Rodrigo Thiago Vioto Cunha

Victor Manuel Cirilo Romero

Desenvolvimento de Ambiente Interativo de Simulação e Análise de Planos

Monografia de Conclusão de Curso apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro

São Paulo
2008

Rodrigo Thiago Vioto Cunha

Victor Manuel Cirilo Romero

Desenvolvimento de Ambiente Interativo de Simulação e Análise de Planos

Monografia de Conclusão de Curso apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro

Área de concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. José Reinaldo Silva

São Paulo
2008

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com anuência de seu orientador. São Paulo, 10 de Dezembro de 2008.

Assinatura do autor.

Assinatura do autor.

Assinatura do orientador.

Ficha Catalográfica

Cunha, Rodrigo Thiago Vioto
Romero, Victor Manuel Cirilo
Desenvolvimento de Ambiente Interativo de Simulação e Análise de Planos. São Paulo, 2008. 57 p.

Monografia de Conclusão de Curso (Graduação) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Mecatrônica e de Sistemas Mecânicos.

1. Prototipagem Virtual. 2. Planejamento Automático. 3. Realidade Virtual. 4. Análise de Planos. I. Universidade de São Paulo. Escola Politécnica. II. Departamento de Engenharia de Mecatrônica e de Sistemas Mecânicos.

Aos nossos pais...

Agradecimentos

Agradecemos primeiramente a Deus, fonte de infinita bondade e perseverança, sem as quais nada deste trabalho teria sido possível.

Ao nosso orientador, Prof. José Reinaldo Silva, e ao doutorando Tiago Stegun Vaquero pelo constante auxílio durante todo o projeto.

Por fim, aos nossos pais, namoradas e amigos, por todo apoio, carinho e compreensão.

Resumo

As situações a serem analisadas pelos Protótipos Virtuais (PVs), especialmente as inseridas em um ambiente de engenharia mecatrônica, têm a necessidade de uma ferramenta de simulação satisfatória. Além disso, os problemas de Planejamento Automático (PA), via de regra, não possuem uma solução analítica fechada, e demandam ferramentas de validação de planos. Com o objetivo de demonstrar a aplicabilidade de uma proposta de uso da Prototipagem Virtual aplicada em problemas de PA, criou-se uma ferramenta de simulação tridimensional ligada ao itSIMPLE, ferramenta de modelagem e análise de problemas de PA desenvolvida pelo DesignLab da Escola Politécnica da USP. Implementada com o uso do freeware Blender, essa interface permite que o usuário visualize, em um ambiente tridimensional, a execução de planos relacionados a um tipo de problema específico: a movimentação de agentes autônomos em um ambiente com obstáculos. Dessa forma, procurou-se mostrar o potencial dessa abordagem para o design de produtos de engenharia mecatrônica.

Abstract

The situations to be analysed by Virtual Prototype (VPs), especially those within a mechatronics engineering environment, need satisfactory simulation tool. Moreover, the problems of Automated Planning (AP) usually don't have an analytical and closed solution, and therefore demand plan validation tools. With the objective to show the applicability of a proposal of Virtual Prototyping usage applied in AP problems, a tridimensional simulation tool was created. This tool is linked to itSIMPLE, an AP problems modeling and analysis tool, developed by DesignLab from Escola Politécnica da USP. This interface was implemented by the usage the freeware Blender, and it enables the user to visualize, in a tridimensional environment, the execution of plans related to a specific kind of problem: the movimentation of autonoms agents in a environment with obstacles. Thus the objective was to show the potential of this approach for the design of mechatronics engineering products.

Conteúdo

Lista de Figuras

1 Introdução

1.1 Apresentação

O desenvolvimento de produtos e serviços feito por empresas faz com que muito tempo e recursos sejam empregados na fase de prototipagem e testes. Isso ocorre devido à importância do desenvolvimento de protótipos para definições importantes do projeto final do produto. Assim é importante que surjam alternativas válidas para que as empresas sejam capazes de desenvolver protótipos de forma eficiente e menos custosa.

Uma dessas alternativas é o uso dos chamados Protótipos Virtuais (PVs). Em um ambiente virtual de desenvolvimento, a proposta é criar modelos tridimensionais do protótipo a ser desenvolvido. Esse ambiente, que procura simular as características do ambiente real em que o protótipo estará inserido, bem como as propriedades físicas mais importantes do produto, deve ser capaz de fornecer as informações mais relevantes sobre o desempenho de um protótipo, as mesmas que seriam obtidas caso o protótipo tivesse sido desenvolvido fisicamente.

As vantagens na utilização de protótipos virtuais são inúmeras, sendo a mais evidente o fato da redução de desenvolvimento de modelos físicos para testes. Sendo o ambiente virtual de desenvolvimento suficientemente avançado para simular as condições físicas do ambiente e possibilitando a manipulação do protótipo de forma completa, os resultados de testes obtidos virtualmente são os mesmos que seriam obtidos com um protótipo físico. Isso contribuiria bastante na eliminação de custos do projeto, já que um protótipo físico só seria desenvolvido numa fase avançada de projeto, já com uma série de melhorias oriundas de seus modelos virtuais previamente testados e verificados.

Dentre as situações possíveis de serem analisadas com o uso de PVs, este trabalho procurou focar-se naquelas inseridas no âmbito do Planejamento Automático (PA). Esse tipo de problema possui um escopo bem definido e pode ser analisado de forma objetiva em um ambiente virtual. Além disso, os problemas

de PA têm a necessidade de uma ferramenta que possa simular de forma satisfatória os planos encontrados pelos softwares de planejamento. Isso permitiria a verificação desses planos, com a oportunidade de se encontrar tanto falhas na modelagem do problema como possibilidades de melhoria do protótipo, evidenciando a utilidade da PV.

No entanto, a grande dificuldade no estudo desses problemas é conseguir modelar, analisar e classificar os domínios de planejamento de uma forma completa e satisfatória. Até algum tempo atrás, a área de pesquisa em PA focava apenas no desenvolvimento de algoritmos de planejamento (chamados de planejadores) para resolução de problemas, e pouca atenção era dada para o processo de análise e modelagem dos modelos dos problemas de planejamento. Isso pode gerar certas dificuldades, pois se a modelagem não for bem feita, o algoritmo de planejamento nunca encontrará uma boa resposta. Atualmente esse cenário vem mudando, o que resultou na utilização de muitos conceitos da Engenharia do Conhecimento e da Análise de Requisitos, que auxiliam no entendimento, modelagem e caracterização de problemas de planejamento mais complexos e próximos da realidade. Nesse sentido, há hoje uma grande necessidade de uma ferramenta que possa, de uma forma simples, intuitiva e eficiente, contemplar todo o ciclo de vida de um problema real de planejamento. Assim, a ferramenta deve auxiliar o projetista a ter uma visão global do sistema, desde a sua concepção, requisitos e modelagem, até a sua análise dinâmica e a eventual implementação de um plano de ações que solucionem o problema. É com o intuito de preencher essa lacuna que a ferramenta itSIMPLE (*Integrated Tools Software Interface for Modeling PLanning Environments*) foi desenvolvida pelo DesingLab da Escola Politécnica da USP. Portanto, a utilização e evolução dessa ferramenta estiveram inseridas no escopo deste trabalho durante todo o período de desenvolvimento.

Com isso, o ambiente virtual proposto pelo trabalho utiliza as funcionalidades do itSIMPLE para adentrar no campo do Planejamento Automático. Dessa forma, buscou-se o desenvolvimento de uma ferramenta integrada que aliasse a Prototipagem Virtual com conceitos de PA. Além disso, procurou-se também realizar-se análise de problemas específicos de planejamento, para se demonstrar os conceitos propostos pelo presente trabalho.

1.2 Motivação

A Prototipagem Virtual está sendo cada vez mais usada para o desenvolvimento de produtos, pois tem se mostrado de fato uma maneira consistente de se

verificar a eficácia de um produto. Existem muitas áreas em que os PVs podem ser aplicados, e a mecatrônica aparece aí de forma destacada. Protótipos de dispositivos ou ambientes mecatrônicos são bons exemplos de aplicação desse tipo de abordagem.

Essa tecnologia pode facilitar muito o projeto de um produto, visto que ela torna esse processo algo muito mais intuitivo e dinâmico. Aliada a práticas como CAD e CAM, ela representa uma nova abordagem bastante sólida de desenvolvimento de produtos. Portanto, é importante que novas propostas sejam desenvolvidas nessa área, indo ao encontro das novas tecnologias aplicáveis à engenharia mecatrônica.

Todo esse desenvolvimento é compatível com uma necessidade existente atualmente na área de Planejamento Automático, que é a de se tratar problemas de planejamento mais próximos da realidade. Para isso, é de suma importância, entre outras coisas, que haja um ambiente de simulação e visualização dos planos. Atualmente, o itSIMPLE possui uma interface de simulação de planos em UML, mas essa não é a melhor solução para esse tipo de tarefa. As linguagens gráficas, como a UML, apesar de serem extremamente úteis no processo de modelagem, se mostram limitadas na medida em que se aumenta a quantidade de objetos e relacionamentos em seu diagrama, o que dificulta a visualização. Além disso, problemas como de espaço físico, posicionamento e colisões seriam muito melhor representados (e conseqüentemente verificados) em um ambiente tridimensional de representação.

Além disso, é importante que o ambiente seja capaz de oferecer ao projetista a capacidade de interagir com o sistema de planejamento através de um sistema virtual. Essa interação, que significa o ato de alterar os estados do sistema gerados ao longo da execução do plano, pode simular situações não previstas, mas que acabam acontecendo na realidade. Assim, uma ferramenta que possibilitasse essa interação contribuiria para que a análise se tornasse cada vez mais próxima da realidade. O sistema, ao receber a nova informação, deve ser capaz de absorvê-la, incorporá-la ao planejamento e criar um novo plano, como se fosse um desvio do plano original, conferindo mais realismo ao processo de análise e verificação.

1.3 Objetivos

O objetivo deste trabalho foi o desenvolvimento de uma proposta de aplicação da Prototipagem Virtual para a análise e verificação de problemas de Planeja-

mento Automático. Com isso, procurou-se desenvolver uma ferramenta integrada de simulação tridimensional dos planos gerados dentro do itSIMPLE, possibilitando a verificação, por parte do usuário, tanto do modelo abstrato de planejamento como do modelo virtual.

Para a implementação dessa interface, será utilizada a ferramenta gratuita Blender. Esse software oferece inúmeras vantagens que o fizeram ser escolhido para este projeto, dentre as quais se destacam o fato dele ser uma ferramenta de modelagem tridimensional leve, de grande aceitação e, principalmente, por ter uma engine integrada, com a qual é possível criar jogos com uma grande fidelidade a fenômenos físicos, como colisões, ação da gravidade, propriedades de corpos rígidos, etc. Além disso, o modelo pode ser animado com o uso de scripts escritos na linguagem Python, nativa no Blender, representando uma plataforma consistente para o tipo de trabalho proposto.

É importante ressaltar que a interface desenvolvida não é interativa, pelo menos não diretamente. A proposta do trabalho foi de criar uma interface tridimensional de simulação, e não ainda de interação, pois uma interface tridimensional interativa, que respondesse em tempo real a comandos de um usuário, seria um trabalho muito mais complexo, indo muito além do escopo deste projeto. Isso significa que o plano gerado no itSIMPLE é simulado nessa interface, como se fosse um filme, cujo roteiro é o próprio plano.

Dessa forma, para que o usuário possa realizar interações com o sistema, com o intuito de criar situações imprevistas, fez parte do trabalho criar, no itSIMPLE, a possibilidade de se realizar essa interação. Para isso, é usada a própria interface UML já existente, porém com as modificações necessárias para que o projetista possa realizar alterações nos estados do plano e depois replanejar o sistema, já com as alterações feitas.

2 Revisão da Literatura

2.1 Prototipagem Virtual

Segundo Cecil e Kanchanapiboon (1), um Protótipo Virtual é um modelo de Realidade Virtual (RV) físico ou de engenharia que pode imitar ou simular o comportamento, resposta, aparência e geometria de um determinado objeto, sistema ou ambiente, com um grau de realismo comparável ao objeto, sistema ou ambiente real. Assim, prototipagem virtual é o processo de se criar e utilizar um PV em aplicações diversas.

