Ferramenta de geração de interfaces gráficas para PDAs

Marcelo de Paiva Guimarães[1] **, Bruno Barberi Gnecco, Marcelo Knorich Zuffo**

Av. Prof. Luciano Gualberto, 158 – Trav. 3 – Butantã

CEP: 05508-900 – São Paulo – SP – Brasil

Laboratório de Sistemas Integráveis – Escola Politécnica – USP

Tel: (+55 11) 3091-5254 – Fax: (+55 11) 3091-5665

{paiva, brunobg, mkzuffo}@lsi.usp.br

Abstract. *This paper explores the possibilities of using portable devices in multiprojection environments, such as CAVEs and Power Walls. A new component of the Glass library is presented: the graphical user interface generator. The Glass library is a framework for virtual reality applications running in graphic clusters. The application generated by this component communicates transparently with a node of the cluster, which processes the events and maintains the synchrony of the rendered images.*

Resumo. *Este trabalho tem como objetivo explorar os recursos do uso de dispositivos portáteis em ambientes de multiprojeção, como CAVEs e Power Walls. Para isto, apresenta o componente de geração de interfaces gráficas para PDAs da biblioteca Glass. A biblioteca Glass é um ambiente de desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos. A aplicação gerada por este componente comunica-se de forma transparente com um nó do aglomerado gráfico, o qual trata os eventos e mantém as imagens sincronizadas no ambiente.*

1. Introdução

Os dispositivos nas aplicações de Realidade Virtual (RV) proporcionam a sensação de imersão, através da conexão do usuário com o ambiente virtual. Para tanto, estes dispositivos atuam de duas formas [1]: lendo os movimentos realizados pelos usuários (pelas várias partes de seu corpo); ou impressionando os sentidos dos usuários, a fim de simular sensações. Tradicionalmente, a RV utiliza dispositivos de interface especializados de alto custo, como luvas, 3-D *joysticks* e capacetes de visualização, que muitas vezes, limitam a liberdade dos usuários e são difíceis de serem manipulados por usuários leigos.

Nos últimos anos os computadores PDAs (*Personal Digital Assistant*) têm sido utilizados largamente. Como conseqüência, um grande número de pessoas estão familiarizadas e o preço tem diminuído. Além disto, a qualidade destes dispositivos tem melhorado, como a velocidade dos processadores e a capacidade de armazenamento

[1] Apoiado pelo Instituto Adventista de Ensino

(*RAM*), o que tem possibilitado utilizá-los para tarefas complexas, como visualização de imagens gráficas, execução de vídeos e aplicações de áudio. Outro ponto importante, é que eles são compatíveis com cartões de rede sem fio IEE 802.11 e com outras interfaces, como GPRS [2], *Bluetooth* [4], Serial[4] e Infra-Vermelho[4].

A facilidade da utilização de PDAs em ambientes de multiprojeção já foi estudada em diversos projetos, como Tweek[5], JAIVE [6] e pelo Virtual Heritage [7]. A contribuição deste trabalho é além de permitir que as aplicações de RV sejam manipuladas por PDAs, é de facilitar a geração da interface gráfica que será executada no PDA. Este se comunicará com um nó do aglomerado gráfico[8], para que os eventos sejam processados e as imagens do ambiente geradas de maneira sincronizada. A ferramenta desenvolvida é um componente da biblioteca de geração de aplicações de Realidade Virtual baseadas em aglomerados gráficos, a Glass. Esta biblioteca garante a sincronização e distribuição dos dados entre os nós do aglomerado.

Os detalhes desta pesquisa serão apresentados nas próximas seções. A seção 2 apresenta uma visão geral da biblioteca Glass. A seção 3 o tratamento de eventos pelas aplicações Glass. A seção 4 apresenta o gerador de interfaces gráficas para PDAs. A seção 5, a utilização deste gerador, apresentado um estudo de caso e finalmente, na seção 6, as conclusões.

