4
Scenario: Sistema de visualização de panoramas com objetos ativos em plataformas celulares.

Essa dissertação tem o objetivo de apresentar um protótipo que possibilite a navegação em ambientes virtuais e que seja executado em celulares que não possuam grande capacidade de processamento. Outro objetivo é que a aplicação atinja um grande número de celulares comercializados atualmente.

O protótipo deve ser capaz de apresentar os ambientes com grande qualidade visual, de preferência com imagens foto-realistas e com a presença de objetos ativos em cena.

Tais objetos devem permitir que o usuário tenha algum tipo de interação com os mesmos, podendo ser para própria navegação dentro do ambiente, quanto para a execução de outra ação.

Esta aplicação, além das características apresentadas acima, conta ainda com a ubiqüidade inerente às plataformas celulares.

A partir de um sistema com essas características pode-se criar uma aplicação de um passeio virtual por um museu, que poderia ser acessada no próprio estabelecimento, facilitando a localização do usuário.

Um segundo exemplo é a visualização do ambiente em que o usuário se encontra em uma outra época, seja, por exemplo, a fachada externa ou a decoração interna de um imóvel nos anos 50. Outro exemplo seria a visualização de projeto arquitetônico ou obra de paisagismo, no exato local onde ela será executada.

O protótipo também poderia reproduzir uma loja virtual e utilizando as interações com os objetos presentes na cena como mecanismo para se obter maiores informações sobre o produto e até mesmo efetuar uma compra.

O sistema foi modelado sabendo que a maioria dos celulares não possuem uma capacidade de processamento elevada, e com a idéia de manter o protótipo simples o bastante para que a velocidade de navegação chegue bem próxima a de reprodução de um filme, além de evitar o tráfego contínuo de dados, ajudando a torná-lo atrativo para usuários finais.

Para as implementações do sistema foi utilizada a plataforma *Java 2nd Edition Micro Edition* (J2ME) utilizando o *Mobile Information Device Profile 2.0*

PUC-Rio - Certificação Digital Nº 0321113/CC

(MIDP 2.0) com suporte a gráficos 3D (spec. M3G JSR-184). A escolha foi feita por se tratar de uma plataforma aberta e com bastantes funções úteis, tanto de manipulação de gráficos 3D, quanto de bitmaps, além de já estar presente no mercado em grande escala, apesar de nem todas as implementações necessitarem de renderização de gráficos 3D.

O código fonte, os binários e as instruções, para todas as versões do protótipo, podem ser encontradas no sitio http://www.icad.puc-rio.br/scenario.

A seguir é apresentada a arquitetura e as implementações do sistema proposto.

4.1. Arquitetura do sistema

O sistema proposto tem como principal funcionalidade a navegação em um ambiente virtual.

Na modelagem atual um ambiente é representado por diversas imagens panorâmicas obtidas a partir de pontos distintos, que correspondem às localizações em que o usuário pode estar a cada momento. Tais imagens podem ser geradas a partir da modelagem de ambientes virtuais, ou a partir de fotografias de ambientes reais. Nesta dissertação cada um dessas visualizações do ambiente, a partir desses pontos pré-fixados, são denominadas *cenas.*

A navegação então é feita de forma discreta, através de cortes secos, onde o usuário só pode estar nos pontos correspondentes ao centro de cada uma das cenas.

Além de uma imagem do panorama, cada cena também é composta de objetos ativos, permitindo ao usuário interagir com o ambiente virtual.

4.1.1. Visualização e Navegação entre cenas

Com o objetivo de manter o protótipo simples e tendo como alvo os dispositivos com menor capacidade de processamento do que os chamados Smartphones, optou-se pela visualização do ambiente utilizando a modelagem de panoramas (Chen,1995).

Levando-se em consideração a baixa capacidade de armazenamento dos dispositivos presentes no mercado, para criar-se uma navegação completa em

PUC-Rio - Certificação Digital Nº 0321113/CC

um ambiente virtual, optou-se por uma fazê-la de forma discreta entre pontos fixos do ambiente.

Para a implementação de uma navegação de forma contínua, seria necessária uma abordagem como a apresentada por Aliaga(2001), chamada de *Plenoptic Stitching*.