Ainda segundo os mesmo autores, um PV deve ter uma série de características, a saber:

- aparência: um PV deve possuir geometria, topologia e aparência acuradas, refletindo características da peça, objeto, sistema ou ambiente a que se destina;
- simulação: PVs devem ser capazes de simular características relevantes para o problema, incluindo comportamento com respostas em tempo real;
- representação: um PV é uma representação digital ou computadorizada;
- interface: um PV deve possuir a habilidade de interfacear com gráficos de realidade virtual, incluindo suporte a aplicações imersivas ou semi-imersivas.

É importante frisar que um PV não é simplesmente um modelo ou representação criada com auxílio de um computador (CAD/CAM), pois deve possuir as características citadas acima, sendo a principal delas o uso de RV, possibilitando interação imersiva ou semi-imersiva. O projetista deve poder "entrar" de fato no ambiente, interagir com ele e observar as respostas das suas ações em tempo real. Dispositivos mecatrônicos são, via de regra, muito complexos, pois integram um grande número de peças, além de serem formados por partes mecânicas, elétricas, e eletrônicas, software, etc. Assim, o desenvolvimento de protótipos nesse tipo

de projeto torna-se algo muito lento e custoso. Por isso, o uso de PVs no projeto desse tipo de produto torna-se bastante vantajoso (2).

A prototipagem virtual pode ser aplicada de diversas formas. As mais comuns são as chamadas Manufatura Virtual e Projeto Virtual de Produto. No primeiro caso, os modelos virtuais são voltados ao desenvolvimento de ambientes de manufatura. Assim, os PVs são destinados a aplicações como projeto de layout de chão de fábrica (incluindo aí as representações tanto das instalações como das próprias máquinas), prototipagem de tarefas de montagem de produtos e PVs de atividades de usinagem de baixo nível, como cortes em peças. Essa aplicação está mais próxima da proposta realizada neste trabalho, apresentada no capítulo seguinte. A Figura 2.1 mostra um exemplo desse tipo de ambiente.

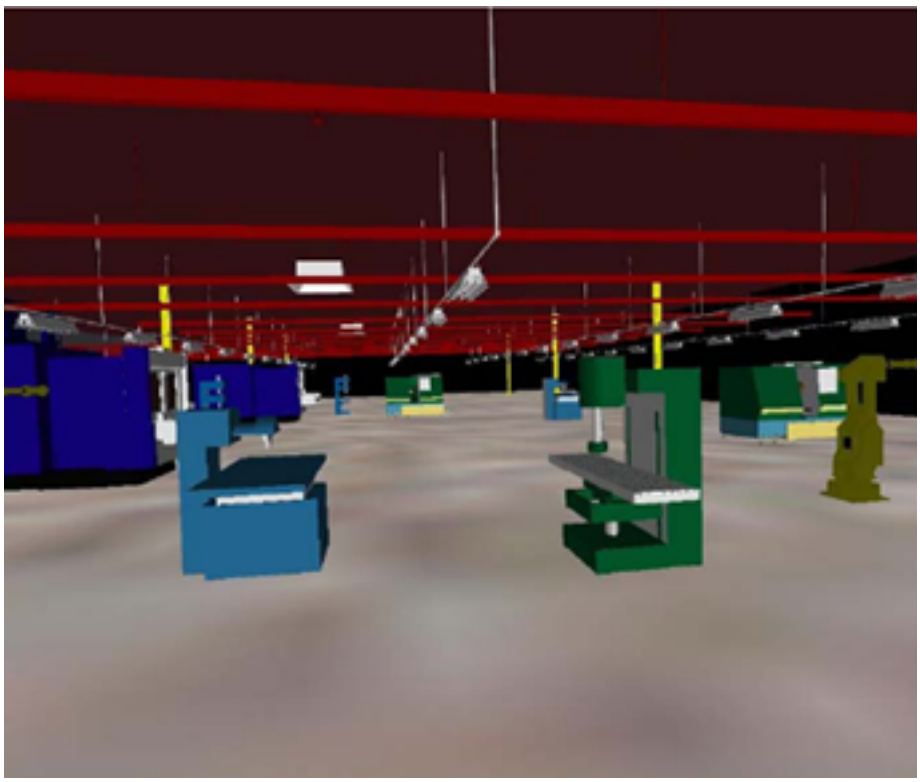

Figura 2.1: Exemplo de PV para layout de chão de fábrica (1)

Já no Projeto Virtual de Produto, os PVs são usados para ajudar engenheiros a conceituar, refinar e modificar projetos de produto mais cedo no seu ciclo de desenvolvimento. Como dito anteriormente, esse tipo de abordagem auxilia muito o projeto de soluções mecatrônicas, visto que o projetista pode interagir com o protótipo da mesma forma como se ele fosse real, verificando toda a integração entre as várias peças do mecanismo. Um exemplo desse uso é mostrado na Figura 2.2. O conceito de Prototipagem Virtual tem sido cada vez mais utilizados para desenvolvimento de novos produtos e serviços, pois apresenta inúmeras vantagens, destacando-se aí a possibilidade de se testar os protótipo sem a necessidade de

sua construção física, o que diminui custos e aumenta a velocidade do processo de desenvolvimento. Além disso, os custos de desenvolvimento de PVs caíram muito nos último dez anos, o que passou a tornar viável esse tipo de prática.

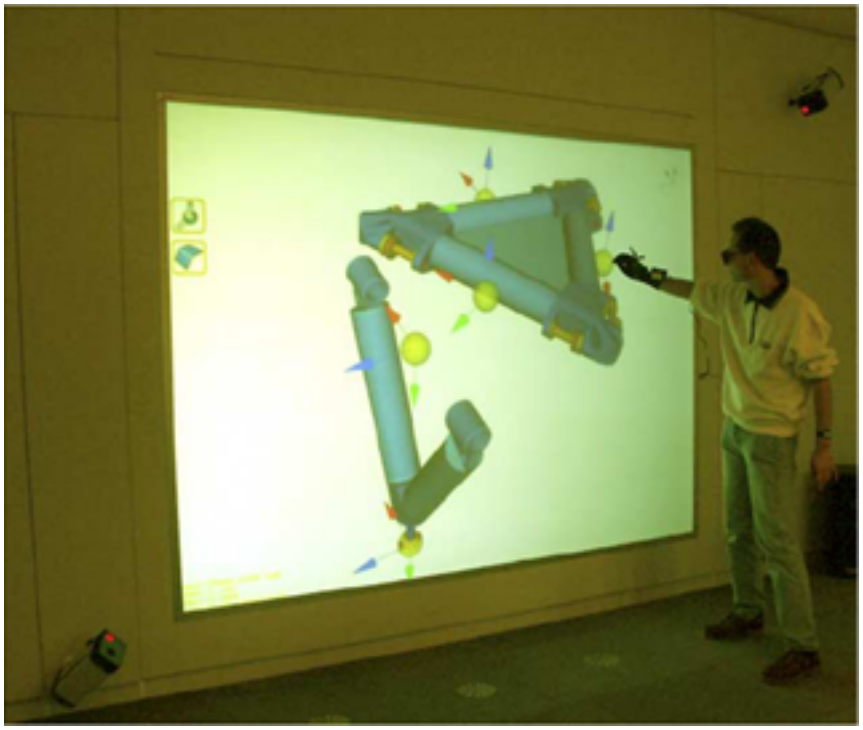

Figura 2.2: Exemplo de interação com PV de mecanismo (2)

2.2 Planejamento Automático

Planejar é uma palavra bastante genérica, que pode obter inúmeros significados diferentes dependendo do contexto. Mas, de uma forma geral, podemos dizer que planejar é escolher e organizar ações, cujos efeitos podemos prever ou estimar, de forma a obter um resultado desejado. O ato de planejar demanda inteligência, visto que se deve tomar decisões entre quais ações escolher, quando executar determinada ação, tudo isso com vistas em um objetivo final bem determinado. Se passarmos a realizar esse processo inteligente de forma automática, com auxílio de computadores, adentra-se no território da Inteligência Artificial (IA). Assim, o Planejamento Automático (PA) é o campo da IA que estuda o processo deliberado de planejar de forma computacional (3).

A Figura 2.3 abaixo mostra o modelo conceitual de PA. Nela podemos ver um sistema qualquer sendo controlado por um controlador que recebe instruções de um planejador. Deve-se notar que o planejador possui, de antemão, uma descrição (ou seja, um modelo do sistema) tanto do estado inicial como dos objetivos a serem alcançados. Além disso, o planejador recebe do controlador o status de

execução, ou seja, o planejador tem a possibilidade de revisar seus planos, de forma a manter o sistema no caminho do seu objetivo. Dessa forma, o planejador envia ao controlador planos, para que sejam executados sobre o sistema. O controlador, além de atuar em Σ, aplicando as ações descritas nos planos oriundos do planejador, também observa o sistema, por meio de sensores. Assim, em caso das ações não terem sido aplicadas de forma satisfatória ou, por outro lado, caso um evento alheio às atuações do controlador seja detectado, o controlador realimenta o planejador, que realiza um replanejamento, baseado nas observações realizadas.

Figura 2.3: Modelo conceitual de Planejamento Automático (3)

Por muito tempo, os esforços dos pesquisadores da área de PA estiveram concentrados apenas nos algoritmos de planejamento aplicados a problemas simples, que não necessariamente possuíssem alguma aplicação real. Essa fase foi importante para o desenvolvimento de conceitos que hoje são utilizados em problemas reais. No entanto, por causa desse foco, deu-se historicamente muito mais atenção ao processo de planejamento e aos softwares que fazem esse planejamento, os planejadores, do que ao processo completo de planejamento, que envolve também análise de requisitos, modelagem, engenharia do conhecimento, etc. Assim, durante muito tempo, a comunidade de PA esteve muito mais preocupada com questões como eficiência e desempenho dos planejadores. Essas são questões importantíssimas, sem dúvida, já que em problemas reais o tempo de resposta é imprescindível para o controle de um sistema. No entanto, o processo de modelagem se mostrou igualmente importante, e faltavam ferramentas nesse aspecto.

Quando o foco da comunidade de PA foi aberto, passando também a englobar todo o processo de modelagem de domínios e problemas, e passou-se a aplicar

os conceitos de planejamento em problemas mais realistas, percebeu-se que os modelos utilizados para o planejamento são cruciais para se obter resultados satisfatórios.

2.3 Ferramentas de Modelagem — itSIMPLE

Algumas ferramentas de auxílio à modelagem têm sido desenvolvidas, no intuito de assistir o projetista nesse momento crucial que é a elaboração do modelo que descreve o sistema que será objeto do planejamento. Entre elas pode-se citar o GIPO III (4) e PlanWorks (5), além é claro da ferramenta que faz parte deste trabalho, o itSIMPLE (6) (7). Todas elas possuem suas interfaces próprias de modelagem de domínios e de planejamento, mas o que as une é o fato de procurarem oferecer ao projetista uma interface amigável para a criação e utilização de domínios e problemas de PA.

Um problema de planejamento normalmente é atacado utilizando-se a linguagem PDDL (*Planning Domain Definition Language*) (8), que atualmente representa a única linguagem padronizada que os planejadores suportam para realizar o planejamento. Nessa linguagem, um problema de planejamento é tratado pelo planejador a partir de duas definições distintas, a saber:

- o domínio, onde o projetista define os tipos (classes) que compõem o problema, atributos que esses tipos ou o próprio sistema como um todo pode apresentar, e ações, definidas basicamente por suas pré-condições e seus efeitos; pode-se dizer que no domínio são definidas as regras que regem todo o sistema;
- e o problema propriamente dito, onde o projetista cria instâncias dos tipos definidos no domínio, atribui ao sistema um conjunto de fatos e define no mínimo dois estados para o sistema (início e meta).

Essas definições nada mais são que arquivos com código PDDL carregados pelo planejador e utilizados no processo de planejamento. Assim, a dificuldade reside no fato dessa linguagem ser relativamente complexa e não intuitiva, até para usuários mais avançados, quanto mais para um usuário comum. Além disso, a PDDL possui limitações inerentes (9) (10), que dificultam o processo de modelagem. A proposta do itSIMPLE é utilizar outras linguagens para interfacear com o projetista e, a partir dessas linguagens, criar um modelo PDDL para o planejador.

