IX Symposium on Virtual and Augmented Reality - SVR 2007, p.197-203. Petrópolis, RJ, Maio 2007.

Uma Metodologia para Especificar Interação 3D utilizando Redes de Petri

Rafael Rieder*
FACIN, PPGCC/PUCRS
Av. Ipiranga, 6681, Partenon
Porto Alegre, RS, Brazil
rieder@inf.pucrs.br

Alberto B. Raposo†
TECGRAF, PUC-Rio
Rua Marquês de São Vicente, 225, Gávea
Rio de Janeiro, RJ, Brazil
abraposo@tecgraf.puc-rio.br

Márcio S. Pinho‡
FACIN, PPGCC/PUCRS
Av. Ipiranga, 6681 - Partenon
Porto Alegre, RS, Brazil
pinho@pucrs.br

ABSTRACT

This work presents a methodology to model and to build 3D interaction tasks in virtual environments using Petri nets, a technique-decomposition taxonomy and object-oriented concepts. Therefore, a set of classes and a graphics library are required to build an application and to control the net dataflow. Operations can be developed and represented as Petri Net nodes. These nodes, when linked, represent the interaction process stages. The integration of these approaches results in a modular application, based in the Petri Nets formalism that allows specifying an interaction task, and also to reuse developed blocks in new virtual environments projects.

Keywords

interaction tasks, Petri nets, specification

1. INTRODUÇÃO

O desenvolvimento de aplicações de Realidade Virtual (RV), em especial aquelas destinadas à pesquisa científica, ainda utilizam processos de modelagem e implementação pouco estruturados e formais, levando, na maioria dos casos, a reescrita de código e a dificuldade em obter uma análise da aplicação antes de seu desenvolvimento.

Para o entendimento de uma aplicação computacional, é útil usar alguma ferramenta de descrição formal que defina comportamentos, tais como Redes de Petri (RdP), Unified Modeling Language (UML) e Máquinas de Estado Finitos. Estas ferramentas permitem compreender e avaliar cada etapa de funcionamento do sistema, além de possibilitar a geração automática de código a partir de diagramas e a modelagem em diferentes níveis de abstração.

Smith [17] destaca que a ausência de uma descrição formal em ambientes virtuais (AVs) dificulta a avaliação de semelhanças entre técnicas de interação (TIs) diferentes, o que acaba levando a "reinvenção" de técnicas. Além disso, conforme aborda Navarre [9], descrições informais facilitam ambigüidades nas implementações.

Formalismos já têm sido usados para modelar TIs e tarefas de interação [9]. Hynet [20], ICO [12] e Flownet [21] são alguns exemplos de formalismos existentes, baseados em RdP. O uso destes formalismos ajuda, por exemplo, na detecção de falhas em sistemas ainda em tempo de projeto.

Além de formalismos, pesquisadores procuram desenvolver taxonomias capazes de documentar e especificar AVs num nível de detalhe mais próximo da concepção do usuário. Conforme Lindeman [6], a RV ainda é uma tecnologia que está em fase de definição, e necessita ser classificada e categorizada. De acordo com Bowman [1], esta classificação e categorização permitiriam entender o conjunto de técnicas utilizadas no desenvolvimento de AVs. Uma vez identificadas, poderiam também ser usadas na organização do projeto interativo.

Os trabalhos que apresentam taxonomias procuram identificar as etapas do processo interativo [1], classificar TIs [2] [6] [15] e organizar o controle de sistema [3]. Estas abordagens provêm a separação de contextos, dividindo o sistema em partes menores (componentes), responsáveis por encapsular uma determinada funcionalidade. Isto permite que componentes sejam reutilizados por novos projetos, possibilitando a combinação destes com o objetivo de criar novas TIs, por exemplo.

Tanto o emprego de formalismos, como de taxonomias, visam otimizar o tempo de projeto e desenvolvimento de AVs. Uma proposta de integração destas abordagens pode permitir a especificação de sistemas de acordo com o nível de conhecimento do usuário, além de permitir o detalhamento de cada

*Rafael Rieder é aluno de pós-graduação da Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS.

†Alberto Barbosa Raposo é professor e pesquisador da Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio.

‡Márcio Serolli Pinho é professor e pesquisador da Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS.

etapa do processo de desenvolvimento de software.

Baseado nestes trabalhos anteriores, nossa proposta descreve uma metodologia capaz de modelar e implementar componentes que representem as etapas do processo interativo. A união destes componentes permite a descrição de uma tarefa de interação, onde a seqüência de passos a serem desenvolvidas pelo usuário é controlada por uma RdP.

Nossa metodologia integra três abordagens de modelagem: o formalismo de RdP, a taxonomia de decomposição proposta por Bowman [1] e conceitos de orientação a objeto. RdP são utilizadas para representar graficamente o comportamento de um AV, com base na divisão das tarefas do processo interativo da taxonomia de decomposição de Bowman. O uso destas abordagens gera um modelo que pode ser codificado com auxílio de um conjunto de classes em C++, o qual define a estrutura e o funcionamento do sistema.

A escolha de RdP para especificação de tarefas em AVs surge naturalmente quando se utiliza uma taxonomia como a de Bowman, pois as tarefas de interação podem ser interpretadas como transições, enquanto os estados assumidos pela aplicação podem ser modelados como lugares na RdP. A partir dessa definição, é possível definir componentes independentes capazes de representar determinadas funcionalidades. Esta separação em unidades é importante para, por exemplo, desenvolver um framework de TIs, ou permitir a geração automática de código, a partir do modelo de especificação já testado e validado.

Este documento encontra-se assim organizado: primeiramente, uma revisão de trabalhos é mostrada na Seção 2. A Seção 3 apresenta a definição da metodologia proposta, baseada no formalismo de RdP e no processo interativo de Bowman. Já a Seção 4 aborda a aplicação desta metodologia, demonstrando os passos necessários para modelagem e a geração de código que implementa o AV. A Seção 5 apresenta testes que validam a especificação, enquanto a Seção 6 mostra a possibilidade de hierarquização dos modelos. A Seção 7 apresenta as vantagens desta abordagem, bem como os objetivos que se pretende alcançar com trabalhos futuros.

2. TRABALHOS RELACIONADOS

Diferentes mecanismos têm sido propostos pela comunidade de RV para descrever e implementar tarefas de interação, buscando compreender a dinâmica das aplicações e possibilitar a padronização de funcionalidades.

HyNet [20] é uma metodologia de especificação de TIs que integra três abordagens de modelagem bem fundamentadas na literatura. RdP de alto nível representam a base formal da especificação, definindo a semântica e permitindo a representação gráfica dos eventos discretos da aplicação. Equações diferenciais permitem a descrição do comportamento contínuo do sistema, enquanto que os conceitos de orientação a objeto permitem aumentar o poder de expressividade da metodologia, proporcionando modelos sucintos e compactos.

