UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA DE VISUALIZAÇÃO 3D DE UM SISTEMA DE ARQUIVOS

VINÍCIUS CAMPOS DA ROSA KRAUSPENHAR

BLUMENAU
2007

2007/1-43

VINÍCIUS CAMPOS DA ROSA KRAUSPENHAR

FERRAMENTA DE VISUALIZAÇÃO 3D DE UM SISTEMA DE ARQUIVOS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Mauro Marcelo Mattos - Orientador

BLUMENAU
2007

2007/1-43

FERRAMENTA DE VISUALIZAÇÃO 3D DE UM SISTEMA DE ARQUIVOS

Por

VINÍCIUS CAMPOS DA ROSA KRAUSPENHAR

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Mauro Marcelo Mattos, Doutor – Orientador, FURB
Membro:	Prof. Antônio Carlos Tavares, Pós-Graduado – FURB
Membro:	Prof. Dalton Solano dos Reis, Mestre – FURB

Blumenau, 10 de julho de 2007

Dedico este trabalho aos meus pais, familiares e amigos que me auxiliaram em todos os momentos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha mãe Vera e minha tia Therezinha que incentivaram em todos os momentos minhas decisões e me apóiam em todos os momentos.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Prof. Mauro Marcelo Mattos, pela confiança e entusiasmo depositados em todas as etapas deste trabalho.

A melhor maneira de predizer o futuro é inventá-lo.

Alan Kay

RESUMO

Este trabalho descreve o desenvolvimento de uma ferramenta de visualização tridimensional de um sistema de arquivos em ambiente Windows, utilizando conceitos da metáfora de memória espacial. Gerenciar arquivos em um ambiente onde o espaço disponível para armazenamento e a quantidade de informações cresce abruptamente tem sido um problema para a maioria dos usuários. A ferramenta apresenta uma interface onde é possível criar diversos objetos e associá-los a arquivos e/ou diretórios, onde uma alteração nestes componentes do sistema de arquivos influi em uma representação nos objetos associados. A ferramenta foi desenvolvida no ambiente Visual Studio 2005 utilizando a linguagem de programação C++ e o motor gráfico OGRE.

Palavras-chave: Interfaces gráficas. Sistemas de arquivos. Visualização de informações.

ABSTRACT

This work describes the development a file system tri-dimensional visualization tool using the metaphor of spatial memory. The tool has an interface where it is possible to create several metaphorical objects and to associate them to archives and/or directories. Changes in any components of the file system influences the representation of the associated objects. The tool was developed in the Visual Studio 2005 using the programming language C++ and graphical engine OGRE.

Key-words: Graphical user interfaces. File systems. Visualization of information.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

WMI – *Windows Management Instrumentation*

API – *Application Programming Interface*

BCC – Curso de Ciências da Computação – Bacharelado

DSC – Departamento de Sistemas e Computação

OGRE – *Object-Oriented Graphics Rendering Engine*

GUI – *Graphic User Interface*

CEGUI – Crazy's Eddie Graphic User Interface

OIS – *Object Oriented Input System*

LISTA DE SÍMBOLOS

% - por cento

- sustenido

SUMÁRIO

1 INTRODUÇÃO

Durante os últimos anos, a evolução das tecnologias de armazenamento e a redução do custo dos dispositivos permitiram que cada usuário tivesse sob sua propriedade centenas de arquivos de diferentes interesses e necessidades. Com o advento da Internet, compartilhar arquivos tornou-se uma tarefa trivial, onde um número imensurável de informações está sendo disponibilizado a todo o momento, permitindo a facilidade para se adquirir cada vez mais informações.

Gerenciar documentos de forma eficiente nesse ambiente tem sido um problema constante para a maioria dos usuários. Assim como a quantidade de arquivos aumenta, a necessidade de uma organização eficiente dos dados mostra-se cada vez mais relevante. Farhoomand e Drury (2002) concluem em sua pesquisa que 62% dos usuários tem dificuldade ou impossibilidade de gerenciar seus documentos.

A grande maioria das soluções para este problema envolve uma hierarquia de pastas e arquivos. Algumas pesquisas têm criado protótipos experimentais, normalmente se baseando nas metáforas de árvores hierárquicas, cronológicas ou memória espacial (HENDERSON, 2004, p. 20).

Em 1998, um projeto da Microsoft apresentou uma nova técnica para gerenciamento de documentos chamada Data Mountain (ROBERTSON et al., 1998, p. 153). Esta técnica consiste em uma interface criada especialmente para aproveitar a habilidade humana de guardar informações sobre um determinado ambiente e sua orientação espacial. A neurociência trata esta habilidade como memória espacial (UNIVERSITY OF BRISTOL, 2001). A Figura 1 mostra a execução do Data Mountain.

Fonte: Robertson et al. (1998, p. 153).

Figura 1 – Tela de execução do Data Mountain

O protótipo do Data Mountain foi desenvolvido especificamente para substituir o sistema de lista de Favoritos do navegador Internet Explorer, apresentando em um ambiente 3D um plano inclinado com miniaturas das janelas do navegador. Testes realizados pela própria Microsoft comprovaram que o modelo de gerência apresentado tornou-se uma alternativa viável (ROBERTSON et al., 1998, p. 160).

Analisando o problema apresentado na gerência de arquivos e os resultados apresentados pelo projeto Data Mountain, surge a idéia de criar um ambiente nos moldes do Data Mountain onde seja possível aproveitar a habilidade humana de memória espacial nas tarefas de adquirir, guardar, manipular e, principalmente, recuperar e usar objetos do sistema de arquivos [1].

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta para gerência e visualização 3D de um sistema de arquivos tendo como base a metáfora da memória espacial [2].

Os objetivos específicos do trabalho são:

a) desenvolver objetos gráficos tridimensionais para representar os objetos do sistema de arquivos;
b) criar um ambiente tridimensional sobre o qual serão aplicadas as metáforas representativas de objetos do sistemas de arquivos;
c) desenvolver um mecanismo que possibilite ao usuário selecionar uma imagem qualquer e sobre ela identificar regiões que possam ser associadas a objetos do sistema de arquivos;
d) desenvolver mecanismos de sincronização dos objetos do ambiente com os objetos do sistema de arquivos;
e) permitir a navegação e manipulação dos objetos do ambiente virtual.

---

[1] Objetos do sistema de arquivos correspondem a arquivos, atalhos, diretórios locais, diretórios remotos e unidades periféricas.

[2] No contexto deste projeto, o conceito de metáfora de memória espacial está ligado à utilização de imagens como forma de permitir ao usuário realizar operações de manipulação de componentes do sistema de arquivos através de uma linguagem associada ao contexto da imagem selecionada.

1.2 ESTRUTURA DO TRABALHO

O primeiro capítulo apresenta uma introdução do trabalho, os objetivos a serem apresentados e a estrutura do trabalho.

O segundo capítulo contempla a fundamentação teórica do trabalho e descreve o conceito de sistema de arquivos, estruturas hierárquicas, memória espacial e visualização de informações. Além disto, descreve o motor gráfico OGRE que foi utilizado no desenvolvimento da ferramenta apresentada e as funções da API do Windows utilizadas para sincronização com o sistema de arquivos.

Todos os diagramas de casos de uso, classes e diagramas de seqüência, juntamente com detalhes sobre a implementação, técnicas e ferramentas utilizadas são apresentadas no terceiro capítulo.

A descrição da metáfora de representação é descrita no quarto capítulo.

O quinto capítulo descreve as considerações finais sobre o trabalho, incluindo sugestões para extensões em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentados conceitos que influenciam diretamente o entendimento geral do trabalho. Inicialmente serão abordados conceitos de sistema de arquivos e a estrutura de organização hierárquica. Posteriormente serão apresentados estudos relacionados à utilização da memória espacial em ambientes virtuais, o projeto Data Mountain e a visualização de informações. Em seguida, os arquivos semânticos, o motor gráfico OGRE e as técnicas para obtenção de dados do núcleo do sistema operacional utilizadas para prover a sincronização das informações. Serão também contemplados neste capítulo os trabalhos correlatos e o estado da arte.

2.1 SISTEMAS DE ARQUIVOS

De acordo com Tanenbaum e Woodhull (2000, p. 27), o sistema de arquivos é a forma que o sistema operacional usa para representar a informação em um espaço de armazenamento, apresentando um modelo abstrato e independente de dispositivos. Neste modelo são implementadas chamadas de sistema para as operações de criar, remover, ler e escrever arquivos.

O sistema de arquivos é uma parte fundamental do sistema operacional, pois fornece uma visão abstrata dos dados persistidos, além de ser responsável pelo serviço de nomes, acesso a arquivos e de sua organização geral.

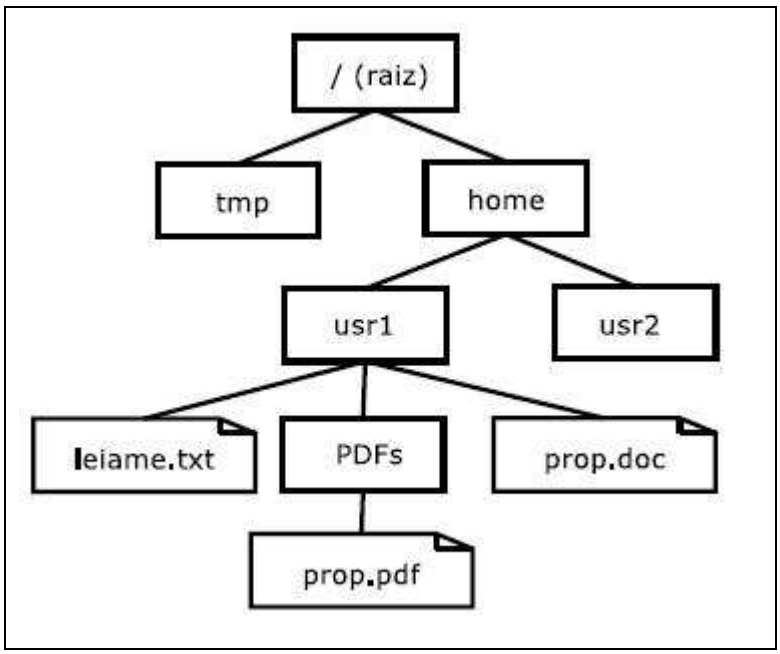
A maioria dos sistemas de arquivos possui o conceito de diretório, que consiste em uma forma de organizar e agrupar arquivos. As mesmas chamadas de sistema utilizadas para operações com arquivos também são aplicadas aos diretórios. Neste modelo, a estrutura hierárquica é formada e pode ser constituída por inúmeros níveis, a partir de um diretório raiz.

Carvalho (2005, p. 3) explica que a maioria dos arquivos armazenados em um sistema de arquivos possui um nome e um caminho, utilizados para a identificação única em tal sistema. Um caminho representa um nó de uma estrutura de diretórios. A localização de um arquivo nesta estrutura parte da raiz, percorrendo os nós correspondentes aos diretórios até encontrar a informação desejada.

2.1.1 ESTRUTURA HIERÁRQUICA

Uma estrutura hierárquica tem como princípio organizar informações em ordem de importância ou disparidade. Normalmente consiste em uma estrutura de árvore onde cada nó da estrutura pode ter apenas um correspondente superior. Na gerência de arquivos, a estrutura hierárquica é comumente utilizada, apresentando diretórios e arquivos como elementos da estrutura (TANENBAUM; WOODHULL, 2000, p. 27; VILLAS, 1993, p. 12).

A Figura 2 mostra uma representação de uma estrutura hierárquica de um sistema de arquivos, apresentando a árvore de diretórios, subdiretórios e arquivos.



Fonte: Carvalho (2005, p. 3).