O itSIMPLE foi desenvolvido pelo DesignLab da Escola Politécnica da USP no intuito de suprir a falta de uma ferramenta que contemplasse não só o processo de planejamento, mas também todo o processo de modelagem de domínios e problemas até a simulação e análise de planos oriundos dos planejadores. Para tanto, o itSIMPLE utiliza linguagens como a UML (*Unified Modeling Language*) (11) e OCL (*Object Constraint Language*) (12) para modelagem de domínios e problemas de planejamento, utilizando diagramas de caso de uso, de classes, de estados e de objetos. Com essas definições, a ferramenta é capaz de criar um modelo em PDDL e então comunicar-se com planejadores em sua interface de planejamento e simulação de planos. Além disso, o itSIMPLE utiliza a linguagem XML (*eXtensible Markup Language*) (13), para armazenagem de informações e interface entre linguagens, dado o fato da XML ser uma linguagem de marcação, e portanto própria para realizar essa transferência de dados dentro do fluxo de trabalho da ferramenta. A Figura 2.4 abaixo mostra a interface UML do itSIMPLE.

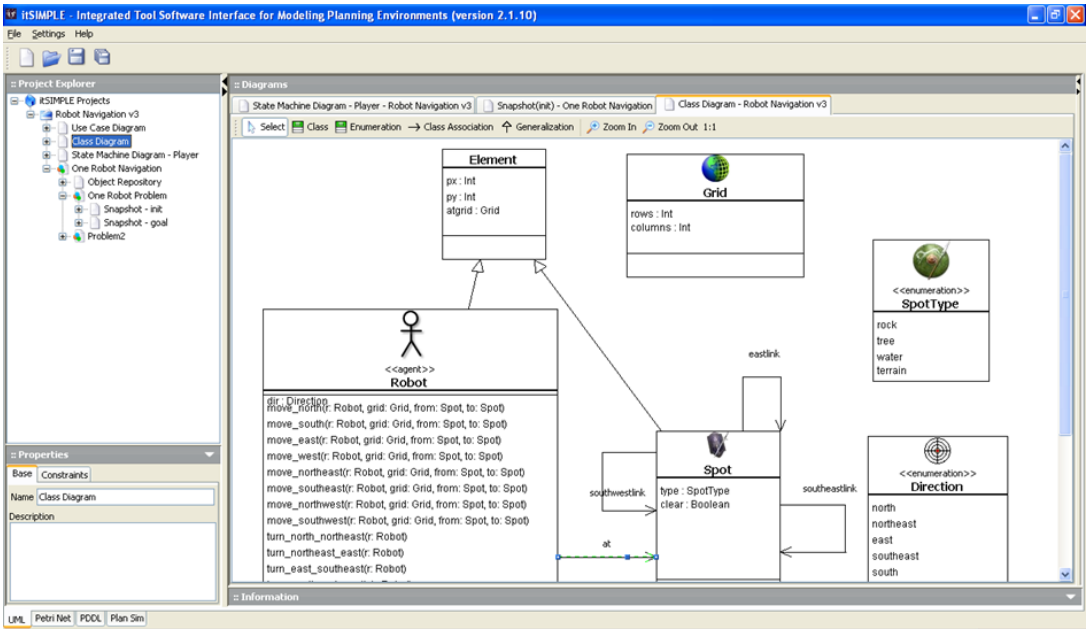

Figura 2.4: Interface UML atual do itSIMPLE

2.4 Análise e Simulação de Planos

Como já foi dito, os planejadores costumam prover os planos que eles encontram em formato puramente textual, uma simples lista de ações, apresentadas linha a linha, sem maiores preocupações com a utilidade desse plano. Essa postura reside no fato já exposto aqui do foco, por parte da comunidade de PA, em analisar atributos como eficiência e desempenho do planejador, e não priorizando, por exemplo, a utilidade e a qualidade que poderia ser dada aos planos bem como

formas estruturadas de expressar tais planos.

O conceito de análise de planos é bastante novo, mas muitos trabalhos já têm sido feitos nessa área. Eles mostram que os planos podem ter uma utilidade muito maior se guardassem, por exemplo, informações sobre o processo de planejamento, o que permitiria que os projetistas pudessem entender a lógica das heurísticas por trás do processo de planejamento (14). Além disso, a análise de planos possibilita ao projetista verificar se o modelo projetado realmente representa o que ele imaginava, se há erros de modelagem ou se há novas definições a serem adicionadas ao modelo. Com isso, poderia ser verificado de fato o caminho que o sistema percorreria caso a execução do plano fosse realizada, e estados inconsistentes ou indesejados poderiam ser previstos.

É importante frisar que o processo de análise de planos estudado neste trabalho não é um processo de validação do plano, mas sim a abertura da possibilidade de o projetista verificar se o plano gerado corresponde às suas expectativas. O fato do plano alcançar o objetivo não significa necessariamente que ele seja um bom plano, dado que ele pode visitar estados indesejados pelo projetista. No entanto, classificar um plano em "bom" ou 'ruim" é uma tarefa subjetiva, e que, com uma ferramenta de análise de planos, pode ser realizada pelo próprio projetista, durante a elaboração do seu modelo.

As ferramentas de modelagem citadas até aqui possuem interfaces próprias de simulação e análise de planos. O itSIMPLE utiliza a mesma linguagem UML da modelagem para a visualização dos estados gerados pela execução do plano, bem como para geração do gráficos de variáveis particulares do problema que o projetista queira acompanhar (15). Com isso, a análise do plano se torna uma tarefa muito mais facilitada, possibilitando a extração de informações úteis do plano obtido pelo planejador.

2.5 Planejamento Automático em Ambientes Virtuais

Sempre com o objetivo de utilizar o Planejamento Automático em ambientes cada vez mais próximos da realidade, essas técnicas começaram a ser utilizadas em ambientes virtuais, que simulam com uma fidelidade considerável um ambiente real. Assim, problemas de planejamento podem ser visualizados por meio de animações em ambientes tridimensionais, que simulam como seria o comportamento real do sistema.

Um exemplo de sucesso dessa técnica é o jogo eletrônico chamado F.E.A.R. (16), um jogo de tiro em primeira pessoa que utiliza conceitos de PA para comandar os inimigos do jogador. Nesse jogo, cada personagem possui um objetivo particular e a lógica é capaz de criar planos particulares para esses personagens, cujas ações disparam animações na tela que dão ao jogo a sua fluência. Este jogo demonstra um exemplo do uso de planejadores em tempo real, ou seja, ao mesmo tempo em que se controla o sistema existem eventos inesperados acontecendo, particularmente, as ações do jogador humano dentro jogo.

Há, no entanto, uma série de problemas a serem enfrentados na elaboração de um ambiente virtual de PA, e que, de certa forma, são semelhantes aos problemas que seriam encontrados no controle de um sistema real. Entre esses problemas, descritos por Dini et al. (17), pode-se destacar, por exemplo, o fato de um ambiente ser dinâmico, ou seja, não se poder assumir que entre as execuções das ações o estado do sistema permanece constante.

Além disso, existe o problema do tratamento dos dados do sistema. Num sistema real, muitas vezes os estados são parcialmente observáveis onde não se pode afirmar com certeza que o observador tem total ciência de todas as variáveis do ambiente, o que traz uma série de problemas, como a necessidade da estimação do estado do sistema. Já num ambiente virtual isso não existe, pois todas as variáveis do sistema estão disponíveis e são conhecidas. Assim, o problema reside no fato desse conjunto de dados ser muito grande, o que pode tornar operações como a avaliação de uma pré-condição em algo extremamente custoso. Uma das soluções para isso é levar em conta apenas as variáveis relevantes para o processo, o que acaba trazendo outro problema, que é determinar quais dados são relevantes para o sistema.

3 Proposta

Os conceitos apresentados na seção anterior foram estudados no princípio do projeto, de forma a se poder encontrar uma ferramenta que pudesse aplicar esses conceitos e resolver algumas das questões pertinentes ao assunto. Assim, a proposta deste trabalho é demonstrar como a utilização da Prototipagem Virtual pode contribuir na resolução de um problema de engenharia.

Dessa forma, foi realizado o desenvolvimento de uma interface de simulação de planos em um ambiente virtual, de forma que o projetista pudesse verificar a evolução do sistema por ele projetado por meio de um modelo tridimensional previamente elaborado. O modelo desenvolvido simula a movimentação de agentes ou veículos autônomos em um ambiente com obstáculos que devem ser evitados. Assim, o itSIMPLE é utilizado para o desenvolvimento do modelo abstrato e dos problemas a serem resolvidos, bem como para obtenção de planos, e o ambiente tridimensional simula a movimentação desses agentes em um cenário.

A interface 3D foi desenvolvida numa ferramenta de modelagem 3D chamada Blender[1], disponibilizada gratuitamente na internet. Essa ferramenta, como qualquer outra, possui vantagens e desvantagens, mas se mostrou a melhor opção pelos seguintes motivos:

- é multiplataforma, ou seja, o Blender é projetado para funcionar nos sistemas operacionais mais utilizados, como Windows, Linux e Mac OS, entre outros;
- é gratuita, de uso livre e disponível facilmente na internet;
- é bastante leve computacionalmente em comparação com outras ferramentas semelhantes, podendo ser utilizadas em computadores medianos;
- possui inúmeras funcionalidades de modelagem 3D, como posicionamento de luz e câmera, texturas, renderização, animação etc.;

[1] http://www.blender.org

- possui uma interface de animação integrada com uma engine embarcada para jogos, o que facilita o processo de animação;
- pode utilizar scripts para controle de animações.

A principal desvantagem da ferramenta é sua interface pouco intuitiva e totalmente diferente de outras ferramentas utilizadas para o mesmo fim, o que dificultou bastante o seu uso, principalmente nos estágios iniciais. Existem muitos tutoriais disponíveis sobre a ferramenta, inclusive no próprio site dos desenvolvedores, que procuram minimizar essa dificuldade, mas ainda sim muito tempo teve de ser dedicado ao entendimento das suas funções.

Mesmo assim, o Blender mostrou-se a melhor opção para este projeto. O que pesou muito para a sua escolha foi o fato dele já possuir, junto à interface de modelagem, módulos de definições de propriedades de corpo rígido, adição de atuadores, sensores e as principais ferramentas necessárias para a criação de animações com modelos tridimensionais. Além disso, o Blender possui uma engine de jogos interna, sendo até possível criar jogos com essa ferramenta. Uma engine é um ambiente que reúne todas as características necessárias para que modelos possam representar objetos reais, como massa, gravidade, física de corpos rígidos etc. Essa engine facilita em muito o processo de animação, já que basta que os modelos sofram atuações e sua movimentação ocorrerá da forma como seria no mundo real.

Além disso, essa interface de animação do Blender pode ser comandada por scripts, utilizando para isso a linguagem Python[2]. Essa linguagem tem as características de ser orientada a objetos e ser interpretada, não compilada, e tem seu uso muito difundido atualmente, de modo destacado na indústria de jogos eletrônicos. Assim, ela pode ser utilizada, no âmbito deste projeto, para realizar as animações que cada ação do plano representa.

Para tanto, essa interface tridimensional de simulação deve se comunicar com o itSIMPLE, de forma a aproveitar tudo que essa ferramenta já possui na questão de simulação e análise de planos. O itSIMPLE já possui os dados gerados pela obtenção de um plano, portanto para aproveitar suas funcionalidades, um arquivo contendo os dados de simulação é gerado pelo itSIMPLE e compartilhado com a interface tridimensional. Utilizando scripts em Python, a interface pode utilizar as informações disponibilizadas pelo itSIMPLE para realizar a animação do modelo tridimensional.

[2]http://www.python.org

É importante frisar também que esse modelo tridimensional, bem como os scripts de animação, devem ser criados pelo próprio projetista. Isso ocorre por que cada problema possui ações e objetos próprios, e seria um trabalho extremamente complexo a criação um sistema genérico de animação, que pudesse criar e animar os modelos baseados unicamente em informações da modelagem, onde quer que essa modelagem fosse realizada.

O objetivo, portanto, é a criação de um ambiente integrado de planejamento e prototipagem, que reúne o itSIMPLE como ferramenta de modelagem abstrata e geração de planos, e o Blender como ambiente de modelagem tridimensional. Essas ferramentas devem se comunicar de forma a trocar a informações relevantes para o processo de visualização e animação do ambiente 3D, criando assim um Protótipo Virtual do modelo abstrato criado no itSIMPLE. O centro desse ambiente é o próprio usuário, que realiza a modelagem e a verifica no ambiente virtual, obtendo informações úteis sobre o sistema real, além de realimentar o modelo com novas informações oriundas da análise de cada situação. A Figura 3.1 ilustra essa integração entre os ambientes.

