UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Uma Arquitetura para Animar Agentes Autônomos em Ambientes Virtuais Usando o Modelo BDI

Dissertação de mestrado

por

JORGE ALBERTO TORRES

Profª. Dra. Luciana Porcher Nedel
Orientadora

Prof. Dr. Rafael Heitor Bordini
Co-orientador

Porto Alegre, dezembro de 2003

Agradecimentos

Há muitas pessoas que merecem reconhecimento ao longo do meu curso de mestrado, mas dirijo meus agradecimentos àqueles professores que tiveram maior participação em meu trabalho. Estes professores, que serão apresentados a seguir, foram importantes não só por sua competência e conhecimentos técnicos, mas principalmente pelo interesse, simplicidade e amizade que sempre demonstraram. Assim, foi uma grande alegria ter convivido com pessoas especiais e um orgulho ter trabalhado com professores que considero exemplos.

Aos professores Carla e Laschuk, agradeço por terem me recebido tão bem como aluno especial nas disciplinas de computação gráfica e pelo interesse que sempre demonstraram em ajudar-me a ingressar no mestrado. Naquela época, eu não tinha nenhuma experiência na área de computação gráfica, e toda a dificuldade para acompanhar uma turma de alunos já bastante avançados foi compensada pela excelente qualidade das aulas e pelo interesse dos dois professores em ajudar-me.

Ao meu co-orientador, Rafael Bordini, agradeço pela amizade e simplicidade com que sempre considerou meus questionamentos, por mais esquisitos que pudessem ser. Admiro que alguém tão detalhista em seu trabalho seja ao mesmo tempo aberto a ouvir sugestões e comentários de alguém de outra área de pesquisa. Mesmo antes de participar oficialmente do meu trabalho, sempre esteve disponível para ajudar-me, e não esqueço das vezes em que preocupou-se em me passar conselhos importantes.

À minha orientadora, Luciana, quero agradecer a capacidade de reunir amizade e liderança ao longo do nosso trabalho. Reconhecendo que sou um aluno de trato não muito fácil, por ser teimoso, crítico e não aceitar as coisas sem um certo grau de debate, admiro a forma como ela sempre tratou as diversas situações. Sem abrir mão de exigir tarefas e cobrar prazos, soube ser aberta a conversas e tratou eventuais momentos de baixa produção de forma compreensiva, ganhando minha confiança. Além disso, quase sempre conseguiu tornar divertido o trabalho muitas vezes enfadonho de pesquisar ou resolver problemas de implementação. Enfim, agradeço por ter estado sempre presente ao longo do curso e por ter indicado as melhores possibilidades de caminhos a seguir, permitindo que eu fizesse as escolhas.

Gostaria de mencionar também todos os colegas de laboratório, que ajudaram a tornar a tarefa mais leve. Agradeço pela amizade, pelas ajudas e dicas e pela boa vontade para trocar de computador sempre que eu necessitei de alguma máquina específica.

Sumário

Lista de Figuras

Resumo

Humanos virtuais são modelos computacionais de pessoas. Se necessário, podem apresentar uma aparência bastante realista, baseada em princípios fisiológicos e biomecânicos. Além disso, são capazes de comportar-se de forma autônoma e inteligente em ambientes dinâmicos, podendo apresentar até mesmo individualidade e personalidade. Humanos virtuais podem ser utilizados como atores sintéticos. Tais atores têm sido usados em uma série de aplicações com a finalidade de simular a presença de atores reais.

A indústria de jogos por computador requer personagens que sejam capazes de reagir apropriadamente a eventos e circunstâncias inesperadas, e até mesmo de alterar o progresso do jogo com seus cursos de ação autônomos. Um modo natural para desenvolver tais personagens prevê o uso de técnicas de inteligência artificial, em particular aquelas relacionadas às áreas de agentes autônomos e sistemas multiagentes. Neste trabalho, propõe-se o uso do modelo BDI (Belief-Desire-Intention) para modelar agentes cognitivos, com a finalidade de implementar personagens animados. O modelo BDI é uma abordagem bastante conhecida e bem sucedida para o desenvolvimento de agentes autônomos em sistemas multiagentes. Trata-se de uma arquitetura poderosa para sistemas dinâmicos e complexos, nos quais agentes podem precisar agir sob informação incompleta e incorreta sobre o seu ambiente e os outros habitantes.

Esta dissertação reúne um modelo articulado para animação de personagens, o qual requer a especificação de movimento em cada junta individualmente, e um interpretador para AgentSpeak(L), uma linguagem de programação orientada a agentes que implementa a arquitetura BDI. Foi desenvolvida uma interface que permite que o sistema de raciocínio de um agente, baseado em BDI, seja usado para dirigir o comportamento de um personagem em um sistema de animação. O uso de AgentSpeak(L) é uma abordagem promissora para a especificação em alto nível de animações complexas por computador.

O modelo conceitual e sua implementação são apresentados em capítulos distintos. Esta separação visa simplificar a compreensão do modelo proposto, permitindo primeiro analisá-lo em um nível mais alto de abstração, para então verificar detalhes de programação. Este trabalho apresenta também duas animações 3D, usadas para ilustrar a abordagem proposta. A principal animação apresentada envolve um agente situado em um ambiente dinâmico; o agente continuamente percebe o ambiente e raciocina para determinar como agir sobre ele, baseado em seu estado mental BDI. A outra aplicação é bastante simples, mas útil para mostrar algumas questões que são relevantes para obter-se mais eficiência em programas AgentSpeak(L).

PALAVRAS-CHAVE: Personagens Autônomos, Humanos Virtuais, Atores Sintéticos, Animação de Personagens Inteligentes, Agentes Racionais, Arquitetura BDI, AgentSpeak(L)

Abstract

Virtual humans are computational models of people. If necessary, they can portray a very realistic appearance, based on biomechanical and physiological principles. Besides, they are able to behave in an autonomous and intelligent way in dynamic environments, and even to exhibit individuality and personality. Virtual humans can be used as synthetic actors. Such kind of actors have been used in several applications, such as games, in order to simulate the presence of real actors.

The computer-game industry requires characters that are able to react appropriately to unexpected events and circumstances, and even to change the game progress with their autonomous courses of actions. A natural way for developing such characters is by the use of artificial intelligence techniques, in particular those related to the areas of autonomous agents and multi-agent systems. In this work, the use of the Belief-Desire-Intention (BDI) model for cognitive agents in order to implement animated characters is proposed. The BDI model is a well-known and successful approach for the development of autonomous agents in multi-agent systems. It is a very powerful architecture for dynamic and complex systems where agents may need to act under incomplete and incorrect information on other agents and their environment.

This work brings together an articulated model for character animation, which requires the specification of motion on each joint individually, and an interpreter for AgentSpeak(L), an agent-oriented programming language that implements the BDI architecture. I have developed an interface that allows the BDI-based agent reasoning system to be used for guiding the behaviour of a character in an animation system. The use of AgentSpeak(L) is a promising approach for the high-level specification of complex computer animations.