Essa modelagem, além de necessitar de um sistema de autoria complexo, também necessita de um grande poder computacional e uma grande quantidade de dados sobre o ambiente. Como já exposto anteriormente, o protótipo proposto é direcionado para plataformas celulares, que possuem baixa capacidade de processamento e de armazenamento de dados quando comparada aos computadores pessoais.

Em uma modelagem de panorama com suporte geométrico (Chen,1995), a imagem que representa o fundo do ambiente é aplicada à face interna do sólido, e a câmera centrada no interior deste podendo variar os sua direção, descrita na equação plenóptica (Adelson,1991) pelos ângulos de azimute e elevação.

Deste modo, o ambiente pode ser decomposto em cenas, onde cada uma é representada por uma imagem panorâmica que corresponde aos valores da equação plenóptica no ambiente para um ponto (Vx,Vy,Vz) pré-definido.

Assim, o usuário do sistema pode executar uma navegação discreta entre esses pontos pré-definidos, visualizando uma cena de cada vez.

Além da implementação de modelagem de panorama com suporte geométrico como proposto por Chen(1995), outras duas implementações foram criadas, tentando explorar algumas das possibilidades para a visualização das cenas.

PUC-Rio - Certificação Digital Nº 0321113/CC

4.1.2. Repositório remoto de Cenas

Na modelagem do sistema proposto o repositório das cenas tornou-se necessário por diversos fatores, sendo o principal permitir que ambientes complexos pudessem ser representados por um grande número de cenas. Como os telefones celulares normalmente não contam com grande capacidade de armazenamento, a representação de um ambiente composto, se todas as cenas tivessem que estar presentes no dispositivo, seria inviável.

Primeiro, o repositório torna-se interessante, uma vez que permite a criação de servidor que possua alguma inteligência ao controlar o acesso as cenas que representem diversos ambientes complexos. Dessa forma, uma única

versão do cliente permitiria que usuário escolhesse entre vários ambientes diferentes. Controles poderiam ser implementados no servidor, e assim oferecer apenas alguns ambientes, com base em diversos tipos de informações tais como critérios de segurança, uma busca feita pelo usuário ou até sua localização física.

Um exemplo seria um ambiente que representa um shopping center, disponível para os clientes que estivessem transitando dentro dele. Ao solicitar a primeira cena para iniciar a navegação, o usuário poderia receber aquela que representa uma visualização deste shopping no ponto mais próximo de sua localização real . Tal aplicação poderia ser usada tanto para ajudar o usuário a se localizar quanto para fazer propaganda de produtos e promoções.

Outro fator interessante é a possibilidade de geração de cenas dinamicamente, permitindo a alteração dos objetos presentes na cena, além da própria imagem do panorama. Assim objetos poderiam ter sua localização alterada e condições climáticas ou correspondência com o horário do acesso poderiam ser reproduzidas.

É claro que há muitas possibilidades de expansão, porém, tais funcionalidades acarretam uma maior complexidade do sistema e não serão aprofundadas neste trabalho.

4.1.3. Interação com objetos presentes na cena

Uma aplicação voltada apenas para a navegação em um ambiente virtual, tem seus atrativos. Poderia utilizar fotos de boa qualidade, mesmo no pequeno display dos telefones celulares, simulando visitas a pontos turísticos apresentando informações históricas ou curiosidades sobre o local.

Porém é interessante que outras funcionalidades sejam adicionadas ao sistema. Isto é feito através da interação do usuário com os objetos presentes na cena. Várias formas de disparar os eventos associados a cada objeto podem ser apresentadas como opções em um menu, um clique sobre o objeto, ou apenas a proximidade com o ângulo de visão da câmera.

Seguindo a implementação desta funcionalidade no *QuickTime VR*, nesta aplicação os eventos são disparados a partir do clique em cima dos objetos, porém utilizando-se um cursor centrado na tela ao invés do ponteiro do mouse.

PUC-Rio - Certificação Digital Nº 0321113/CC

Como a aplicação foi estruturada em módulos através do *design pattern* de estados (Gamma et al.,1995), que será apresentado adiante, módulos de interação podem ser adicionadas facilmente.