Figura 2 – Exemplo de uma árvore de diretórios

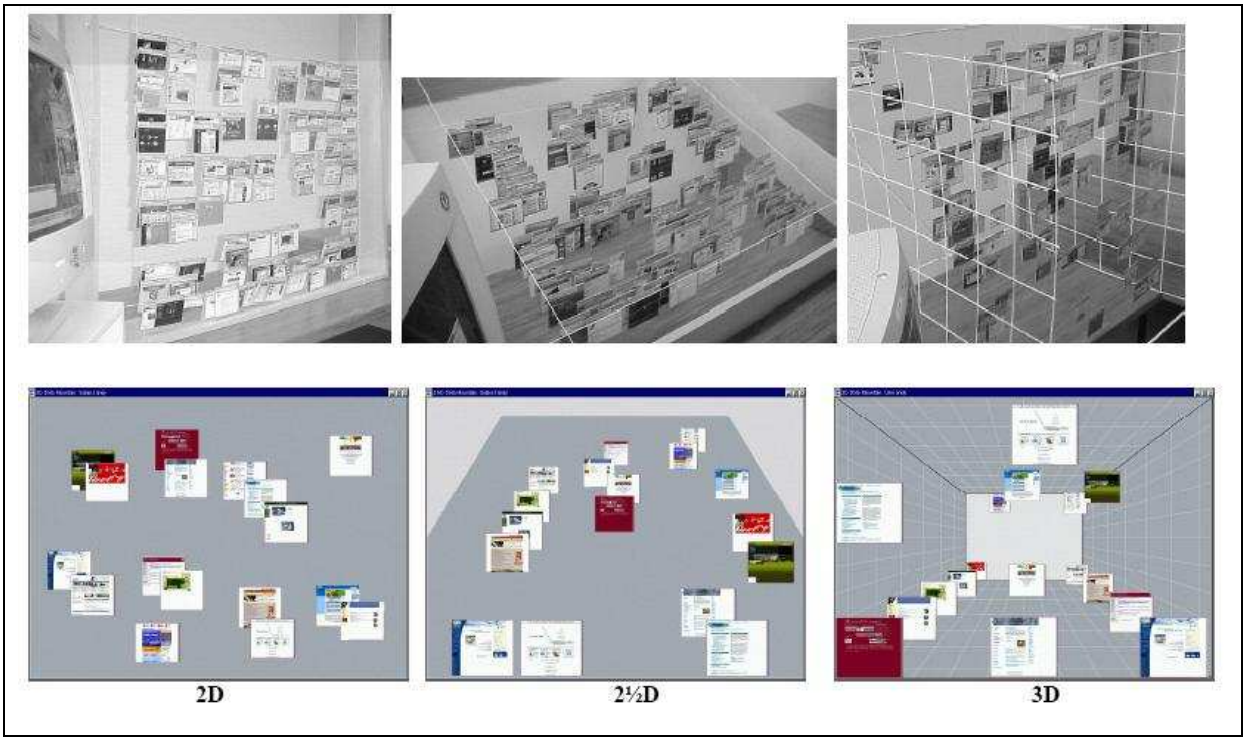
Utilizando ferramentas de gerência como o Windows Explorer, presente no sistema operacional Windows da Microsoft, é possível criar diretórios, subdiretórios e colocar arquivos dentro destes diretórios, definindo assim a estrutura organizacional.

Apesar da organização de arquivos ser uma tarefa bastante pessoal, Henderson (2004, p. 21) identificou que é comum usuários utilizarem um padrão de organização, normalmente composto pela seguinte hierarquia de conceitos: gênero, tarefa, assunto e tempo.

2.2 MEMÓRIA ESPACIAL EM AMBIENTES VIRTUAIS

Cockburn e McKenzie (2001, p. 138) procuraram identificar a efetividade da memória espacial em ambientes de modelos físicos reais e em modelos virtuais correspondentes, partindo dos resultados apresentados pelo Data Mountain.

Utilizando seis ambientes para o estudo, três ambientes reais físicos provendo uma experiência para os usuários sem as limitações de visualização virtual e três ambientes virtuais equivalentes nas perspectivas 2D, $2^1/_2$D e 3D, vários usuários foram submetidos às tarefas de organizar páginas e posteriormente acessar as mesmas na organização criada. O ambiente $2^1/_2$D corresponde à visualização apenas do plano inferior em perspectiva. A Figura 3 apresenta os ambientes utilizados.



Fonte: adaptado de Cockburn e McKenzie (2001, p. 139).

Figura 3 – Ambientes virtuais utilizados para o estudo de memória espacial

A maioria dos usuários apresentou dificuldade na tarefa de organizar os objetos no ambiente $2^1/_2$D e 3D, porém, ficou comprovada a capacidade espacial do cérebro humano de acessar rapidamente os documentos nestes ambientes. Nos testes realizados, o acesso a um item organizado anteriormente no ambientes 2D físico foi de 4,2 segundos, enquanto no ambiente 3D foi de 3,7 segundos (COCKBURN; MCKENZIE, 2001, p. 6).

A utilização de um ambiente tridimensional também apresenta algumas facilidades, como a possibilidade de movimento e rotação, permitindo alterar o ponto de vista das

informações (COCKBURN; MCKENZIE, 2001, p. 145).

2.3 DATA MOUNTAIN

O projeto Data Mountain (ROBERTSON et al., 1998) tem como objetivo desenvolver uma ferramenta de gerenciamento de documentos baseada especificamente em conceitos e estudos sobre a utilização da memória espacial em ambientes virtuais.

O protótipo, desenvolvido no intuito de ser uma alternativa aos Favoritos do Internet Explorer, apresenta um plano inclinado onde são apresentadas diversas janelas onde cada uma representa uma página da internet.

Como os objetos deste protótipo devem representar páginas da internet, cada objeto pode ser distinguido por ser uma pequena imagem da própria página que representa.

Alguns detalhes da implementação visam facilitar a utilização da memória espacial, como efeitos de animação ao movimentar os objetos e a utilização do plano inclinado para prover o conceito de profundidade, facilitando a organização dos objetos.

Um estudo comparando o modelo de organização do sistema de Favoritos do Internet Explorer e o modelo de organização do Data Mountain provou que apesar da maior dificuldade do usuário em organizar inicialmente os objetos, houve maior facilidade de acesso aos objetos, meses depois da organização inicial, no modelo proposto pelo Data Mountain (ROBERTSON et al. 1998, p. 161).

2.4 VISUALIZAÇÃO DE INFORMAÇÕES

Conforme Freitas et al. (2001, p. 143), “visualização de informações é uma área de aplicação de técnicas de computação gráfica, geralmente interativas, visando auxiliar o processo de análise e compreensão de um conjunto de dados, através de representações gráficas manipuláveis”.

Uma representação visual de dados representa um alto grau de abstração. Os dados em si não estão disponíveis ao usuário e o mesmo não possui interesse em acessá-los diretamente. Ao invés disso, prefere analisar padrões ou características semelhantes e trabalhar em cima

destes conjuntos.

> A visualização de informações é o uso de representação visual, interativa e suportada por computador, de dados abstratos para ampliar a cognição. O objetivo de representar o dado abstrato visualmente consiste em auxiliar os indivíduos a enxergarem um fenômeno no dado, usando a percepção para diminuir o esforço cognitivo (CARD et al. 1999, p. 17, tradução nossa).

O dado e suas peculiaridades, sem nenhuma representação visual, são considerados por Romani e Rocha (2001, p. 171) um dado bruto. Através de transformações de dados que contém valores derivados ou estruturas, o dado bruto é transformado numa relação ou conjunto de relações que são mais estruturadas e mais facilmente mapeadas para formas visuais.

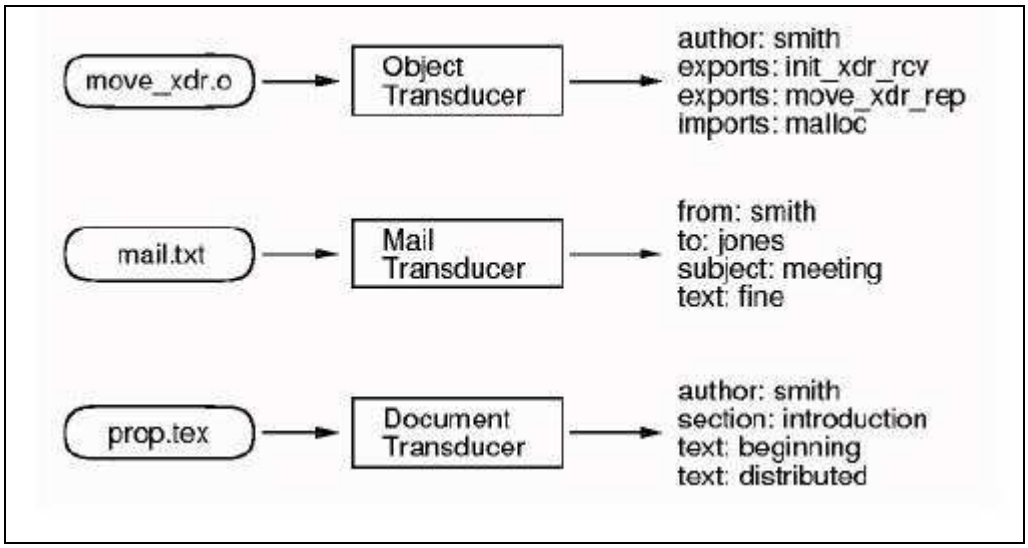
Uma técnica de visualização é baseada numa representação visual e em mecanismos de interação que possibilitam ao usuário manipular essa representação no intuito de melhor compreender um conjunto de dados (Freitas et al., 2001, p. 143). A escolha da técnica de representação da informação depende inteiramente do tipo de informação que será utilizada e das necessidades do usuário.

2.5 ARQUIVOS SEMÂNTICOS

De acordo com Gifford et al. (1988, p. 1), um arquivo semântico é uma informação armazenada que provê acesso flexível e associativo aos dados do sistema pela extração automática de atributos dos arquivos através de tradutores específicos por formato. Pesquisas são efetuadas através destes atributos, definindo grupos de dados semelhantes.

A compatibilidade com os atuais sistemas de arquivos é possível através do conceito de diretório virtual, onde o resultado de cada pesquisa será armazenado. Através dos atributos dos arquivos, é possível determinar agregações de informações, associações entre arquivos e associações entre grupos de informações.

A utilização de arquivos semânticos possibilita a definição de comportamento dos usuários, definindo grupos de dados mais acessados de acordo com o tempo (Gifford et al., 1988, p. 2). A Figura 4 apresenta um exemplo de tradução.



Fonte: Gifford et al. (1988, p. 2).

Figura 4 – Exemplo de tradução semântica

2.6 OGRE

OGRE é um motor gráfico 3D orientado a objetos, escrito em C++, e construído para facilitar o desenvolvimento de aplicações que utilizam aceleração gráfica em 3D. A biblioteca de classes do OGRE abstrai todas as funções de outras bibliotecas mais complexas para desenvolvimento de aplicações em 3D, como OpenGL e DirectX (TORUS KNOT SOFTWARE, 2007).

Este motor gráfico utiliza uma hierarquia flexível de classes permitindo que uma cena possa ser construída livremente, em tempo real e incluindo novos objetos sem a necessidade de que a aplicação precise ser reescrita para suportar estas adições. Desta forma, é possível modificar qualquer aspecto visual sem alterar o comportamento da aplicação.

Além de ser orientado a objetos, o OGRE é orientado à cena. Uma cena é uma representação abstrata de todos os componentes que são apresentados em um mundo virtual. Podem ser componentes estáticos como interiores de um local, modelos como árvores, cadeiras ou personagens, fontes de iluminação ou câmeras utilizadas para visualizar o ambiente.

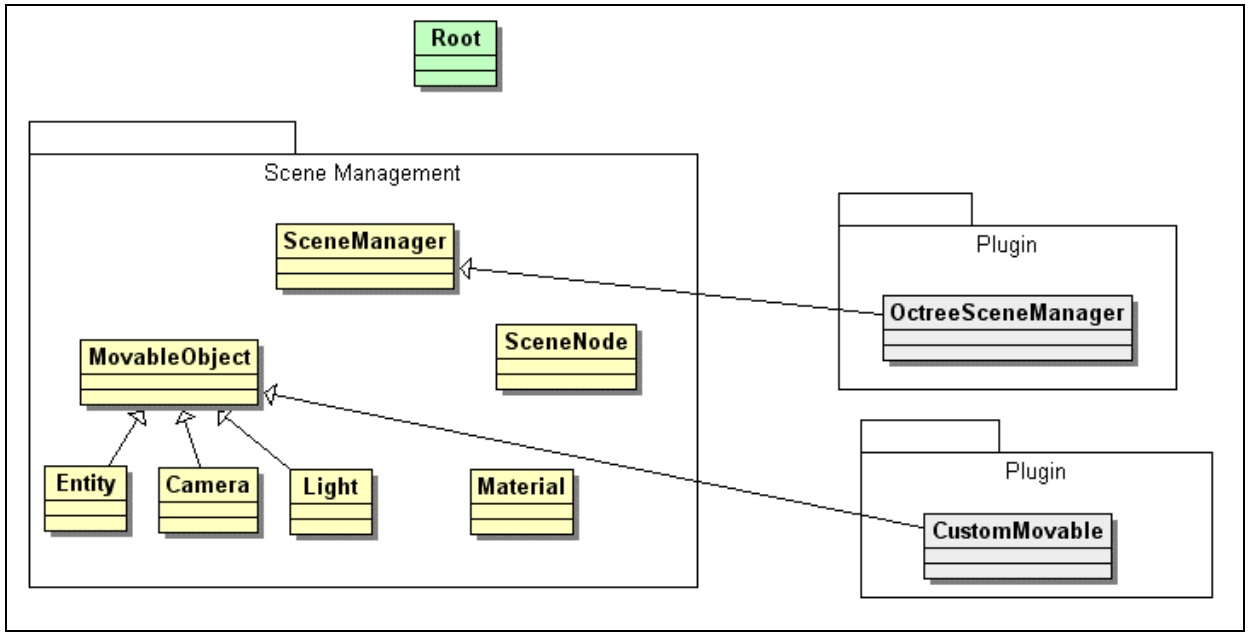
Desta forma, todo componente que será representado no ambiente ou exercerá um comportamento que afeta outros componentes deverá ser anexado à um nó de cena. Este nó de cena é responsável por definir diversas propriedades de seu objeto anexado, como posição,

tamanho e orientação.