The conceptual model and its implementation are presented in distinct chapters. This separation aims at simplifying the comprehension of the proposed model, allowing its analysis first at a higher abstraction level, and after that to check programming details. This work also presents two 3-D animations used to illustrate the proposed approach. The main animation presented involves an agent that is situated in a dynamic environment; the agent continuously perceives the environment and reasons on how to act upon it based on its BDI mental state. The other application is quite simple, but useful to show some issues that are relevant for obtaining better performance from AgentSpeak(L) programs.

KEYWORDS: Autonomous Characters, Virtual Humans, Synthetic Actors, Intelligent Character Animation, Rational Agents, BDI Architecture, AgentSpeak(L)

1. Introdução

Humanos virtuais são modelos computacionais de pessoas. Se necessário, podem apresentar uma aparência bastante realista, baseada em princípios fisiológicos e biomecânicos. Restrições funcionais podem ser aplicadas a eles, a fim de que seus movimentos satisfaçam as limitações humanas. Além disso, ao invés de ficarem restritos a movimentos fixos, humanos virtuais são capazes de comportar-se de forma autônoma e inteligente em ambientes dinâmicos, podendo apresentar até mesmo individualidade e personalidade [BAD 99].

De acordo com Gratch et al. [GRJ 2002], a construção de um humano virtual requer um esforço multidisciplinar, reunindo problemas tradicionais de inteligência artificial com uma série de questões que abrangem desde computação gráfica até ciência social. Humanos virtuais devem agir e reagir em seu ambiente simulado, e para isso precisam ser capazes de raciocínio e planejamento. Para que possam manter uma conversação, é explorada a pesquisa em linguagem natural, incluindo reconhecimento e síntese de fala, além de compreensão e geração de linguagem natural. A criação de corpos humanos que possam ser controlados em tempo real requer técnicas de computação gráfica e animação. Finalmente, para que um agente pareça humano, deve comportar-se como tal. Assim, a pesquisa em humanos virtuais deve fazer uso de teorias de psicologia e comunicação para representar, de forma apropriada, comportamento não verbal, emoções e personalidade. Para combinar os diversos componentes, o principal desafio é manter a consistência entre o estado interno do agente (por exemplo, objetivos, planos, emoções) e os diversos canais de representação do comportamento (por exemplo, fala e movimentos do corpo). Quando estes canais entram em conflito, o agente parece desajeitado e pouco realista.

Humanos virtuais podem ser utilizados como atores sintéticos. Um ator sintético, virtual ou digital é definido como um ator autônomo, com aparência humana e completamente gerado por computador [THA 95]. Tais atores têm sido usados em uma série de aplicações com a finalidade de simular a presença de atores reais. Estas aplicações abrangem áreas diversas, entre as quais incluem-se educação, treinamento, terapia, segurança e entretenimento. Os atores que são inseridos num ambiente virtual podem ser usados tanto para povoar este ambiente, apresentando comportamentos padronizados, quanto para atuar como protagonistas, interagindo entre si e com os usuários (os quais são representados através de avatares). O principal objetivo na implementação destes atores é torná-los realistas, tanto em aparência e gestos quanto em comportamento, a ponto de serem confundidos com participantes reais. Além disto, a execução de seus movimentos deve ocorrer em tempo real.

1.1. Motivação

Os primeiros modelos de atores sintéticos usavam abordagens comportamentais predominantemente reativas, dando ênfase a aspectos relacionados aos mecanismos de percepção do ambiente, à implementação dos movimentos e à memória. Em seguida,

passaram a receber maior importância processos cognitivos como personalidade, emoções e atitude social. O uso de tais processos na elaboração do estado interno dos atores fornece-lhes maior credibilidade, no sentido de parecerem menos artificiais. De acordo com Allbeck e Badler [ALL 2000], são as ações e a comunicação (não verbal) de um personagem, mais do que sua aparência, que precisam parecer similares às de pessoas reais para dar credibilidade a uma animação. No artigo citado, são examinados vários fatores que influenciam a comunicação não verbal, tais como idade, importância social, estado emocional, humor, personalidade, gênero, cultura, entre outros. As ações de um ator virtual precisam ser coerentes com seu estado cognitivo interno para que ele possa apresentar um comportamento consistente.

Em aplicações de longa duração, tais como jogos do tipo *Adventure*, para que um ator sintético tenha credibilidade, é fundamental que apresente planejamento a longo prazo. A capacidade de planejar é importante para que o comportamento do ator seja condizente com o que se espera dele, mantendo-se estável ao longo da aplicação. Isto não quer dizer que ele não possa surpreender e adotar atitudes inesperadas, desde que estas sejam coerentes com seu estado interno. Um ator pode, por exemplo, usar sua inteligência para parecer bobo, enganando um oponente de forma premeditada, com a finalidade de obter vantagem a partir disso. Funge et al. [FUN 99] foram os primeiros a salientar que não basta aos personagens autônomos reagirem apropriadamente a estímulos do ambiente. É preciso que o comportamento destes personagens possa ser dirigido, através do controle de seu conhecimento e do modo como suas ações são planejadas a partir deste conhecimento. A partir de sua proposta de modelagem cognitiva, surgiram novos projetos propondo o uso de diferentes arquiteturas cognitivas.

A possibilidade de aperfeiçoar o aspecto de planejamento em atores sintéticos, visando principalmente a utilização destes em jogos de longa duração, é a motivação do trabalho descrito nesta dissertação. Para isso, sugere-se a implementação do raciocínio dos atores através da lógica *Belief-Desire-Intention* (BDI). Jogos de longa duração requerem personagens autônomos capazes de planejar suas ações de acordo com seus objetivos e comprometer-se com este planejamento até haver necessidade de reconsiderá-lo. Além disso, eles precisam responder com rapidez a eventos no ambiente que possam causar mudanças temporárias de humor e estratégia. O modelo BDI de agentes racionais satisfaz estas necessidades e pode ser útil na criação de jogos mais imprevisíveis.

De acordo com Georgeff et al. [GEO 98], BDI é possivelmente o modelo de agentes com raciocínio prático mais conhecido e mais estudado na comunidade de agentes inteligentes. A principal razão para isso é que BDI é um modelo filosófico respeitável de raciocínio prático humano (originalmente desenvolvido por Bratman [BRA 87]), possuindo um conjunto de arquiteturas implementadas e várias aplicações bem sucedidas em ambientes dinâmicos e complexos. No trabalho aqui apresentado, dois motivos adicionais levaram à escolha do modelo BDI :

- a abstração lógica do modelo permite implementar raciocínio complexo de forma natural, devido à sua proveniência do raciocínio prático humano. É bem mais adequado para a modelagem de agentes com comportamento sofisticado do que usar métodos numéricos ou baseados em regras simples;

- a possibilidade de implementação de comportamento coerente, mesmo em caso de falhas ou situações não previstas em um ambiente dinâmico e complexo.