Baseado na metodologia HyNet, a Flownet [21], desenvolvida para a descrição de comportamentos dinâmicos de AVs, apresenta como diferencial uma notação gráfica que permite a especificação de TIs e do comportamento dos objetos do AV. Uma ferramenta também é oferecida para o processo de análise de modelos, visando auxiliar o desenvolvedor na compreensão dos resultados.

O formalismo ICO, por sua vez, consiste em uma técnica de descrição formal que permite a modelagem e implementação de sistemas interativos [12]. Esta técnica faz uso de orientação a objetos e RdP para descrever aspectos estruturais e comportamentais de sistemas, permitindo a prototipação de interfaces gráficas.

O projeto e desenvolvimento de AVs também pode ser analisado sob o ponto de vista das TIs utilizadas, buscando classifica-las com o objetivo de melhor entender seus componentes e, conseqüentemente, as possibilidades de reuso de código gerado para novas aplicações.

Lindeman [6], por exemplo, apresenta uma taxonomia que decompõe TIs de acordo com a forma de manipulação envolvida (direta ou indireta), as ações discretas e contínuas do sistema, e os graus de liberdade oferecidos. Tal abordagem procura auxiliar o projetista na identificação dos parâmetros existentes em cada TI, possibilitando a construção de novas formas de interação.

Bowman [2] apresenta duas taxonomias complementares para a classificação de TIs. A primeira, baseada em metáforas, tem por objetivo facilitar a compreensão das técnicas empregadas num AV por meio da representação de alguma ação ou situação do mundo real que sirva como referência ao usuário no momento da interação virtual. Já a segunda, baseada na decomposição de tarefas, tem por objetivo a análise do processo interativo. Conforme Bowman [1], a separação de tarefas em simples componentes permite que cada um destes seja analisado e testado de maneira independente, como forma de avaliar a usabilidade e efetividade de uma TI em um determinado AV.

Frameworks de RV também procuram separar funcionalidades em componentes, visando abstrair a complexidade de determinadas ações do sistema, além de oferecer o reuso de componentes em diferentes projetos.

Figueroa [4] propõe uma arquitetura de desenvolvimento de TIs, baseada em *pipes* e filtros, no qual fontes de informação (como, por exemplo, dispositivos) geram um fluxo de dados que são propagados entre filtros interconectados. Em seu trabalho, a linguagem de marcação InTML, baseada no X3D [23], foi desenvolvida para servir de *front-end* às bibliotecas de RV. Desta forma, TIs podem ser construídas e tratadas como componentes externos, independentes da aplicação. Isto permite que técnicas possam ser integradas dentro de uma tarefa, criando novas formas de interação. Além disso, facilita o reuso de código e abstrai a complexidade de um AV.

Outros *frameworks*, como o Unit [10], usam o conceito de unidades para representar nodos em um fluxo de dados. Cada unidade pode apresentar diferentes propriedades e um número qualquer de conexões entre outras unidades. Unit também organiza TIs em uma camada de abstração entre os dispositivos de entrada e a aplicação, como a InTML. No

entanto, ela possibilita que TIs sejam modificadas ou trocadas em tempo de execução (desde que especificadas em projeto).

Wingrave [22] propõe uma arquitetura baseada em máquinas de estado hierárquicas que permitem a comunicação entre o projetista e o programador de uma interface 3D, facilitando o reuso e o gerenciamento de código.

Em uma perspectiva diferente dos trabalhos acima citados, Ying [25] busca entender o processo interativo através da análise de código de aplicações de RV já implementadas. Usando os conceitos de engenharia reversa, a partir do código de um AV são extraídas informações relacionadas à interação do usuário. Tais informações são organizadas e armazenadas em um arquivo XML que serve de base para a geração de um modelo de RdP. Conforme MacWilliams [7], estas representações gráficas são úteis para o *debugging* da aplicação. O avaliador pode ter uma visão do processo interativo em execução na rede, enquanto o usuário interage normalmente com a aplicação.

Pela análise dos trabalhos citados, nota-se que as abordagens têm vantagens e objetivos particulares, mas, em geral, não abordam todo o ciclo de desenvolvimento de aplicações em computador. Neste sentido, este projeto desenvolveu uma metodologia que busca organizar o processo de produção de um software de RV do projeto à implementação, tendo como base uma taxonomia que estrutura a interação em AVs, permitindo a modelagem hierárquica do projeto e a geração de código baseado na programação orientada a objetos.

3. BASE DA METODOLOGIA

Para a definição desta metodologia, optou-se pelo emprego de RdP por esta apresentar um formato de especificação formal com diferentes formas de extensão, bastante fundamentado e consolidado pela comunidade científica [24]. Conforme Murata [8], uma RdP é uma ferramenta de modelagem gráfica e matemática para especificação e análise de sistemas concorrentes e dinâmicos, onde aplicações de RV se enquadram.

Além disso, o poder de expressão de seu formalismo, juntamente com as ferramentas de análise e simulação existentes, permitem que a estrutura e o comportamento de um sistema sejam previstos e testados antes da fase de implementação.

A representação gráfica adotada para a metodologia baseia-se no uso de Redes de Petri Coloridas (RdP-C) e Redes de Petri Hierárquicas (RdP-H), como forma de distinguir os diferentes tipos de dados a serem manipulados por uma aplicação e permitir a hierarquização dos modelos. Ambas abordagens são uma extensão das RdP que têm por objetivo reduzir o tamanho dos modelos. Por convenção, este trabalho irá referir-se à RdP-C pelo simples termo de "RdP", dado que AVs sempre operam com tipos de dados distintos, e seus modelos são representados como tal.

A proposta também procura aproximar a concepção do usuário ao trabalho do projetista e do desenvolvedor, modelando a aplicação sob a perspectiva das tarefas que o usuário deve desempenhar no AV. Estas tarefas podem ser agrupadas em "tarefas-base" que, quando organizadas por características similares, procuram definir as etapas do processo interativo. Conforme já mencionado, Bowman e Hodges [1] propõem uma taxonomia para técnicas de seleção e manipulação que prevê a divisão deste processo em três tarefas elementares, seguindo esta ordem: **seleção**, **manipulação** e **liberação**.

Este trabalho procura adaptar essa taxonomia, separando a tarefa de seleção em duas etapas distintas. A primeira, denominada **seleção**, é responsável pelo processo de indicação do objeto que se deseja manipular. Já a segunda, denominada **anexação**, trata do processo de confirmação da seleção. Ambas apresentam *feedbacks* que informam o usuário sobre sua execução. As tarefas de manipulação (posicionamento e orientação) e liberação permanecem inalteradas, seguindo a concepção original de Bowman e Hodges [1].