Todos os componentes anexados aos nós de cena são controlados pela classe `SceneManager`. Toda vez que um componente é carregado no ambiente, a classe `SceneManager` fica a par da posição deste componente, assim como toda a vez que uma câmera é criada para visualizar o ambiente, a classe `SceneManager` é notificada sobre suas propriedades.

Estas informações são necessárias para que a classe `SceneManager` possa prover a visualização dos nós de cena no ambiente e a interação que há entre eles.

A Figura 5 apresenta o diagrama das principais classes utilizadas para manipulação de componentes no OGRE.



Fonte: adaptado de Torus Knot Sofware (2007).

Figura 5 – Diagrama das principais classes do OGRE

Ao contrário de outros motores gráficos destinados a jogos, outros recursos como GUI, controle de colisão e interação com dispositivos de entrada como mouse e teclado, são disponibilizados por outras bibliotecas externas que fazem integração com o OGRE. Desta forma, é possível escolher qual implementação será utilizada e qual oferece mais recursos para a ferramenta que se deseja construir.

Como interface gráfica, foi escolhida a CEGUI (Crazy Eddie's GUI System) por apresentar documentação de fácil acesso para integração com o OGRE, ser abrangente no número de componentes e customizável. A CEGUI é uma biblioteca livre, orientada a objetos e independente de plataforma, que disponibiliza diversos componentes gráficos, como botões, menus, listas, etc (CEGUI, 2005).

Para a interação com os dispositivos de entrada, foi escolhida a OIS *(Object-Oriented*

Input System). A OIS é uma biblioteca livre escrita em C++ e independente de plataforma, utilizada para tratar qualquer dispositivo de entrada (WRECKEDGAMES, 2006). Por essas razões e por ser utilizada amplamente nas versões mais recentes do OGRE, ela foi escolhida para ser utilizada na implementação do trabalho.

2.7 TÉCNICAS PARA OBTENÇÃO DE DADOS DO SISTEMA OPERACIONAL

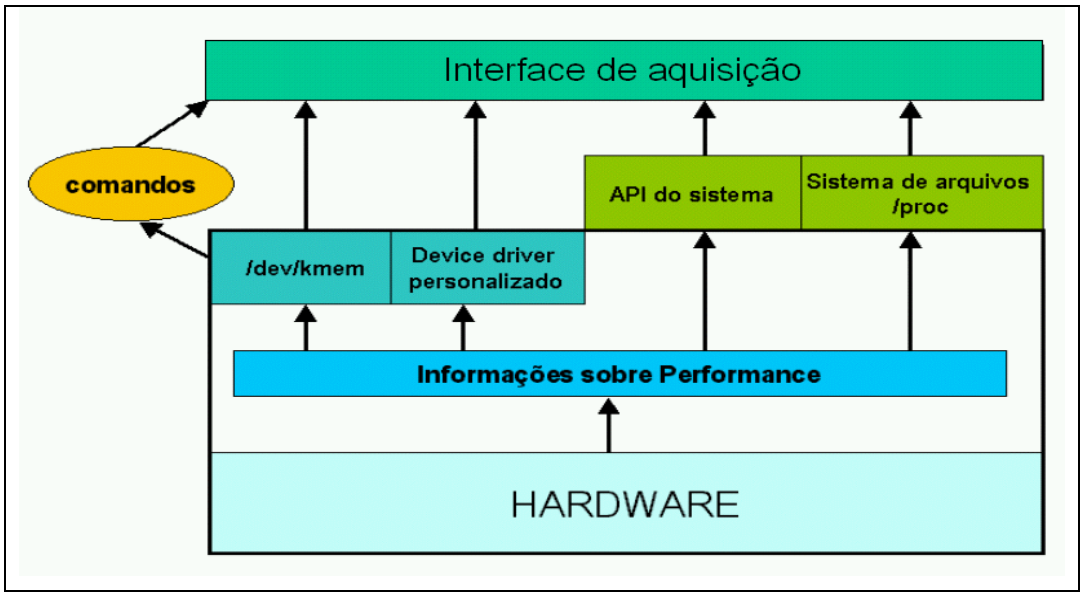
Skjellum et al. (2000) informa que existem diversas técnicas para obter informações sobre o estado dos sistemas operacionais. De acordo com a Figura 6 as técnicas podem ser descritas como as seguintes:

a) através do acesso direto a estruturas de dados do núcleo;
b) através de *device drivers* específicos;
c) através de uma interface com o sistema de arquivos;
d) através de APIs específicas como WMI (*Windows Management and Instrumentation*);
e) através de interface com o sistema operacional como *uptime* do Unix.

Ainda segundo Skjellum et al. (2000), o monitoramento de performance deve ser uma atividade que cause a menor intrusão possível ao sistema. Podem ser consideradas como fontes de intrusão a leitura de informações do sistema e a execução de notificação de eventos.

Para o presente trabalho, foi utilizada a técnica de API específicas, no caso, a WMI.

A WMI é uma infra-estrutura para gerenciamento de dados e operações no sistema operacional Windows (MICROSOFT, 2007). Com as funções disponibilizadas pela WMI, é possível identificar alterações em determinados pontos do sistema de arquivos.



Fonte: Mattos (2003, p. 85).

Figura 6 - Formas de obtenção de informações do sistema operacional

A função `FindFirstChangeNotification` da API é utilizada para criar uma ligação com as alterações que ocorrem em um determinado objeto do sistema de arquivos.

As alterações que serão notificadas são filtradas de acordo com uma série de váriáveis próprias da API. Os parâmetros desta função são detalhados na Figura 7.

```
HANDLE FindFirstChangeNotification(
LPCTSTR lpPathName,                 // Ponteiro para nome do diretório a ser examinado
BOOL bWatchSubtree,                 //  Flag para monitoração de diretório ou árvore de diretório
DWORD dwNotifyFilter                // Condição de filtro para esperar
);
```

lpPathName	Ponteiro para um string terminado em '\0' que especifica o caminho do diretório a ser monitorado.
bWatchSubtree	Especifica se a função irá monitorar o diretório ou a árvore de diretório. TRUE: monitora árvore de diretório FALSE: monitora apenas o diretório
dwNotifyFilter	Condição a ser aguardada: FILE_NOTIFY_CHANGE_FILE_NAME: modificação do nome do arquivo: renomear, criar, ou apagar arquivo. FILE_NOTIFY_CHANGE_DIR_NAME: modificação do nome do diretório: renomear, criar, ou apagar diretório. FILE_NOTIFY_CHANGE_ATTRIBUTES: modificação do atributo no diretório ou árvore de diretórios examinada. FILE_NOTIFY_CHANGE_SIZE: modificação do tamanho de arquivo na árvore ou diretório. FILE_NOTIFY_CHANGE_LAST_WRITE modificação da data da última escrita no arquivo. FILE_NOTIFY_CHANGE_SECURITY: modificação do descritor de segurança no diretório ou árvore de diretórios.

Fonte: Seixas (2000, p. 22).

Figura 7 – Parâmetros da função `FindFirstChangeNotification`

2.8 TRABALHOS CORRELATOS

Ferramentas comerciais e acadêmicas foram desenvolvidas tendo em vista resolver o problema de gerenciamento de arquivos. Dentre elas, foram escolhidas o Task Gallery, BumpTop, Tactile 3D, XCruiser, FSV e StepTree que apresentam uma proposta semelhante.

Em 1999, a Microsoft desenvolveu uma interface para expandir a janela do *desktop* em um ambiente 3D com ilimitadas janelas de *desktop*, chamada Task Gallery (ROBERTSON et al., 1999, p. 1) para o sistema operacional Windows. O conceito é tentar criar uma ilusão baseada na habilidade humana de visão espacial, aonde as pessoas vão intuitivamente navegar no ambiente. Os estudos utilizando o Task Gallery provaram que os usuários utilizaram o sistema de forma bastante natural, tendo comparado o ambiente ao mundo real (ROBERTSON et al., 1999, p. 5).

A interface apresentada pelo Task Gallery utiliza a técnica *Perspective Wall* apresentada por Mackinlay et al. (1991), porém não existe uma definição de que a informação principal está associada à região central da imagem 3D e as áreas laterais contém informações de contexto geral. A Figura 8 mostra a tela de execução do Task Gallery.

Fonte: Robertson et al. (1999, p. 1).

Figura 8 – Tela de execução do Task Gallery

BumpTop (AGARAWALA; BALAKRISHNAN, 2006, p. 1) é um protótipo que está sendo desenvolvido pela Universidade de Toronto. Consiste em um ambiente de gerência de arquivos em 3D para o sistema operacional Windows utilizando conceitos de física na manipulação de documentos, como atrito e massa. Utiliza também um conceito de pilhas para organizar documentos. A visualização corresponde a uma mesa tridimensional onde é possível criar pilhas e distribuir documentos aleatoriamente. A proximidade dos itens com a

tela corresponde à utilização freqüente dos mesmos. Pilhas de documentos normalmente estão ordenadas cronologicamente, pois os itens mais atuais serão inseridos no topo da pilha.

Este projeto apresenta uma alternativa para a gerência de documentos, ignorando completamente a atual estrutura hierárquica para apresentar um modelo mais próximo ao mundo real. A Figura 9 representa a execução do aplicativo BumpTop.

Fonte: Agarawala e Balakrisshnan (2006, p. 3).

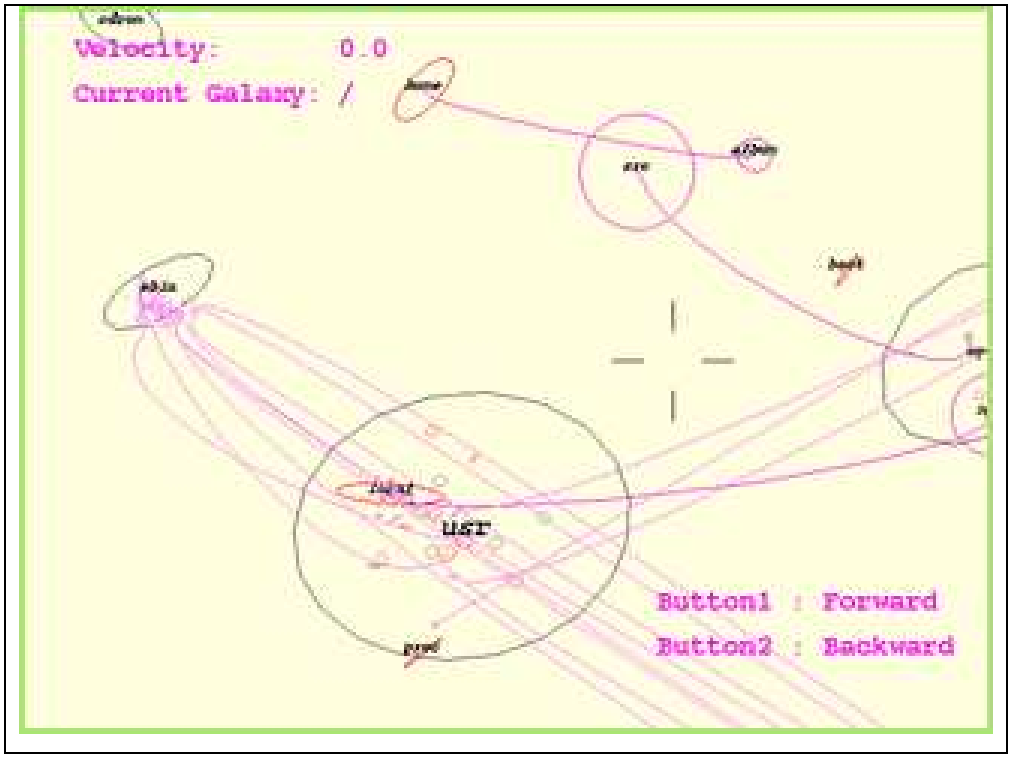
Figura 9 – Arquivos de imagens e aplicativos no ambiente BumpTop

Entre as ferramentas desenvolvidas para a visualização de sistemas de arquivos utilizando técnicas semelhantes, pode-se citar a Tactile 3D (UPPER BOUNDS INTERACTIVE, 2006) onde é apresentado um ambiente semelhante a um jogo tridimensional. A Figura 10 demonstra a visualização do sistema de arquivos na ferramenta Tactile 3D.