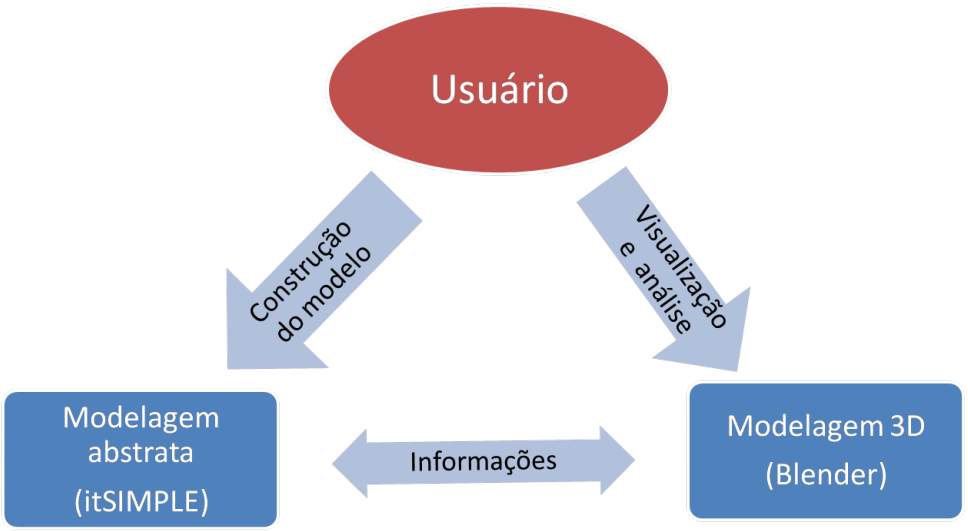

Figura 3.1: Esquema de integração proposto

Essa interface, no entanto, não deve ser ainda um ambiente interativo, ou seja, o usuário não poderá atuar no sistema ao mesmo em tempo que o plano é executado. Uma implementação desse tipo seria muito semelhante a um jogo eletrônico, e fugiria do escopo do trabalho. Ainda assim, é importante que exista alguma forma do projetista poder atuar nos sistema, de forma a poder representar eventos imprevistos para o planejador. Por isso, foi proposta inicialmente a criação, dentro do itSIMPLE, de uma interface de edição de estados do plano e replanejamento como forma de preparação para um projeto que contemple a interação. Assim, o projetista poderia interagir com o plano dentro do itSIMPLE,

encontrar um novo plano, para aí sim verificar o resultado da sua interação na interface de simulação tridimensional.

O esquema para esse processo de modificação de estados e replanejamento é mostrado na Figura 3.2, que contém um fluxograma desse processo. Como pode ser visto, após realizar as mudanças nos estados, é dada a opção ao usuário de replanejar o sistema. Assim, ele pode verificar o impacto das suas alterações no plano a partir do novo plano gerado.

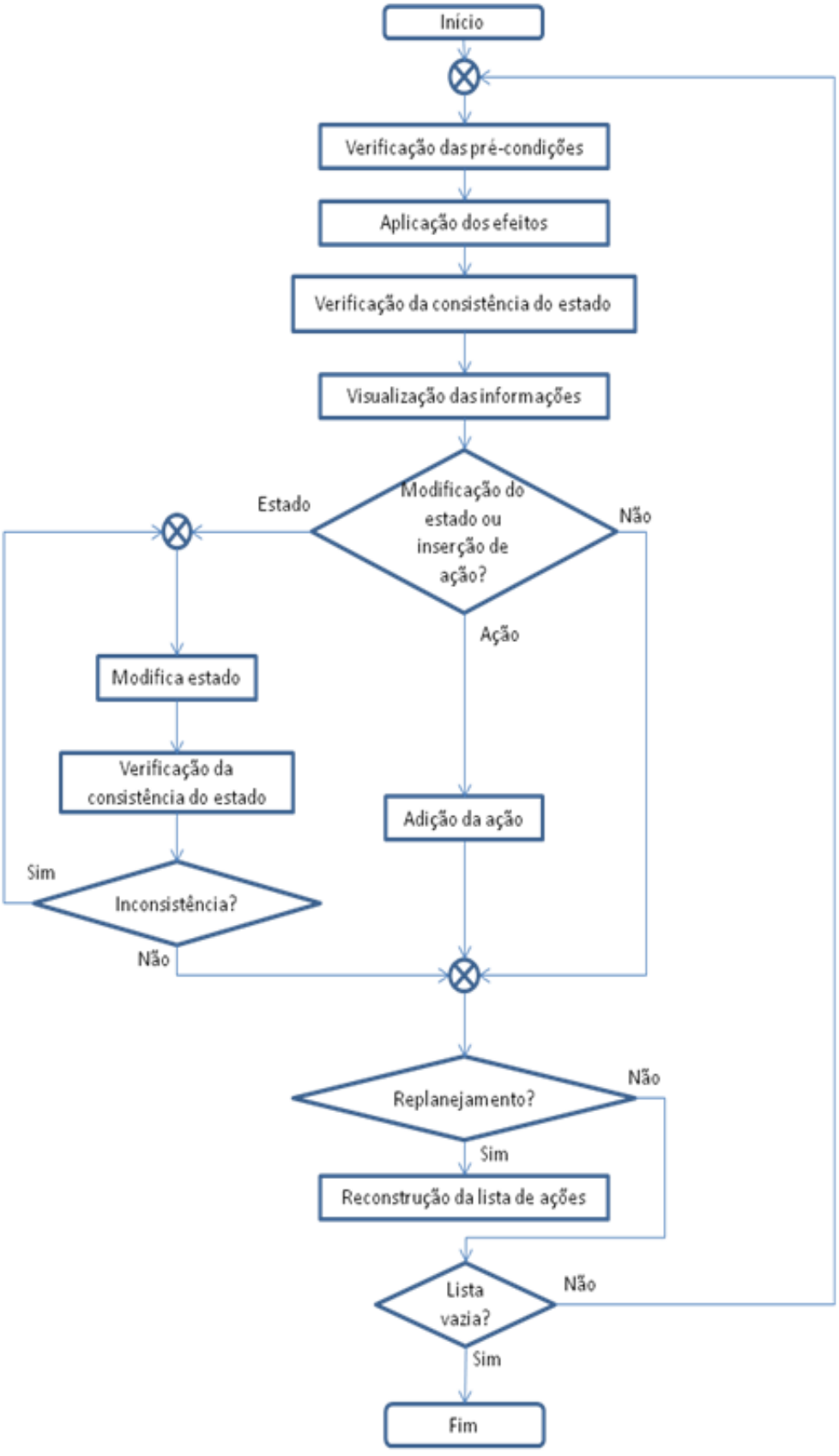

Figura 3.2: Processo de modificação de estados e replanejamento

4 Resultados Obtidos

4.1 Descrição do Funcionamento

O objetivo do sistema integrado entre itSIMPLE e interface é a verificação de possíveis erros encontrados em uma resposta de planejador e modelo quando este é aplicado a um ambiente modelado tridimensionalmente. Este modelo possibilita a verificação de erros de movimentação, geometrias e limites físicos como centros de gravidade, pontos de contato no choque e outros elementos que possam invalidar uma solução modelada.

O sistema de visualização e planejamento não é composto de apenas um modulo central que gerencia toda a sua utilização, mas sim de diversos sistemas que se comunicam e realizam estas tarefas. Podemos dividir basicamente o sistema em duas grandes interfaces, a de planejamento e a de visualização, uma dependente do itSIMPLE e a outra do Blender.

O processo de uso deste sistema pode ser dividido nas seguintes fases:

- criação do modelo no itSIMPLE;
- planejamento;
- geração de arquivo de saída com plano e interface no itSIMPLE;
- criação automática de ambiente tridimensional no software Blender;
- animação do ambiente.

O processo de modelagem no itSIMPLE é feito utilizando-se a sua interface UML. Os diagramas mais importantes são o de classes, o de estados e o de objetos. No primeiro, é feita uma descrição abstrata do domínio, em que as classes representam os tipos de objetos que farão parte do problema. Com as classes, definem-se as características de cada tipo; por exemplo, uma classe Robô, que representa um tipo de agente do problema, deve ter informações como a posição do robô, sua orientação etc., bem como as ações que um robô pode executar,

como movimentar-se em uma determinada direção. Um exemplo de diagrama de classes pode ser visto na Figura 2.4 (página 20).

No diagrama de estados são definidas as condições para a execução de ações. Assim, os estados que cada tipo de objeto pode ter são definidos, bem como as ações que levam de um estado a outro, com suas pré e pós condições. A Figura 4.1 ilustra um exemplo de diagrama de estados.

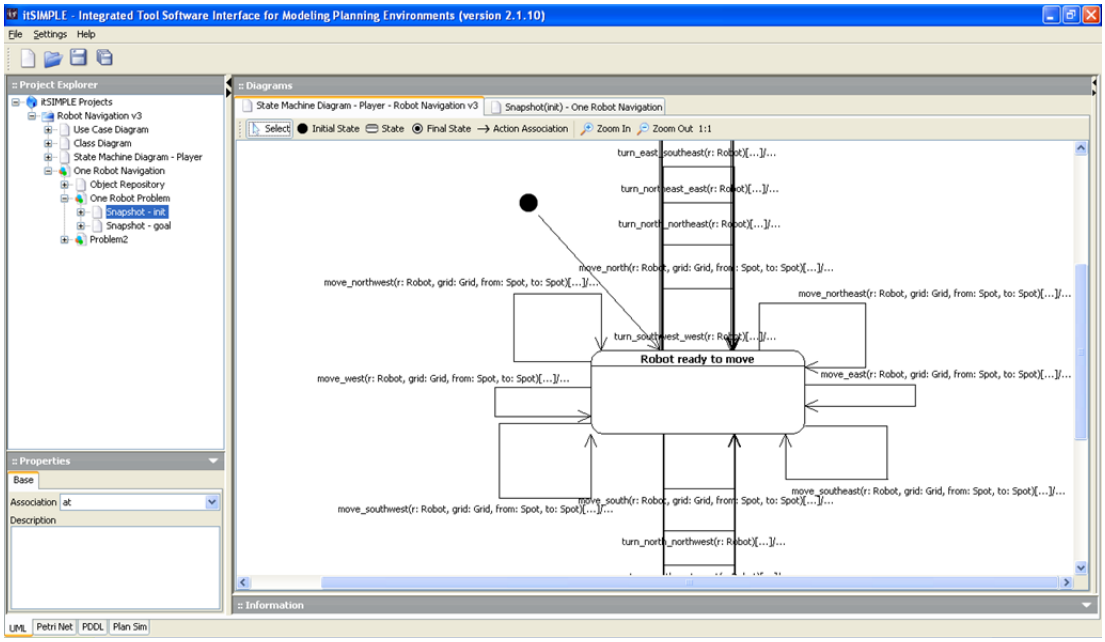

Figura 4.1: Exemplo de diagrama de estados no itSIMPLE

Por fim, nos diagramas de objetos são definidas as situações que serão resolvidas pelo planejador. Dessa forma, são criados objetos, cada um de um tipo oriundo do diagrama de classes, e suas propriedades são configuradas. No exemplo da classe Robô, um objeto da classe robô teria uma posição e uma orientação definidas. A Figura 4.2 mostra um exemplo de diagrama de objetos.

Com isso são criadas as cenas final e inicial, definindo um problema, a ser resolvido pelo planejador. O itSIMPLE possui um mecanismo interno de chamada a planejadores, bem como uma interface de visualização e análise de planos. Essa interface também utiliza a linguagem UML (com diagramas de objetos) para visualização dos planos, o que pode ser bastante útil. Em muitos casos, todavia, uma interface tridimensional pode oferecer ao usuário um poder de análise muito maior, o que reitera a proposta deste trabalho de uma solução que utiliza o conceito de Prototipagem Virtual.