2. Glass

O objetivo da Glass é agilizar e facilitar o processo de desenvolvimento de aplicações de RV baseadas em aglomerados gráficos. Esta biblioteca consiste de um conjunto escalonável de componentes que podem ser utilizados pelas aplicações. As aplicações são construídas reutilizando os componentes que estão disponíveis, conforme a necessidade, habilitando-se assim a execução dos serviços oferecidos pela biblioteca. Desta forma, aplicações de RV, como por exemplo: Simuladores de Vôos, Sistemas de Visualização, Sistemas de Apoio a Decisão, Sistemas de Ensino, Jogos, poderão ser executadas em sistemas de multiprojeção, com baixo custo, de forma eficiente e otimizada.

Atualmente a Glass é composta basicamente pelos seguintes componentes:

- *Comunicação*: Fornece soluções para comunicação, sincronização e compartilhamento de variáveis (tipos comuns de dados e estruturas) entre os nós. Além disto, a possiblidade de chamada remota de métodos.

- *Dispositivos*: Fornece recursos para que os dados gerados pelos dispositivos sejam tratados e enviados para os nós que estão interessados. Além disto, recursos para enviar os dados das aplicações para os dispositivos de visualização ou de *force-feedback*;

Essa biblioteca está sendo desenvolvida em C++ de tal forma que oferece abstração dos dados e independência de outras bibliotecas. Deste modo, os tipos comuns de dados e outros poderão ser criados e manipulados, além disto, as soluções já disponíveis poderão ser reutilizadas. A reutilização de outras bibliotecas é uma opção dos desenvolvedores, que podem, por exemplo, utilizar a biblioteca gráfica OpenGL [9] ou

a OpenGL Performer [10] para implementação das aplicações. Um *binding* para Java está em desenvolvimento.

3. O tratamento das interações pelas aplicações Glass

A sensação de imersão gerada pelo uso dos dispositivos é obtida por intermédio do tratamento imediato das ações dos usuários [3]. Este tratamento depende da combinação do tempo de processamento das entradas e o número de dispositivos no ambiente. O tratamento pode ser feito por um nó específico ou por diversos nós do aglomerado gráfico, isto dependerá da arquitetura implementada no sistema.

O nós que manuseiam os dados de entrada dos dispositivos não precisam estar executando as aplicações; contudo, estes devem receber as entradas, processar e enviar o resultado para o nós que estão executando a aplicação. Os eventos devem ser recebidos por todos os nós interessados; caso isto não aconteça, incoerências no ambiente podem ocorrer. Também não é necessário que apenas um nó seja o responsável pelo tratamento dos dados de entrada, assim, a aplicação pode ser projetada de tal forma que um nó seja responsável pelos dados de entradas do teclado, enquanto um outro é responsável pelos dados de um *tracker*.

O tratamento correto desses dados é de extrema importância para as aplicações de multiprojeção, pois o ponto de vista de cada imagem deve ser preciso para que não ocorra incoerência na imagem. Quando a aplicação receber um dado, esta deve executar a rotina de tratamento apropriada, caso contrário, as imagens não serão geradas corretamente. Após geradas, as imagens devem ser apresentadas de forma síncrona.

A Glass fornece um componente, denominado Dispositivos, que garante a distribuição correta dos dados de entrada/saída dos dispositivos para os nós responsáveis pelo tratamento. Desta maneira, os dados de entrada gerados pelos dispositivos *(PDAs, trackers, mouses, haptics e outros)* são encaminhados para os nós responsáveis pelo tratamento e os dados gerados pelas aplicações são enviados para os dispositivos de *force-feedback*.

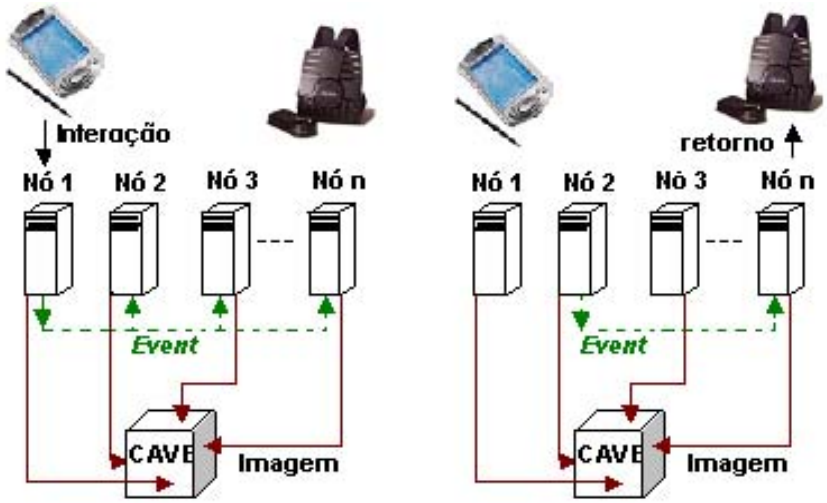
Essas funcionalidades são disponibilizadas pela variável do tipo *Event*, que garante a entrega dos dados de entrada/saída dos dispositivos para os nós interessados. Esta variável é utilizada nas seguintes situações:

- *Para entrada de dados dos dispositivos:* antes da geração da imagem de cada *frame* (quadro), as orientações de posicionamento são obtidas, armazenadas nas variáveis do tipo *Event* e enviadas para todos os nós. A seguir, os nós calculam os pontos de visão (*viewpoints*) dos usuários, tendo como resultado as coordenadas de cada olho. Estas coordenadas são utilizadas para a geração das imagens estereoscópicas, assim os nós geram as imagens simultaneamente e apresentam, de maneira sincronizada, as imagens nos dispositivos de visualização.

- *Para saída de dados dos dispositivos (force-feedback):* após o tratamento de uma interação realizada por um dispositivo entrada, a aplicação pode ter que gerar um dado de retorno para um dispositivo de *force-feedback*. Quando este retorno é gerado por uma aplicação, ele é enviado, utilizando uma variável do tipo *Event*

para o nó onde está conectado este dispositivo, para que ocorra a ação sobre o usuário.

A Figura 1 exemplifica o funcionamento da utilização das variáveis *Event*. A Figura 1(a) mostra o momento em que o usuário interagiu com um dispositivo, no caso um PDA que está transmitindo os eventos para o Nó 1, o envio do dados (*Event*) para os outros nós (linha pontilhada), e por fim, os nós tratando estes dados e enviando (linha contínua) as imagens para o CAVE. A Figura 1 (b) mostra o momento em que o Nó 2 gera um dado (*Event*) de retorno e este é enviado (linha pontilhada) para o Nó n, onde está acoplado um dispositivo de *force-feedback.*



(a) Entrada de dados **(b) Saída de dados**

Figura 1– Entrada/Saída da variável Event

4. Gerador de interface gráfica

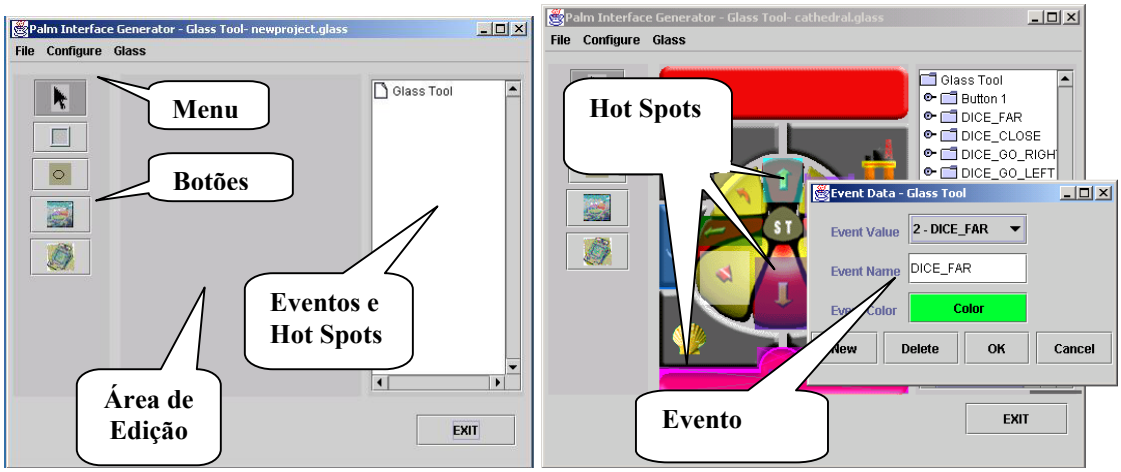
Uma ferramenta para a fácil geração da interface gráfica para PDAs foi criada. Esta ferramenta foi escrita em Java, e permite definir os *hot spots* de uma imagem (gif, jpeg) de maneira gráfica, associando-os a eventos. Um *hot spot* é uma área de uma imagem que ao ser pressionada disparará a transmissão de um evento associado para um nó específico do aglomerado gráfico, o qual é responsável pelo tratamento dos eventos deste PDA. A interface gráfica visível do PDA é a imagem, sem os *hot spots*, que foi criada com o objetivo de representar o sistema de controle da aplicação. Tendo os *hot spots* de uma imagem, esta ferramenta gera o código Java para o aplicativo do PDA, e o trecho de código para o tratamento dos eventos recebidos nos nós responsáveis por seu processamento.