4.2. Implementação

Como citado anteriormente foi utilizada para a implementação do sistema a plataforma *Java 2nd Edition Micro Edition* (J2ME) utilizando o *Mobile Information Device Profile 2.0* (MIDP 2.0) . A escolha foi feita por se tratar de uma plataforma aberta e com bastantes funções úteis, tanto de manipulação de gráficos 3D, quanto de bitmaps, além de já estar presente no mercado em grande escala.

A aplicação proposta foi implementada utilizando o Design Pattern de Estados(Gamma et al.,1995).

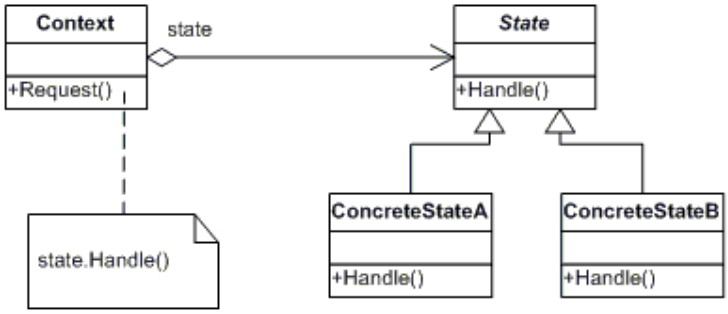

Figura 5 - Diagrama UML que representa o design pattern de estados

O *pattern* de estados é um padrão de modelagem comportamental. Ele permite que um objeto mude seu comportamento de acordo com o seu estado interno, aparentando mudar de classe.

O Contexto define a interface com os usuários do sistema, além de manter uma referência para o estado corrente.

O Estado define a interface para encapsular o comportamento associado com cada um dos *estados concretos* do contexto.

Por fim os *estados concretos* são os que implementam o comportamento do contexto associado a cada um.

Desta maneira os eventos gerados pelo sistema são tratados pelo estado corrente, e somente ele. Além disso, é possível controlar os recursos utilizados por cada estado, liberando-os durante a transição entre estados diferentes.

PUC-Rio - Certificação Digital Nº 0321113/CC

Na plataforma J2ME todas as rotinas de desenho no *display* do celular e captura dos eventos de tecla são tratados pelas classes derivadas da classe *Displayble*.

Como em um *MIDlet* existe apenas um objeto *Displayable* corrente, a própria plataforma permite a separação da aplicação em estados. Porém, alguns pontos de entrada de eventos externos são definidos no *MIDlet*, como é o caso do evento de pausa, invocado quando a aplicação precisa ter sua execução suspensa, como quando o celular recebe uma chamada de voz por exemplo.

Já o evento de inicio da execução é invocado tanto quando a aplicação é iniciada, quanto esta retorna de um estado suspenso anterior.

Com o padrão de Estados tanto os eventos de pausa e reinício da execução podem ser corretamente tratados de forma simples, quanto os recursos podem ser gerenciados, impedindo, por exemplo, que os recursos utilizados no módulo de visualização de cenas possam prejudicar o funcionamento do módulo de recuperação de cenas.

A figura 6 representa a arquitetura principal e a figura 7 mostra o diagrama de estados do sistema.