A principal dificuldade apresentada pelo modelo BDI é implementar as funções definidas na teoria, de forma eficiente [WOO 99]. Muitos esforços têm sido feitos no sentido de diminuir as lacunas entre a filosofia do modelo e sua implementação, e um dos resultados de pesquisa visando unificar teoria e prática é a linguagem AgentSpeak(L) [RAO 96]. AgentSpeak(L) é uma linguagem orientada a agentes, baseada na teoria BDI, com restrições necessárias para a implementação prática dos seus conceitos em tempo real. No modelo aqui proposto, AgentSpeak(L) é usada como linguagem para a especificação em alto nível de personagens racionais em animações. Esta é, pelo que se sabe, uma área de aplicação para a qual o modelo BDI não foi aplicado, ainda que muitos tipos de animação necessitem de atores com o tipo de raciocínio sofisticado que agentes AgentSpeak(L) apresentam.

1.2. Objetivo

O objetivo deste trabalho é criar personagens que sejam representados por corpos articulados em ambientes virtuais tri-dimensionais e que tenham capacidade de planejar suas ações usando lógica BDI em tempo real. Com esta finalidade, é proposta uma arquitetura dividida em três níveis: um nível corresponde ao sistema articulado que define o corpo; outro nível, implementado através de um interpretador para AgentSpeak(L), é o mecanismo responsável pelo planejamento e raciocínio; e um nível intermediário, o qual é o enfoque principal do trabalho, é encarregado da comunicação necessária entre os dois primeiros. Este nível controla quais informações do ambiente podem ser percebidas pelo corpo e as envia continuamente ao interpretador AgentSpeak(L). Além disso, determina como deve ser a movimentação do corpo articulado para que este possa executar as ações definidas pelo raciocínio.

É importante salientar que este trabalho não tem a pretensão de criar atores sintéticos completos, pois concentra-se apenas nos aspectos relacionados ao estabelecimento da arquitetura proposta, enfatizando a capacidade de planejamento e raciocínio com múltiplos focos de atenção em ambientes dinâmicos. Ainda não estão sendo considerados outros fatores importantes relacionados à credibilidade dos personagens, tais como influência e expressão coerente de emoções e personalidade, além de funções sensoriais de percepção.

1.3. Organização

O Capítulo 2 desta dissertação apresenta uma revisão bibliográfica destacando as principais características de humanos virtuais e seu uso como atores sintéticos. São abordados diversos aspectos relativos a sistemas multiagentes e animação comportamental, incluindo representação e animação de corpos articulados, percepção de eventos externos que ocorrem no ambiente virtual, expressão e influência de emoções, memória e tomada de decisão. São

analisadas também as arquiteturas de agentes inteligentes reativos e cognitivos mais importantes.

O Capítulo 3 descreve o modelo proposto neste trabalho para produzir atores sintéticos articulados com raciocínio cognitivo usando lógica BDI. É explicado como são compostos os corpos articulados e a forma como eles são integrados ao mecanismo de raciocínio, o qual é implementado através de um interpretador AgentSpeak(L). Além disso, são apresentados os outros componentes do modelo: ambiente virtual e avatares.

O Capítulo 4 explica em detalhes como este modelo foi desenvolvido. Além de apresentar as classes implementadas e descrever o uso do interpretador AgentSpeak(L), salienta as técnicas utilizadas para integrar todos os elementos do modelo. O Capítulo 5 apresenta duas aplicações desenvolvidas como estudos de caso, uma delas criada de forma experimental durante a definição do modelo e a outra produzida com a finalidade de ilustrar a utilização da arquitetura criada. Por fim, o Capítulo 6 avalia o resultado do trabalho e sugere extensões para ele.

2. Atores Sintéticos

2.1. Introdução

Neste capítulo, será apresentada uma revisão bibliográfica incluindo os mais importantes componentes envolvidos na criação de humanos virtuais e na utilização dos mesmos como atores sintéticos. A divisão do capítulo em seções foi estabelecida visando separar cada componente que foi considerado individualmente relevante, embora a maioria dos trabalhos pesquisados tratem deles em conjunto. É importante salientar que a proposta desta dissertação envolve atores sintéticos individuais. Desta forma, esta revisão não compreende trabalhos relacionados com coordenação e comunicação de agentes.

2.1.1. Animação Comportamental

A técnica de animação comportamental foi apresentada por Reynolds [REY 87] e permite que um personagem seja animado a partir da modelagem de seu comportamento, sem que o animador precise ocupar-se com a especificação dos movimentos que levam a tal resultado. Os comportamentos podem variar desde simples planejamento de caminho até interações emocionais complexas entre personagens.

Um sistema de animação comportamental em geral é composto por um ambiente virtual e por atores autônomos que se movimentam nesse ambiente. O comportamento de um ator é influenciado pelas informações capturadas do ambiente sobre os objetos e sobre os outros atores participantes. Essa informação é usada pelo modelo comportamental para selecionar qual ação tomar, e esta resulta em um procedimento de controle motor [PAR 2001].

Em seu trabalho, Reynolds estudou em detalhes o comportamento de grupos de animais, tais como bandos de pássaros, cardumes de peixes e manadas de animais terrestres. A animação desse tipo de comportamento usando técnicas tradicionais é praticamente impossível. Na abordagem adotada por Reynolds, cada pássaro decide sua trajetória sozinho, sem intervenção do animador, e o comportamento do bando é o resultado da interação dos comportamentos individuais. Cada pássaro segue três objetivos: fuga de colisões, tentativa de igualar a velocidade dos vizinhos próximos e tentativa de aproximar-se dos vizinhos do bando. O animador fornece informação sobre a trajetória do líder e sobre a distância mínima entre os pássaros.

Reynolds [REY 87] destaca ainda a importância de que os pássaros tenham percepção apenas parcial de seu ambiente, uma vez que informação completa e perfeita levaria a falhas de comportamento. Assim, seu modelo de percepção tenta disponibilizar aos atores (pássaros) aproximadamente a mesma informação que um animal real obtém de seu

ambiente. Uma vez que o animador não controla diretamente os movimentos do personagem, este pode apresentar um comportamento inesperado, dificultando o processo de animação.

Haumann e Parent [HAU 88] descrevem simulação comportamental como um meio de obter movimento global complexo simulando regras simples de comportamento entre atores ligados localmente. Os autores apresentam uma aplicação criada para fazer experiências com simulação comportamental e usam esta aplicação para criar uma biblioteca de atores que comportam-se fisicamente, sendo capazes de reproduzir o movimento de objetos flexíveis de modo realista. Algumas propriedades úteis da simulação comportamental são apontadas: primeiro, é mais intuitivo obter um comportamento de grupo através da manipulação de atores do que a partir de um conjunto de equações diferenciais; segundo, a capacidade de adaptação dos atores às mudanças no ambiente elimina a necessidade de reelaborar especificações de movimento; por último, é possível experimentar várias situações diferentes, simplesmente colocando os atores na cena e deixando-os atuar.