A linguagem da aplicação que é executada no PDA é Java, podendo ser executada em qualquer sistema operacional compatível (Windows CE, Linux). Esta aplicação apresenta a interface gráfica de interação para o usuário. Quando o usuário realiza uma interação, o evento é capturado e transmitido para um nó do aglomerado gráfico, utilizando-se *sockets*.

4.1. Gerador de interface gráfica

Na forma de Aplicação Java, Figura 2(a), o gerador de interface gráfica da Glass é executado pelo interpretador Java. A interface é constituída por botões, menus e por uma área de desenho, que mostra os *hot spots*, no qual cada um está associado a um evento. Os botões têm como objetivo permitir selecionar, criar e apagar *hot spots* (retângulos e elipses). Além disto, os botões acionam a leitura da imagem e a geração do código que serão executados no o PDA, bem como a geração do trecho de código que será executado no nó que tratará os eventos vindos do PDA.

Como mostra a Figura 2(b), a cada *hot spot* é associado um evento, que possui um número, nome e cor. Quando o usuário clicar em um *hot spot*, o número do evento associado será transmitido para o nó responsável pelo tratamento dos dados de entrada. Vários *hot spots* podem estar associados ao mesmo evento. O menu fornece recursos para criar, fechar e salvar os projetos, além disto, permite que o tamanho da área da imagem seja alterada. A área apresentada na Figura 2 é 320 x 240, pois está configurado para um iPAQ H3970.



(a) (b)

Figura 2 – Gerador de interface gráfica para PDA

4.2 Código do PDA

O gerador de interfaces gráficas cria automaticamente uma aplicação Java que usa como interface gráfica a AWT. A escolha de Java como linguagem permite a fácil portabilidade não somente para os diversos sistemas operacionais de PDAs, mas também para outros portáteis e mesmo *desktops*, permitindo, por exemplo, o controle de aplicativos remotamente.

Para gerar o código, os desenvolvedores não precisam conhecer Java, devem apenas carregar a imagem que será executada no PDA, criar os devidos *hot spots*, fazer as associações de cada *hot spot* com um determinado evento e acionar a geração de código. A Figura 3 mostra o momento em que o gerador de código requisita ao desenvolvedor o nome do nó (*Server Name)* do aglomerado gráfico responsável pelo tratamento dos eventos, o número da porta de comunicação (*Port*), o nome da aplicação

Java (*Java Code*) e o nome do arquivo que será gravado o trecho de código do nó do responsável pelo tratamento do evento (C *Code*).

