex: c para Character, i para Item, l para Link)
isHasSeveralFaces()	Define se o objecto tem várias imagens que o representam (conforme a sua orientação) ou não
isRobot()	Define se o objecto é controlado por um humano

Tabela A.2 : Informação disponibilizada pela vista AdvObjInterface.

PeopleContainerInterface

Este objecto representa a vista do objecto PeopleContainer do motor de jogo. Esta vista herda a informação de AdvObjInterface e acrescenta duas listas de objectos (do tipo AdvObjInterface): uma lista de personagens e uma lista de items. Este objecto serve apenas como uma vista intermédia, que representa um objecto capaz de incluir no seu interior itens e personagens.

...	...
getItemList()	Devolve a lista de itens presentes no objecto.
getPeopleList()	Devolve a lista de personagens presentes no objecto

Tabela A.3 : Informação disponibilizada pela vista PeopleContainerInterface.

RoomInterface

Este objecto representa a vista do objecto Room do motor de jogo. Esta vista extende a vista PeopleContainerInterface de forma a representar uma sala no jogo. Para tal, é acrescentada informação sobre os caminhos disponíveis a partir da sala e, opcionalmente, uma imagem com um efeito para a sala (por exemplo para simular chuva ou nevoeiro).

...	...
getEffect()	Devolve uma ImageRef com o efeito a incluir na sala.

getWays()	Devolve uma lista de LinkInterfaceMap com os caminhos (LinkInterface) possíveis de ser seguidos a partir da sala e a sua respectiva direcção (N, S, E, W, U, D).
getWaysWithImages()	Devolve uma lista com as direcções que possuem uma imagem no seu caminho.

Tabela A.4 : Informação disponibilizada pela vista RoomInterface.

VehicleInterface

Este objecto representa a vista do objecto Vehicle do motor de jogo. Ao contrario do que acontece com o objecto Vehicle, esta vista estende a vista PeopleContainerInterface. A razão para esta escolha teve a ver com o facto de se tornar necessário (externamente) separar um normal Item de um PeopleContainer. Assim, a informação adicional disponibilizada por esta vista tem a ver com o modo de visualização do conteúdo de um veiculo e a imagem interna do mesmo.

...	...
getInnerImage()	Devolve a imagem que representa o interior deste veiculo.
getShowPolicyPeople()	Devolve o modo como as personagens deve ser apresentadas dentro do veiculo. (ver Dev. Ref. Sec. 3.9.7.1).
getWaysWithImages()	Devolve uma lista com as direcções que possuem uma imagem no seu caminho.

Tabela A.5 : Informação disponibilizada pela vista VehicleInterface.

LinkInterface

Vista do objecto Link usado pelo motor de jogo. Esta vista representa um caminho entre duas salas (Rooms) no motor de jogo.

...	...
getFromIds()	Devolve os identificadores das duas salas (Rooms) que o caminho liga.
getDirection()	Devolve a direcção do caminho. No caso de o caminho ser bidireccional, este valor mostra a direcção do caminho a partir do identificador da primeira sala.

Tabela A.6 : Informação disponibilizada pela vista LinkInterface.

PlayerInterface

Vista do objecto Player usado pelo motor de jogo. Tal como o objecto Player herda de AdvObject, PlayerInterface também herda de AdvObjInterface.

A informação adicional presente nesta vista identifica o jogador e respectivo jogo, sendo essa

informação: WorldInterface (o jogo) associado ao jogador, o RoomInterface em que se encontra e a sua respectiva orientação nele. Caso exista um objecto actualmente em foco pelo jogador, este é identificado. Também se encontram associados a cada jogador o identificador de um SkinInterface (de um dos disponíveis no jogo) e um ClientInterface descritos mais à frente.

…	…
getMsgBoard()	ClientInterface associado ao jogador.
getFacing()	Orientação actual do jogador.
getFocus()	Devolve o objecto actualmente em foco pelo jogador.
getPrevPanelIds()	Referencia aos painéis do jogo actualmente disponíveis ao jogador. Tipicamente é composto por o identificador de um painel especifico da sala mais o identificador de um painel especifico do jogador.
getProfile()	Map com informação sobre o utilizador. Algumas chaves: “name”, “skin”, “audio”, “screenwidth”, “screenheight”
getRoom()	RoomInterface actualmente associado ao jogador
getSkin()	SkinInterface associada ao jogador.
getContainer()	VehicleInterface em que o jogador se encontra inserido (este poderá ser nulo).
getWorld()	WorldInterface associado a este jogador.