Com base nessas definições, a metodologia de modelagem desenvolvida neste trabalho prevê que cada elemento de uma RdP (**lugares**, **transições**, **arcos** e **marcas**) represente um determinado papel durante o processo interativo de uma aplicação de RV.

Lugares são elementos que **determinam o estado atual** da interação. Eles dispõem de canais de comunicação, capazes apenas de receber e transmitir informações necessárias para o funcionamento da rede, não produzindo dados.

Para que uma tarefa de seleção esteja disponível ao usuário, por exemplo, é necessário que o **lugar** que representa o estado de seleção contenha dados que indiquem, ao menos, a forma de apontamento, os objetos disponíveis para seleção e que nenhum objeto está sendo manipulado neste momento. A Figura 1 representa esta situação usando um **lugar** e uma **transição**.

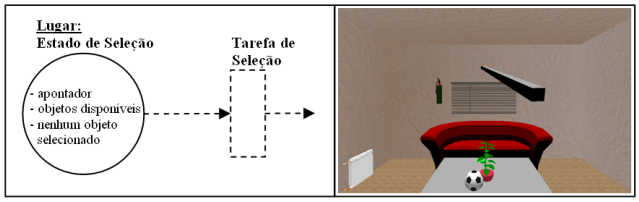

Figura 1: O *Estado de Seleção* fornece as informações necessárias que possibilitam ao usuário executar a *Tarefa de Seleção*.

Transições são elementos que **realizam o processamento**, alterando o comportamento da aplicação. Elas representam, em alto nível, as tarefas elementares de interação. Assim como os **lugares**, as **transições** dispõem de canais de comunicação para a recepção e transmissão de informações pela rede. No entanto, elas podem gerar novos dados, inserindo-os na rede.

Um exemplo de seu funcionamento pode ser representado pelo exato momento em que o usuário confirma a seleção de um objeto. A **transição** responsável por esta tarefa é então disparada, executando a operação que anexa o objeto ao apontador do usuário. A Figura 2 representa este exemplo.

Arcos determinam a **ordem de execução da rede**. Eles são responsáveis pelo transporte de dados entre um **lugar** e

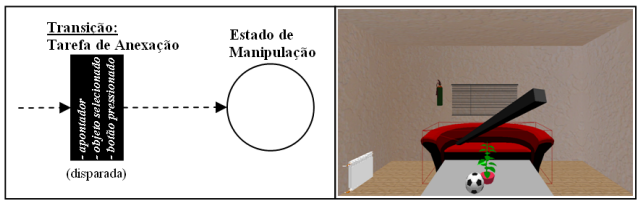

Figura 2: Exemplo de transição que executa a anexação de um objeto selecionado para um apontador. Após esta tarefa, a aplicação passa para o *Estado de Manipulação*.

uma **transição** (e vice-versa), indicando pré e pós-condições para a execução das **transições** ou para o estabelecimento de um estado. Conseqüentemente, os **arcos** definem a ordem de execução das tarefas no AV.

Para manipular um objeto, por exemplo, é necessário antes executar tarefas de seleção e anexação que fornecem informação sobre o objeto escolhido pelo usuário. A Figura 3 apresenta esta seqüência, rotulando os **arcos** com os dados necessários para a realização de cada tarefa.

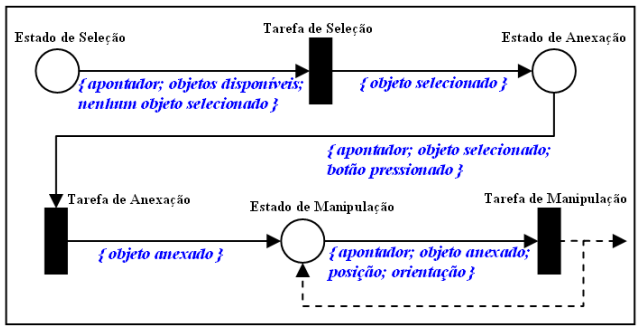

Figura 3: Arcos entre lugares e transições definindo a ordem de execução e os recursos de cada etapa do processo interativo.

Marcas representam os **recursos disponíveis** para o funcionamento da aplicação, bem como da RdP. Dados como objetos geométricos, menus, vetores de posicionamento e clique de botões são exemplos de possíveis **marcas**. Tais recursos são provenientes da própria aplicação ou de dispositivos de entrada (como *mouse*, teclado ou rastreador), podendo ser armazenados em **lugares** da rede e terem seus conteúdos atualizados por **transições**. Nesta metodologia, as **marcas** distinguem-se umas das outras pelo tipo de dado que elas encapsulam. A Figura 4 apresenta um exemplo onde os recursos de uma aplicação são representados na RdP na forma de ícones.

Figura 4: Objetos e sua representação como marcas numa RdP.

Por ser baseado em RdP, a metodologia também exige a definição de **marcas iniciais**. Sob o ponto de vista de uma aplicação, estas **marcas** referem-se à **configuração inicial do sistema**. Dados vindos dos dispositivos de entrada e a lista dos objetos geométricos que compõem um AV (fornecida pela aplicação) podem ser considerados exemplos de **marcas iniciais**, e precisam ser fornecidas para algum **lugar** da RdP. Seguindo o formalismo, deve se especificar ao menos um **lugar** que guarde estas **marcas**.

De acordo com as regras de formação das RdP, cabe lembrar que um nodo (**lugar** ou **transição**) não pode ser ligado diretamente a outro nodo de mesmo tipo. Portanto, sempre entre dois **lugares** haverá uma **transição**, da mesma forma que entre duas **transições** haverá um **lugar**.

4. APLICANDO A METODOLOGIA

4.1 Plataforma de Teste

Para ilustrar o uso desta metodologia na especificação do processo interativo em AVs, foi utilizada uma aplicação de Quebra-Cabeça Virtual [16], onde o objetivo principal do usuário é escolher e encaixar corretamente cada peça do jogo, realizando para tal tarefas de seleção e manipulação de objetos.

A Figura 5 apresenta as características do cenário virtual. Inicialmente, as peças do quebra-cabeça localizam-se misturadas ao lado direito da área de visualização, e a caixa para montagem localiza-se à esquerda. A ordem de escolha dos blocos não é um quesito levado em consideração pela aplicação. Para interagir na aplicação, usa-se a técnica de mão virtual.

Esta aplicação foi construída utilizando a linguagem C++, a OpenGL, a GLUT e a biblioteca gráfica SmallVR [13] [14], uma biblioteca que facilita o desenvolvimento de aplicações de RV, abstraindo diferentes aspectos de implementação como o controle de dispositivos e o gerenciamento de um grafo de cena, não perdendo a estrutura básica de um programa com a GLUT. Ela traz suporte a dispositivos convencionais, como teclado, *mouse* e monitor, e não-convencionais, como rastreadores e óculos de RV.