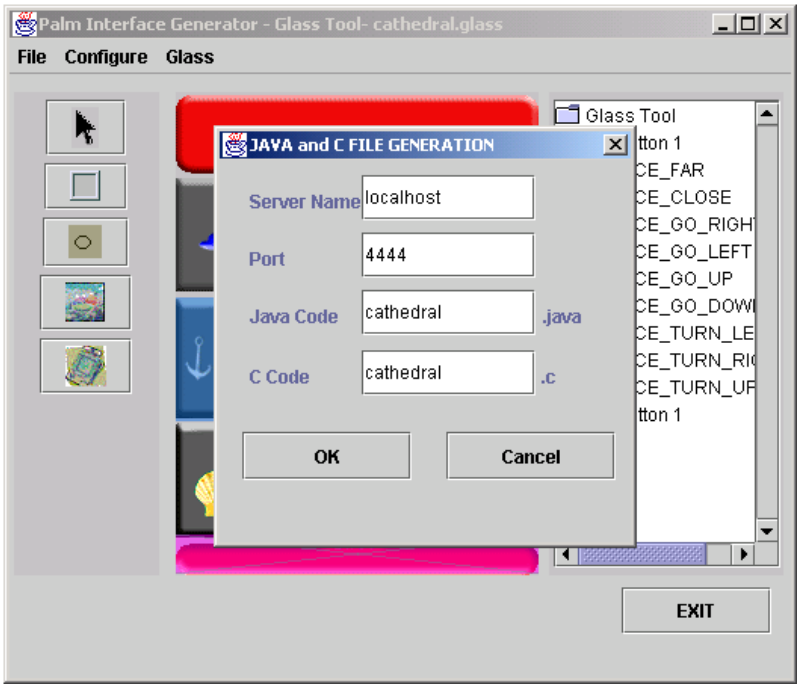


Figura 3 – Geração da aplicação para o PDA

A aplicação que é gerada automaticamente e que é executada no PDA, tem como responsabilidade a apresentação da interface gráfica, e a detecção das entradas de dados do usuário, seu processamento e envio para o nó do aglomerado gráfico responsável, de maneira completamente transparente. Atualmente, não é possível mudar as configurações da interface de forma dinâmica, como por exemplo, um *hot spot*. Mas, estão sendo desenvolvidos recursos para que as configurações (imagem, *hot spots*) possam ser recarregadas em tempo de execução.

A Figura 4 mostra um exemplo de código que foi gerado automaticamente para o PDA. Quando o construtor `Projeto` for invocado, este criará `(desenho.put("1",new Retangulo(4,56,114,87,1)))` as áreas de *hot spot*s, utilizando os parâmetros que foram obtido pela ferramenta de geração. Neste código é apresentado o método `mouseClicked`, que é responsável pela captura do evento do usuário e pela chamada do método de transmissão de evento, o `palm.Send(Integer.toString(tmp.val))`, o parâmetro passado é o valor correspondente da área pressionada (*hot spot*). Após gerado, este código deve ser compilado com a Máquina Virtual Java compatível e copiado para o PDA.

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
class Projeto extends Frame implements MouseListener{
      . . .
   public void mouseClicked(MouseEvent e) {
      int i;
      Forma tmp;
      for (i=1;i<=23;i++){
            tmp = (Forma)desenho.get(Integer.toString(i)); ;
            if (tmp.Selecionado(e.getX(), e.getY()))
               {palm.Send(Integer.toString(tmp.val));}
            }
       }
  public Projeto () {
      . . .
      desenho.put("1",new Retangulo(4,56,114,87,1));//direita
      desenho.put("2",new Retangulo(5,85,69,115,1));
      desenho.put("3",new Retangulo(5,219,35,292,2)); //esquerda
      desenho.put("4",new Retangulo(35,244,66,293,2));
                     . . .
      }
```

Figura 4 – Código que será executado no PDA

A Figura 5 mostra o trecho de código do nó responsável pelo tratamento dos eventos. Este código servirá como base inicial para a implementação total da rotina. Neste exemplo, quando o usuário clicar no *hot spot* denominado *direita*, o valor um (1) será transmitido para um nó, o qual chamará a rotina apropriada para este evento (`switch`).

```
enum event = {direita=1,
              esquerda=2
};