Tabela A.7 : Informação disponibilizada pela vista PlayerInterface.

ImageRef

Vista do objecto Image usado pelo motor de jogo. A informação contida nesta vista é utilizada na apresentação da imagem, de acordo com a lógica a ela associada pelo motor de jogo (atributos como o tamanho e posição da imagem).

Este objecto poderá representar qualquer elemento multimédia que utilize a tag <IMG> de HTML.

getBaseline()	(ver Dev. Ref. Sec. 2.10)
getHeight()	Largura do objecto.
getMobieSrc()	URL do elemento multimedia para disponibilização em dispositivos com ecrã reduzido.
getShowX()	(ver Dev. Ref. Sec. 2.10)
getShowY()	(ver Dev. Ref. Sec. 2.10)
getShowAreaX1()	(ver Dev. Ref. Sec. 2.10)
getShowAreaX2()	(ver Dev. Ref. Sec. 2.10)
getSrc()	URL do elemento multimédia.
getWidth()	Comprimento do objecto.

Tabela A.8 : Informação disponibilizada pela vista ImageRef.

ClientInterface

Este objecto representa a vista do objecto Client usada pelo motor de jogo. Este objecto tem como principal função auxiliar a adaptação do jogo à sua apresentação no cliente alvo e ao mesmo tempo transmitir a informação resultante da interacção de um jogador com o jogo.
Esta vista possui: as mensagens de chat de um determinado jogador; informação sobre a última interacção do jogador com o motor de jogo; texto associado à ultima interacção com o motor de jogo (console); e o tamanho do ecrã pensado aquando da criação do jogo actual.

getBackgroundSound()	URL de um clip de som de fundo.
getConsole()	Devolve texto presente na console
getLastContact()	Devolve a data do último pedido efectuado
getMessages()	Devolve as lista de mensagens (do tipo Message) do jogador
getSound()	URL de um clip de áudio para ser executado.
getWorldScreenWidth()	Devolve a largura do ecrã do jogo
getWorldScreenHeight()	Devolve o comprimento do ecrã do jogo
isBackgroundSoundLoop()	Define se deve haver repetição ou não do clip de som de fundo.
isHasNewMessages()	Devolve a existência ou não de novas mensagens recebidas pela ultima interacção.

Tabela A.9 : Informação disponibilizada pela vista ClientInterface.

Message

Este objecto representa a vista do objecto Message usada pelo motor de jogo no sistema de chat. Cada mensagem contém um identificador, a data da sua criação, o seu emissor (identificador e nome) e o seu conteúdo propriamente dito.

getDate()	Data de criação da mensagem
getFrom()	Identificador do emissor da mensagem
getId()	Identificador da mensagem
getSenderName()	Nome do emissor
getText()	Conteúdo da mensagem

Tabela A.10 : Informação disponibilizada pela vista Message.

WorldMessagesInterface

Este objecto representa a vista do objecto Messages usada pelo motor de jogo. É nesta vista que estão contidas as mensagens locais usadas pelo jogo, a lista de comandos disponíveis e a respectiv codificação usada. O processo de adaptação destas mensagens a cada língua encontra-se descrito em Dev. Ref. Sec. 1.3.2.

getCharset()	Codificação usada no texto.
getCmds()	Devolve lista com os nomes dos comandos disponíveis no jogo.
getMsgs()	Devolve lista com as mensagens disponíveis

	no jogo.

Tabela A.11 : Informação disponibilizada pela vista WorldMessagesInterface.

PanelInterface

Vista do objecto Panel usado pelo motor de jogo. Esta vista é utilizada para representar um painel de comandos no jogo, contendo para o efeito a lista dos elementos (CtrlInterface) incluídos no painel e o seu respectivo identificador no jogo (World).

getControls()	Lista de comandos associados a este painel
getId()	Identificador do painel

Tabela A.12 : Informção a disponibilizada pela vista PanelInterface.