Modelos comportamentais podem ser usados para produzir animação de alto nível envolvendo seres humanos e animais [THA 96]. Alguns exemplos de comportamentos baseados em percepção a partir de sensores virtuais foram desenvolvidos pelo grupo de pesquisa de Thalmann. Em um destes trabalhos, Noser et al. [NOS 95a] propõem um modelo cujo objetivo é permitir que um ator virtual explore um ambiente desconhecido, construindo modelos mentais e mapas cognitivos a partir dessa exploração. Durante ou após a construção dos mapas, o ator pode planejar caminhos, navegar e encontrar lugares com sucesso. O modelo divide-se em um sistema de navegação global e outro de navegação local. O primeiro requer uma estrutura de memória e tem como tarefa o planejamento de caminhos até alvos especificados. O segundo usa informação capturada do ambiente para atingir objetivos e para evitar obstáculos.

Este sistema de navegação local é uma extensão da abordagem descrita por Renault et al. [REN 90] e baseia-se no uso de Autômatos de Deslocamento Local (ADL). Um ADL é um procedimento contendo instruções que permitem a um ator sintético deslocar-se em uma parte específica de seu ambiente. Alguns exemplos desses autômatos são ADL *deslocamento-em-um-corredor*, ADL *desvio-de-obstáculo*, ADL *cruzando-com-outro-ator-sintético*, ADL *passagem-em-uma-porta*, etc. ADLs simples podem compor outros mais genéricos, e a modularidade desse sistema possibilita reações a um grande número de situações do cotidiano. O sistema possui um mecanismo de visão sintética através do qual obtém informação do ambiente. Baseado nesta informação e no estado interno do ator, um mecanismo controlador decide quais objetivos criar e quais ADLs executar para atingi-los. O controlador administra também o relógio interno do ator, a fim de atualizar seu objetivo global e local em intervalos regulares, os quais são determinados por uma variável contendo a taxa de atenção do ator.

Thalmann e Noser [THA 99] estenderam a abordagem descrita acima para implementar um jogo de tênis baseado em sensores. O sistema de visão (descrito na seção 2.3.1) é usado por cada jogador para reconhecer a bola em movimento, estimar sua trajetória e localizar o oponente para o planejamento da estratégia de jogo. As características geométricas da quadra, por sua vez, fazem parte do conhecimento dos jogadores. Através de um sistema de audição, o jogador tem conhecimento dos eventos que ocorrem ao longo do jogo. Autômatos acompanham a trajetória da bola pelo sistema de visão, calculam o tempo e a posição de colisão da bola com a raquete, executam a jogada, etc.

Noser e Thalmann [NOS 96] apresentam ainda um sistema de animação comportamental baseado em um L-System [PRU 90]. As regras de produção podem ser usadas para modelagem e desenvolvimento do ambiente, e também para a definição de comportamentos complexos. O sistema de produção associa primitivas geométricas (cubos, esferas, etc.) a seus símbolos. Os atores são representados por um símbolo especial, e, dependendo do propósito da aplicação, sua forma geométrica pode variar desde primitivas simples até estruturas de esqueleto complexas ou mesmo superfícies de corpos deformáveis.

Bécheiraz e Thalmann [BEC 98] propõem um sistema para produzir animação comportamental contendo um modelo que gera emoções sentidas pelos atores autônomos. Este sistema é composto por quatro módulos, lidando com percepção, geração de emoções, seleção de comportamentos e execução de ações. O módulo de percepção fornece aos módulos de comportamento e de emoção o estado de percepção, composto por objetos, atores e ações de atores. O módulo de comportamento faz uso da percepção e das emoções para selecionar um comportamento e então controla o movimento do ator através do módulo de ação. Este modelo está ilustrado na Figura 2.1.

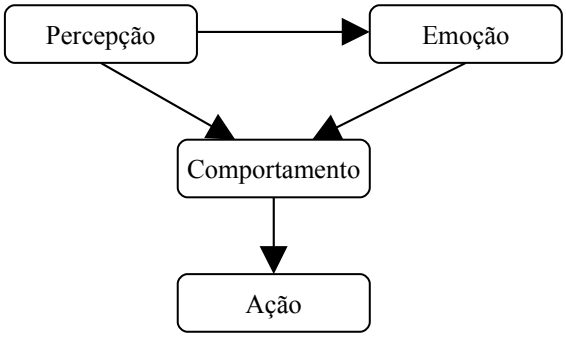


Figura 2.1 - Estrutura do modelo comportamental (adaptada de [BEC 98])

As emoções sentidas por um ator são causadas por sua percepção e afetam sua reação subseqüente. Dois atores podem apresentar diferentes reações à mesma percepção, dependendo de como são afetados por ela. Para possibilitar essa variação, as emoções geradas possuem uma intensidade associada a elas, e os atores um limite próprio de tolerância para cada uma. O módulo de comportamento decompõe um comportamento em outros mais simples, os quais, por sua vez, ainda podem ser decompostos. Cada nível desta hierarquia contém um ou mais comportamentos, que são executados de forma seqüencial ou concorrente. No final da estrutura hierárquica encontram-se os comportamentos que executam uma ou mais ações, as quais animam um modelo humano genérico e são controladas pelo módulo de ação.

Tu e Terzopoulos [TU 94] apresentam um modelo comportamental para a criação de peixes artificiais. O sistema motor compreende o modelo dinâmico do peixe e um conjunto de controladores motores. Cada um destes controladores é um procedimento parametrizado encarregado de executar uma função motora específica, tal como "nadar à frente" ou "virar à esquerda". Eles traduzem parâmetros de controle como velocidade ou ângulo de giro em ações musculares. O peixe artificial possui um gerador de intenções, que é seu centro "cognitivo". A cada passo de tempo este gerador emite uma intenção baseada nos hábitos do peixe, em seu estado mental e na informação sensorial. O sistema comportamental então escolhe e executa uma rotina de comportamento que, por sua vez, seleciona os controles motores apropriados. O gerador de intenções controla também um mecanismo de atenção para filtrar somente informação sensorial condizente com os objetivos correntes.

Creatures [GRA 97] é um software que simula um ambiente no qual agentes fazem o papel de animais de estimação virtuais, podendo interagir com objetos, com outros agentes e com participantes reais. O sistema baseia-se em técnicas desenvolvidas em pesquisas nas áreas de Vida Artificial e Comportamento Adaptativo, e sua arquitetura interna é inspirada na biologia animal. Os agentes ("criaturas") são modelados geneticamente e usam redes neurais artificiais para simular o controle senso-motor e aprendizado, além de estruturas bioquímicas responsáveis pelo metabolismo energético e pela regulagem hormonal de comportamento.