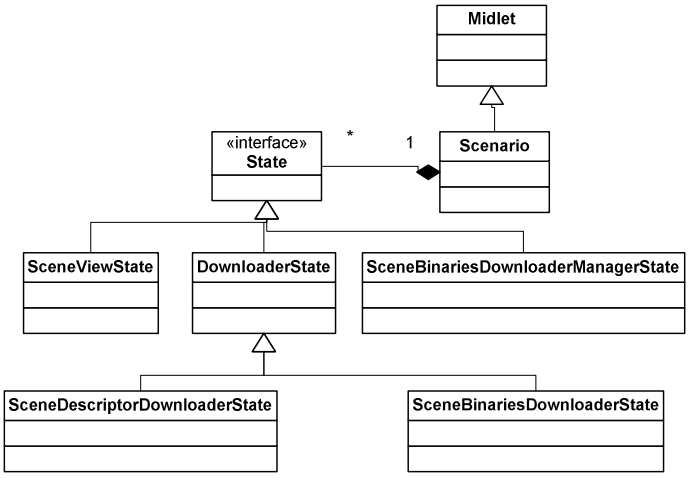

Figura 6 - Diagrama que representa a arquitetura principal do sistema proposto

PUC-Rio - Certificação Digital Nº 0321113/CC

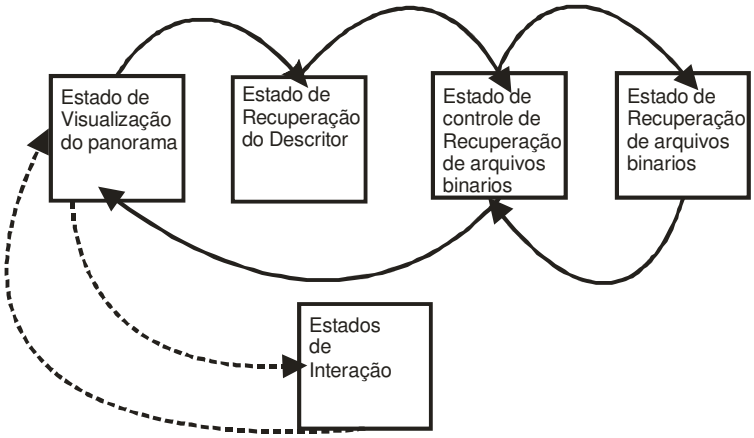

Figura 7 - O diagrama de estados do sistema proposto

A seguir são apresentados, mais detalhadamente, a arquitetura e o funcionamento dos módulos da aplicação.

4.2.1. Módulo de Visualização

O módulo de visualização é o responsável pelo controle interno da cena, controlando a câmera e o início da interação entre usuário e objetos, além de disparar a navegação para as cenas vizinhas.

Com o objetivo de verificar as possibilidades de implementação, três abordagens para o módulo de visualização das cenas foram feitas.

A primeira mais fiel às técnicas de IBR, manipulando os *pixels* da imagem do panorama utilizando técnicas de *warping*, seguindo a visualização feita com um suporte cilíndrico como em (Chen,1995).

Para a segunda foram utilizadas rotinas de renderização de gráficos 3D disponíveis para a plataforma J2ME, tendo o suporte cilíndrico representado por um polígono e com a imagem do panorama mapeada em sua face interna como textura.

Uma vez que esta API não esta amplamente disponível nos celulares presentes no mercado, a terceira implementação utiliza a imagem do panorama diretamente como fundo da cena.

Basicamente todas as três implementações apresentam a mesma arquitetura, onde o estado, *SceneViewState*, possui um *Canvas*, chamado *SceneViewCanvas* e um controlador chamado *CanvasController*.

PUC-Rio - Certificação Digital Nº 0321113/CC

A seguir são expostos os detalhes de implementação.

4.2.1.1. Visualização utilizando rotinas 3D

Como citado anteriormente nesta implementação o suporte cilíndrico foi feito através de um polígono, e a imagem do panorama é mapeada como textura interna.

Assim uma câmera virtual é colocada no centro do cilindro e uma luz ambiente é utilizada para iluminar o cilindro uniformemente.

Na atual implementação apenas o ângulo de azimute da câmera é variado, possibilitando que o usuário tenha uma visualização completa da cena em 360º, e um efeito de zoom é feito através do movimento da câmera em círculos centrados no eixo do cilindro.

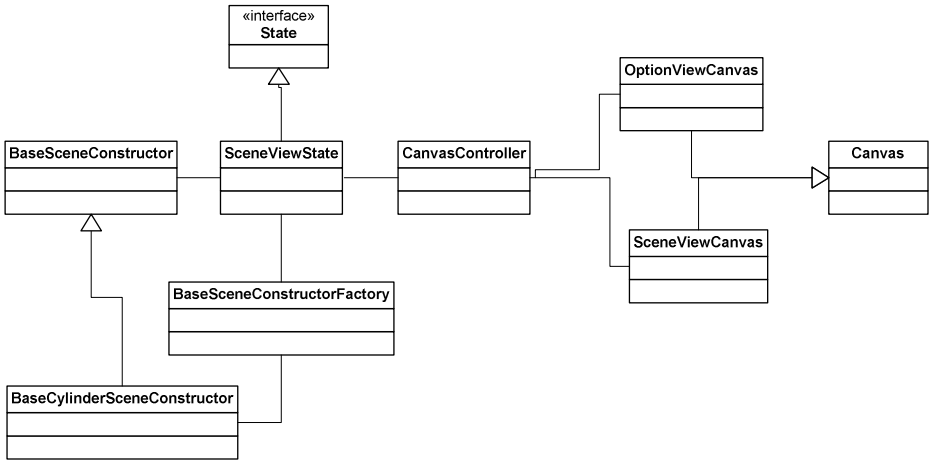

Figura 8 - Diagrama que representa a arquitetura do modulo de visualização da implementação utilizando rotinas 3D.