CtrlInterface

Este objecto representa a vista do objecto Ctrl usado pelo motor de jogo. Esta vista é utilizada para apresentar um elemento de um painel do jogo[1], sendo composta pela identificação do elemento e respectiva descrição, identificador do evento associado ao comando e respectivo modelo (O, OO, TO, TT), imagem do elemento e o seu tipo. Alguns dos campos desta vista só se encontram preenchidos em determinados tipos de elementos.

getDescription()	(ver Dev. Ref. Sec. 2.5).
getDropDownList()	Lista de valores do botão (apenas usado quando o Ctrl é uma *DropDownList*).
getEventId()	Identificador do evento a accionar pelo elemento (quando se trata de um botão).
getEventModel()	Modelo do evento (O, OO, TO, TT).
getIcon()	URL do ícone associado ao comando.
getId()	Identificador do elemento.
getImage()	ImageRef da imagem associada ao elemento.
getName()	(ver Dev. Ref. Sec. 2.5).
GetType()	Tipo de comando. 0 – Ghost 1 - Button 2 - Label 3 - Carriage Return 4 - TextBox 5 - ImageButton 6 - Submit 7 - Dropdown

[1] Todos os elementos possiveis encontram-se descritos na Secção 2.5 de [7]

	8 - Checkbox 9 - Password 10 - Hidden 11 - Map

Tabela A.13 : Informação disponibilizada pela vista CtrlInterface.

SkinInterface

Este objecto representa a vista do objecto Skin usado pelo motor de jogo. O seu conteúdo é praticamente igual ao objecto original: toda a informação sobre o estilo a adoptar na apresentação dos vários elementos do jogo. Grande parte da informação aqui presente serve para personalizar a interface Web, no entanto é também aqui que são guardados os valores de alguns elementos multimédia usados pelo motor de jogo. Um destes casos são os ícones de navegação, o ícone predefinido para cada tipo de objecto (Item, Character, Player...).

getAlinkStyle()	(ver Dev. Ref. Sec. 4.2)
getBodyBackground()	(ver Dev. Ref. Sec. 4.2)
getBodyBgColor()	(ver Dev. Ref. Sec. 4.2)
getBodyStyle()	(ver Dev. Ref. Sec. 4.2)
getButtonStyle()	(ver Dev. Ref. Sec. 4.2)
getChatEraseBtnStyle()	Estilo usado no botão de apagar uma mensagem
getChatExpandBtnStyle()	Estilo usado no botão de expandir o conteúdo de uma mensagem
getChatExpTextStyle()	Estilo usado no texto de uma mensagem quando expandida.
getChatMsgStyle()	Estilo usado em cada linha de uma tabela de mensagens
getChatTableStyle()	Estilo usado para a tabela de mensagens
getChatTextStyle()	Estilo usado no texto de uma mensagem (quando não expandida)
getCtrlBanner()	(ver Dev. Ref. Sec. 4.2)
getIcoCharacter()	(ver Dev. Ref. Sec. 4.2)
getIcoItem()	(ver Dev. Ref. Sec. 4.2)
getIcoPlayer()	(ver Dev. Ref. Sec. 4.2)
getIcoRev()	(ver Dev. Ref. Sec. 4.2)
getIcoRotL()	(ver Dev. Ref. Sec. 4.2)
getIcoRotR()	(ver Dev. Ref. Sec. 4.2)
getIcoWayN()	(ver Dev. Ref. Sec. 4.2)
getIcoWayS()	(ver Dev. Ref. Sec. 4.2)
getIcoWayE()	(ver Dev. Ref. Sec. 4.2)
getIcoWayW()	(ver Dev. Ref. Sec. 4.2)
getIcoWayU()	(ver Dev. Ref. Sec. 4.2)
getIcoWayD()	(ver Dev. Ref. Sec. 4.2)