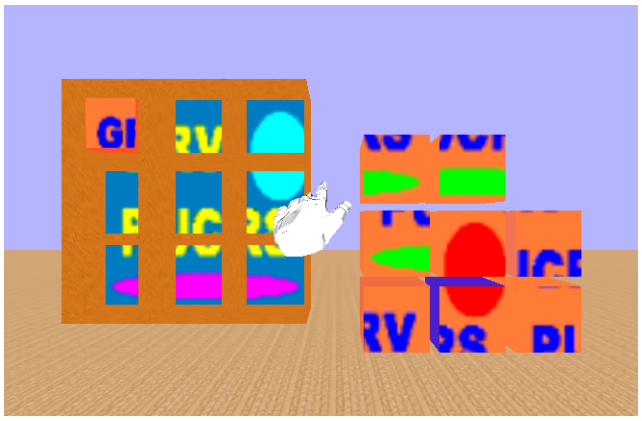

Figura 5: Quebra-Cabeça Virtual usando a técnica de mão virtual.

Para o emprego da metodologia, três passos devem ser seguidos pelo projetista da aplicação, quais sejam:

- Identificar as tarefas do AV, de acordo com a taxonomia de decomposição de Bowman, bem como os estados assumidos após a execução de cada tarefa;
- Definir uma RdP com tarefas e estados identificados pelo passo anterior;
- Implementar o modelo, utilizando um conjunto de classes especialmente desenvolvido para construir a RdP e controlar sua execução.

4.2 Identificando as Etapas do Processo Interativo

A aplicação inicia no *Estado de Seleção* (Figura 6), no qual o usuário move um apontador (mão virtual) no ambiente, procurando por um objeto para seleção. A partir deste estado, a *Tarefa de Seleção* realiza um teste de colisão entre o apontador e as peças do quebra-cabeça. Se a colisão existir (um objeto sendo apontado), o *Estado de Anexação* é habilitado.

A partir deste ponto, se o usuário mantiver pressionado o botão de seleção, a *Tarefa de Anexação* é disparada. Esta tarefa "cola" o objeto ao apontador, estabelecendo o *Estado de Manipulação*.

Uma vez estabelecido este estado, a *Tarefa de Manipulação* é disparada, permitindo que o usuário reposicione o objeto usando a mão virtual. A localização deste objeto tem por base a posição do apontador onde o mesmo está anexado. Para simplificar a especificação, algumas operações de *feedback* normalmente fornecidas ao usuário foram omitidas.

Enquanto a peça está sendo movimentada, uma ação de "soltar" o objeto pode ser realizada. Se o botão de seleção for solto, o *Estado de Liberação* é habilitado, o qual imediatamente dispara a *Tarefa de Liberação*, que separa o objeto do apontador.

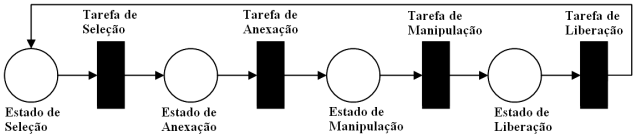

Figura 6: RdP em alto nível representando as quatro tarefas básicas de interação.

4.3 Construindo o Modelo de RdP

Depois de identificadas as tarefas da aplicação em um alto nível de abstração, procede-se o mapeamento que divide as tarefas em partes menores, baseando-se nas operações que cada uma desempenha [1]. A Tabela 1 apresenta a decomposição das tarefas elementares, enquanto que a Figura 7 mostra a nova configuração da RdP.

Seguindo a metodologia, passa-se a identificar os recursos necessários (dados) para cada etapa do processo interativo,

Tabela 1: Detalhando as quatro tarefas básicas de interação.

Tarefas de Alto Nível	Operações Básicas
Tarefa de Seleção	- Subtarefa de Indicação - Subtarefa de *Feedback* de Indicação
Tarefa de Anexação	- Subtarefa de Confirmação - Subtarefa de *Feedback* de Confirmação
Tarefa de Manipulação	- Subtarefa de Posicionamento
Tarefa de Liberação	- Subtarefa de Separação - Subtarefa de *Feedback* de Separação

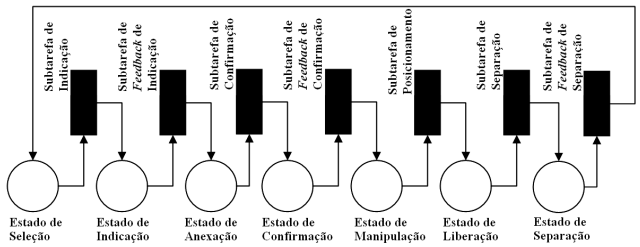

Figura 7: RdP representando o detalhamento do processo interativo da aplicação de Quebra-Cabeça Virtual.

representando-os na RdP como **marcas**. Para tanto, os **arcos** da rede devem ser rotulados com estas **marcas** que, neste caso, são representadas por ícones.

Os **lugares** *Estado de Seleção*, *Estado de Anexação*, *Estado de Manipulação* e *Estado de Liberação* necessitam constantemente de informações sobre o estado dos dispositivos e de variáveis de controle da aplicação. Para tanto, **marcas** referentes a estes dados são depositadas nestes **lugares**.

O modelo completo da RdP que representa o processo interativo do Quebra-Cabeça Virtual é apresentado pela Figura 8. Os dispositivos são representados por um triângulo, enquanto a aplicação é representada por um hexágono. Estas formas são meramente ilustrativas e servem apenas para compreensão do funcionamento da rede.

O passo de simulação da rede é controlado pela aplicação, que testa cada **transição** toda vez que a visão do usuário precisa ser alterada. Em outras palavras, todas as **transições** são verificadas a cada ciclo de *rendering*, disparando as **transições** de acordo com a existência de pré-condições (**marcas** necessárias em cada **lugar**).

Um ciclo completo de execução da RdP pode ser analisado e interpretado da seguinte forma: inicialmente, a aplicação e o dispositivo enviam **marcas** para os **lugares** *Estado de Seleção*, *Estado de Anexação*, *Estado de Manipulação* e *Estado de Liberação*. Como a **transição** *Subtarefa de Indicação* recebe todas as **marcas** necessárias do *Estado de*

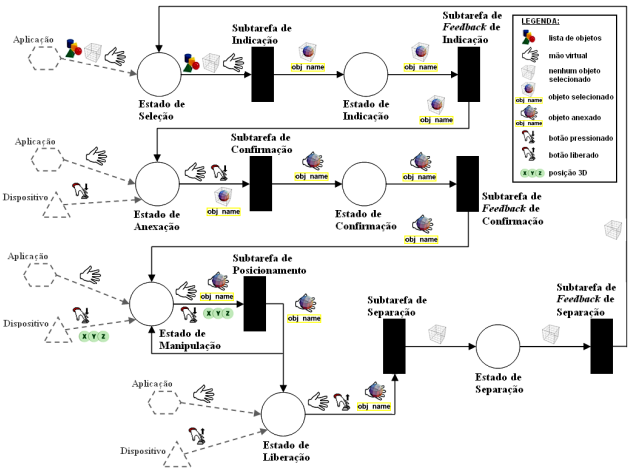

Figura 8: RdP com as marcas necessárias para habilitar as transições que representam o processo interativo do Quebra-Cabeça Virtual.