2.2. Percepção

Seres humanos recebem informação sensorial do ambiente em que estão inseridos a partir de seus cinco sentidos: visão, audição, tato, olfato e paladar. Esta informação é então transmitida ao cérebro, que encarrega-se de interpretar os sinais recebidos e de integrá-los. Percepção é, portanto, a combinação das sensações capturadas pelos sensores com a interpretação feita pelo cérebro, possibilitando ao ser humano extrair conhecimento do ambiente [PRI 9?, WRA 94].

Humanos virtuais, por sua vez, são modelos computacionais de seres humanos. Com o objetivo de alcançar realismo, devem apresentar, além de uma representação gráfica convincente, comportamento similar ao humano. Suas ações e reações devem ser imprevisíveis e baseadas na combinação de seu estado interno, de seus objetivos e da percepção de eventos externos.

A implementação de humanos virtuais requer o uso de agentes perceptivos capazes de sentir o mundo através de sensores virtuais e de interpretar os sinais recebidos gerando conhecimento acessível a processos internos de raciocínio e de tomada de decisões. A quantidade de informação absorvida pelos sensores em um determinado momento pode ser maior do que a capacidade de processamento do agente. Uma solução para este problema é incluir um sistema para focalizar a atenção do agente em informação relevante. Terzopoulos e Rabie [TER 97] apresentam um mecanismo deste tipo.

Em ambientes virtuais, participantes reais são graficamente representados por avatares que interagem e se comunicam com outros avatares e também com agentes autônomos. A importância da percepção é diferente para cada um desses dois grupos (agentes e avatares). Para os participantes reais busca-se proporcionar a melhor imersão possível no ambiente virtual através da produção de estímulos virtuais realistas e da diminuição dos estímulos provenientes do mundo real.

Considerando agora os agentes autônomos, representados por humanos virtuais, a percepção permite que tenham autonomia e que apresentem comportamentos cotidianos de forma natural e espontânea, podendo inclusive expressar emoções. No entanto, uma vez que aplicações em tempo real exigem resposta imediata, a interpretação das informações sensoriais obtidas precisa ser muito rápida, caso contrário é necessário que os agentes tenham acesso à representação interna do ambiente virtual. Neste caso, informações importantes são extraídas diretamente, não chegando a ser interpretadas. Este acesso à informação extra torna

o agente dependente do ambiente, ao mesmo tempo lhe trazendo uma vantagem às vezes indesejada com relação aos participantes reais.

A seguir, serão apresentadas considerações sobre visão, audição, tato e olfato e sua utilização em humanos virtuais.

2.2.1. Visão

Os olhos são responsáveis pelo sentido da visão, sendo que uma imagem observada é projetada na retina. O nervo ótico transmite a imagem para o cérebro, que então a corrige e interpreta. A íris é um músculo que controla as dimensões da pupila, aumentando-a para captar mais luz em ambientes mais escuros e diminuindo-a em ambientes mais claros. O cristalino é um elemento que muda de forma conforme a distância do objeto sendo observado, a fim de focalizar a luz que entra no olho.

Quando inseridos em ambientes virtuais, os participantes humanos visualizam imagens em três dimensões. A sensação de imersão visual pode ser incrementada através do uso de dispositivos de realidade virtual, tais como HMDs (*Head Mounted Displays*) ou CAVEs (*Cave Automatic Virtual Environments*). Independente dos dispositivos disponíveis, a interpretação de imagens tri-dimensionais fornece uma série de informações importantes. É possível identificar um objeto, estimar sua posição e seu tamanho, calcular o momento de futuras colisões, etc.

O mesmo tipo de percepção deve ser obtido pelos agentes, a fim de que possam reagir adequadamente. No modelo proposto por Noser e Thalmann [NOS 95b], é implementada uma visão sintética com memória visual para os atores, que não depende da modelagem do ambiente. Esta implementação funciona mesmo para ambientes definidos proceduralmente (L-Systems) e objetos fractais, os quais não possuem representação geométrica. O ambiente é renderizado em uma janela de 50 por 50 pixels a partir do ponto de vista do ator sintético (câmera sintética posicionada entre os olhos do ator). O ator tem acesso ainda à sua posição, ao z-buffer (para saber a que distância estão os objetos) e à cor dos objetos, a qual permite identificá-los (para o ator, há uma tabela de correspondência entre cor e objeto).

Outro exemplo apresentado pelos autores é um jogo de tênis, no qual os agentes são raquetes que usam a visão sintética para localizar a bola pela cor dos pixels, calculando então sua velocidade e posição. Como a janela de visão do agente tem um tamanho limitado, é preciso controlar o foco para que a bola possa ser vista independentemente de estar muito próxima ou afastada. Assim que o agente visualiza a bola, o número de pixels com sua cor é comparado com valores limiares mínimo e máximo. Se o número de pixels da cor da bola ultrapassar o limite superior, é sinal de que a bola está muito próxima, e o ângulo de visão deve ser mais aberto. Em contrapartida, se o número de pixels correspondentes à bola for menor que o limite inferior, então a bola está muito longe, e é preciso fechar o ângulo de visão para não perdê-la de vista.

Terzopoulos e Rabie [TER 97] apresentam um sistema de visão retinal ativa para peixes sintéticos que segue os mesmos princípios. Na primeira versão deste sistema (1994), os peixes obtinham informação sobre o ambiente através do acesso direto à sua representação interna. Para torná-los mais naturais e independentes da modelagem do ambiente, foi implementada a visão sintética. Cada peixe possui modelos de cores de objetos de seu interesse armazenados. Se algum objeto captado pela visão corresponde a um desses modelos, então o peixe o enquadra em seu nível de foco mais fechado e passa a persegui-lo.

2.2.2. Audição

Os ouvidos são os órgãos responsáveis pelo sentido da audição. As ondas sonoras entram pela orelha e chegam até o canal auditivo, em cujo final está o tímpano, uma membrana que vibra com a passagem das ondas sonoras e que transmite as ondulações aos três ossos do ouvido médio, martelo, bigorna e estribo. A vibração chega então ao ouvido interno, formado pela cóclea e pelos canais semicirculares. A cóclea é um tubo em forma de caracol que pega as vibrações do estribo e as transforma em impulsos nervosos, os quais são transmitidos ao cérebro, que distingue os sons. Os canais semicirculares são responsáveis pelo equilíbrio. Em seu interior há um líquido cujo movimento informa ao cérebro a posição da cabeça, e mudanças súbitas de velocidade. Isto permite ao corpo perceber se está em movimento ou queda, mesmo sem usar outros sentidos. O som é medido em decibéis e pode ser classificado entre agudo, médio e grave.