getIconLabelStyle()	(ver Dev. Ref. Sec. 4.2)
getId()	(ver Dev. Ref. Sec. 4.2)
getLinkStyle()	(ver Dev. Ref. Sec. 4.2)
getList1BgColor()	(ver Dev. Ref. Sec. 4.2)
getList2BgColor()	(ver Dev. Ref. Sec. 4.2)
getListStyle()	(ver Dev. Ref. Sec. 4.2)
getName()	(ver Dev. Ref. Sec. 4.2)
getPanelBackground()	(ver Dev. Ref. Sec. 4.2)
getPanelBgColor()	(ver Dev. Ref. Sec. 4.2)
getPanelStyle()	(ver Dev. Ref. Sec. 4.2)
getPicFader()	(ver Dev. Ref. Sec. 4.2)
getPicPlayer()	(ver Dev. Ref. Sec. 4.2)
getPicSpacer()	(ver Dev. Ref. Sec. 4.2)
getResizeBtnStyle()	Estilo do botão de expansão/minimização de uma tabela (de itens, pessoas ou estado).
getSndNewmsg()	(ver Dev. Ref. Sec. 4.2)
getStylesheet()	(ver Dev. Ref. Sec. 4.2)
getSysMsgStyle()	(ver Dev. Ref. Sec. 4.2)
getTitleStyle()	(ver Dev. Ref. Sec. 4.2)

Tabela A.14 : Informação disponibilizada pela vista SkinInterface.

1.2.3 Vistas de Administrador

Para apresentar a informação devolvida a um administrador após a invocação de um serviço no motor de jogo, foram criadas 5 objectos. Todas os objectos criados tiveram como base o objecto World, no entanto não existe uma correspondência directa com qualquer objecto presente no motor de jogo.

WorldDump

Este objecto disponibiliza informação sobre o estado actual de um jogo: todas as suas propriedades e elementos.

getName()	Devolve o nome do jogo actual
getMsgCharset()	Devolve a codificação usada no texto do jogo.
getProperties()	Devolve uma lista de todas as variáveis presentes no jogo e os seus respectivos valores.
getElements()	Devolve todos os elementos (itens, personagens, salas...) presentes no jogo e os seus respectivos valores.

Tabela A.15 : Informação disponibilizada por WorldDump.

WorldLog

Este objecto disponibiliza o ficheiro de log de um jogo.

getName()	Devolve o nome do jogo.
getVersion()	Devolve a versão do jogo.
getMsgCharset()	Devolve a codificação usada no texto do jogo.
getDebugMode()	(Ver Dev. Ref. 1.2.2)
getLogDump()	String com todo o conteúdo do log do jogo.

Tabela A.16 : Informação disponibilizada por WorldLog.

WorldOptimizer

Este objecto disponibiliza informação sobre a performance de um determinado jogo.

getName()	Devolve o nome do jogo actual
getMsgCharset()	Devolve a codificação usada no texto do jogo.
getVersion()	Devolve a versão do jogo.
getStats()	Devolve a lista de estatísticas de execução (OptimizerStat) do jogo.
getTotalCons()	Devolve o valor total do consumo de tempo do jogo.
getTotalFreq()	Devolve o numero total de chamadas a Eventos/Funções/Subrotinas.
getRuntimeErrors()	Devolve o numero de erros detectados durante a execução do jogo.

Tabela A.17 : Informação disponibilizada por WorldOptimizer.

OptimizerStat

Este objecto disponibiliza algumas estatísticas da execução de um determinado evento, função ou sub-rotina.

getName()	Devolve o nome do elemento ao qual as estatísticas se referem
getCpuTime()	Devolve o tempo usado do processador por esse elemento (em milissegundos).
getCpuPercent()	Devolve a percentagem de tempo do processador que o elemento usou em relação ao tempo total gasto pelo jogo.
getNoCalls()	Devolve o numero de chamadas a este elemento
getAvg()	Devolve o tempo médio que cada chamada ocupou de processador (em milissegundos).

Tabela A.18 : Informação disponibilizada por OptimizerStat.

1.3 Dados adicionais

Para terminar a especificação da camada de interacção com o motor de jogo, resta apenas definir o conjunto de constantes usadas no mesmo.

1.3.1 Tipos de AdvObjInterface

- Item - “i!”
- Room - “r!”
- Character - “c!”
- Link - “w!”

1.3.2 Tipos de CtrlInterface

- Ghost - 0
- Button - 1
- Label - 2
- CR - 3
- Textbox - 4
- ImageButton - 5
- Submit - 6
- DropDown - 7
- CheckBox - 8
- Password - 9
- Hidden - 10
- Map - 11

1.3.3 Tipos de modos de visualização (Show mode)

- Offscreen - 0
- OnScreen_Image - 1
- OnScreen_Icon - 2

1.3.4 Tipos de vistas de cena

- Look_1stPerson - 0
- Look_3rdPerson - 1

Anexo B - Diagrama UML das classes usadas na camada de serviços.