Fonte: Upper Bounds Interactive (2006).

Figura 10 – Visualização de diretórios no ambiente Tactile 3D

A ferramenta XCruiser (SHINYAMA, 2003) aplica a metáfora de galáxias para o sistema de arquivos, onde em um ambiente 3D, arquivos são representados como planetas e diretórios por galáxias, como representa a Figura 11.



Fonte: adaptado de Shinyama (2003).

Figura 11 – Visualização de uma partição Unix no ambiente XCruiser

Para representar o volume de informações de um sistema de arquivos, as ferramentas FSV (RICHARD, 1999) e StepTree (TIBSOFT, 2003) fornecem um gráfico 3D contendo o mapeamento de um dispositivo de armazenamento onde o tamanho das regiões do gráfico

correspondem ao tamanho dos arquivos e diretórios.

2.9 ESTADO DA ARTE

Atualmente a gerência de arquivos é limitada às estruturas hierárquicas e modos de visualização restritos a representar estas estruturas. A maioria dos usuários está adaptada a este conceito apesar de reconhecer suas limitações.

Novas formas de gerência de arquivos utilizando novas metáforas exploram possibilidades de facilitar a relação entre informações virtuais e os objetos físicos, onde a metáfora de representação muda para cada usuário de acordo com a sua realidade e necessidade.

A utilização de ambientes tridimensionais permite uma maior facilidade na identificação de objetos virtuais com objetos reais e a navegação entre os objetos tridimensionais permite a utilização da memória espacial na organização das informações.

Metáforas tridimensionais são utilizadas comumente para representar simuladores, jogos eletrônicos e mais recentemente em recursos dos sistemas operacionais como a visualização de imagens.

3 DESENVOLVIMENTO DO TRABALHO

Para detalhar o processo de desenvolvimento descrito no seguinte trabalho, serão abordados os temas a seguir:

a) análise e especificação dos requisitos;
b) especificação através de diagramas de classes e casos de uso;
c) softwares utilizados;
d) estrutura de um programa utilizando OGRE;
e) interação com o ambiente;
f) monitoramento dos objetos do sistema operacional.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta possui os seguintes requisitos funcionais:

a) mapear os objetos do sistema de arquivos para uma metáfora de um ambiente 3D: a ferramenta deverá gerar uma interface baseada em uma cidade onde diretórios intermediários serão representados por ruas, diretórios finais serão representados por casas e os arquivos serão representados por personagens localizados em ruas e casas de acordo com a sua localização no sistema de arquivos;
b) permitir a manipulação de objetos do sistema de arquivos através das tarefas associadas ao contexto da metáfora selecionada: a manipulação de objetos do sistema de arquivos deve ser vinculada à utilização dos objetos na metáfora. Desta forma, deve ser possível vincular objetos do sistema de arquivos aos objetos da metáfora;
c) através de *callbacks* do sistema operacional, notificar alterações efetuadas nos objetos do sistema de arquivos assim como a inclusão de dispositivos de armazenamento atualizado a representação dos objetos no ambiente 3D: a ferramenta deve estar preparada para receber as notificações do sistema operacional e representa-las no contexto da metáfora. Ou seja, quaisquer alterações nos objetos do sistema de arquivos devem ser representadas nos objetos vinculados na metáfora. A inclusão de dispositivos de armazenamento

representará a possibilidade de vincular os mesmos aos objetos da metáfora.

A ferramenta apresenta os seguintes requisitos não funcionais:

a) ser implementado utilizando o ambiente Visual Studio 2005;
b) ser implementado na linguagem de programação C++;
c) ser compatível com o sistema operacional Windows 2000 e XP.

3.2 ESPECIFICAÇÃO

Nesta seção serão apresentados os diagramas de casos de uso e diagramas de classe.

Os diagramas foram desenvolvidos utilizando a ferramenta Enterprise Architect versão 6.5 da empresa Sparx Systems.

3.2.1 Diagramas de casos de uso

Os diagramas de casos de uso apresentados nesta seção tem como ator o usuário que fará a organização dos objetos no ambiente e o próprio ambiente ao receber as mensagens do sistema operacional.

Na Figura 12 é descrito o usuário como ator que utilizará a interface e efetuará a organização de seus arquivos no ambiente, criando objetos e manipulando os mesmos.

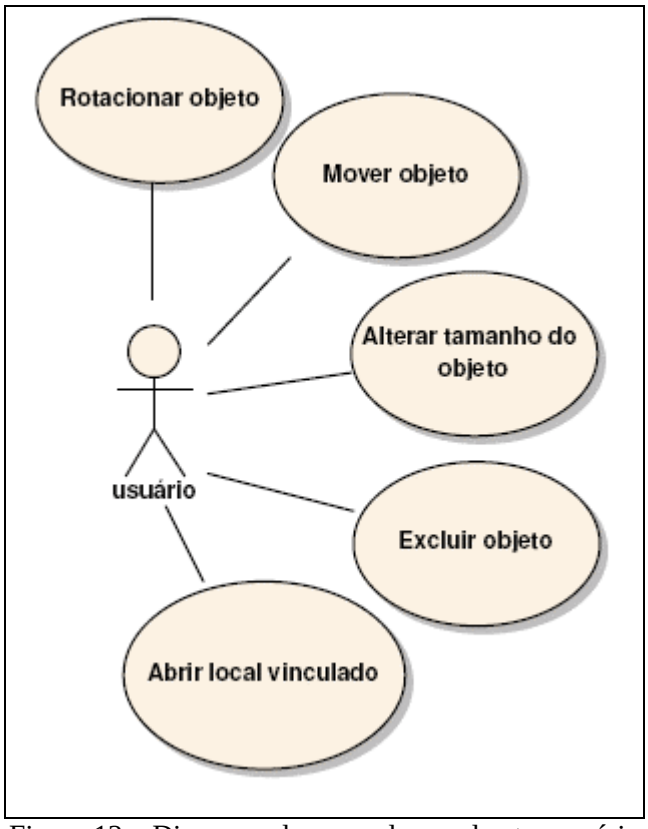


Figura 12 – Diagrama de casos de uso do ator usuário

Na Figura 13 o ambiente é o ator que receberá a mensagem do sistema operacional no momento em que os arquivos vinculados foram alterados e efetuará a tratamento nos objetos do sistema.

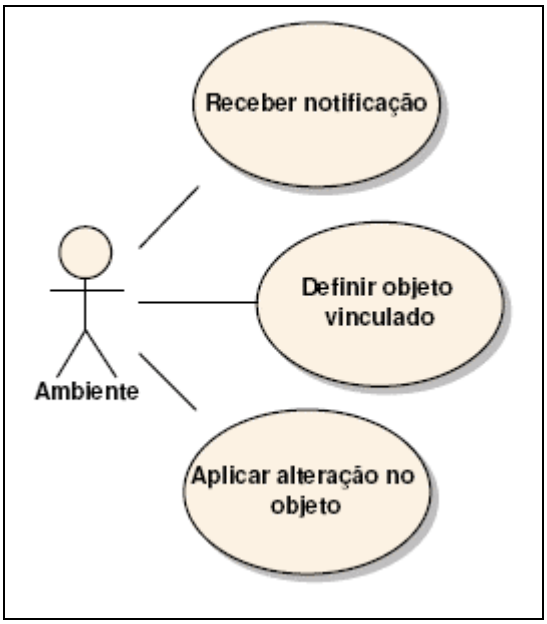


Figura 13 – Diagrama de casos de uso do ator ambiente

3.2.2 Diagramas de classes

Os diagramas de classes apresentados aqui demonstram a implementação efetuada para a construção da ferramenta.

A Figura 14 demonstra a arquitetura do sistema em termos de um diagrama de classes da aplicação. A seguir cada uma das classes é apresentada em detalhes.

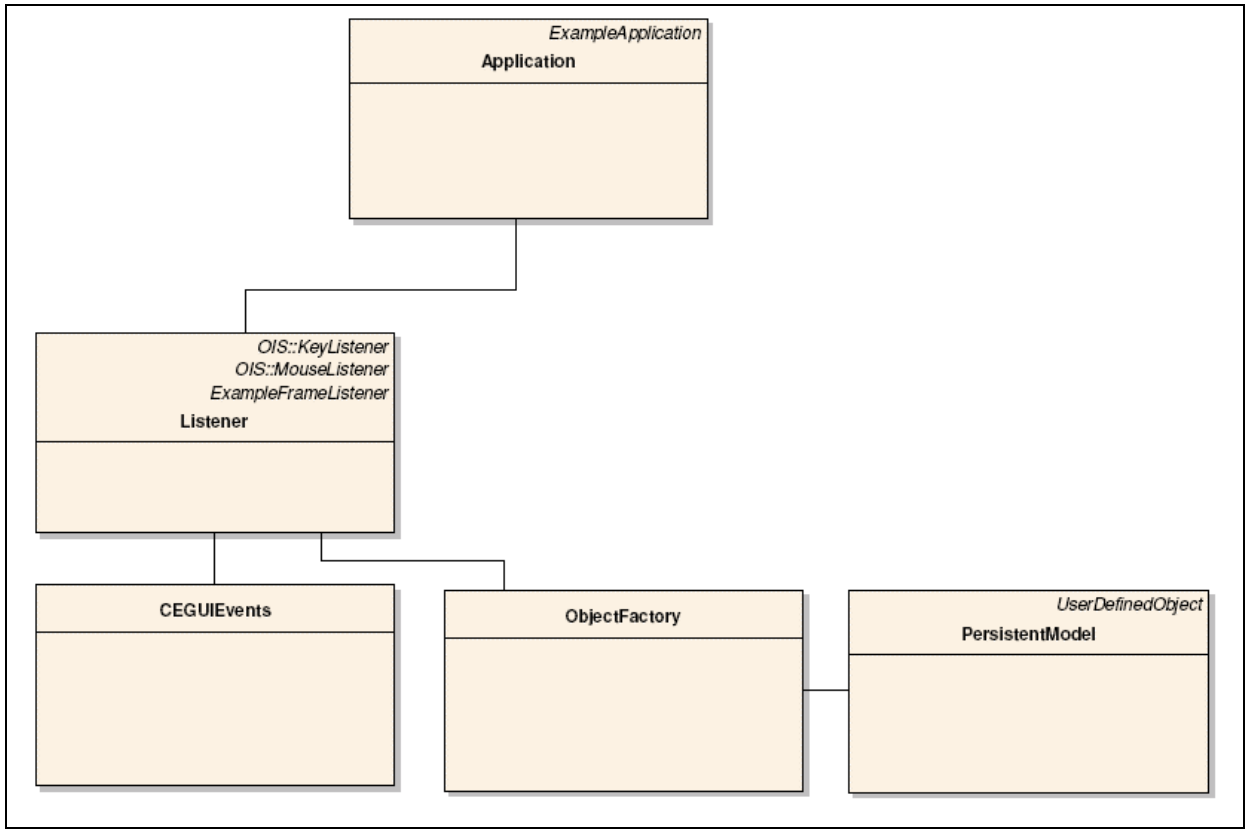


Figura 14 – Diagrama de classes da aplicação

3.2.2.1 Classe `Application`

A Figura 15 apresenta a classe `Application` que é utilizada para inicializar o ambiente e seus diversos componentes, como luz, câmera, terreno, GUI e o objeto da classe `Listener` que irá efetuar a representação visual do ambiente e receber os eventos de mouse e teclado.