A audição permite localizar objetos no espaço, especialmente quando se movem. Isto ocorre, porque os sinais sonoros em um ambiente 3D possuem também uma direção de propagação, a qual é captada pelos sensores auditivos. Em um modelo ideal para captação e reprodução de som ambiental, seriam usados infinitos microfones com precisão máxima, dispostos de forma circular, cada um captando entrada de som em uma única direção pré-determinada. Alto-falantes, colocados exatamente da mesma forma que os microfones, reproduziriam o som previamente captado. Neste caso o som seria reconstituído sem perder nenhum detalhe. Modelos reais para reprodução de som ambiental, contudo, costumam usar um número par de microfones (4, 8, 12, etc) posicionados de forma circular e separados por ângulos iguais (respectivamente, 90º, 45º, 30º, etc) para captação. Embora a direção das ondas sonoras possa ser recriada com precisão, não há um método para reproduzir um campo sonoro com fidelidade completa para quaisquer ângulo e freqüência, uma vez que a informação sonora que não está centralizada nas diretivas dos microfones precisa ser filtrada e tratada [HIR 94].

Sistemas de *display* de som ambiental podem ser integrados aos sistemas de *display* visual disponíveis para a imersão de seres humanos em ambientes virtuais. A integração com HMDs é obtida através da utilização de um sistema de fones de ouvido com capacidade de reprodução de som ambiental, chamado *Convolvotron* [FOS 91]. Este sistema é composto por um processador de sinais digitais em tempo real que recebe sinais de quatro portas de entrada, filtra esses sinais através de coeficientes determinados pelas posições e orientações da fonte e do ouvinte, e posiciona cada sinal no espaço de percepção 3D do ouvinte. Os dados resultantes são então apresentados nos fones de ouvido. Já a integração de *display* de som

ambiental em *displays* imersivos (CAVEs) é simples, uma vez que o *display* visual absorve pequenos erros no *display* de som. Isto acontece, porque há um domínio da visão sobre os outros sentidos. Se uma pessoa estiver vendo um objeto movimentar-se e ouvir um som, associará este som ao objeto, mesmo que a direção do som reproduzido não seja perfeita.

Noser e Thalmann [NOS 95b] apresentam um modelo acústico para humanos virtuais inseridos como agentes em um ambiente virtual. Neste modelo, os microfones captores são os ouvidos dos agentes, e os sons emitidos são lidos de arquivos com formato AIFF (*Audio Interchange File Format*), uma vez que qualquer fonte sonora (sintética ou real) pode ser convertida para este formato. O som pode ser renderizado a cada quadro da animação para cada microfone em tempo real. Neste trabalho (1995), foram considerados apenas ambientes contendo sons simples e poucos microfones. A mudança na freqüência do som ao aproximar-se ou afastar-se de um objeto pode ser sentida, pois o sistema leva em conta a movimentação tanto das fontes sonoras como dos agentes.

Em um caso ideal, os agentes usariam os mesmos fatores fisiológicos de interpretação de sinais sonoros que permitem aos seres humanos localizar um som em um ambiente. Entretanto, sistemas de realidade virtual demandam que o reconhecimento semântico do som e a conseqüente reação sejam rápidos. Assim, uma das opções viáveis seria os agentes terem acesso a dados internos do ambiente, o que os tornaria dependentes. A outra alternativa seria os agentes receberem o mesmo sinal sonoro estéreo e digitalizado que os participantes reais recebem através dos fones de ouvido e interpretá-lo. Por uma questão de eficiência computacional, essa interpretação visa somente "posicionar os olhos do agente" e passar o sinal de áudio para um módulo de reconhecimento de fala. Como os eventos sonoros do modelo possuem informação de tempo de início, posição e orientação da fonte, o agente consegue calcular, a partir dessas informações e de sua própria posição no ambiente, a distância, o atraso e a amplitude do som captado.

2.2.3. Tato

No tato, as sensações do ambiente são captadas através dos neurônios sensoriais ou dos corpúsculos sensoriais, que ficam logo abaixo da pele. A sensação de forma dos objetos é registrada pelos neurônios sensoriais e enviada para os neurônios de associação, que a repassam para os neurônios efetuadores. Estes recebem os impulsos e mandam ordens imediatas para o corpo. Já as sensações de temperatura, pressão e dor são captadas pelos corpúsculos sensoriais. Elas são então transmitidas aos neurônios sensoriais, seguindo o caminho descrito acima.

A modelagem tradicional para a implementação de sensores virtuais de tato baseia-se na detecção de colisões geométricas entre superfícies. Esta abordagem pode acusar um número grande de colisões, tornando difícil modelar as respostas dos agentes. Além disso, ela não fornece sensores capazes de captar fatores ambientais, tais como vento e temperatura, pois não permite controlar objetos gerados proceduralmente (sem representação geométrica explícita). Assim, uma alternativa viável é a modelagem de campos de força para implementar sensores táteis [THA 99]. A geração dos campos de força é feita a partir de

modelos físicos, e a detecção desses campos pelos agentes ocorre através de uma função definida para calcular a intensidade do campo de força global em uma determinada posição. O valor retornado por esta função deve ser comparado com um valor limite, que representa colisão. Dessa forma, até mesmo vento pode ser sentido.

Esta abordagem é apropriada para aplicações em tempo real, pois reduz o tempo de análise e reação dos agentes [PAR 2001]. Isto ocorre porque o número de pontos sensoriais ligados a um agente pode ser pequeno, simplificando o controle de sua resposta comportamental (ao contrário da detecção tradicional de colisões).

2.2.4. Olfato

O nariz é o órgão humano associado ao olfato. Na parte superior das narinas encontram-se as membranas olfativas, que captam os cheiros que entram pelo nariz. Logo acima das membranas encontram-se os nervos olfativos, que enviam as informações de cheiros para o bulbo olfativo. Este último passa então a sensação ao cérebro. Um dos nervos olfativos possui terminações nervosas livres distribuídas na cavidade nasal e conecta-se a diferentes regiões do cérebro, fornecendo uma maneira de ativar rapidamente reflexos de proteção como espirro e interrupção de inalação.

A reprodução de cheiros, mesmo para fragrâncias simples, não tem muita precisão. Em uma situação ideal, um número limitado de odores básicos armazenados deveria ser capaz de gerar qualquer cheiro através de combinações, de forma similar à usada em impressoras a jato de tinta para a reprodução de cores. Entretanto, há uma diferença entre a sensibilidade humana aos odores e às cores, o que torna bem mais custosa a geração de cheiros. Diferentemente das cores, odores não são aditivos, ou seja, o fato de aumentar ou diminuir a quantidade de uma fragrância básica não implica em uma alteração correspondente no cheiro resultante. Isto ocorre porque existem cerca de mil tipos diferentes de receptores no nariz, e cada um destes reage a um pequeno grupo de odores, podendo haver intersecção entre os odores que os diferentes receptores captam. Assim, na prática, não há como criar um reprodutor genérico para qualquer cheiro. A partir de testes, contudo, é possível reproduzir cheiros pré-determinados para uma aplicação específica [KAY 2000], [ZYB 99].

Para a percepção de cheiros, existem modelos de simulação de processamento neuro-biológico buscando a compreensão de como os odores são identificados. Redes neurais artificiais são usadas para análise de dados e reconhecimento de padrões. Sensores eletrônicos de cheiro (*narizes eletrônicos*) são simplificações dos sistemas biológicos. Segundo [ZYB 99], esta simplificação não é crítica para realidade virtual.