1.1 Adapters

Figura B.1 : Diagrama UML das classes StringMap e LinkInterfaceMap.

1.2 Vistas de utilizador

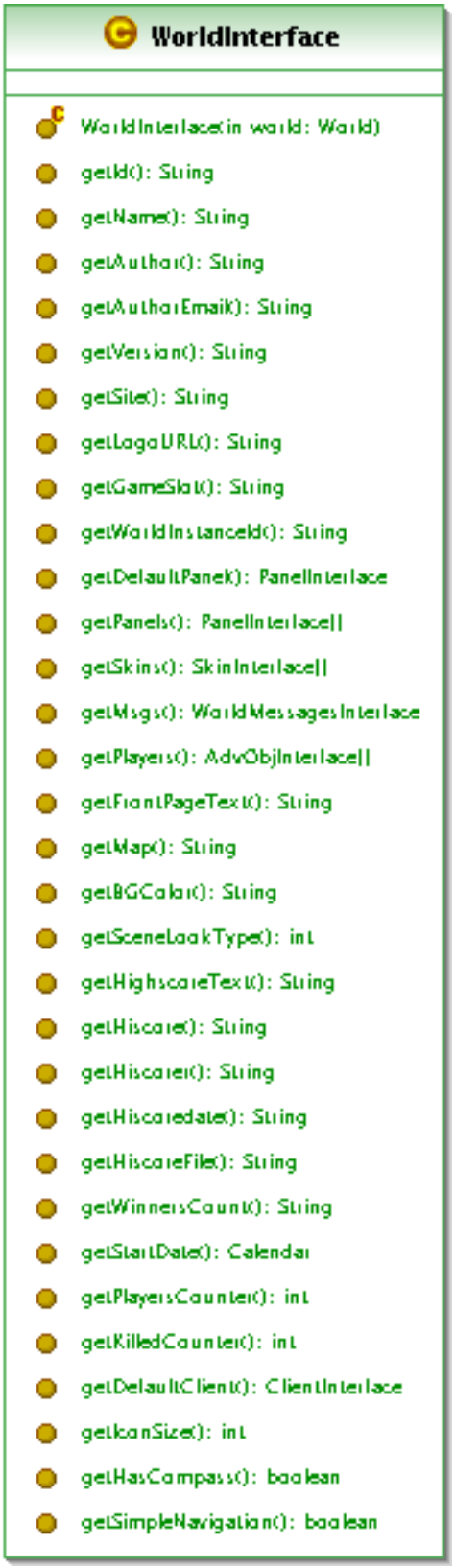

Figura B.2 : Diagrama UML da classe WordInterface.