A *SceneViewState* é a base do módulo e é a responsável pela criação da cena, através da classe *BaseSceneConstructor*, com base na descrição e imagens recuperadas pelo modulo de navegação entre cenas.

Nela são instanciados os objetos que representam o cilindro, a câmera virtual, a iluminação, e os objetos. Com adição de objetos que representem esferas ou cubos, é possível a inclusão de cenas modeladas utilizando estas figuras como suporte geométrico.

Um único suporte geométrico foi utilizado para todas as instâncias de cenas criadas para este protótipo e ele é instanciado a partir da classe *BaseCylinderSceneConstructor*.

PUC-Rio - Certificação Digital Nº 0321113/CC

Este foi modelado por polígonos aproximando um cilindro por um *mesh* de dezoito lados, representado por cento e vinte e três triângulos e com suas normais invertidas, apontando para o centro, de forma que a imagem panorâmica seja projetada em sua face interna quando mapeada como textura.

Os valores utilizados na construção do cilindro foram capturados a partir de um primeiro modelo gerado com a ferramenta 3D Studio Max e exportado como um arquivo H3T, formato criado pela empresa Hi Corp desenvolvedora do Mascot Capsule. Este é um *engine* 3D para dispositivos moveis, e está presente em vários celulares como os fabricados pela Sony Ericcson. Esta empresa também desenvolveu ferramentas para a conversão de arquivos H3T para o formato M3G, que descreve os modelos na plataforma *Mobile Graphics 3D* do J2ME.

Todos esses elementos são organizados em um grafo de cena e expostos para a classe *SceneViewCanvas*, responsável apenas pela correta visualização do modelo completo.

A classe *CanvasController* processa os eventos gerados pelo usuário. Apesar do ponto de entrada destes eventos ser o *SceneViewCanvas*, por este ser um objeto do tipo *Displayable*, eles são encaminhados à classe *CanvasController*.

Com base nas entradas do usuário este controlador recalcula o posicionamento da câmera na cena, e verifica a interação do usuário com os objetos presentes na cena. Atualmente esta interação é feita quando o objeto se encontra de frente para a câmera e o usuário pressiona a tecla de seleção do telefone celular. Esta rotina foi implementada utilizando o método *pick* e a classe *RayIntersection*, presentes na API do *Mobile 3D Graphics*.

A combinação deste método e desta classe permite recuperar a instância de um objeto na cena a partir de sua interseção com um raio de origem em um ponto específico da *viewport* corrente e de orientação igual a da câmera, que também é utilizada como parâmetro do método citado.

Além disso, a plataforma ainda permite que informações de escopo e identificação sejam associadas a cada objeto, criando uma classificação para os mesmos. Nesta implementação esses mecanismos são utilizados para definir que tipos de interação cada tipo de objeto expõe.

Com base nesta classificação, o *CanvasController* expõe um menu permitindo que o usuário escolha entre os tipos de interação expostos por aquele objeto e dispara um pedido de transição de estado, encaminhando a execução para o modulo responsável pela ação correspondente.

PUC-Rio - Certificação Digital Nº 0321113/CC

Atualmente apenas o escopo de navegação entre cenas está definido, assim, a única opção de interação disponível é a de navegação entre cenas.

4.2.1.2. Visualização utilizando rotinas 2D e manipulação de imagens

Para um protótipo livre das rotinas da API M3G utilizada na implementação descrita anteriormente, duas outras foram feitas.

Na implementação por *warping*, o cilindro de suporte é uma curva parametrizada, então a imagem de fundo é projetada nele. Depois, de acordo com as informações da câmera virtual, a imagem é projetada novamente na tela. Para este procedimento é necessário que a imagem seja decomposta em pixels, possibilitando sua manipulação.