Seleção, esta é imediatamente disparada, coletando seus dados (a lista de objetos do cenário, o apontador e uma variável indicando que não existe objeto selecionado). Esta **transição** realiza um teste de colisão entre o apontador do usuário e os objetos do cenário. Existindo colisão com algum objeto, uma nova **marca** é gerada, representando o objeto em intersecção. Transcorrida esta etapa, a aplicação passa para o *Estado de Indicação*. Uma vez estabelecido, este estado dispara a **transição** *Subtarefa de Feedback de Indicação*, responsável por aplicar um realce sobre este objeto, de maneira a destacá-lo dos demais objetos do AV.

Em seguida, o **lugar** Estado de Anexação recebe a **marca** que representa o objeto selecionado. Neste estado, ele já dispõe das **marcas** referentes ao apontador utilizado e de um possível botão pressionado pelo usuário. Quando todos estes dados estiverem presentes neste estado, a **transição** *Subtarefa de Confirmação* é disparada, "colando" o objeto selecionado no apontador.

A **marca** que representa o objeto selecionado é transmitida para o *Estado de Confirmação*. Nestas condições, a *Subtarefa de Feedback de Confirmação* é disparada, a qual emite um sinal sonoro comunicando o sucesso no processo de anexação. Após isto, uma **marca** que encapsula o nome do objeto selecionado é transmitida para o *Estado de Manipulação*, o qual define o início do processo de manipulação, onde o usuário procura posicionar corretamente o objeto na caixa de montagem. Este estado sempre dispõe das **marcas** que representam o apontador (recebida da aplicação), os dados que chegam do dispositivo de rastreamento e o botão que está sendo pressionado pelo usuário. Com a chegada da **marca** do objeto selecionado, a **transição** de *Subtarefa de Posicionamento* é disparada. Esta **transição** atualiza a posição do objeto, baseada nos dados do rastreador, gerando e enviando uma nova **marca** do objeto selecionado para os *Estados de Liberação e Manipulação*. E, enquanto o botão de seleção estiver sendo pressionado, a **transição** *Subtarefa de Posicionamento* é repetidamente disparada, permitindo ao usuário reposicionar o objeto tantas vezes quanto necessárias.

Se o botão for liberado, o *Estado de Manipulação* não oferecerá mais as pré-condições necessárias para a *Subtarefa de Posicionamento* ser acionada. Ao mesmo tempo, o *Estado de Liberação* recebe uma **marca** informando que o usuário soltou o botão, além das **marcas** que representam o objeto selecionado e o apontador utilizado. Nestas circunstâncias, a **transição** *Subtarefa de Separação* é disparada. Esta **transição** "solta" o objeto no AV em sua nova posição. Seqüencialmente, o *Estado de Separação* recebe uma **marca** que dispara a **transição** *Subtarefa de Feedback de Separação*, que emite um sinal sonoro comunicando que o processo de liberação foi realizado com sucesso. Uma **marca** é então repassada para o *Estado de Seleção*, designando que uma nova seleção já pode ser executada.

4.4 Fase de Implementação do Modelo

Uma vez concluída a modelagem, é possível iniciar o procedimento de implementação. Com o objetivo de automatizar o processo de geração de código, foi desenvolvido um conjunto de classes na linguagem C++, descrito pelo diagrama UML apresentado na Figura 9. Estas classes representam os elementos da RdP e têm como base o mecanismo de *signals* e *slots* [18] para a comunicação entre **lugares** e **transições**.

Conforme Trolltech [18], *signals* são notificações (mensagens) de um objeto para outro que sinalizam a ocorrência de um determinado evento, enquanto que *slots* são respostas do objeto receptor com relação a estas mensagens. Sob o paradigma de orientação a objeto, esta resposta é um método a ser chamado e executado. Neste projeto, este mecanismo foi implementado usando a *classe QObject* do *toolkit* Qt [18]. Com este recurso, **lugares** e **transições** podem ser implementados como objetos interconectados, permitindo a passagem das **marcas** durante a simulação da RdP.

As classes que definem **lugares**, **arcos** e **marcas** podem ser diretamente usadas para instanciar objetos. Já **transições** devem ser implementadas como novas classes derivadas da classe abstrata que representa uma **transição** genérica. Esta abordagem força o desenvolvedor a codificar alguns métodos essenciais para a simulação do modelo, bem como permite o reuso de classes já desenvolvidas e utilizadas em projetos anteriores.

Para começar a implementação do modelo, inicialmente, o projetista deve instanciar um objeto da *classe PetriNet* (veja Figura 10, linha 01). Este objeto representa a **Rede** como um todo, e é usado para iniciar a simulação da RdP e armazenar **lugares** e **transições**. Além disso, dispõe de métodos que encapsulam e restauram os conteúdos das **marcas**, e realizam a conexão entre os nodos da RdP.

Depois de instanciado o objeto que representa a **Rede**, criam-se os **lugares** e **transições** (veja Figura 10, linhas 02 e 03), além das **marcas** que circularão pela rede (veja Figura 10, linha 04). **Lugares** devem ser instanciados a partir da *classe Place*, enquanto que **transições** são instanciadas a partir de classes derivadas da *classe Transition* (por exemplo, a *classe Indication* da Figura 10), e **marcas** a partir da *classe Token*. Posteriormente, **lugares** e **transições** são adicionados a lista do objeto **Rede** (linhas 05 e 06).