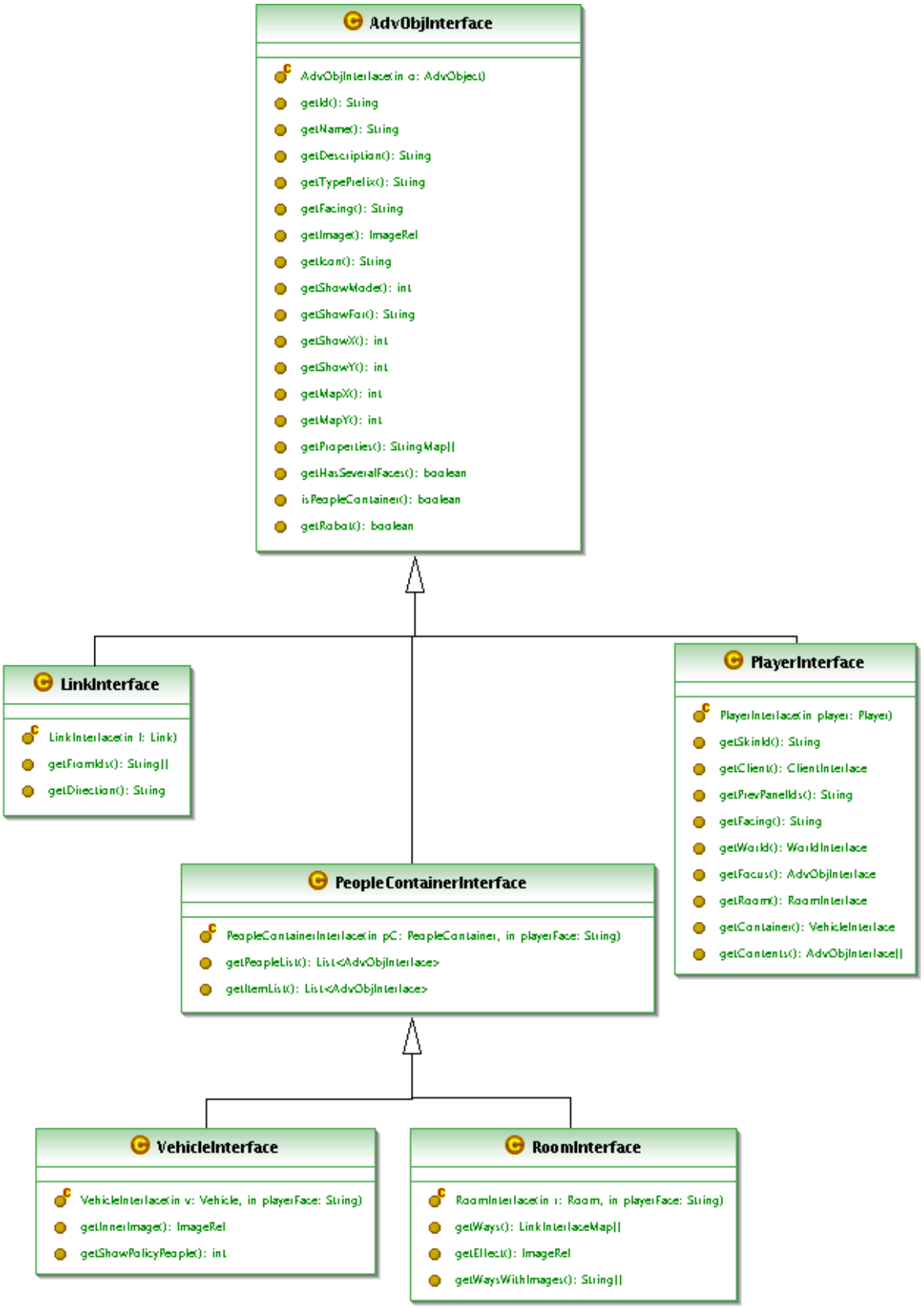

Figura B.3 : Diagrama UML da classe AdvObjInterface e das respectivas classes que a estendem.