Esta classe é herdada da classe `ExampleApplication`, disponível na distribuição do OGRE. A classe `ExampleApplication` contém as inicializações padrões do OGRE, como a leitura dos *plugins* gráficos que estão disponíveis.

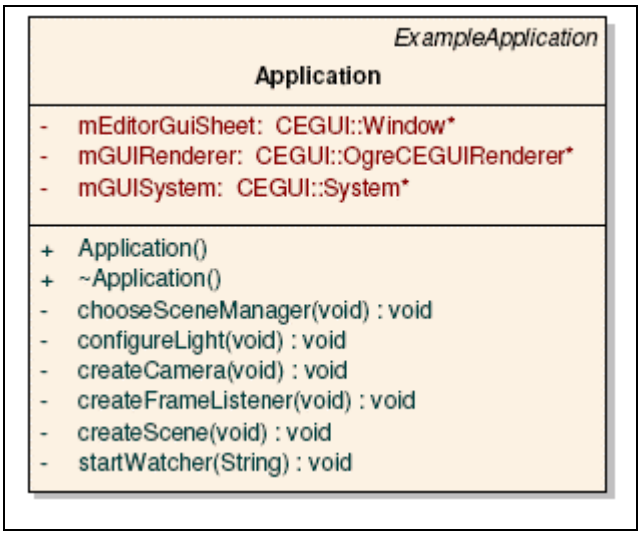


Figura 15 – Diagrama de classe `Application`

O método `createScene` efetua estas inicializações, e é executado após o método `chooseSceneManager` que efetua a configuração da cena, que pode ser:

a) `ST_GENERIC`: tipo genérico, efetua uma performance satisfatória na maioria das cenas;
b) `ST_EXTERIOR_CLOSE`: tipo utilizado para cenas de tamanho relativamente pequeno e que envolvam um terreno estático;
c) `ST_EXTERIOR_FAR`: utilizado para cenas ao ar livre onde o tamanho do terreno precisa ser muito extenso;
d) `ST_INTERIOR`: utilizado para ambientes pequenos como interiores. É otimizado para prover resultados mais rápidos em relação a interseções com paredes e corredores.

A classe `Application` também inicializa o monitoramento do sistema de arquivos através do método `startWatcher`.

3.2.2.2 Classe `Listener`

A classe `Listener`, representada na Figura 16, corresponde à manipulação dos eventos da OIS e a apresentação das modificações que estes eventos produzem no ambiente. Qualquer evento do mouse, teclado ou do sistema de arquivos gera uma chamada nos métodos desta classe para que seja devidamente tratado no ambiente.

Na versão 1.2 da OGRE, o tratamento dos dispositivos de entrada era nativo. A partir da versão 1.4 este tratamento passou a ser integrado com a biblioteca OIS. Por esta razão que

a classe `Listener` herda as propriedades tanto da classe `ExampleFrameListener` como das classes `MouseListener` e `KeyListener`.

Figura 16 – Diagrama da classe `Listener`

Os métodos `frameStarted` e `frameEnded` são executados toda vez que é efetuada a renderização. Nestes métodos são executados todos os métodos que irão representar uma alteração na visualização das cenas, como o recebimento de uma notificação do sistema de arquivos através do método `receiveWatcherMessage` e o controle da posição da câmera na movimentação da mesma através de comandos de mouse e teclado no método `controlCameraPosition`.

3.2.2.3 Classe `CEGUIEvents`

A OGRE utiliza como padrão a CEGUI (Crazy Eddie's GUI System) para disponibilizar janelas, botões e outros componentes semelhantes para interface gráfica. Esta classe declara quais métodos são utilizados quando eventos da CEGUI são executados, como pressionar um botão, selecionar um item de uma lista e mover uma janela.

A Figura 17 demonstra a classe `CEGUIEvents`.

Figura 17 – Diagrama da classe `CEGUIEvents`

Esta classe efetua a ligação com a biblioteca `FileManagement` que contém as funções utilizadas para acessar os objetos do sistema operacional, necessárias para efetuar os vínculos.

O principal método desta classe é o `setupEventHandlers`. Este método é responsável por efetuar a ligação dos eventos dos objetos da CEGUI com os métodos da classe.

3.2.2.4 Classe `ObjectFactory`

Esta classe é utilizada para abstrair a gerência dos objetos na interface. Consiste em métodos para criar, localizar, excluir, alterar formato dos objetos e associação com objetos do sistema de arquivos. A Figura 18 mostra os métodos e atributos desta classe.

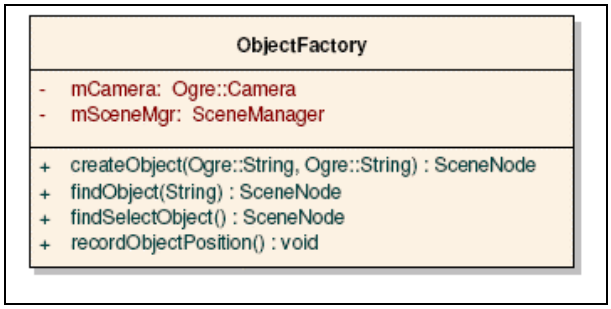


Figura 18 – Diagrama da classe `ObjectFactory`

O método `createObject` efetua a criação de um novo objeto e sua ligação com um nó de cena além de criar o objeto de persistência e efetuar a ligação entre eles.

3.2.2.5 Classe `PersistentModel`

Todos os objetos criados pelo usuário são persistidos pela classe `PersistentModel`. Esta classe é responsável por guardar dados dos objetos como posição, nome e arquivos vinculados. A Figura 19 demonstra o diagrama desta classe.

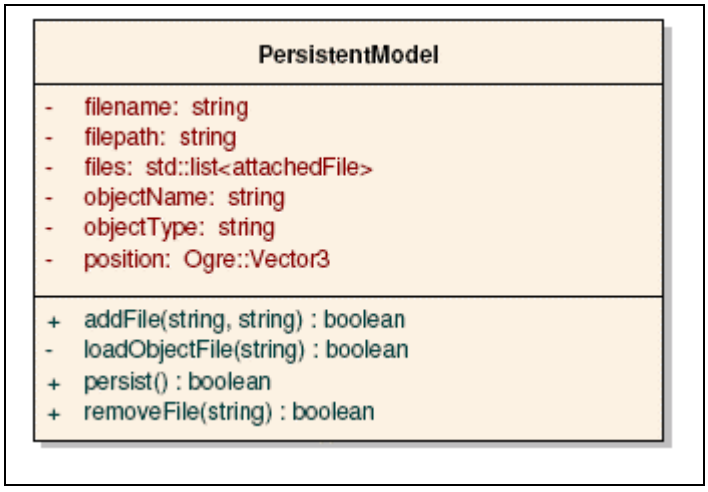


Figura 19 – Diagrama da classe `PersistentModel`

O método `loadObject` efetua a leitura dos arquivos de persistência e cria os objetos no ambiente, utilizando a instância do objeto da classe `ObjectFactory`.

É importante verificar que esta classe é herdada da classe `UserDefinedObject`. Esta

classe pertence ao OGRE e é necessária para que possa ser feita à ligação entres os objetos da classe `PersistentModel` e os objetos da classe `MovableObject` do OGRE através do método `setUserObject da classe MovableObject`.

3.3 IMPLEMENTAÇÃO

Nesta seção do trabalho são abordadas características dos softwares utilizados. Além disto, detalhes sobre a implementação do mecanismo de sincronização com o sistema operacional, a persistência dos objetos, geração dos modelos e do ambiente tridimensional, geração de interfaces com a biblioteca CEGUI, interação da CEGUI com o ambiente e interação com a biblioteca OIS.

3.3.1 Softwares utilizados

Para efetuar a codificação foi utilizada a IDE Visual Studio C++ 2005 em conjunto com o motor gráfico OGRE da empresa Torus Knot Software. Inicialmente a implementação foi feita com a versão 1.2, porém foi modificada para utilizar a versão 1.4 em decorrência de sua liberação estável. O sistema operacional utilizado foi Microsoft Windows XP.

A biblioteca para interpretar comandos de entrada foi a OIS versão 1.0 compatível com a versão 1.4 da OGRE. Para prover a GUI foi utilizada a biblioteca CEGUI versão 1.0 gerando a interface em XML.

A criação dos modelos e conversão dos mesmos para o formato *mesh* utilizado pela OGRE foi efetuada com a ferramenta 3ds Max versão 8 da empresa Autodesk e o *plugin* LEXIExporter 3ds Max 1.0.5 da empresa ITE ApS.

O acesso aos arquivos do sistema operacional foi possível utilizando as funções disponibilizadas pela API Win32.

3.3.2 Estrutura de um programa utilizando OGRE

A OGRE disponibiliza um *framework* contendo os principais objetos para construir

uma aplicação. Este *framework* deve ser adicionado na classe principal da aplicação a ser construída e inicializa o objeto da classe `Root.` Esta classe representa o ponto de início da aplicação, obtendo os sistemas de renderização disponíveis, gerenciamento das configurações salvas e acesso às outras classes do sistema.

A estrutura inicial de uma aplicação que utiliza o OGRE pode ser vista no Quadro 1.

```
#include "ExampleApplication.h"

class TutorialApplication : public ExampleApplication
{
protected:
public:
    Application()
    {
    }

    ~Application()
    {
    }
protected:
    void createScene(void)
    {
    }
};

#if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
#define WIN32_LEAN_AND_MEAN
#include "windows.h"

INT WINAPI WinMain( HINSTANCE hInst, HINSTANCE, LPSTR strCmdLine, INT )
#else
int main(int argc, char **argv)
#endif
{
    // cria o objeto da aplicação
    TutorialApplication app;

    try {
        app.go();
    } catch( Exception& e ) {
#if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
        MessageBox( NULL, e.getFullDescription().c_str(), "Ocorreu um erro!", MB_OK |
MB_ICONERROR | MB_TASKMODAL);
#else
        fprintf(stderr, "Ocorreu um erro: %s\n",
                e.getFullDescription().c_str());
#endif
    }

    return 0;
}
```

Quadro 1 – Estrutura inicial de uma aplicação OGRE

O método `createScene` é responsável por inicializar diversos componentes fundamentais da interface, como câmeras, terreno, luzes e a GUI.

O Quadro 2 apresenta a criação destes componentes.

```
void createScene(void)
{
       // configura luz do ambiente
       configureLight();
      // carrega arquivo que contém o terreno e suas dimensões
      mSceneMgr->setWorldGeometry( "terrain.cfg" );
       // cria a câmera principal
       SceneNode *node = mSceneMgr->getRootSceneNode()->
            createChildSceneNode( "CamNode1", Vector3( -400, 200, 400 ) );
    . . .
    // configuração da CEGUI
    mGUIRenderer = new CEGUI::OgreCEGUIRenderer(mWindow,
                      Ogre::RENDER_QUEUE_OVERLAY, false, 3000,mSceneMgr);
    mGUISystem = new CEGUI::System(mGUIRenderer);
    . . .
    //carrega o layout
      CEGUI::Window* sheet =
                      CEGUI::WindowManager::getSingleton().loadWindowLayout(
                                                 (CEGUI::utf8*)"Demo13.layout");

       mGUISystem->setGUISheet(sheet);
      //inicia as threads que irão monitorar o sistema de arquivos startWatchers();
}
```

Quadro 2 – Criação dos componentes primários do OGRE

Utilizando apenas a classe `Application` é possível construir uma aplicação simples em OGRE. Uma aplicação mais complexa envolvendo interação com o usuário envolve a utilização de uma segunda classe, `ExampleFrameListener`.

A classe `ExampleFrameListener` é utilizada para receber notificações antes e depois de apresentar uma cena. No presente trabalho foi criada a classe `Listener` que herda as propriedades da classe `ExampleFrameListener`.

Na classe `Application`, o método `createFrameListener` é responsável pela criação do objeto `Listener` e por vincular este objeto à classe `Root` do OGRE conforme mostra o Quadro 3.

```
void createFrameListener(void)
{
       mFrameListener= new Listener(mWindow, mCamera, mSceneMgr,
                      mGUIRenderer, CEGUI::WindowManager::getSingleton());
       mFrameListener->showDebugOverlay(false);
       mRoot->addFrameListener(mFrameListener);
}
```

Quadro 3 – Instanciação de um objeto da classe Listener

Dois métodos são fundamentais nesta classe, o método `frameStarted` e o método `frameEnded`, que correspondem ao recebimento de notificações antes e após a renderização.