Na plataforma J2ME, isto é feito lendo o valor das cores de cada pixel para um vetor de inteiros e então recalculando a posição desses pixels em um segundo vetor, utilizado para criar a imagem final.

Em tal abordagem mostrou baixa performance, que será comentada no próximo capítulo.

Em uma segunda implementação sem utilizar rotinas 3D, a imagem do panorama foi utilizada diretamente como imagem de fundo , sem aplicar nenhum tipo de transformação na imagem. Apenas faz-se o cálculo de quanto que um pixel da imagem de fundo representa em relação a cada grau de rotação da câmera virtual. A partir dessa informação, é definido que parte da imagem deve ser desenhada. Nestas duas abordagens, assim como na implementação 3D, apenas o ângulo de azimute tem variação. Não há alteração do ângulo de elevação da câmera virtual

PUC-Rio - Certificação Digital Nº 0321113/CC

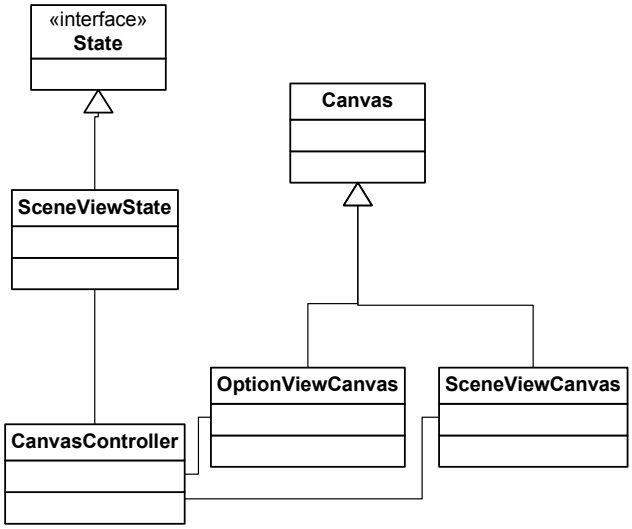

Figura 9 - Diagrama que representa a arquitetura do modulo de visualização da implementação utilizando rotinas 2D.

PUC-Rio - Certificação Digital Nº 0321113/CC

Nestas duas implementações, a classe *ScenarioViewCanvas* é a responsável pelos cálculos de desenho da cena, inclusive dos objetos que são representadas pela classe *Target*, que encapsula a classe *Sprite*, e serão expostas mais à frente.

O efeito de zoom é executado alternando entre versões da imagem do panorama com resoluções diferentes. Estas são previamente redimensionadas e recuperadas a partir do módulo de *download* de cenas que será apresentado ainda nesse capítulo.

Assim como na implementação utilizando as rotinas 3D, a classe *CanvasController* é a responsável por tratar os eventos gerados pelo usuário. Neste caso, a interação entre o usuário e os objetos presentes na cena é feita quase da mesma maneira. Porém , quando o usuário aperta a tecla de seleção, o objeto a ser selecionado é escolhido através da colisão da imagem que representa um alvo posicionado no centro da tela e o *sprite* que representa o objeto.

Também foram incluídas informações de escopo nos objetos.

Assim, como já tinha sido feito na implementação anterior, o usuário escolhe em um menu qual ação deseja executar dentre as definidas pelo escopo do objeto. Apenas a navegação entre cenas foi implementada para os protótipos atuais.

4.2.2.
Módulo de navegação entre cenas

O Módulo de navegação é o responsável pela recuperação da descrição da cena e das imagens a serem usadas, seu tratamento e preparação do módulo de visualização de cenas.

Este módulo é composto por três classes:

- *SceneDescriptorDownloaderState*
- *SceneBinariesDownloaderManagerState*
- *SceneBinariesDownloaderState*