Com o fim da descrição do modelo e do processo de planejamento, o itSIMPLE gera uma solução descrita em passos para o caminho a ser desenvolvido pelo elemento e uma visão de como o espaço será descrito. Esta solução é ligada ao

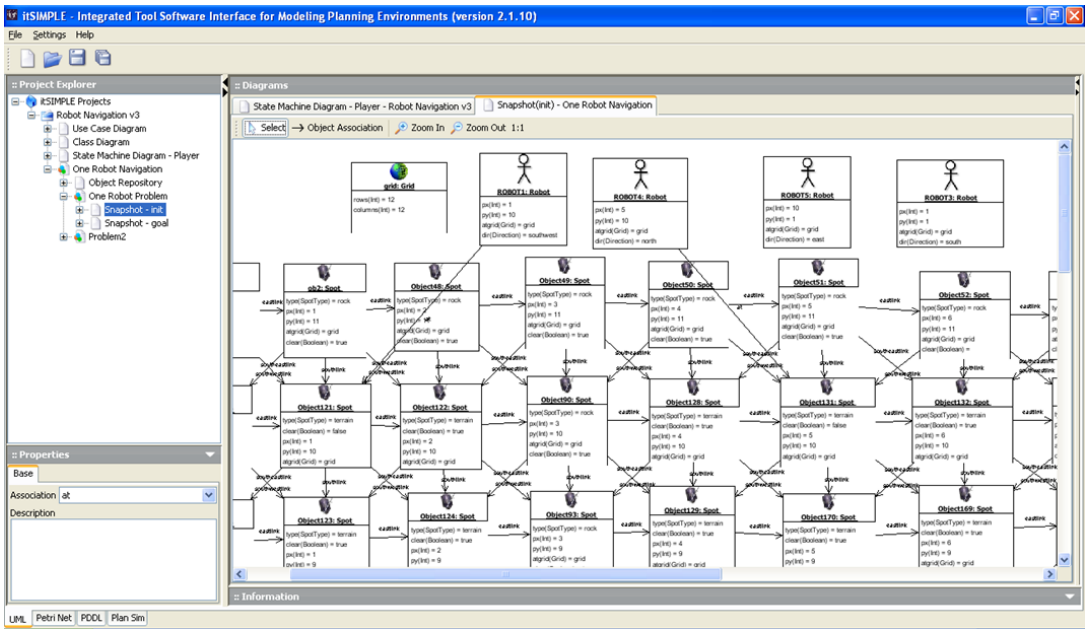

Figura 4.2: Exemplo de diagrama de objetos (cena inicial) no itSIMPLE

Blender por dois arquivos separados onde um irá descrever o plano e o outro irá descrever o ambiente e seus elementos, como obstáculos e atores. Isso habilita a construção automática do Protótipo Virtual, bem como a animação dos agentes. O Apêndice A mostra um plano gerado no itSIMPLE para resolução do problema aqui proposto.

Um script criado para ser utilizado em conjunto com o software Blender irá gerar todo o ambiente tridimensional, com três elementos principais: espaço físico, obstáculos e atores. O espaço físico é basicamente o suporte de todo o sistema, no caso analisado por este trabalho, um plano. Em um modelo mais complexo poderia ser uma geografia de terreno acidentado ou um material áspero. Este objeto, diferente dos outros elementos é sempre um único elemento criado e, juntamente com os atores, é obrigatório.

Os obstáculos são todos elementos que bloqueiam a passagem dos atores e demonstram algumas das restrições planejadas no modelo. Podem por exemplo, representar árvores, rios, pedras em um espaço na natureza, ou podem demonstrar máquinas, colunas e outras estruturas dentro de uma planta. Todos possuem junto com a sua definição (tipo que será criado) a posição que está no ambiente. Este tipo de elemento não é ativo, está sempre estático no ambiente.

Os últimos elementos, os atores, são aqueles que executam efetivamente o plano. São diferenciados dos outros elementos, pois possuem scripts que lêem o plano e permitem a sua animação no cenário sendo, por isso, os de composição mais complexa. Por exemplo, podem representar um trabalhador no chão de

fábrica, um robô na superfície lunar, um bombeiro nos destroços de um incêndio e outros elementos que são ativos no espaço de representação da realidade. Dado a sua complexidade, são os que têm a modelagem mais complicada, por isso deve-se ter uma atenção redobrada sobre estes elementos da visualização do plano. A Figura 4.3 mostra um cenário criado automaticamente no Blender, com o plano, obstáculos e agentes.

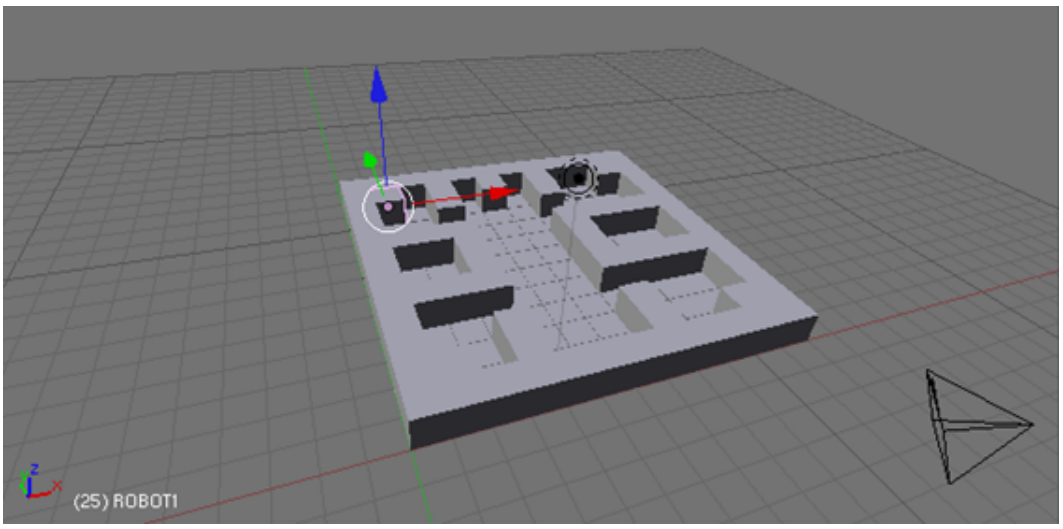

Figura 4.3: Cenário criado automaticamente no Blender

A solução adotada permite não apenas um agente, mas diversos. O planejador, quando da geração do plano, é quem se encarrega de evitar as colisões entre agentes, visto que as ações modeladas inibem esse tipo de erro com suas pré-condições. No entanto, é importante verificar neste momento se elementos geométricos de um ator não criam sobreposição sobre outro ator, pois dado que ambos possuem além do seu raio de movimentação no terreno, um raio de movimentação das ações.

Por exemplo, o espaço ocupado por um robô não é apenas o de sua posição default (também chamada de posição de manutenção), mas de uma área de acesso chamada de envelope que demonstra todo o seu raio de ação devido aos graus de liberdade que o braço (ou braços) permite. Os movimentos permitidos no modelo explorado por esse trabalho para um ator são a translação bidimensional no plano e a rotação, o que permite que ele se desloque para qualquer posição em oito direções, sempre orientado para o sentido de movimentação correto.

A última fase consiste na animação do plano e na verificação da movimentação. Nesta fase, os elementos se deslocam pelo plano demonstrando a trajetória que um ator pré-estabelecido no modelo realizaria. É nesta fase que se verifica se o ator realiza algum tipo de movimentação que cria conflitos com o ambiente e obstáculos. Durante esta fase há a possibilidade de se modificar o posiciona-

mento da câmera para uma melhor visualização dos pontos de interesse. Este posicionamento se dá via setas do teclado e mouse para rotação.

É nesse momento que o projetista tem a oportunidade de verificar o protótipo criado, residindo aí a vantagem de se utilizar a Prototipagem Virtual. No caso deste trabalho, ele pode verificar se o plano gerado é consistente com o ambiente onde os atores vão se movimentar, verificando colisões ou restrições de tamanho (como um ator muito largo tentando passar por um corredor estreito). Assim, se o plano for utilizado para o comando dos agentes no mundo real, ele pode saber de antemão se aquele plano é válido, evitando problemas e reduzindo custos.

Apesar da necessidade de mais do que uma família de software em uso, as diversas fases do projeto se integram de forma simples e são poucas as ações necessárias por parte dos usuários entre uma fase e outra. Há possibilidade de no futuro o software se integrar em apenas um pacote, porém um trabalho profundo no ambiente tridimensional e de física que está incorporado ao software Blender deve ser criado para esta independência. Este não é um trabalho simples, mas possibilitaria um ambiente ainda mais integrado e com funções especializadas em relação ao sistema de visualização de planos.

4.2 Arquivos de Saída Gerados pelo itSIMPLE

A comunicação entre itSimple e a interface gerada no Blender se dá por dois arquivos contendo texto no formato ASCII e extensão txt. Um dos arquivos se chama dominio.txt e neste está toda a informação sobre a criação dos elementos do cenário que consta na representação da realidade. O outro arquivo do mesmo tipo é o plano.txt. Neste arquivo estão todos os passos para a movimentação do ator (ou atores). O arquivo domínio é composto de três partes: Domínio, Obstáculo e Ator. Todas as partes são iniciadas pela chamada "Start_XXXXXX" e finalizam por "End_XXXXXX". Isto serve para controle do script, pois alguns elementos (como os obstáculos e atores) são criados mais do que uma vez e apenas a chamada End_XXXXXX informa que os elementos daquele tipo não serão mais criados.

Na fase de criação do elemento Domínio, no caso em estudo, um plano com dimensões especificadas é criado. A entrada é do tipo "Plano,100,100". Isto significa que um plano com dimensões 100 unidades por 100 unidades será criado. O plano é deslocado em relação a origem do sistema para que o canto inferior esquerdo da geometria fique posicionado na origem do sistema de coordenadas do ambiente Blender. Isto existe para que fique mais claro o entendimento entre

o sistema utilizado no modelo e no Blender, sendo a origem de ambos no canto inferior esquerdo e em ambos crescente para a direita e para cima. O plano recebe ainda uma propriedade, que nada mais é do que uma variável que pode ser acessada por outros elementos do sistema. Essa propriedade indica a quantidade de agentes do modelo, e é utilizada internamente pelo script de animação.

Na segunda parte do arquivo, são criados os obstáculos. Estes podem ser de diversos tipos e tamanhos. O elemento obstáculo é da forma "ROCK,10,10", onde o primeiro elemento é o tipo de obstáculo, o segundo a posição em x e o terceiro posição em y. É importante ressaltar que informações sobre a geometria, como o tamanho do elemento não é definida nesta parte, mas sim no script de geração de cenário quando o tipo de elemento é definido.

Como exemplo disto, seria possível ter os seguintes tipos de obstáculo "Rock": "BigRock" e "SmallRock", onde o primeiro teria uma geometria que ocuparia um espaço de 4x4 unidades de tamanho e o segundo 1x1 unidade. Depois de definido o tipo, o elemento é deslocado para a posição do espaço em relação ao plano definido no domínio de forma discretizada. Esta discretização ocorre pois as geometrias não necessariamente necessitam ter o tamanho que será usado para movimentação no plano. Isto significa que um plano 100x100 pode ser dividido em um tabuleiro 10x10, onde as movimentações ocorram de 10 em 10 unidades. Logo, um objeto em vez de ser indicado como posicionado em 50,30, pode ser posicionado em 5,3. Isto facilita a modelagem do problema, pois cria um espaço visto como um tabuleiro, o que tanto para programação quanto para entendimento do usuário é facilitado. Cada elemento Obstáculo é adicionado a um grupo de elementos do tipo Obstáculo durante a sua criação. Ao fim do último elemento, a chamada de finalização passa para a próxima fase da criação de elementos: os atores.

Os atores são os elementos mais complexos da geração de cenários. Eles possuem diversos tipos de características únicas que os tornam especiais. Na sua essência, a criação é semelhante à de um obstáculo, um elemento geométrico no espaço com posicionamentos. A chamada é do tipo "ROBOT1,1,1,NORTH", sendo o primeiro elemento da seqüência o nome do agente, o segundo a posição em x, terceiro a posição em y e por último uma característica que o difere da linha de criação de obstáculos, o sentido para qual está voltado. No Apêndice B, é mostrado o código que gerou o cenário da Figura 4.3.