3.3.3 Interação com o ambiente

Interação com o ambiente corresponde a qualquer ação do usuário para criar, manipular e excluir objetos presentes no ambiente. Esta interação ocorre com o tratamento dos eventos de mouse e teclado pela classe `Listener` e sua repercussão no ambiente de

acordo com o estado atual.

A classe `Listener` sobrescreve os métodos `mousePressed`, `mouseMoved` e `keyPressed` das classes `KeyListener` e `MouseListener`.

O Quadro 4 apresenta a implementação do método `mouseMoved`. De acordo com o botão do mouse que estiver pressionado, uma ação será executada. Se nenhum dos botões estiver pressionado, o ambiente apresentará o nome do objeto selecionado. Se o botão central for movimentado, a câmera irá aplicar o *zoom* no ambiente.

```
virtual bool mouseMoved (const OIS::MouseEvent &e)
{
       CEGUI::System::getSingleton().injectMouseMove(e.state.X.rel, e.state.Y.rel );
       if (e.state.Z.rel > 0) {
              mTranslateVector.z = -mMoveScale;// Move a câmera para frente
              mCamera->moveRelative(mTranslateVector);
       }
       else if (e.state.Z.rel < 0)  {
              mTranslateVector.z = mMoveScale;// Move câmera para trás
              mCamera->moveRelative(mTranslateVector);
       }

      if ( mLMouseDown ) {   onLeftDragged( e );  }
      else if ( mRMouseDown ){  onRightDragged( e );  }
       else {  showSelectedObject( e ); }
       return true;
}
```

Quadro 4 – Método sobrescrito `mouseMoved`

No caso de um objeto estar selecionado, o botão esquerdo estar pressionado e o mouse for movimentado, a ferramenta efetuará a tentativa de movimentação do objeto. O Quadro 5 demonstra o código utilizado para este fim.

```
bool onLeftDragged( const OIS::MouseEvent &e ) {
       if (mCurrentObject == NULL)
              return true;
      . . .
       //efetua o cálculo da posição do mouse
       Ogre::Real x = Ogre::Real(e.state.X.abs) / Ogre::Real(e.state.width);
       Ogre::Real y = Ogre::Real(e.state.Y.abs) /Ogre::Real(e.state.height);

      //cria o raio de pesquisa
       Ray mouseRay = mCamera->getCameraToViewportRay( x, y );

      //efetua a busca
       mRaySceneQuery->setRay( mouseRay );
       mRaySceneQuery->setSortByDistance( true );

       RaySceneQueryResult &result = mRaySceneQuery->execute();
       RaySceneQueryResult::iterator itr;
       for ( itr = result.begin( ); itr != result.end(); itr++ )  {
              if ( itr->worldFragment ) {
                   mCurrentObject->setPosition(itr->worldFragment->singleIntersection.x,
                                               mCurrentObject->getPosition().y,
                                               itr->worldFragment->singleIntersection.z );
                   x -= objLabel->getWidth().d_scale / 6;
                   y += objLabel->getHeight().d_scale / 6;
                  objLabel->setPosition(CEGUI::UVector2(CEGUI::UDim(x,x),CEGUI::UDim(y,y)));
                . . .
              }
       }
       return true;
}
```

Quadro 5 – Movimentação de objeto

De acordo com a posição do mouse e a intersecção desta posição com o terreno, a posição do objeto será definida. Clicando com o botão direito do mouse em um objeto será apresentado um menu de opções conforme mostra a Figura 20.

Eventos de teclado também são tratados no sistema, porém, são utilizados apenas para a movimentação da câmera.

Figura 20 – Menu de opções do objeto

3.3.4 Monitoramento dos objetos do sistema operacional

O método `createScene` da classe `Application` também é responsável por executar o método `startWatcher`. Este método cria uma *thread* para cada *drive* do sistema operacional e efetua o monitoramente de qualquer alteração que ocorrer no *drive*.

Este monitoramento é possível através de chamadas de funções da API Win32. Em um primeiro momento da implementação uma *thread* era criada para cada arquivo vinculado, porém, esta solução comprometia o desempenho da aplicação. Desta forma, foi efetuada a alteração para criar apenas uma *thread* para cada *drive* e utilizada a opção de monitorar os subdiretórios.

O Quadro 6 apresenta o código fonte do método `WatchDirectory` que efetua o monitoramento onde é possível verificar que a cada vez que uma notificação do sistema operacional é disparada, o método `RunClient` é executado. Este método envia uma mensagem contendo o local que disparou a notificação. Esta mensagem é recebida pelo

método `receiveWatcherMessage`, da classe `Listener`, que irá definir qual objeto é responsável por aquela notificação.

```
void WINAPI WatchDirectory(void){
   DWORD dwWaitStatus;
   setModifications();
   // aguarda até uma notificação ser disparada pelo sistema operacional
   while (true)    {
      dwWaitStatus = WaitForMultipleObjects(2,dwChangeHandles, FALSE,1000);
      switch (dwWaitStatus) {
         case WAIT_OBJECT_0: // um arquivo foi criado ou excluído
            RunClient(lpDrive, WAIT_OBJECT_0);
            if ( FindNextChangeNotification(dwChangeHandles[0]) == FALSE )
                ExitProcess(GetLastError());
            break;
          case WAIT_OBJECT_0 + 1: // um diretório foi criado ou excluído
            RunClient(lpDrive, WAIT_OBJECT_0 + 1);
            if (FindNextChangeNotification(dwChangeHandles[1]) == FALSE)
               ExitProcess(GetLastError());
            break;
         case WAIT_OBJECT_0 + 2:  // um diretório ou arquivo foi acessado
            RunClient(lpDrive, WAIT_OBJECT_0 + 2);
            if (FindNextChangeNotification(dwChangeHandles[1]) == FALSE)
                ExitProcess(GetLastError());
            break;
```

Quadro 6 – Método `WatcherDirectory` utilizado para monitoramento

O Quadro 7 apresenta os métodos `RunClient` e `receiveWatcherMessage`.

```
static int RunClient(int typeMessage){//int argc, char* argv[])
       HANDLE hMailslot;

       hMailslot = CreateFile( g_szMailslot, GENERIC_WRITE,FILE_SHARE_READ,
                               NULL, OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);

       if (INVALID_HANDLE_VALUE == hMailslot) {
              return 1;  //Erro
       }

       SIZE_T cbBytes;

       //envia a mensagem
       BOOL bResult = WriteFile( hMailslot, szBuffer,strlen(szBuffer)+1,&cbBytes,NULL);
      . . .
void receiveWatcherMessage() {
       bResult = ReadFile(hMailslot,szBuffer, sizeof(szBuffer),&cbBytes,&overlap);

       WaitForSingleObject(hMailslot, 1);
       mCurrentObject = mFactory->findObject(szBuffer);
       if ((strcmp(szBuffer, myBufferCreated) == 0) ||
           (strcmp(szBuffer, myBufferChanged) == 0) ||
           (strcmp(szBuffer, myBufferRenamed) == 0) ||
           (strcmp(szBuffer, myBufferDeleted) == 0))    {
              HWND JanelaPrincipal = FindWindow(NULL, "Ogre Render Window");
              SetForegroundWindow (JanelaPrincipal);
              if (strcmp(szBuffer, myBufferCreated) == 0)
                     mFactory->setObjectCreated(mCurrentObject, szBuffer);
           . . .
```

Quadro 7 – Métodos para comunicação das notificações

É possível visualizar que o método `receiveWatcherMessage` ao receber uma notificação procura o objeto vinculado, ativa a janela do ambiente e efetua o tratamento de acordo com o tipo de mensagem recebida.

4 METÁFORA DE REPRESENTAÇÃO

Para representar os objetos do sistema de arquivos no ambiente tridimensional, foi criada a metáfora de uma cidade, com seu centro, ruas, habitantes e casas. Esta metáfora foi criada por ser de fácil relacionamento entre os objetos do sistema de arquivos e os objetos da metáfora, uma vez que um ambiente composto por diversas ruas que demonstram o percorrer de um caminho até a informação desejada assemelha-se à organização hierárquica.

A Figura 21 apresenta um esboço da representação do sistema de arquivos na metáfora.

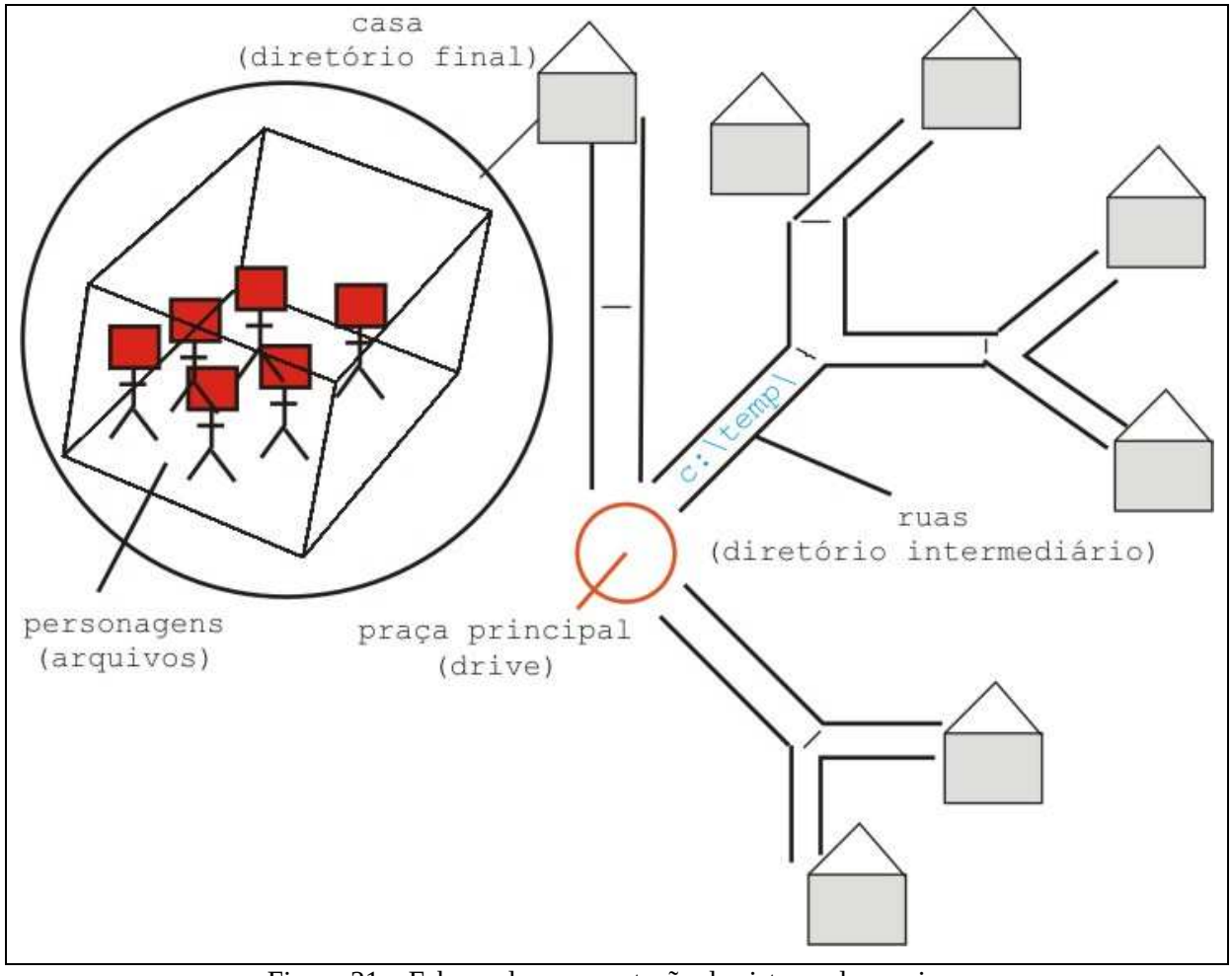


Figura 21 – Esboço da representação do sistema de arquivos

Utilizando esta metáfora é possível também estender as representações adicionando objetos da metáfora que corresponderão a outros objetos do sistema de arquivos que não aqueles representados aqui, como mídias removíveis e periféricos diversos.

A Figura 22 mostra o ambiente tridimensional com a configuração apresentada na Figura 21.