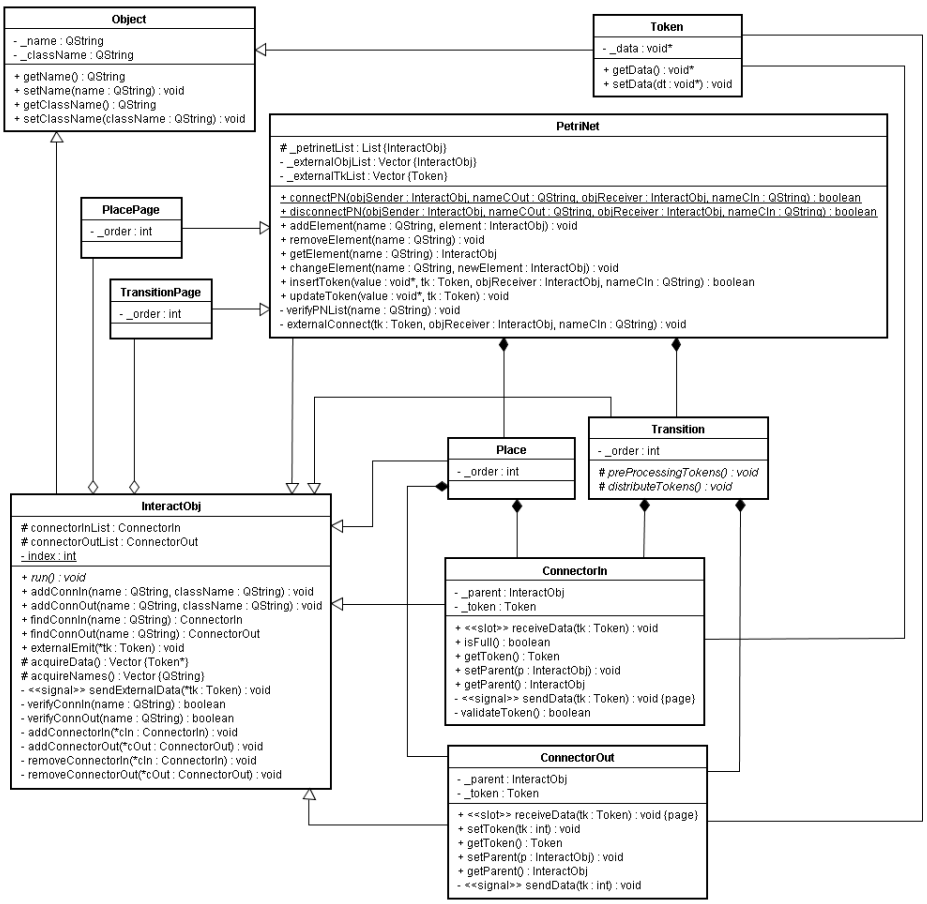

Figura 9: Conjunto de classes usado para implementar o modelo de RdP.

Conforme visto na Seção 3, **lugares** e **transições** trocam **marcas** através de canais de comunicação. Para estabelecer uma ligação entre estes canais, conectores devem ser criados e adicionados aos **lugares** e **transições**. Cada conector é identificado por um nome e do tipo de dado que irá suportar (linhas 07 e 08). Após isto, para estabelecer a comunicação entre dois objetos, o projetista deve explicitamente conectá-los, definindo o objeto emissor e seu conector, bem como o objeto receptor e respectivo conector (linha 09).

No caso específico das **transições**, é preciso instanciar objetos de classes derivadas da *classe Transition* que representem as tarefas do processo interativo simbolizadas no modelo. Para o exemplo apresentado pela Figura 7, foram identificadas e criadas as seguintes classes, bem como definidas as suas funcionalidades:

- **Indication**: detecta a colisão entre objetos do AV com o apontador do usuário;
- **Indication_Feedback**: aplica um realce no objeto selecionado;
- **Confirmation**: anexa o objeto selecionado ao apontador;
- **Confirmation_Feedback**: notifica a anexação;
- **Positioning**: atualiza a posição do objeto em manipulação;
- **Detachment**: libera o objeto do apontador;
- **Detachment_Feedback**: notifica a liberação.

De forma semelhante aos filtros definidos por Figueroa [4], a interface da classe abstrata, que dá origem as classes derivadas de **transições**, obriga o desenvolvedor a implementar três métodos virtuais responsáveis pela coleta e distribuição de **marcas** e pelo processamento dos dados dentro das **transições**.

O método *preProcessingTokens* recebe marcas, efetua o desempacotamento destas, e converte-as para tipos de dados aceitos pela aplicação. Já o método *distributeTokens* realiza o empacotamento dos dados da aplicação para dentro de uma marca, devolvendo-os para a RdP. Finalmente, o método *run* permite ao desenvolvedor inserir o código específico da aplicação, realizando o que for necessário para o funcionamento do sistema.

A conexão entre a RdP e os elemento externos, como dados vindos dos dispositivos ou dados específicos da aplicação (como lista de objetos, apontadores, menus, etc) pode ser feita através do método *insertToken*, disponível na *classe*

Place (Figura 10, linha 10). Esta função-membro permite que **marcas** sejam adicionadas e atualizadas em **lugares**, tanto na criação da RdP, quanto durante sua simulação.

Após todas essas etapas, a execução da RdP pode ser iniciada através da chamada do método *run* do objeto da classe *PetriNet* (Figura 10, linha 11). Este procedimento executa os métodos *run* de todos os objetos adicionados à sua lista. Para garantir a sincronização entre a função de desenho da aplicação e a simulação da RdP, o desenvolvedor precisa evocar o método *run* da **Rede** no início de cada ciclo de *rendering*.

Line	Code
01	`PetriNet *pn = new PetriNet();`
02	`Place *pSel = new Place();`
03	`Indication *tInd  = new Indication();`
04	`Token *tkObjs  = new Token();`
05	`pn->addElement("Estado de Seleção", pSel);`
06	`pn->addElement("Subtarefa de Indicação", tInd);`
07	`pSel->addConnOut("objs", "vector");`
08	`tInd->addConnIn("objs", "vector");`
09	`pn->connectPN(pSel, "objs", tInd, "objs");`
10	`pn->insertToken(vecObjs, tkObjs, pSel, "objs");`
11	`pn->run();`

Figura 10: Um simples exemplo de código gerado na fase de implementação.

Desta forma, a fase de implementação pode ser resumida nesta seqüência de passos:

- Derivar novas classes da *classe base Transition* para representar tarefas do AV;
- Instanciar o objeto que representa toda a RdP;
- Instanciar os objetos correspondentes aos **lugares**, **transições** e **marcas**;
- Adicionar estes objetos à **Rede**;
- Adicionar conectores para **lugares** e **transições**;
- Realizar a ligação entre **lugares** e **transições**;
- Definir o valor das **marcas** iniciais;
- Definir pontos onde a aplicação atualizará a RdP;
- Executar a RdP.

5. TESTES DO MODELO

Depois de modelado e implementado a aplicação de Quebra-Cabeça Virtual utilizando a metodologia proposta, alguns testes foram realizados para validar o comportamento do modelo e da aplicação em situações críticas como dados inválidos, errados ou redundantes.

O modelo previamente apresentado descreve situações válidas que podem ocorrer durante o processo interativo. Com dados válidos, a aplicação foi testada por diferentes usuários, sendo que a RdP executou apropriadamente.

No entanto, algumas situações inválidas podem ocorrer, impedindo a execução da rede e, conseqüentemente, bloqueando a interação no AV. Para tanto, é importante verificar se o modelo comporta-se adequadamente nestes casos, não atuando de maneira inesperada.