A face em que está voltado é importante para ajudar na veracidade da movimentação. No modelo desenvolvido neste trabalho, foi considerado que para que um objeto se desloque em um sentido, o mesmo precisa estar voltado para esta

direção. Isto é geometricamente importante em vários casos. Se o modelo está representando um conjunto de rotas aéreas, o fato de um avião estar virado ou não para o lado do aeroporto é muito importante para o pouso. Um robô estar também com o sentido virado para o equipamento ou peça que o mesmo ir atuar também pode fazer a diferença. Além disso, o agente pode não caber em um caminho em que caberia se possuísse uma orientação diferente, de acordo com suas propriedades geométricas.

Existe uma limitação que ainda não torna o processo de criação do ambiente totalmente automático. Devido a uma limitação do software Blender, algumas ferramentas de controle dos elementos no ambiente de simulação não podem ser adicionadas via script, sendo possível a sua adição apenas de forma manual, como mostrado na Figura 4.4, na parte inferior da tela. Esta é uma limitação ainda não contornável. Estes elementos são de três tipos: sensores, atuadores e controladores. O Blender utiliza estes mecanismos para que, dentro de seu ambiente de simulação física, o elemento possa interagir com os outros elementos da cena.

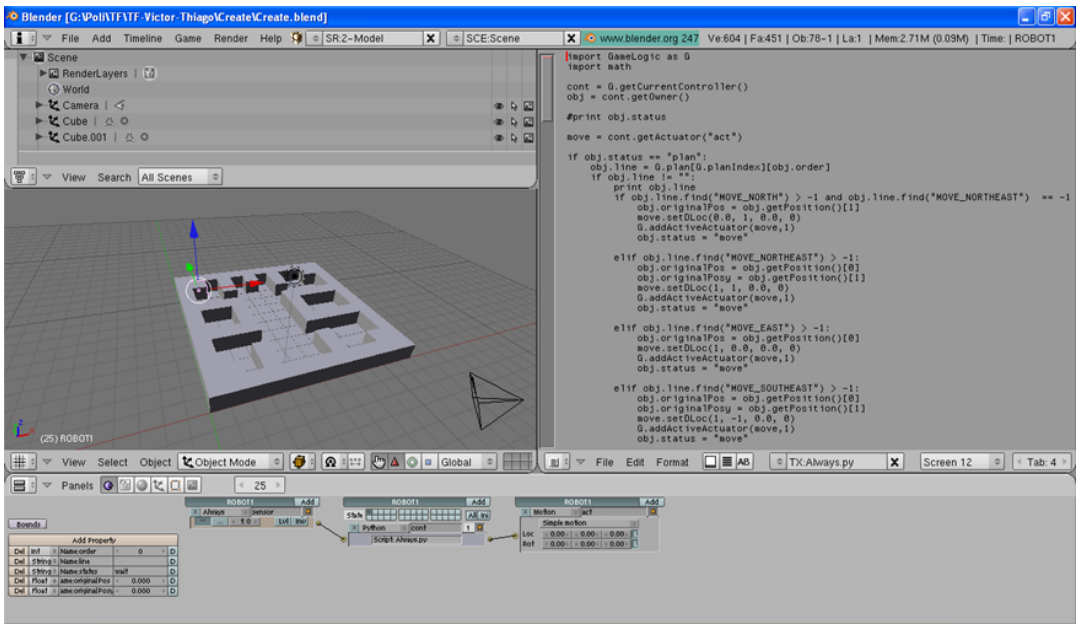

Figura 4.4: Janela de controles do Blender

A interação ocorre da seguinte forma: os sensores são os elementos que criam a interação entre os atores e o universo de simulação. Dentre as possibilidade de sensores que um objeto pode ter, estão incluídos entradas do teclado, proximidade com outros elementos, mudanças de estado (ou propriedade) ou mesmo um tipo de sensor que esteja sempre ativo, que fique a todo o momento sendo acionado (sensor do tipo *Always*, "sempre" em português).

Outro elemento de controle são os atuadores. Os atuadores criam as reações

dinâmicas do elemento no ambiente. Uma força, uma rotação, uma velocidade, entre outros, podem ser as reações físicas impostas em um elemento. É desta forma que conseguimos que, no ambiente de simulação, o elemento se desloque de um ponto para outro ou gire. A forma com que esta movimentação foi realizada neste projeto será explicada mais à frente.

O último elemento do conjunto é o controlador. O controlador une os sensores aos atuadores e verifica se as condições de um satisfazem a execução do outro. Isto ocorre da seguinte forma: um sensor recebe uma condição que altera o seu sinal para verdadeiro; neste momento o controlador (que pode ser de diversos tipos lógicos, como exemplo, AND (e) ou OR (ou)) verifica se a condição para ativar um atuador é suficiente. Um controlador do tipo AND, ligado a apenas um sensor e um atuador, logo que o sensor envia um sinal verdadeiro, aciona o atuador. Caso o mesmo controlador AND estivesse ligado a dois sensores e um controlador, a atuação ocorreria apenas quando ambos sensores fossem verdadeiros. Existe ainda um tipo de controlador que é imprescindível para este trabalho: o controlador do tipo Python.

Basicamente, toda a inteligência necessária para um elemento do tipo ator realizar a movimentação no espaço está localizada em seu controlador Python. O controlador Python permite ainda a movimentação da câmera, a atuação de um sensor preso ao plano para controle de diversas entidades, enfim, basicamente todo o funcionamento do ambiente de visualização está apoiado na inteligência descrita nos scripts Python anexados aos agentes do modelo.

Entretanto, como dito acima, a parte de adição de elementos de controle ainda não é um processo automatizado. A adição destes elementos faria com que o usuário não necessitasse de utilizar nenhum elemento da interface do Blender. É importante frisar que mesmo não sendo automática, esta passagem é muito simples e pode ser feita em poucos segundos mesmo pelo usuário mais inexperiente. Após a finalização da criação dos elementos do ambiente e atribuição dos elementos de controle aos atores, a movimentação pode ser iniciada, sendo apenas necessário iniciar o ambiente de simulação física do Blender.

4.3 Animação do Ambiente

Apesar de o conceito de animação ser algo muito amplo, neste trabalho será considerado de uma forma mais restrita. Esta restrição se dá na movimentação dos atores sobre um plano bidimensional e sua rotação em torno do eixo normal

ao plano (eixo z). Isto confere três graus de liberdade aos agentes. Existem muitos casos que podem ser analisados e visualizados com esses três graus:

- veículos auto-guiados (AGVs - Auto Guided Vehicle) em uma indústria;
- definição de rotas para tratores em colheitas;
- busca de saídas para "labirintos" planos;
- modelo caça e caçador;
- outras situações que possam ter sua movimentação transportada para o plano bidimensional.

O mecanismo para esta movimentação se dá, como descrito no capítulo anterior, por um script Python com toda a inteligência da movimentação anexa ao elemento Ator. Este script, que pode ser visto no Apêndice C, trabalha em conjunto com outro script que está anexado ao controle da geometria de base do ambiente (plano).

O ambiente, por meio do script anexado à geometria de base (este script será chamado doravante de God), lê o arquivo e inicializa uma lista de ações com os eventos de movimentação, que podem ser uma translação (linear) ou uma rotação. Esta lista não apenas incorpora as movimentações de um dos atores, mas de todos.

O script God tem também outra função, a de controlar as seqüências de eventos. O processo de animação trabalha em um sistema de turnos (ou passos). Um turno somente é finalizado quando todos os atores terminam a sua ação. É possível, no entanto, que um ator nem se movimente no turno, o que faz com que ele fique parado enquanto outros agentes se movimentam. O script God é mostrado no Apêndice D.

Cada ator verifica na sua linha de seqüência, criada pelo script God, a ação que irá executar e a realiza durante o turno. As ações possíveis são:

- deslocar para o norte: MOVE_NORTH;
- deslocar para o sul: MOVE_SOUTH;
- deslocar para o leste MOVE_EAST;
- deslocar para o oeste MOVE_WEST;
- deslocar para o nordeste MOVE_NORTHEAST;

- deslocar para o noroeste: MOVE_NORTWEST;
- deslocar para sudeste: MOVE_SOUTHEAST;
- deslocar para sudoeste: MOVE_SOUTHWEST;
- girar no sentido horário:

  TURN_NORTH_NORTHEAST;

  TURN_NORTHEAST_EAST;

  TURN_EAST_SOUTHEAST;

  TURN_SOUTHEAST_SOUTH;

  TURN_SOUTH_SOUTHWEST;

  TURN_SOUTHWEST_WEST;

  TURN_WEST_NORTHWEST;

  TURN_NORTHWEST_NORTH;

- girar no sentido anti-horário:

  TURN_NORTH_NORTHWEST;

  TURN_NORTHWEST_WEST;

  TURN_WEST_SOUTHWEST;

  TURN_SOUTHWEST_SOUTH;

  TURN_SOUTH_SOUTHEAST;

  TURN_SOUTHEAST_EAST;

  TURN_EAST_NORTHEAST;

  TURN_NORTHEAST_NORTH.

Estes movimentos disparam um atuador que compõe o grupo de controle dos atores e que quando direcionados realizam a movimentação a um sentido. Caso o elemento do tipo Ator não esteja com a face frontal virada para o caminho desejado, o mesmo deve girar antes de se mover. A distância percorrida é sempre uma quantidade fixa e deve se relacionar com a escala utilizada para definir a divisão discreta do plano geométrico para que um elemento sempre se desloque para o centro da próxima célula do plano.

Como foi mencionado anteriormente, no caso em que o ator termina o movimento antecipadamente a outros atores, ele deve aguardar o resto das outras

movimentações daquele passo para iniciar sua próxima movimentação (se houver). Isso ocorre para manter o sincronismo da animação, já que o planejador se encarrega de encontrar a melhor seqüência de ações e quais ações podem ser realizadas ao mesmo tempo.

5 Trabalhos Futuros

Um projeto onde o objetivo final é a criação de uma interface onde a realidade possa ser representada da forma mais clara possível é definitivamente um projeto ambicioso e complexo. A realidade possui um número de variáveis praticamente ilimitado e que dificilmente será coberto por algum ambiente de representação gráfica. Isto deixa claro que qualquer projeto que se propuser a este desafio será expandido a cada dia e das mais diversas formas. Como não é o objetivo neste momento a descrição de todas estas evoluções possíveis que a interface e o sistema de planejamento possam ter, é importante saber pelo menos o que é importante para que esta ferramenta se torne ainda mais interessante.

A primeira evolução fundamental seria a completa capacidade de se gerar cenários e atores sem a necessidade de intervenção do usuário. Isto traria maior independência à necessidade de se saber algo sobre o software Blender. Nem sempre os usuários estão dispostos a aprender, mesmo que minimamente, a utilizar uma ferramenta para um fim que não seja especificamente o dela. Para uma pessoa que está preocupada em softwares de planejamento, ter de lidar com um software de desenho tridimensional é uma grande oportunidade para se desmotivar. Por isso, esta liberdade ao Blender é importante.

Indo ao encontro desta idéia, a total eliminação do Blender no processo seria algo importante. Para isso, a interface tridimensional poderia ser encarada como mais um módulo do itSIMPLE, e deveria funcionar totalmente integrado à esta ferramenta, o que não estava no escopo deste trabalho, mas representaria um avanço importante.

Há também a necessidade de se importar elementos externos e de se criar uma biblioteca de elementos básicos para modelagem. Elementos simbólicos (como carros, aviões, navios e pessoas) e obstáculos (como montanhas, rios e árvores) poderiam constar em uma biblioteca básica, que permitisse ao usuário simular com qualidade o seu modelo, sem necessitar da criação de seus próprios modelos tridimensionais. Para isso, seria necessária uma evolução na fase de criação automática de ambiente com a inclusão da capacidade de importação de elementos

externos, além da própria criação dessa biblioteca com tipos de modelos tridimensionais básicos.

Existe ainda a possibilidade de se expandir a interface para visualização de informações em tempo real de diversas variáveis do sistema na tela de animação. O exemplo adotado neste trabalho não faz uso dessa interface com elementos de texto sendo mostrados em tela, já que essa é uma tarefa bastante custosa de se fazer no Blender, que não possui uma interface preparada para textos. A sua possibilidade, ainda assim, criaria um número de opções bastante grande para geração de relatórios interativos para problemas de planejamento, bem como facilitaria a observação das propriedades dos elementos do ambiente por parte do usuário.