2.3. Memória

Memória é uma das características fundamentais para que um ator sintético apresente comportamento autônomo inteligente, e geralmente é definida como o poder de reproduzir ou recordar o que foi aprendido e retido, especialmente através de mecanismos associativos [THA 99]. A percepção permite ao ator apresentar um comportamento reativo, mas uma estrutura de memória é necessária para que ele possa implementar algum tipo de planejamento ou raciocínio.

No modelo de exploração de um ambiente desconhecido, como o proposto por Noser et al. [NOS 95a], o sistema de navegação global do ator utiliza sua memória visual para planejar sua movimentação, evitando caminhos errados testados anteriormente e obstáculos já detectados. Essa memória é implementada através de uma estrutura de ocupação do tipo *octree*, a qual é atualizada continuamente a partir da percepção visual do ator. A *octree* possui a propriedade de representar o ambiente 3D. Cada nodo livre da *octree*, o qual não está ocupado com um obstáculo, é interpretado como um nodo do grafo, e considera-se que todos os voxels vizinhos estão conectados por arestas. Em uma primeira abordagem, um caminho é representado por uma seqüência de nodos livres. Para evitar uma explosão combinatória das possibilidades, é recomendada uma busca em profundidade utilizando algumas heurísticas.

No modelo proposto por Tu e Terzopoulos [TU 94], em um mundo dinâmico complexo, a memória permite que peixes artificiais tenham alguma persistência em suas intenções. Se o comportamento corrente é interrompido por um evento de mais alta prioridade, o gerador de intenções pode armazenar a intenção corrente e alguma informação associada em uma memória de curta duração, com capacidade para um item único. Essa informação pode ser usada para reiniciar o comportamento interrompido. A persistência serve para tornar comportamentos de longa duração, tais como alimentação e acasalamento, mais robustos. Se um peixe está em acasalamento, por exemplo, e uma colisão iminente é detectada, uma intenção para evitar a colisão é gerada, e a intenção de acasalamento é armazenada, juntamente com a identidade do parceiro. Após ultrapassado o obstáculo, o gerador de intenções comanda o mecanismo focalisador de percepção para gerar informação atualizada sobre o parceiro de acasalamento, assumindo que este encontra-se ainda no seu raio de visão.

2.4. Raciocínio

Sistemas desenvolvidos com técnicas convencionais de programação executam suas funções sempre da mesma maneira, seguindo o que foi planejado pelos programadores. A execução pode variar em diferentes circunstâncias, mas ocorre sempre dentro das possibilidades previstas. Frente a uma situação não especificada pelo programador, um sistema assim falha, pois não tem capacidade de adaptação. Para grande parte das aplicações, isto não é problema, visto que estão inseridas em ambientes estáticos. Entretanto, em áreas como a medicina, controle de tráfego, redes de telecomunicação, entretenimento, educação,

negócios e outras, cresce o número de aplicações com tarefas de gerenciamento e controle inseridas em ambientes dinâmicos complexos. Tais aplicações necessitam de agentes, sistemas capazes de decidir por conta própria o que precisam fazer frente a situações imprevistas, a fim de atingir seus objetivos [WOO 99].

Embora não exista uma definição universalmente aceita do termo agente, há consenso de que autonomia é uma característica central. Autonomia significa que um agente é capaz de agir sem a intervenção de humanos ou de outros sistemas, além de possuir algum tipo de controle sobre suas ações e seu estado interno. Com relação aos demais atributos, há pouca concordância, até porque variam para diferentes domínios nos quais os agentes podem estar inseridos. Enquanto para algumas aplicações, por exemplo, a capacidade de aprender é fundamental, para outras é irrelevante. Assim, como definição básica, pode-se dizer que um agente é um sistema computacional situado em algum ambiente e capaz de agir com autonomia neste ambiente, com a finalidade de atingir os objetivos que lhe foram designados.

Através da interpretação dos estímulos captados por sensores (reais no caso de agentes de hardware, virtuais em agentes de software), um agente consegue perceber as mudanças em seu ambiente e então, mediante um processo decisório, escolhe uma ação disponível para executar. Essa decisão é uma questão chave na construção de arquiteturas de agentes, e sua complexidade é afetada pelas características do ambiente no qual o agente está inserido. Na maioria dos domínios, o conhecimento que o agente tem do ambiente é apenas parcial, assim como a influência que exerce no mesmo. Assim sendo, não há como garantir o resultado de uma ação nem antever o estado resultante da mesma com certeza, caracterizando o não-determinismo do ambiente. Um agente precisa inclusive estar preparado para a possibilidade de falha. Em sistemas multiagentes (nos quais dois ou mais agentes interagem com o ambiente e eventualmente entre si) a complexidade é maior, pois outros agentes podem alterar o ambiente.

Wooldridge e Jennings [WOO 95] apresentam duas definições para o termo agente. A descrição mais genérica define agente como um sistema computacional baseado em software (mais usualmente) ou hardware que apresenta as seguintes propriedades:

- autonomia;
- reatividade: agentes são capazes de perceber seu ambiente e responder a mudanças que nele ocorrem;
- pró-atividade: agentes não agem apenas em resposta a seu ambiente, mas tomam iniciativas para atingir seus objetivos;
- capacidade social: agentes são capazes de interagir com outros agentes (e possivelmente com usuários) através de algum tipo de linguagem de comunicação.

Wooldridge [WOO 99] salienta que as três últimas características dão flexibilidade a um agente e são necessárias para que ele possa ser considerado inteligente. De acordo com ele, um agente inteligente deve ser capaz de avaliar seu plano de ação e de alterá-lo quando sentir que o mesmo não vai funcionar ou quando seus objetivos mudarem. Esta reação deve ocorrer em tempo hábil para que dê resultado, mas o agente também não pode ficar continuamente reagindo, sob pena de não conseguir focar-se em seus objetivos. Assim, o balanço ideal entre comportamento dirigido a objetivos e comportamento reativo varia conforme a aplicação e não é fácil de ser atingido. A capacidade social é importante para que o agente possa negociar e cooperar com outros agentes, pois nem sempre será possível atingir seus objetivos sozinho.

Na outra definição [WOO 95], adotada particularmente por pesquisadores da área de Inteligência Artificial, o termo agente tem um significado mais forte e específico do que o anterior. Além das características descritas acima, agentes apresentam propriedades mentais semelhantes às humanas, tais como conhecimento, crenças, intenções, obrigações e até emoções. Outro atributo que pode conferir um aspecto humano a um agente é a possibilidade de representação gráfica do mesmo.