switch (event) {
      case direita:
               // a ser preenchido pelo desenvolvedor
             break;
      case esquerda:
               // a ser preenchido pelo desenvolvedor
             break;
}
```

Figura 5 – Trecho do código responsávelo pelo tratamento dos eventos

A aplicação gerada é responsável por toda a interface gráfica, e também pela detecção de entrada de dados pelo usuário, seu processamento e envio para o nós do aglomerado gráfico, de maneira completamente transparente.

4. Estudo de Caso

Para implementação e análise da pesquisa desse gerador, um aplicativo de visualização de arquiteturas foi desenvolvido e executado no Laboratório de Sistemas Integráveis da Universidade de São Paulo. O aglomerado gráfico utilizado é composto por 6 nós Dual-Pentium III Xeon 1GHz com 75GBytes de disco rígido e 1GByte de memória RAM, interligados por um sistema de interconexão de alta velocidade [SAN01, ZUF01].

A aplicação desenvolvida segue a arquitetura apresentada na Figura 6. Este modelo representa o usuário, que está dentro do CAVE, manipulando uma interface gráfica que está sendo apresentada por um PDA, neste caso um IPAQ H3970. Os eventos gerados pelo usuário na Interface gráfica são transmitidos para o Nó 4 do aglomerado gráfico, o qual trata as interações, gera as imagens e as envia para um dispositivo de multiprojeção, neste caso um CAVE.

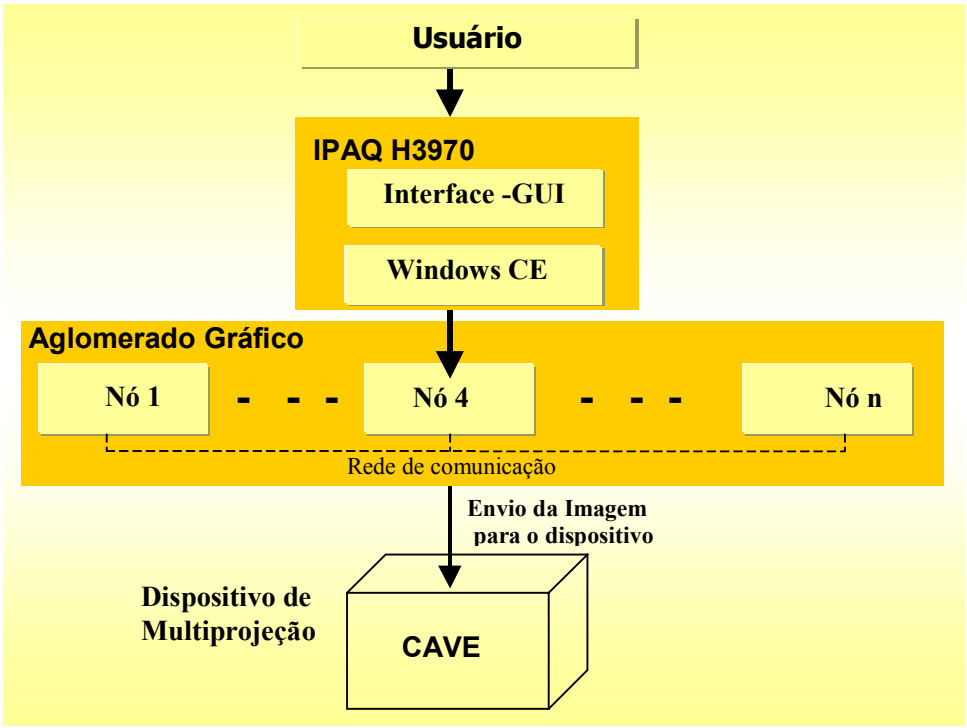


Figura 6 – Arquitetura das aplicações

A seguir, a Figura 7 mostra a aplicação no PDA Esta aplicação simula a visualização de uma catedral em um *CAVE*. A cada interação do usuário, os nós geram as imagens de acordo com o seu próprio ponto de vista. Para manter a coerência na geração das imagens, foi necessário criar dois pontos de sincronização, o de dados e o de *swaplock*. Desta maneira, cada tela apresenta um ponto de vista de forma sincronizada.

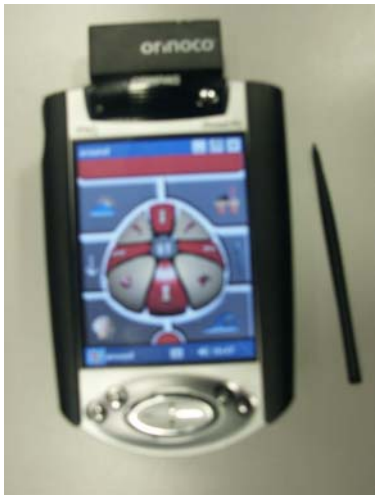


Figura 7 – Aplicação sendo executada no PDA

A Figura 8 mostra um usuário interagindo com a aplicação (Catedral), por intermédio de um PDA. O desempenho da aplicação foi satisfatório, atingindo interatividade, e sendo limitado pelo *hardware* gráfico.

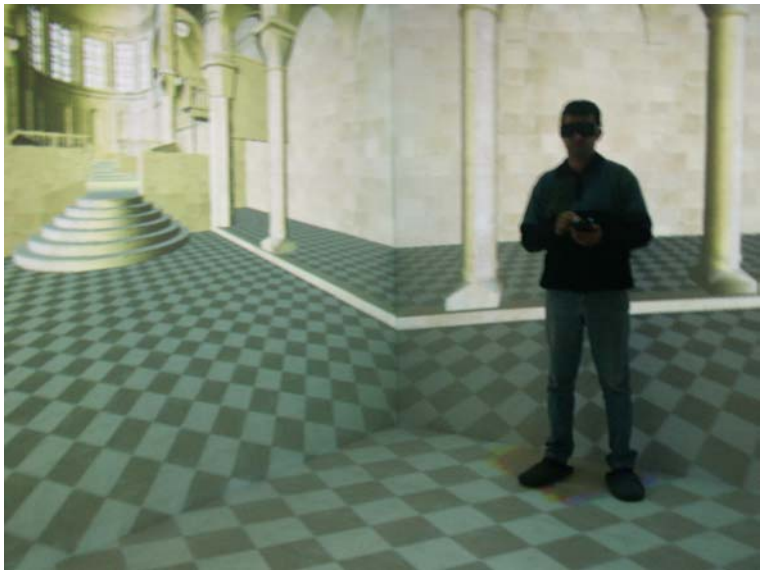


Figura 8 – Usuário manipulando aplicação usando o PDA – Catedral.

Modelo fornecido por Makro Dabrovic (mdabrov@rna.hr)