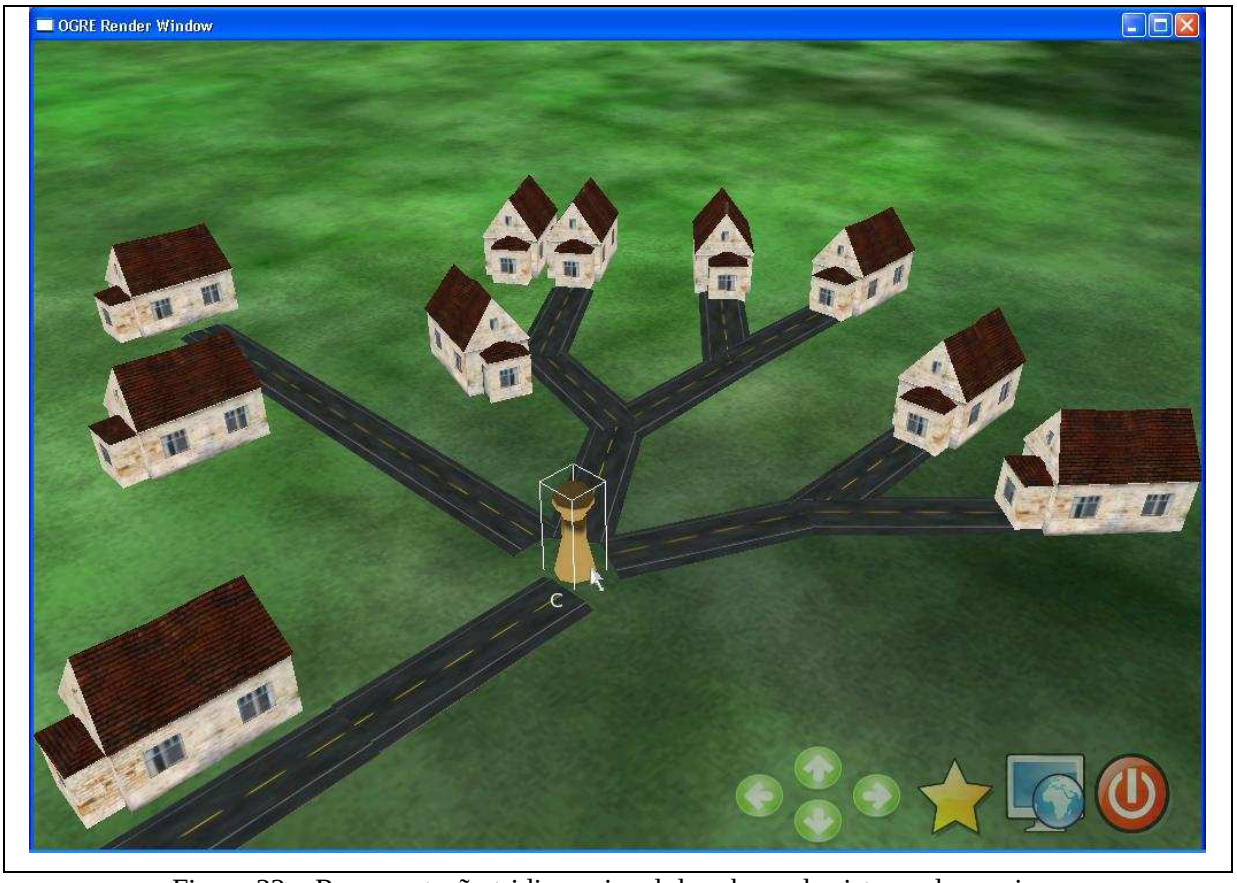


Figura 22 – Representação tridimensional do esboço do sistema de arquivos

Na Figura 22 pode-se visualizar um objeto central correspondente ao *drive* e seus subdiretórios representados por ruas e casas. Uma rua com bifurcações representa um diretório intermediário que contém outros diretórios intermediários, assim como uma rua que conduz a várias casas representa um diretório intermediário que contém outros diretórios finais. A estrutura e o layout são estáticos e limitados devido à complexidade de representar um grande volume de dados em uma nova metáfora de representação.

4.1 IMPLEMENTAÇÃO DA METÁFORA

Todo objeto do sistema de arquivos possui um objeto correspondente no ambiente de acordo com o tipo de objeto. O sistema percorre os diretórios do sistema de arquivos e efetua a criação dos objetos dinamicamente. Diretórios que contenham outros diretórios são representados no ambiente através de objetos na forma de uma estrada. Diretórios que não contenham outros diretórios são representados através de objetos na forma de uma casa. Arquivos são representados através de objetos na forma de um personagem humanóide.

Ao percorrer cada objeto do sistema de arquivos, o ambiente efetua uma chamada ao método `createObject` da classe `ObjectFactory` que irá efetuar a criação da nova entidade e sua ligação ao nó da cena que permitirá a visualização e manipulação do objeto. O Quadro 8 apresenta o método `createObject`.

```
SceneNode* createObject(Ogre::String name, Ogre::String type) {
       Entity *ent;
      // se já existir um objeto com o mesmo nome, não efetua a criação
       if (findObject(name))
              return NULL;

       ent = mSceneMgr->createEntity( "Object_" + String(name), String(type) + ".mesh" );

       SceneNode* mCurrentObject =
                          mSceneMgr->getRootSceneNode()->createChildSceneNode(String(name));
       mCurrentObject->attachObject( ent );
       mCurrentObject->setScale(1.0f, 1.0f, 1.0f);

      // o modelo de persistência é criado/carregado para o objeto
       PersistentModel* mPersistentMode = new PersistentModel(mCurrentObject, String(type));
      . . .
       return mCurrentObject;
}
```

Quadro 8 – Método para criação de objetos

O Quadro 9 apresenta o método `loadDirectory` que demonstra como os objetos do sistema de arquivos são identificados e representados no ambiente.

```
int loadDirectory(LPTSTR path, LPTSTR name, int position, SceneNode* lastObj ){
...
       // busca o primeiro arquivo no diretório
       hFind = FindFirstFile(DirSpec, &FindFileData);
       if (hFind == INVALID_HANDLE_VALUE) { return (-1); }
       // busca todos os outros arquivos do diretório
     while (FindNextFile(hFind, &FindFileData) != 0) {
       if ((FindFileData.dwFileAttributes ==  FILE_ATTRIBUTE_DIRECTORY)){
              lastDirectory = false;
              ...
              SceneNode* mObj = NULL;
              mObj = createRoad(name, Vector3(lastObj->getPosition().x, 0,
                                          lastObj->getPosition().z), 1, true);
       }
       else if (FindFileData.dwFileAttributes == FILE_ATTRIBUTE_ARCHIVE)  {
                 createFileObject(FindFileData.cFileName, Vector3(5,0,5), fullpath);
              }
     }//while
         ...
      dwError = GetLastError();
      FindClose(hFind);
      if (dwError != ERROR_NO_MORE_FILES)
         return (-1);
      if (lastDirectory)      {
        createHouse(name,Vector3(lastObj->getPosition().x,0,lastObj->getPosition().z),true);
      }

  return 0;
}
```

Quadro 9 – Método `loadDirectory`

A Figura 23 apresenta o objeto que representa um diretório que contém outros diretórios.

Figura 23 – Representação de um diretório intermediário

A Figura 24 mostra a representação de um diretório que não contém diretórios.

Figura 24 – Representação de um diretório final

A Figura 25 demonstra a representação de um arquivo no ambiente.

Figura 25 – Representação de um arquivo

A Figura 26 demonstra a representação do local onde são localizados os arquivos recentes.

Figura 26 – Representação dos documentos recentes

A Figura 27 representa o local onde são localizados os arquivos excluídos.

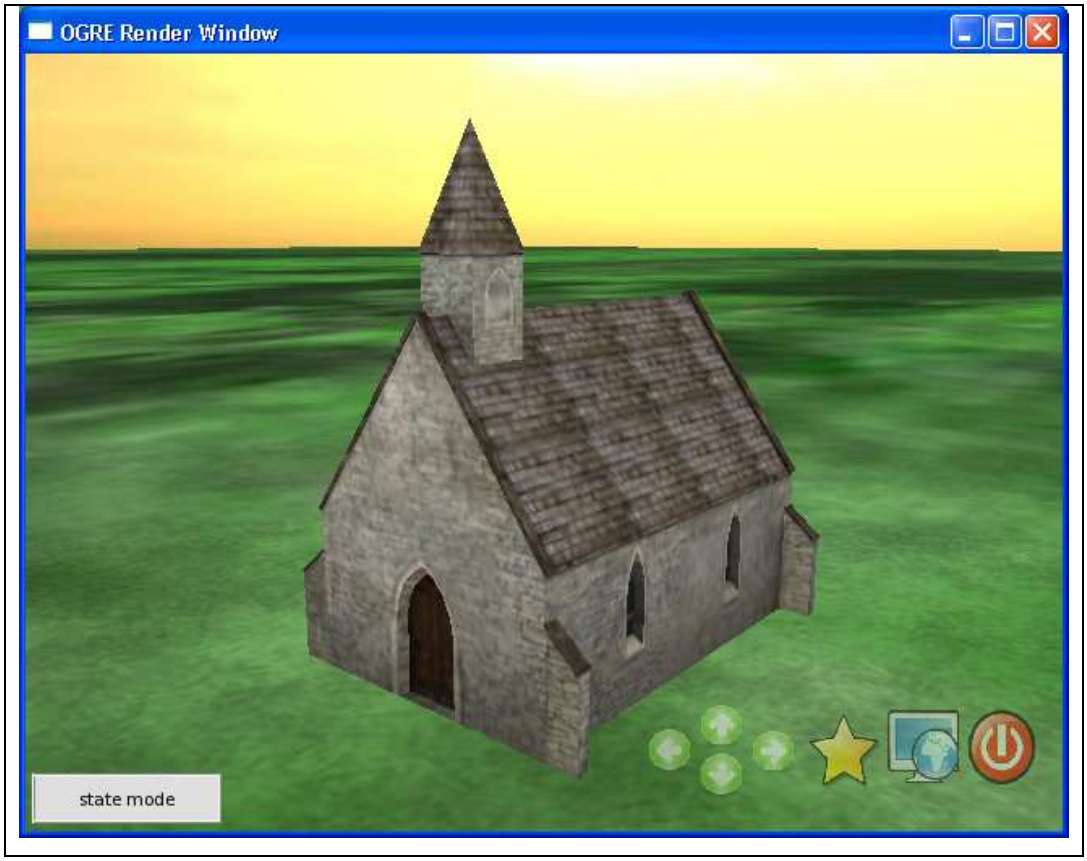


Figura 27 – Representação da lixeira

4.2 PERSISTÊNCIA DOS OBJETOS

Todo o objeto criado no ambiente é persistido através da classe `PersistentModel`. Sempre que o ambiente é carregado, todos os arquivos de persistência são percorridos e seus objetos são apresentados no ambiente. Quando o ambiente é finalizado, estes arquivos são atualizados com os dados correntes dos objetos. O Quadro 10 mostra um exemplo de um arquivo de persistência.

```
[header]
	demonstracao
	building_mushroom
	demonstracao.obj
	c:\models\demonstracao.obj
	[position]
	242.765
	-0.165444
	464.303
	[files]
	1|functions.api|F:\API-GUIDE\
```

Quadro 10 – Exemplo de um arquivo de persistência

A classe `PersistentModel` contém atributos como o nome do objeto, sua posição e os arquivos e/ou diretórios que estão vinculados.

A seção `[header]` do arquivo grava informações pertinentes ao cadastro do objeto, seu nome, seu tipo e seu arquivo de persistência. A seção `[position]` contém sua posição nos eixos x, y e z. A seção `[files]` contém os objetos do sistema operacional vinculados ao objeto do ambiente.

4.3 VÍNCULO COM OS OBJETOS DO SISTEMA OPERACIONAL

Todo objeto do ambiente representa vínculos que existem com os objetos do sistema de arquivos. Estes vínculos são criados no momento em que o objeto do sistema de arquivos é verificado e representado no ambiente.

Clicando com o botão direito do mouse no objeto e selecionando a opção Visualizar vínculos, é apresentado o formulário que apresenta os objetos do sistema operacional que estão vinculados. A Figura 28 apresenta este formulário.