O primeiro teste procurou simular um AV vazio, sem objetos para seleção. Desta forma, quando o usuário pressionava o botão de seleção para indicar um objeto, nada poderia acontecer. Do ponto de vista do modelo, esta situação pode ser modelada pela exclusão da **marca** *Lista de Objetos* da RdP. Com esta configuração, a RdP continuou executando normalmente, porém, seu novo comportamento impediu que as tarefas de seleção e manipulação fossem executadas quando o usuário pressionava o botão, conforme esperado.

Em uma segunda simulação, optou-se pela exclusão do objeto apontador, representado na RdP pela ausência da **marca** *Mão Virtual*. Assim como na situação anterior, a rede continuou executando normalmente, porém impedindo que o usuário selecionasse qualquer objeto, novamente conforme esperado.

No terceiro teste, inseriu-se **marcas** que não pertenciam ao conjunto padrão utilizado, com o intuito de verificar se estas influenciariam ou não no comportamento da RdP. Para tanto, dados vindos do rastreador referentes à orientação do objeto foram introduzidos no **lugar** *Estado de Manipulação*. Conforme esperado, estes dados foram descartados quando a **transição** *Subtarefa de Posicionamento* era disparada.

O objetivo do último teste era simular uma situação onde não houvesse comunicação entre o AV e a RdP. Como conseqüência, o conteúdo das **marcas** da RdP eram gerados sem valores, o que impedia a execução da aplicação.

6. MODELAGEM HIERÁRQUICA

A modelagem hierárquica de uma aplicação usando RdP permite descrever o sistema em diferentes níveis de abstração, simplificando a representação e oferecendo diferentes visões do mesmo sistema. Isto facilita a compreensão da aplicação por usuários com distintos níveis de conhecimento. Além disso, a existência de níveis de hierarquia pode ser útil para tornar claro o funcionamento do modelo como, por exemplo, situações onde um conjunto complexo de operações necessite de uma representação simplificada, em um único módulo (uma TI, por exemplo).

A título de exemplo, imagine que seja interessante agrupar em uma única **transição** todo o processo de seleção. Desta forma, as **transições** *Subtarefa de Indicação*, *Subtarefa de Feedback de Indicação*, *Subtarefa de Confirmação* e *Subtarefa de Feedback de Confirmação*, e os **lugares** *Estado de Indicação*, *Estado de Confirmação* e *Estado de Anexação* podem ser agrupados como uma única entidade, denominada *Tarefa de Seleção*. As **transições** e **lugares** hachurados da Figura 11 destacam o conjunto a ser unificado. Já a Figura 12 mostra o novo modelo como uma RdP-H. Neste caso, a interpretação do modelo continua praticamente a mesma, pois a **transição** "Tarefa de Seleção" recebe as mar-

cas vindas do *Estado de Seleção* e repassa a **marca** *Objeto Selecionado* (indicado e confirmado) para o *Estado de Manipulação*.

Para representar um subconjunto de lugares e transições, uma nova entidade chamada **subrede** precisa ser definida. Nesta metodologia, **subredes** podem ser instanciadas a partir das classes especiais *PlacePage* e *TransitionPage*. A primeira abstrai uma rede que inicia e termina com **lugares**, enquanto a segunda encapsula uma rede que inicia e termina com **transições**.

Para este exemplo, optou-se pela criação de um objeto do tipo *TransitionPage* para representar a *Tarefa de Seleção*. Posteriormente, este objeto foi adicionado a RdP, junto com os demais já adicionados a sua lista. As conexões entre **lugares** e **transições**, bem como a definição de **marcas**, permaneceram inalteradas.

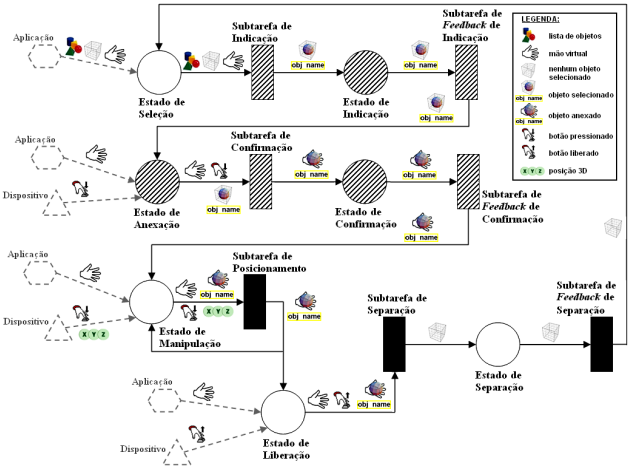

Figura 11: RdP da Figura 8 apresentando os nodos a serem integrados.

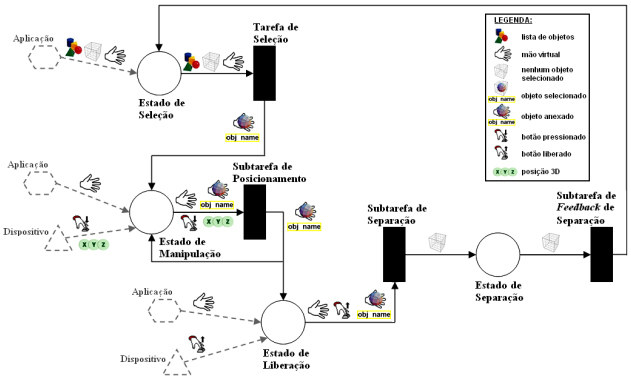

Figura 12: RdP com a *Tarefa de Seleção* hierarquizada, abstraindo detalhes do modelo representado pela Figura 11.

7. CONCLUSÕES

Este trabalho definiu uma metodologia para especificar tarefas de interação de uma aplicação de RV, utilizando o formalismo de RdP como base no processo de modelagem de *software*. A expectativa é que esta metodologia, à medida que vá sendo empregada, permita com que tarefas e técnicas de interação possam tornar-se componentes disponíveis para novos sistemas, uma vez que foram testadas e simuladas previamente.

Esta metodologia também objetiva facilitar o desenvolvimento de aplicações, aproximando a etapa de concepção e projeto de AVs à etapa de desenvolvimento, oferecendo recursos para implementação, testes e documentação do sistema.

Apesar do processo de implementação apresentado ter usufruído dos recursos de uma biblioteca gráfica específica, não existe impedimento para o uso de outras bibliotecas. Os passos da metodologia (e o pacote de classes desenvolvido) são independentes das funcionalidades gráficas, e podem ser adaptadas conforme as necessidades de outras ferramentas como, por exemplo, OpenSceneGraph [11], Crystal Space [19] e X3D [23] ou, até mesmo, serem usados em conjunto.