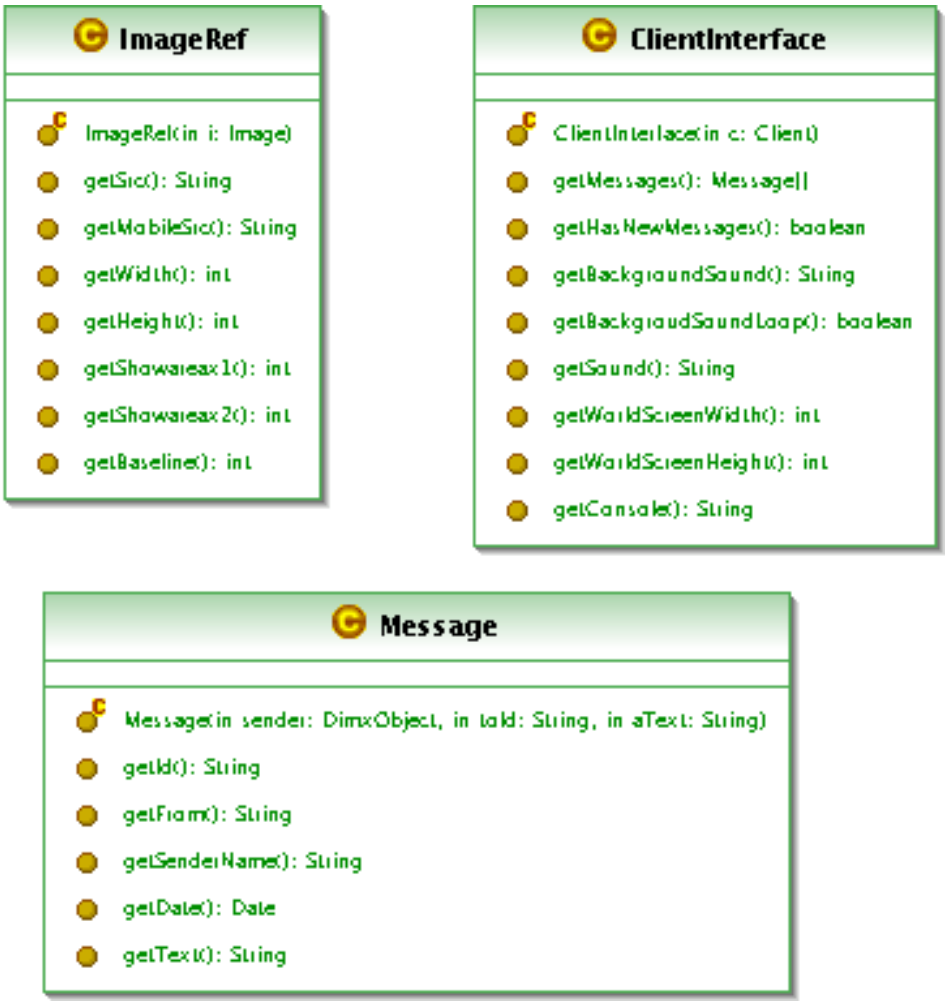

Figura B.4 : Diagrama UML das classes ImageRef, ClientInterface e Message.

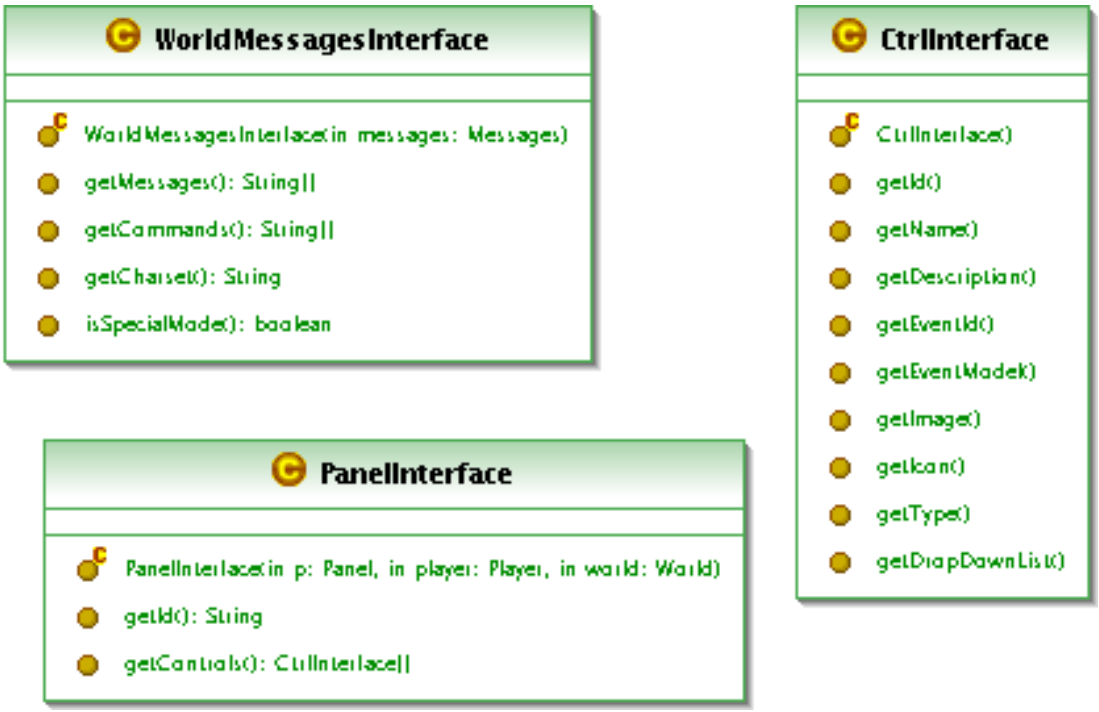

Figura B.5 : Diagrama UML das classes WorldMessagesInterface, CtrlInterface e PanelInterface.

Figura B.6 : Diagram UML da classe SkinInterface.

1.3 Vistas de administrador

Figura B.7 : Diagrama UML das classes usadas nos serviços de administrador.