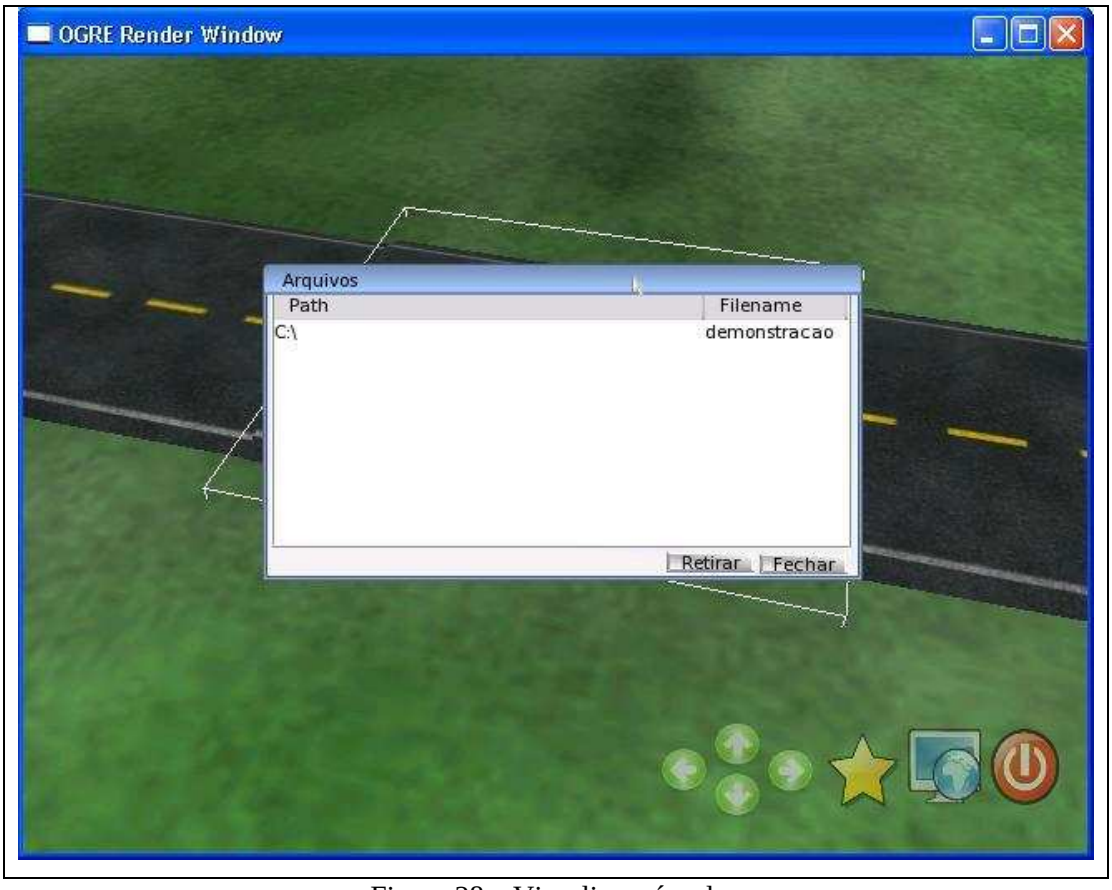


Figura 28 – Visualizar vínculos

Toda vez que um arquivo ou diretório é representado, o objeto da classe `PersistentModel` correspondente ao objeto selecionado é atualizado, inserindo o nome do arquivo e seu caminho em uma lista. O Quadro 11 apresenta o método `addFile` que efetua esta inclusão.

A classe `PersistentModel` contém uma estrutura para guardar os dados do arquivo e esta estrutura é associada à uma lista implementada pela biblioteca STD.

```
bool addFile(string filename, string filepath) {
        unsigned int newindex;
        newindex = this->files.back()._index;
        newindex++;
       // inclui o arquivo na lista de arquivos do objeto
        this->files.push_back(attachedFile(newindex, filename.c_str(), filepath.c_str()));
        // persiste a inclusão
        ofstream saida(this->getFilepath().c_str(), ios::app);
        char buffer [33];
        string path = itoa(newindex, buffer, 10);
        path += "|" + filename + "|" + filepath;
        saida << path.c_str() << endl;
        saida.close();

        return true;
}
```

Quadro 11 – Método `addFile` da classe `PersistentModel`

4.4 RESULTADOS E DISCUSSÃO

Os estudos realizados na criação do Data Mountain e por Cockburn e McKenzie (2001) demonstraram as vantagens da utilização da memória espacial em ambientes virtuais.

Entretanto, estes estudos não apresentam uma ferramenta direcionada ao gerenciamento do sistema de arquivos. Os outros trabalhos correlatos utilizados para gerência de arquivos não fazem uso dos conceitos de memória espacial.

A OGRE como motor gráfico para o desenvolvimento contribui para a inclusão de novos objetos e a expansão da ferramenta para outros sistemas operacionais, modificando as classes de persistência e comunicação com o sistema operacional[3].

Além disso, algumas questões foram percebidas ao longo do desenvolvimento, como a necessidade de um algoritmo para a descoberta da melhor forma de redesenho do sistema de arquivos na metáfora e a substituição do menu de acesso para operações dentro do conceito da metáfora como abrir uma pasta ser correspondente a abrir uma porta.

[3] O ambiente foi testado em uma máquina com processador Pentium 4 de 3.0 Ghz, memória ram de 512 MB e placa de vídeo com 128 MB de memória.

O presente trabalho foi submetido e aceito no workshop de sistemas operacionais realizado na cidade do Rio de Janeiro nos dias 4 e 5 de julho de 2007.

5 CONCLUSÕES

O objetivo deste trabalho em criar uma ferramenta para a gerência de objetos do sistema operacional baseado na metáfora de memória espacial foi concluído com êxito.

Em comparação ao projeto Data Mountain, que serviu de inspiração para o desenvolvimento, foram adicionadas melhorias, como a possibilidade de novos objetos em novos formatos serem incluídos ao ambiente e a navegação dinâmica dentro do ambiente.

Em relação à utilização do motor gráfico OGRE, o mesmo mostrou-se facilmente extensível às necessidades da aplicação provendo as funcionalidades necessárias para a implementação, sem envolver a necessidade de utilizar funções de baixo nível disponibilizadas por bibliotecas gráficas como DirectX e OpenGL.

Apesar da necessidade de utilizar funções da API Win32 para prover o monitoramento dos objetos do sistema de arquivos, tais funções permitiram o monitoramento completo do sistema, sem a necessidade de utilização abusiva de recursos do computador.

Tendo em vista estas considerações, este trabalho mostra-se uma contribuição ao estudo da utilização da memória espacial na gerência de objetos do sistema de arquivos. Destaca-se que este trabalho foi uma prova de conceito, ou seja, buscou demonstrar a aplicabilidade de uma nova metáfora na forma de interação com os objetos do sistema de arquivos. Certamente muito trabalho tem de ser realizado de forma a possibilitar a utilização desta forma de interação por usuários acostumados a gerenciar arquivos na estrutura hierárquica.

5.1 EXTENSÕES

Sugere-se que a ferramenta possa ser melhorada aplicando novas funcionalidades que facilitem a operacionalidade, como controle de colisão entre os objetos aplicando conceitos da física, múltiplos ambientes, possibilidade de modificar a aparência do ambiente em tempo de execução, utilizar conceitos de arquivos semânticos e estender a aplicação para ser utilizada em outros sistemas operacionais. Contudo, o aspecto mais importante a ser trabalhado refere-se ao detalhamento das novas formas de interação com o ambiente metafórico criado e todas as conseqüências desta interação.

REFERÊNCIAS BIBLIOGRÁFICAS

AGARAWALA, A.; BALAKRISHNAN, R. **Keepin' it real:** pushing the desktop metaphor with physics, piles and the pen. Toronto, 2006. Disponível em: <http://honeybrown.ca/Pubs/BumpTop.pdf>. Acesso em: 30 ago. 2006.

CARD, Stuart K. et al. **Readings in information visualization**: using vision to think. San Francisco: Morgan Kaufmann Publishers, 1999.

CARVALHO, Roberto P. **Sistema de arquivos paralelos**: alternativas para a redução do gargalo no acesso ao sistema de arquivos. 2005. 131 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

TURNER, Paul. **CEGUI**. [S.l.], 2005. Disponível em: <http://www.cegui.org.uk>. Acesso em: 20 mar. 2007.

COCKBURN, A.; MCKENZIE, B. Evaluating the effectiveness of spatial memory in 2D and 3D physical and virtual environments. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 22., 2002, Minneapolis. **Proceedings...** Alpha: Sheridan Printing, 2001. p. 138-146.

FARHOOMAND, A. F.; DRURY, D.H. Managerial information overload. **Communications of the ACM**, Nova Iorque, v. 45, n. 10. p. 22-36. Out. 2002.

FREITAS, C. M. et al. Introdução à visualização de informações. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v. 8, n. 2. p. 143-158. Out. 2001.

GIFFORD, David et al. Semantic file systems. In: ACM SYMPOSIUM ON OPERATING SYSTEM PRINCIPLES, 13., 1991, Pacific Grove. **Proceedings…** Pacific Grove: [s.n.], 1988. p. 16-25.

HENDERSON, Sarah. Personal digital document management. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 24., 2004, Vienna. **Proceedings...** Vienna: [s.n.], 2004. p. 20-25.

MACKINLAY, J. D. et al. The perspective wall: detail and context smoothly integrated. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 11., 1991, New Orleans. **Proceedings...** New Orleans: [s.n.], 1991. p. 173-176.

MATTOS, Mauro M., **Fundamentos conceituais para a construção de sistemas operacionais baseados em conhecimento**. 2003. 382 f. Tese (Doutorado em Engenharia de Produção) - Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis.

MICROSOFT. **Windows management instrumentation**. [S.l.], 2007. Disponível em: <http://www.microsoft.com/whdc/system/pnppwr/wmi/default.mspx >. Acesso em: 2 maio 2007.

RICHARD, D. **FSV**: File System Visualizer. [S.l.], 1999. Disponível em: <http://fsv.sourceforge.net/>. Acesso em: 20 set. 2006.

ROBERTSON, George et al. **Data Mountain**: using spatial memory for document management. Vermont, 1998. Disponível em: <http://www.microsoft.com/usability/UEPostings/p153-robertson.pdf>. Acesso em: 15 ago. 2006.

ROBERTSON, George et al. **Task Galley**: a 3D window manager. Vermont, 1999. Disponível em: <http://research.microsoft.com/~marycz/tg2000.pdf>. Acesso em: 20 set. 2006.

ROMANI, Luciana; ROCHA, Heloísa. O uso de técnicas de visualização de informação como subsídio à formação de comunidades virtuais de aprendizagem em EaD. In: WORKSHOP SOBRE FATORES HUMANOS EM SISTEMAS COMPUTACIONAIS, 4., 2001, Florianópolis. **Anais...** Florianópolis: UFSC/SBC, 2001. p. 169-182.

SEIXAS, Constantino. **Fundamentos e aplicações de sistemas de automação**. Belo Horizonte, 2004. Disponível em: <http://www.cpdee.ufmg.br/~seixas/Especializacao/Download/DownloadFiles/ATR_Cap5.pdf>. Acesso em: 2 maio 2007.

SHINYAMA, Y. **XCruiser**. [S.l.], 2003. Disponível em: <http://xcruiser.sourceforge.net>. Acesso em: 20 set. 2006.

SKJELLUM, A., et al. **Systems administration**. Portsmounth: University of Portsmounth, 2000.

TANENBAUM, Andrew S.; WOODHULL, Albert S. **Sistemas operacionais**: projeto e implementação. 2. ed. Tradução Edson Furmankiewicz. Porto Alegre: Bookman, 2000.

TIBSOFT. **StepTree**. [S.l.], 2003. Disponível em: <http://www.tibsoft.com/index.php?page=steptree>. Acesso em: 20 set. 2006.

TORUS KNOT SOFTWARE. **OGRE 3D**: open source graphic engine. [S.l.], 2007. Disponível em: <http://www.ogre3d.org/index.php?option=com_content&task=view&id=19&Itemid=105>. Acesso em: 10 mar. 2007.

UNIVERSITY OF BRISTOL. **Neural basis of spatial memory**. Bristol, [2001?]. Disponível em: <http://www.bris.ac.uk/depts/Synaptic/research/projects/memory/spatialmem.htm>. Acesso em: 28 ago. 2006.

UPPER BOUNDS INTERACTIVE. **Tactile 3D manual**. [S.l.], 2006. Disponível em: <http://www.tactile3d.com/overview/documentation/manual/Manual.html>. Acesso em: 20 set. 2006.

VILLAS, Marcos V. **Estrutura de dados**: conceitos e técnicas de implementação. 5. ed. Rio de Janeiro: Campus, 1993.

WRECKEDGAMES. **OIS manual**. [S.l.], 2006. Disponível em: <http://www.wreckedgames.com/wiki/index.php/WreckedLibs:OIS:Manual>. Acesso em: 10 maio 2007.