Como trabalhos futuros, sugere-se a construção de um editor gráfico capaz de construir a RdP e derivar o código-fonte da aplicação automaticamente, com base no arquivo de descrição do grafo. Para tanto, editores como DIA [5] e YeD [26] estão sendo analisados. Além disso, a ferramenta poderia disponibilizar ao usuário componentes de técnicas já definidos por outros projetos.

Como os estágios de simulação da RdP são controlados pela metodologia, seria interessante também apresentar ao projetista o processo de execução do programa em um gráfico animado, sobreposto ao grafo da RdP, e em paralelo ao uso da aplicação. Atualmente, somente uma saída textual é gerada durante a execução da aplicação.

Da mesma forma, uma idéia interessante seria incorporar esta metodologia a um *framework* de RV, tornando-o uma plataforma completa de desenvolvimento. Neste *framework* de interação, recursos para análise, projeto, desenvolvimento e avaliação de protótipos de AVs poderiam estar incorporados, permitindo que falhas de projeto fossem rapidamente detectadas, e o tempo de construção dos sistemas fosse reduzido.

8. AGRADECIMENTOS

Este trabalho foi parcialmente financiado pelo Tecgraf - Grupo de Tecnologia em Computação Gráfica da Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), laboratório financiado e parceiro de projetos de pesquisa da PETROBRAS. Agradecimentos também para as bolsas de estudo concedidas pelo convênio DELL/PUCRS (Pontifícia Universidade Católica do Rio Grande do Sul) e pela CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) durante o decorrer da pesquisa.

9. REFERÊNCIAS

[1] D. A. Bowman and L. F. Hodges. Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. *Journal of Visual Languages and Computing*, 10(1):37–53, 1999.

[2] D. A. Bowman, E. Kruijff, J. J. L. Jr., and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison-Wesley, 2005.

[3] R. Dachselt and A. Hübner. A survey and taxonomy of 3d menu techniques. In *EGVE '06: Proceedings of the 12th Eurographics Symposium on Virtual Environments*, pages 89–99, 2006.

[4] P. Figueroa, M. Green, and H. J. Hoover. Intml: a description language for vr applications. In *Web3D '02: Proceeding of the seventh international conference on 3D Web technology*, pages 53–58, New York, NY, USA, 2002. ACM Press.

[5] A. Larsson and et. al. DIA, a drawing program - GNOME-related project. Available on: http://www.gnome.org/projects/dia/, 2006.

[6] R. W. Lindeman. *Bimanual interaction, passive-haptic feedback, 3d widget representation, and simulated surface constraints for interaction in immersive virtual environments*. PhD thesis, Faculty of the School of Engineering and Applied Science, George Washington University, 1999. Director-James K. Hahn.

[7] A. MacWilliams, C. Sandor, M. Wagner, M. Bauer, G. Klinker, and B. Brügge. Herding sheep: Live system development for distributed augmented reality. In *ISMAR*, pages 123–132. IEEE Computer Society, 2003.

[8] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, pages 541–580, April 1989. NewsletterInfo: 33Published as Proceedings of the IEEE, volume 77, number 4.

[9] D. Navarre, P. A. Palanque, R. Bastide, A. Schyn, M. Winckler, L. P. Nedel, and C. M. D. S. Freitas. A formal description of multimodal interaction techniques for immersive virtual reality applications. In M. F. Costabile and F. Paternò, editors, *INTERACT*, volume 3585 of *Lecture Notes in Computer Science*, pages 170–183. Springer, 2005.

[10] A. Olwal and S. Feiner. Unit: modular development of distributed interaction techniques for highly interactive user interfaces. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 131–138, New York, NY, USA, 2004. ACM Press.

[11] R. Osfield and D. Burns. Open Scene Graph. Available on: http://www.openscenegraph.org/, 2006.

[12] P. A. Palanque and R. Bastide. Synergistic modelling of tasks, users and systems using formal specification techniques. *Interacting with Computers*, 9(2):129–153, 1997.

[13] M. S. Pinho. SmallVR: Uma ferramenta orientada a objetos para o desenvolvimento de aplicações de realidade virtual. In *SVR '02: Proceedings of the 5th Symposium on Virtual Reality*, pages 329–340, 2002.

[14] M. S. Pinho. SmallVR. Available on: http://www.smallvr.org/, 2006.

[15] I. Poupyrev, S. Weghorst, M. Billinghurst, and T. Ichikawa. A framework and testbed for studying manipulation techniques for immersive VR. In D. Thalmann, editor, *ACM Symposium on Virtual Reality Software and Technology*, pages 21–28, New York, NY, 1997. ACM Press.

[16] R. Rieder, F. B. Silva, A. B. Trombetta, R. A. Kopper, M. C. dos Santos, and M. S. Pinho. Uma avaliação do uso de estímulos táteis em um ambiente virtual. In *SVR '06: Proceedings of the 8th Symposium on Virtual Reality*, pages 135–146, 2006.

[17] S. Smith and D. Duke. Virtual environments as hybrid systems. In *Eurographics UK 17th Annual Conference (EG-UK'99)*, Cambridge, UK, 1999.

[18] Trolltech. Qt Reference Documentation - Open Source Edition. Signals and Slots. Available on: http://doc.trolltech.com/4.1/signalsandslots.html, 2006.

[19] J. Tyberghein and et. al. Crystal Space 3D. Available on: http://www.crystalspace3d.org/, 2006.

[20] R. Wieting. Hybrid high-level nets. In *WSC '96: Proceedings of the 28th conference on Winter simulation*, pages 848–855, 1996.

[21] J. S. Willans and M. D. Harrison. Prototyping pre-implementation designs of virtual environment behaviour. In *EHCI '01: Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, pages 91–108, London, UK, 2001. Springer-Verlag.

[22] C. A. Wingrave and D. A. Bowman. Chasm: Bridging description and implementation of 3d interfaces. In *Proceedings of the Workshop on New Directions in 3D User Interfaces*, pages 85–88, 2005.

[23] X3D. Web3D Consortium - Royalty Free, Open Standards for Real-Time 3D Communication. Available on: http://www.web3d.org/, 2006.

[24] J. Ying. An approach to Petri net based formal modeling of user interactions from x3d content. In *Web3D '06: Proceedings of the eleventh international conference on 3D web technology*, pages 153–157, New York, NY, USA, 2006. ACM Press.

[25] J. Ying and D. Gracanin. Petri net model for subjective views in collaborative virtual environments. In A. Butz, A. Krüger, and P. Olivier, editors, *Smart Graphics*, volume 3031 of *Lecture Notes in Computer Science*, pages 128–134. Springer, 2004.

[26] yWorks the diagramming company. yEd - Java Graph Editor. Available on: http://www.yworks.com/products/yed/, 2006.