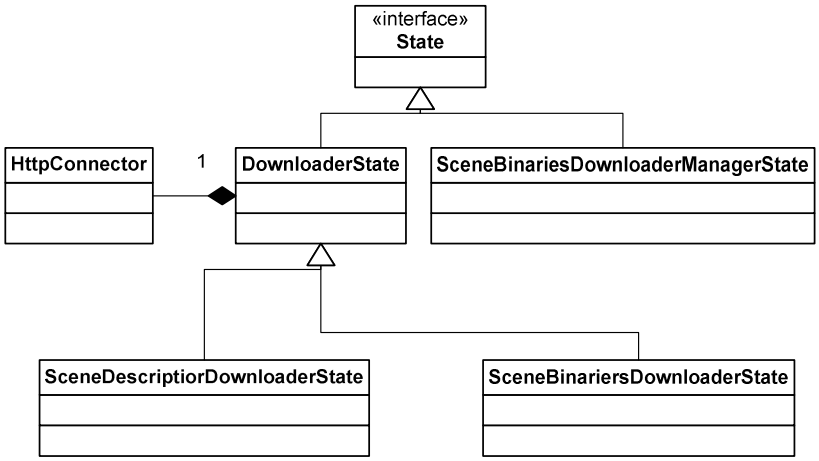

Figura 10 - Diagrama de classes do modulo de recuperação de cenas.

A primeira, como o nome indica, é o estado responsável pelo baixar o descritor da cena.

O descritor atualmente é um arquivo texto separado por espaços e que contem a identificação das imagens do panorama e descrição dos objetos presentes na cena.

Na implementação atual o servidor é apenas um repositório de arquivos, assim, as identificações das imagens são apenas os nomes dos arquivos correspondentes.

A descrição dos objetos inclui seu posicionamento, identificação das imagens que devem ser usadas em sua representação, posicionamento e escopo, além de informações associadas necessárias para a interação com os mesmos.

Como comentado anteriormente apenas o escopo de navegação entre cenas foi implementado e a única informação associada a objetos presente no

PUC-Rio - Certificação Digital Nº 0321113/CC

atual protocolo é a identificação do descritor da cena correspondente. Outra limitação atual, mas que pode ser facilmente modificada, é o posicionamento do objeto ser descrito apenas pela direção, em relação à posição inicial da câmera, em que se encontra.

As imagens e outros eventuais arquivos binários são recuperados pelas outras duas classes que compõem o módulo de navegação entre as cenas: *SceneBinariesDownloaderManagerState* e *SceneBinariesDownloaderState*.

A *SceneBinariesDownloaderManagerState* verifica quais imagens, da lista montada a partir do descritor da cena, estão presentes no *cache* de imagens e solicita ao *SceneBinariesDownloaderState* o *download* destas.

As imagens são posteriormente recuperadas, para o uso pelo módulo de visualização, a partir da classe *ImageLoader* . Esta classe cria um *cache* das imagens utilizadas, evitando assim a replicação quando estas são referenciadas por mais de um objeto.

Abaixo se encontra exemplo de um arquivo de descrição de cena.

```
1 boston.png 2 descriptor2.txt 90 object2.png
descriptor3.txt 105 object3.png
```

Como citado anteriormente, cada um dos componentes são separados por espaços em branco e possuem a seguinte estrutura :

1 – Número de imagens para a representação do panorama
2 – Para cada imagem
   a) Nome da imagem
3 – Número de objetos presentes na cena
4 - Para cada objeto temos:
   a) descritor da cena associada
   b) posicionamento do objeto
   c) imagem que o representa

4.2.3. Objetos presentes na cena

Os objetos presentes na cena são representados por *billboards*, que na plataforma M3G são representados pela classe *Sprite3D*, e na implementação

PUC-Rio - Certificação Digital Nº 0321113/CC

são representados pela classe *Target*, que encapsula a classe *Sprite*, que faz parte das classes básicas da plataforma J2ME . Desta forma a cena pode conter uma grande quantidade de objetos sem comprometer a interação do usuário.

Como já citado, cada objeto pertence a uma ou mais classificações através da definição de escopos, que é oferecida pela própria plataforma. Esta classificação é feita a por um mapa de bits representados em um inteiro (32 bits), onde um objeto de *escopo A* pertence a um *escopo B,* se o resultado da operação *bitwise E* for diferente de zero.

A partir desta classificação é definido qual o tipo de interação é oferecido ao usuário.

Alguns efeitos poderiam ser implementados, como a representação de cada objetos como um conjunto de *billboards*, os chamados *billboards clouds* (Decoret, 2003). Os contrapontos são o aumento do número de imagens referentes a cada objeto, uso de memória, e claro eventuais problemas de processamento se o número de camadas for muito grande.

PUC-Rio - Certificação Digital Nº 0321113/CC