A interação no ambiente de simulação é possível atualmente apenas de forma indireta, utilizando-se a interface de simulação de planos do itSIMPLE. Apesar de útil, o processo ideal seria abrir a possibilidade do usuário atuar no sistema em tempo real, ao mesmo tempo em que o modelo é simulado no Blender. Isso exigiria, além de uma interface tridimensional com sensores apropriados, uma integração total entre o modelo tridimensional e os planejadores, para que o replanejamento pudesse ser feito em tempo de execução. Essa é uma tarefa extremamente complexa e foi deixada, portanto, para trabalhos futuros, mas que tornaria a tarefa de análise muito mais rica e próxima da realidade.

6 Conclusões

A tecnologia da Prototipagem Virtual, com o uso de Protótipos Virtuais no processo de desenvolvimento de produtos é algo cada vez mais usado na engenharia. De forma especial, o desenvolvimento de produtos mecatrônicos é um campo bastante propício ao uso desse tipo de abordagem, visto que dispositivos ou ambientes mecatrônicos lidam com a integração de um grande número de peças e máquinas, mecânicas ou elétricas, além dos softwares de controle e monitoramento inerentes a qualquer operação.

Neste sentido, o presente trabalho procurou mostrar uma forma consistente de se aplicar a Prototipagem Virtual para análise de problemas de engenharia, combinando-se seus conceitos com a área de Planejamento Automático e análise de planos. Dessa forma, este trabalho procurou demonstrar que essas duas áreas podem se completar, criando um ambiente rico para a resolução de problemas reais de engenharia.

De forma a demonstrar que essa proposta é viável e coerente, o trabalho incluiu o desenvolvimento de uma interface gráfica capaz de simular a execução de planos. Assim, a interface desenvolvida na ferramenta de modelagem 3D chamada Blender se comunica com o itSIMPLE, software desenvolvido pelo DesignLab da Escola Politécnica, e que pode ser usado tanto para modelagem de domínios e problemas, bem como para obtenção e análise de planos. Os planos, por sua vez, são obtidos pela integração entre o itSIMPLE e planejadores.

Com isso, foi possível criar um ambiente integrado entre o itSIMPLE e a interface tridimensional para análise de um tipo específico de problema, muito comum na engenharia: a movimentação de agentes em um plano repleto de obstáculos. Procurou-se com esse problema específico demonstrar toda a potencialidade do uso de Protótipos Virtuais juntamente com o Planejamento Automático dentro da engenharia mecatrônica.

Com a integração estabelecida, o ambiente desenvolvido é capaz de criar automaticamente o cenário do problema a partir da modelagem realizada no

itSIMPLE. Em seguida, com pouca intervenção do usuário, o plano gerado pelo planejador pode ser executado e a movimentação planejada dos agentes pode ser visualizada.

A partir daí, o projetista pode realizar inúmeras análises, como o da disposição dos obstáculos (que podem ser máquinas ou estações em um chão de fábrica), o tamanho dos agentes e sua interação com o ambiente, as interações entre vários agentes, entre outras. Esse tipo de análise evidencia a utilidade do ambiente e, por conseguinte, o potencial desse tipo de abordagem. Apesar da interface tridimensional não ser interativa, o que fugiria do escopo do trabalho, é possível que o usuário modifique o ambiente durante a execução de um plano de forma indireta, antes da sua simulação no Blender. Para isso, foi adicionada a possibilidade, dentro da interface de simulação de planos do itSIMPLE, de se atuar no ambiente durante a execução do plano. Quando isso acontece, o usuário pode replanejar o sistema, o que gera um novo plano a partir do ponto de atuação. Assim, esse novo plano pode ser executado na interface tridimensional e ele pode verificar o resultado de sua atuação.

Assim, com o fim do presente trabalho, temos como resultado uma interface tridimensional integrada à ferramenta de modelagem que é o itSIMPLE. Os objetivos foram atingidos na medida em que foi demonstrado com sucesso que a Prototipagem Virtual é uma abordagem bastante consistente para análise de problemas de engenharia, sejam esses problemas oriundos de desenvolvimento de produtos, seja na aplicação desses produtos em algum ambiente.

Apêndice A – Exemplo de Plano

```
0:  TURN_NORTH_NORTHEAST ROBOT01
0:  TURN_NORTH_NORTHEAST ROBOT04
1:  TURN_NORTHEAST_EAST ROBOT01
1:  TURN_NORTHEAST_EAST ROBOT04
2:  TURN_EAST_SOUTHEAST ROBOT01
2:  TURN_EAST_SOUTHEAST ROBOT04
3:  MOVE_SOUTHEAST ROBOT01
4:  TURN_SOUTHEAST_SOUTH ROBOT01
4:  MOVE_SOUTHEAST ROBOT04
5:  MOVE_NORTH ROBOT02
5:  TURN_SOUTHEAST_SOUTH ROBOT04
6:  TURN_NORTH_NORTHEAST ROBOT02
7:  MOVE_NORTHEAST ROBOT02
8:  TURN_NORTHEAST_EAST ROBOT02
8:  MOVE_SOUTH ROBOT01
9:  MOVE_SOUTH ROBOT04
9:  TURN_SOUTH_SOUTHEAST ROBOT01
10:  TURN_SOUTHEAST_EAST ROBOT01
10:  MOVE_EAST ROBOT02
11:  MOVE_EAST ROBOT01
12:  MOVE_EAST ROBOT01
13:  MOVE_EAST ROBOT02
13:  TURN_EAST_SOUTHEAST ROBOT01
14:  TURN_EAST_NORTHEAST ROBOT02
14:  MOVE_SOUTH ROBOT04
14:  TURN_SOUTHEAST_SOUTH ROBOT01
15:  MOVE_NORTHEAST ROBOT02
16:  TURN_NORTHEAST_NORTH ROBOT02
17:  MOVE_NORTH ROBOT02
18:  MOVE_SOUTH ROBOT01
```

```
19:  MOVE_SOUTH ROBOT04
20:  MOVE_SOUTH ROBOT04
21:  MOVE_SOUTH ROBOT01
21:  TURN_SOUTH_SOUTHWEST ROBOT04
22:  MOVE_SOUTH ROBOT01
23:  MOVE_NORTH ROBOT02
23:  TURN_SOUTH_SOUTHEAST ROBOT01
24:  TURN_NORTH_NORTHEAST ROBOT02
24:  MOVE_SOUTHWEST ROBOT04
25:  MOVE_NORTHEAST ROBOT02
26:  TURN_NORTHEAST_NORTH ROBOT02
27:  MOVE_NORTH ROBOT02
28:  MOVE_SOUTHWEST ROBOT04
28:  TURN_NORTH_NORTHEAST ROBOT02
29:  MOVE_SOUTHEAST ROBOT01
29:  TURN_SOUTHWEST_WEST ROBOT04
29:  TURN_NORTHEAST_EAST ROBOT02
30:  MOVE_SOUTHEAST ROBOT01
31:  TURN_SOUTHEAST_EAST ROBOT01
32:  MOVE_EAST ROBOT01
33:  MOVE_EAST ROBOT02
34:  MOVE_WEST ROBOT04
35:  MOVE_WEST ROBOT04
36:  MOVE_EAST ROBOT01
36:  TURN_WEST_SOUTHWEST ROBOT04
37:  MOVE_EAST ROBOT02
37:  TURN_EAST_SOUTHEAST ROBOT01
38:  TURN_EAST_NORTHEAST ROBOT02
38:  MOVE_SOUTHWEST ROBOT04
39:  MOVE_NORTHEAST ROBOT02
39:  TURN_SOUTHWEST_SOUTH ROBOT04
40:  MOVE_SOUTHEAST ROBOT01
41:  MOVE_SOUTH ROBOT04
42:  MOVE_NORTHEAST ROBOT02
43:  MOVE_SOUTHEAST ROBOT01
```

Apêndice B – Codificação do Cenário

```
Start_Dominio
12,12
End_Dominio


Start_Obstaculo
ROCK,1,11
ROCK,0,11
ROCK,2,11
ROCK,3,11
ROCK,4,11
ROCK,5,11
ROCK,6,11
ROCK,7,11
ROCK,8,11
ROCK,9,11
ROCK,10,11
ROCK,11,11
ROCK,0,9
ROCK,0,8
ROCK,0,7
ROCK,0,6
ROCK,0,5
ROCK,0,4
ROCK,0,3
ROCK,0,2
ROCK,0,1
ROCK,0,0
ROCK,1,0
ROCK,0,10
```

```
ROCK,2,0
ROCK,3,0
ROCK,4,0
ROCK,5,0
ROCK,6,0
ROCK,7,0
ROCK,8,0
ROCK,10,0
ROCK,9,0
ROCK,11,0
ROCK,11,10
ROCK,11,9
ROCK,11,8
ROCK,11,7
ROCK,11,6
ROCK,11,5
ROCK,11,4
ROCK,11,3
ROCK,11,2
ROCK,11,1
ROCK,3,10
ROCK,7,9
ROCK,7,10
ROCK,3,9
ROCK,7,2
ROCK,7,1
ROCK,3,2
ROCK,3,1
ROCK,1,7
ROCK,2,7
ROCK,3,6
ROCK,3,5
ROCK,3,7
ROCK,3,4
ROCK,1,4
ROCK,2,4
ROCK,10,7
ROCK,9,7
```

```
ROCK,8,7
ROCK,7,7
ROCK,7,6
ROCK,7,5
ROCK,7,4
ROCK,8,4
ROCK,9,4
ROCK,10,4
End_Obstaculo


Start_Actor
ROBOT01,1,10,NORTH
ROBOT04,5,10,NORTH
ROBOT02,1,1,NORTH
End_Actor
```