5. Conclusões e trabalhos futuros

O uso de PDAs não somente pode facilitar a interação de usuários com aplicativos imersivos, como pode ser o próprio controle do sistema de RV. Além disto, pode facilitar a interação com grande volume de dados e informações alfanuméricas. Este trabalho apresentou uma ferramenta automática que gera o código fonte da interface gráfica para o PDA, em Java, e o trecho de código para ser executado pelo nó do aglomerado gráfico responsável pelo tratamento dos eventos do PDA. O sistema

desenvolvido permite uma interação simples e intuitiva dos usuários com ambientes de RV.

Como trabalhos futuros, planeja-se estender a ferramenta para suportar interfaces gráficas com animação, e a mudança dinâmica da interface gráfica de maneira simples.

6. References

[1] PINHO, M. S. &KIRNER, C. **Uma Introdução à Realidade Virtual. Instituto de Informática** – PUCRS. 2001.

[2] **A Sony Ericsson apresenta cartão modem GPRS para PC** http://www.ericsson.pt/pressroom/other/pressreleases/20020305_0005_SonyEricsson_GC75.pdf.2002.

[3] ZYDA, M. **Networked Virtual Environments: design and Implementation**. SIGGRAPH series.1999.

[4] **The Official Bluetooth Web site**. http://www.bluetooth.com/. Consultado em 27/05/2003.

[5] HARTLING,P & BIERBAUM,A & CRUZ-NEIRA-C. **Tweek: Merging 2D and 3D Interaction in Immersive Environments**. 6th World Multiconference on Systemics, Cybernetics, and Informatics, Orlando, Florida, July 2002

[6] Hill,L. &. Cruz-Neira,C**. Palmtop interaction methods for immersive projection technology systems**. Fourth International Immersive Projection Technology Workshop (IPT 2000).2000.

[7] BENINI, L. & BONFIGLI, M.E. & CALORI, L. & FARELLA,E & RICCÒ, B. **Palmtop Computers for managing Interaction with Immersive Virtual Heritage.** in Proceedings of EUROMEDIA2002, pp. 183-189.

[8] **SGI Graphics Cluster™: The Cluster Architecture Challenges, the SGI™ Solution**. Visualization Solutions Development Group. White Paper. 2001. Disponível em http://www.sgi.com/products/legacy/vis_systems.html. Acessado em novembro de 2001.

[9] **OPENGL**. The Industry's Foundation for High Performance Graphics. Disponível em http://www.opengl.org/. Acesso em janeiro de 2003.

[10] **OPENGL Performer**. Disponível em http://www.sgi.com/software/performer/. Acesso em janeiro de 2003.