Apêndice C – Script Object.py

```
import GameLogic as G
import math
cont = G.getCurrentController()
obj = cont.getOwner()
move = cont.getActuator(''act'')
if obj.status == ''plan'':
    obj.line = G.plan[G.planIndex][obj.order]
    if obj.line != '''':
        if obj.line.find(''MOVE_NORTH'') > -1 and
        obj.line.find(''MOVE_NORTHEAST'') == -1 and
        obj.line.find(''MOVE_NORTHWEST'') == -1:
            obj.originalPos = obj.getPosition()[1]
            move.setDLoc(0.0, 0.5, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''move''
        elif obj.line.find(''MOVE_NORTHEAST'') > -1:
            obj.originalPos = obj.getPosition()[0]
            obj.originalPosy = obj.getPosition()[1]
            move.setDLoc(0.1, 0.1, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''move''
        elif obj.line.find(''MOVE_EAST'') > -1:
            obj.originalPos = obj.getPosition()[0]
            move.setDLoc(0.1, 0.0, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''move''
        elif obj.line.find(''MOVE_SOUTHEAST'') > -1:
            obj.originalPos = obj.getPosition()[0]
            obj.originalPosy = obj.getPosition()[1]
            move.setDLoc(0.1, -0.1, 0.0, 0)
```

```
        G.addActiveActuator(move,1)
        obj.status = ''move''
    elif obj.line.find(''MOVE_SOUTH'') > -1 and
    obj.line.find(''MOVE_SOUTHEAST'') == -1 and
    obj.line.find(''MOVE_SOUTHWEST'') == -1:
        obj.originalPos = obj.getPosition()[1]
        move.setDLoc(0.0, -0.1, 0.0, 0)
        G.addActiveActuator(move,1)
        obj.status = ''move''
        elif obj.line.find(''MOVE_SOUTHWEST'') > -1:
        obj.originalPos = obj.getPosition()[0]
        obj.originalPosy = obj.getPosition()[1]
        move.setDLoc(-0.1, -0.1, 0.0, 0)
        G.addActiveActuator(move,1)
        obj.status = ''move''
    elif obj.line.find(''MOVE_WEST'') > -1:
        obj.originalPos = obj.getPosition()[0]
        move.setDLoc(-0.1, 0.0, 0.0, 0)
        G.addActiveActuator(move,1)
        obj.status = ''move''
    elif obj.line.find(''MOVE_NORTHWEST'') > -1:
        obj.originalPos = obj.getPosition()[0]
        obj.originalPosy = obj.getPosition()[1]
        move.setDLoc(-0.1, 0.1, 0.0, 0)
        G.addActiveActuator(move,1)
        obj.status = ''move''
    elif obj.line.find(''TURN_NORTH_NORTHEAST'') > -1 or
    obj.line.find(''TURN_NORTHEAST_EAST'') > -1 or
    obj.line.find(''TURN_EAST_SOUTHEAST'') > -1 or
    obj.line.find(''TURN_SOUTHEAST_SOUTH'') > -1 or
    obj.line.find(''TURN_SOUTH_SOUTHWEST'') > -1 or
    obj.line.find(''TURN_SOUTHWEST_WEST'') > -1 or
    obj.line.find(''TURN_WEST_NORTHWEST'') > -1 or
    obj.line.find(''TURN_NORTHWEST_NORTH'') > -1:
    obj.originalPos = math.acos(obj.getOrientation()[0][0])
    if obj.getOrientation()[0][1] < 0:
        obj.originalPos = 2*math.pi - obj.originalPos
        move.setDRot(0.0, 0.0, -0.05, 1)
```

```
            G.addActiveActuator(move,1)
            obj.status = ''move''
        elif obj.line.find(''TURN_NORTH_NORTHWEST'') > -1 or
        obj.line.find(''TURN_NORTHWEST_WEST'') > -1 or
        obj.line.find(''TURN_WEST_SOUTHWEST'') > -1 or
        obj.line.find(''TURN_SOUTHWEST_SOUTH'') > -1 or
        obj.line.find(''TURN_SOUTH_SOUTHEAST'') > -1 or
        obj.line.find(''TURN_SOUTHEAST_EAST'') > -1 or
        obj.line.find(''TURN_EAST_NORTHEAST'') > -1 or
        obj.line.find(''TURN_NORTHEAST_NORTH'') > -1:
            obj.originalPos = math.acos(obj.getOrientation()[0][0])
            if obj.getOrientation()[0][1] > 0:
                obj.originalPos = 2*math.pi - obj.originalPos
            move.setDRot(0.0, 0.0, 0.05, 1)
            G.addActiveActuator(move,1)
            obj.status = ''move''
        elif obj.line.find(''WAIT'') > -1:
            obj.status = ''wait''
            G.ok[obj.order] = True
if obj.status == ''move'':
    pos = obj.getPosition()
    if obj.line.find(''MOVE_NORTH'') > -1 and
    obj.line.find(''MOVE_NORTHEAST'') == -1 and
    obj.line.find(''MOVE_NORTHWEST'') == -1:
        if pos[1] - obj.originalPos > 1:
            move.setDLoc(0.0, 0.0, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
            G.ok[obj.order] = True
    elif obj.line.find(''MOVE_NORTHEAST'') > -1:
        if pos[0] - obj.originalPos > 1
        and pos[1] - obj.originalPosy > 1:
            move.setDLoc(0.0, 0.0, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
            G.ok[obj.order] = True
    elif obj.line.find(''MOVE_EAST'') > -1:
        if pos[0] - obj.originalPos > 1:
```

```
            move.setDLoc(0.0, 0.0, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
            G.ok[obj.order] = True
    elif obj.line.find(''MOVE_SOUTHEAST'') > -1:
        if pos[0] - obj.originalPos > 1 and
        pos[1] - obj.originalPosy < -1:
            move.setDLoc(0.0, 0.0, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
            G.ok[obj.order] = True
    elif obj.line.find(''MOVE_SOUTH'') > -1 and
    obj.line.find(''MOVE_SOUTHEAST'') == -1 and
    obj.line.find(''MOVE_SOUTHWEST'') == -1:
        if pos[1] - obj.originalPos < -1:
            move.setDLoc(0.0, 0.0, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
            G.ok[obj.order] = True
    elif obj.line.find(''MOVE_SOUTHWEST'') > -1:
        if pos[0] - obj.originalPos < -1 and
        pos[1] - obj.originalPosy < -1:
            move.setDLoc(0.0, 0.0, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
            G.ok[obj.order] = True
    elif obj.line.find(''MOVE_WEST'') > -1:
        if pos[0] - obj.originalPos < -1:
            move.setDLoc(0.0, 0.0, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
            G.ok[obj.order] = True
    elif obj.line.find(''MOVE_NORTHWEST'') > -1:
        if pos[0] - obj.originalPos < -1 and
        pos[1] - obj.originalPosy > 1:
            move.setDLoc(0.0, 0.0, 0.0, 0)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
```

```
        G.ok[obj.order] = True
    elif obj.line.find(''TURN_NORTH_NORTHEAST'') > -1 or
    obj.line.find(''TURN_NORTHEAST_EAST'') > -1 or
    obj.line.find(''TURN_EAST_SOUTHEAST'') > -1 or
    obj.line.find(''TURN_SOUTHEAST_SOUTH'') > -1 or
    obj.line.find(''TURN_SOUTH_SOUTHWEST'') > -1 or
    obj.line.find(''TURN_SOUTHWEST_WEST'') > -1 or
    obj.line.find(''TURN_WEST_NORTHWEST'') > -1 or
    obj.line.find(''TURN_NORTHWEST_NORTH'') > -1:
        angle = math.acos(obj.getOrientation()[0][0])
        if obj.getOrientation()[0][1] < 0:
            angle = 2*math.pi - angle
        if obj.line.find(''TURN_NORTH_NORTHEAST'') > -1 and
        abs(angle - obj.originalPos) > 6:
            obj.originalPos = 0
        if angle - obj.originalPos > math.pi/4 or
        (obj.line.find(''TURN_NORTHWEST_NORTH'') > -1 and
        obj.originalPosy < 0 and obj.getOrientation()[0][1] >= 0):
            move.setDRot(0.0, 0.0, 0.0, 1)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
            G.ok[obj.order] = True
        obj.originalPosy = obj.getOrientation()[0][1]
    elif obj.line.find(''TURN_NORTH_NORTHWEST'') > -1 or
    obj.line.find(''TURN_NORTHWEST_WEST'') > -1 or
    obj.line.find(''TURN_WEST_SOUTHWEST'') > -1 or
    obj.line.find(''TURN_SOUTHWEST_SOUTH'') > -1 or
    obj.line.find(''TURN_SOUTH_SOUTHEAST'') > -1 or
    obj.line.find(''TURN_SOUTHEAST_EAST'') > -1 or
    obj.line.find(''TURN_EAST_NORTHEAST'') > -1 or
    obj.line.find(''TURN_NORTHEAST_NORTH'') > -1:
        angle = math.acos(obj.getOrientation()[0][0])
        if obj.getOrientation()[0][1] > 0:
            angle = 2*math.pi - angle
        if obj.line.find(''TURN_NORTH_NORTHWEST'') > -1 and
        abs(angle - obj.originalPos) > 6:
            obj.originalPos = 0
        if angle - obj.originalPos > math.pi/4 or
```

```
        (obj.line.find(''TURN_NORTHEAST_NORTH'') > -1 and
        obj.originalPosy > 0 and obj.getOrientation()[0][1] <= 0):
            move.setDRot(0.0, 0.0, 0.0, 1)
            G.addActiveActuator(move,1)
            obj.status = ''wait''
            G.ok[obj.order] = True
        obj.originalPosy = obj.getOrientation()[0][1]
if obj.status == ''wait'':
    if G.next:
        obj.status = ''plan''
        G.ok[obj.order] = False
```

Apêndice D – Script God.py

```
import GameLogic as G
import Blender
cont = G.getCurrentController()
obj = cont.getOwner()
if hasattr(G, ''file'') == 0:
   filestr = 'C:/plan.txt'
   G.file = open(filestr)
   G.nActions = -1
   #create an empty plan
   G.plan = []
   while True:
      line = G.file.readline()
      if not line:
      break
   line = line.replace(''('','''')
   line = line.replace('')'','''')
   tokens = line.split()
   index = int(tokens[0].split(':'')[0])
   #update G.nLines, when necessary
   if index > G.nActions:
      G.nActions = index
   try:
      step = G.plan[index]
   except:
   # create a list filled with ''wait''
      step = [''WAIT'']*obj.nObjects
      G.plan.insert(index, step)
   #get the robot order from his name
   robots = list(Blender.Group.Get(''ROBOTS'').objects)
   for robot in robots:
```

```
            if robot.name == tokens[2]:
                order = robot.getProperty(''order'').getData()
                break
            #insert the action in the right position
            step[order] = tokens[1]
        # the number of actions is the highest index plus 1
        G.nActions += 1
if hasattr(G, ''ok'') == 0:
    G.ok = [True]*obj.nObjects
if hasattr(G, ''next'') == 0:
    G.next = False
if hasattr(G, ''planIndex'') == 0:
    G.planIndex = -1
if G.planIndex < G.nActions - 1:
    if not False in G.ok:
        #this means the objects have already moved
        #increment planIndex
        G.planIndex += 1
        #set G.next to True
        G.next = True
    if not True in G.ok:
        #this means the objects are starting to move
        G.next = False
else:
    #stop moving
    G.next = False
```

Referências

1 CECIL, J.; KANCHANAPIBOON, A. **Virtual engineering approaches in product and process design.** The International Journal of Advanced Manufacturing Technology, Volume 31, Numbers 9-10, Londres: Springer, Janeiro 2007 , pp. 846-856 (11)

2 SHEN, Q., GAUSEMEIER, J., BAUCH, J., RADKOWSKI, R. **A cooperative virtual prototyping system for mechatronic solution elements based assembly.** Advanced Engineering Informatics 19(2): 169-177 (2005)

3 GHALLAB M.; NAU D.; TRAVERSO P.. **Automated Planning: Theory and Practice.** San Francisco: Morgan Kaufmann, 2004. 635p.

4 MCCLUSKEY, T. L., SIMPSON, R. M. **Tool Support for Planning and Plan Analysis within Domains embodying Continuous Change.** Workshop on Plan Analysis and Management, ICAPS 2006, Cumbria, UK.

5 DALEY, P., FRANK, J., IATAURO M., MCGANN, C.; TAYLOR, W. **PlanWorks: A Debugging Environment for Constraint-Based Planning Systems.** Entry in the 1st International Knowledge Engineering Competition, 2005

6 VAQUERO, T. S. **itSIMPLE: Ambiente Integrado de Modelagem e Análise de Domínios de Planejamento Automático.** Dissertação de Mestrado, Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, EPUSP. 2007.

7 VAQUERO, T. S., ROMERO, V. M. C., TONIDANDEL, F., SILVA, J. R. **itSIMPLE 2.0: An Integrated Tool for Designing Planning Domains.** In: ICAPS, 2007. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling. Menlo Park, California, USA, AAAI Press, 2007.

8 MCDERMOTT, D., HENDLER, J. **Planning: What it is, What it could be, An Introduction to the Special Issue on Planning and Scheduling.** Artificial Intelligence, 76:1-16, 1998.

9 BACCHUS, F. **The Power of Modeling - a Response to PDDL2.1.** Journal of Artificial Intelligence Research 20:125 - 132, 2003.

10 BODDY, M. **Imperfection Match: PDDL2.1 and Real Applications.** Journal of Artificial Intelligence Research 20:133 - 137, 2003.

11 OMG (Object Management Group) **Unified modeling language specification: version 1.4.** http://www.omg.org/uml, 2001.

12 OMG (Object Management Group), **OCL 2.0 - Object Constraint Language.** http://www.omg.org/cgibin/doc?ptc/2003-10-14, 2003.

13 BRAY, T.; PAOLI, J.; MCQUEEN, C.M.; MALER, E.; YERGEAU, F. **Extensible Markup Language (XML) 1.0.** 3^{rd} Edition, 2004. Disponível em: http://www.w3.org/TR/REC-xml/.

14 WICKLER, G., POTTER, S. and TATE, A. **Recording Rationale in <I-N-C-A> for Plan Analysis.** Workshop on Plan Analysis and Management, ICAPS 2006, Cumbria, UK.

15 ROMERO, V. M. C., VAQUERO, T. S., TONIDANDEL, F., and SILVA, J. R. **Analysis and Management of Plans provided by Automated Planning Systems.** In: VIII SBAI - Simpósio Brasileiro de Automação Inteligente, Florianópolis, 2007.

16 ORKIN, J. **Three States and a Plan: The AI of F.E.A.R.**, Game Developer's Conference Proceedings, 2006.

17 DINI D. M.; LENT M. V.; CARPENTER P.; IYER K. **Building robust planning and execution systems for virtual worlds.** In Proceedings of the Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE), Marina del Rey, California, 2006.