Existem diversas arquiteturas possíveis para planejar e implementar agentes, desde os reativos até os cognitivos. Os agentes reativos, descritos na Seção 2.4.1, são muito simples e não possuem representação do seu conhecimento (ou seja, não armazenam nenhuma informação sobre o ambiente ou outros agentes), ficando este implícito em suas regras de comportamento. Estes agentes não possuem memória de suas ações passadas nem podem planejar seu comportamento futuro. Suas reações dependem unicamente da percepção das mudanças do ambiente a cada instante. Os agentes cognitivos, ao contrário, mantêm uma representação explícita de seu ambiente e dos outros agentes do sistema, têm memória do passado, podem comunicar-se com outros agentes usando comunicação de alto nível (baseada em atos de fala) e possuem um mecanismo de controle deliberativo. Este processo deliberativo permite-lhes decidir, através de algum tipo de raciocínio, sobre seus objetivos próprios e sobre os planos (cursos de ação) que serão usados para atingi-los [ALV 97]. A Seção 2.4.2 trata de agentes cognitivos e apresenta os modelos BDI e Soar, que estão entre os modelos cognitivos mais importantes.

2.4.1. Agentes Reativos

A arquitetura de agentes reativos mais conhecida é a arquitetura de subsunção (*subsumption architecture*) [BRO 86], que possui duas características principais [ALV 97]:

- os agentes têm à sua disposição um conjunto de comportamentos elementares, cada qual podendo ser executado por uma ação individual;

- os comportamentos são ordenados seguindo uma hierarquia de prioridade. O mecanismo de escolha da ação a ser executada obedece à percepção contínua do ambiente e a essa hierarquia. Por exemplo, um comportamento para evitar obstáculos deve ter alta prioridade. Entretanto, ele só será executado se a percepção do ambiente indicar que há um obstáculo à frente. Em caso contrário, será buscado o próximo comportamento na lista, e assim sucessivamente.

Um exemplo clássico de uso desta arquitetura é o sistema dos *robôs mineradores* [STE 90], cujo objetivo é fazer um conjunto de robôs encontrar amostras de minerais e levá-las para uma base central. O ambiente e a localização dos minerais são desconhecidos, mas as amostras encontram-se sempre agrupadas. Dois mecanismos são usados para resolver o problema: um sinal emitido pela base, cujo gradiente varia conforme a distância; e um sistema de sinalização, que permite implementar comunicação indireta entre os robôs.

Este sistema foi implementado de duas maneiras. A primeira não faz uso da possibilidade de comunicação indireta. Cada robô procura aleatoriamente até encontrar uma amostra de minerais. Quando consegue encontrar, detecta o sinal da base e segue a variação

crescente de seu gradiente. Chegando à base, deposita seu carregamento e volta a movimentar-se aleatoriamente. O comportamento individual dos agentes é construído a partir das regras abaixo, obedecendo à prioridade indicada:

1. se detectar um obstáculo, evitá-lo;
2. se estiver carregado e na base central, descarregar;
3. se estiver carregado, seguir o sinal da base central (no sentido do maior gradiente);
4. se perceber um mineral e ainda não estiver carregado, pegá-lo;
5. realizar movimento randômico.

Agindo individualmente, os robôs conseguem recolher os minerais, mas perdem muito tempo para encontrá-los, pois a busca é sempre aleatória. A segunda implementação, por sua vez, utiliza o mecanismo de sinalização dos robôs e adota uma solução cooperativa. Aproveitando o fato de que os minerais encontram-se agrupados, cada vez que um robô encontra uma amostra, ele marca uma trilha até a base central (pode usar migalhas, bandeiras, etc.). Qualquer robô que encontre a trilha vai segui-la até a base, se estiver carregado, ou na direção dos minerais (seguindo o gradiente decrescente do sinal). As regras a seguir estabelecem o comportamento cooperativo dos agentes:

1. se detectar um obstáculo, evitá-lo;
2. se estiver carregado e na base, descarregar;
3. se estiver carregado e não estiver na base, marcar trilha com 2 migalhas e seguir o gradiente crescente do sinal;
4. se perceber um mineral e ainda não estiver carregado, pegá-lo;
5. se perceber migalhas, retirar uma migalha da trilha e seguir o gradiente decrescente do sinal;
6. realizar movimento randômico.

Esta abordagem é bem mais eficiente que a primeira, sendo que os robôs apresentam um comportamento parecido com colônias de formigas, seguindo as trilhas até que as amostras terminem, para só então procurar novos focos. Seguindo a definição de agente inteligente apresentada, pode-se considerar que estes agentes podem ser assim caracterizados, visto que apresentam todos os atributos requeridos. São autônomos, apresentam comportamento voltado a atingir seus objetivos, e reagem à percepção que têm do ambiente. Para completar, comunicam-se, embora de forma indireta, e cooperam com outros agentes.

Uma característica importante de sistemas reativos é o fato de que sua funcionalidade deve emergir a partir dos comportamentos individuais dos agentes no ambiente. Isto torna este tipo de sistema complicado de ser construído, visto que não há uma metodologia para a criação dos agentes. Eles são construídos através de um processo de tentativa e erro, com a finalidade de obter o comportamento desejado para o sistema como um todo.

2.4.2. Agentes Cognitivos

As ações dos agentes cognitivos são internamente reguladas por seus objetivos, os quais, juntamente com suas decisões e planos, são baseados em crenças. Posto que um agente

cognitivo pode ter mais de um objetivo ativo na mesma situação, ele precisa ter um mecanismo de escolha baseado em algum raciocínio. A relação entre crenças e objetivos, gerando e controlando comportamentos, pode ser considerada como uma forma complexa e flexível de regra composta de condição e ação [CAS 98].

Dois dos principais modelos de agentes cognitivos são o BDI (Belief-Desire-Intention) e a arquitetura Soar (State, Operator And Result). O embasamento filosófico do modelo BDI vem da teoria do raciocínio prático humano, que envolve dois processos importantes: decidir quais objetivos busca-se atingir (deliberação) e como agir para atingi-los [WOO 99]. Soar é uma arquitetura baseada em regras, em seu nível mais baixo de representação, contendo mecanismos que a tornam eficiente para aplicações complexas e dinâmicas. Ambos modelos surgiram praticamente na mesma época, em meados da década de 80, e foram utilizados em várias aplicações de larga escala. Soar utiliza uma abordagem empírica para o projeto da arquitetura; alguns dos seus princípios básicos são conhecidos a partir da construção de sistemas. Sistemas BDI, por sua vez, são criados seguindo a filosofia e a lógica do próprio modelo [GEO 98].

2.4.2.1. Arquitetura Soar

A arquitetura Soar visa construir sistemas inteligentes que interagem, de forma autônoma, com ambientes complexos habitados por outras entidades, incluindo outros agentes inteligentes e participantes reais. Soar combina reatividade e deliberação, e seus agentes possuem flexibilidade para responder a uma determinada situação de várias formas diferentes. Para garantir a reatividade dos agentes, é utilizado um algoritmo de manutenção para atualizar crenças sobre o ambiente. Com relação ao processo deliberativo, todo o conhecimento relevante a uma determinada situação é identificado e aplicado através de um mecanismo associativo.

O conhecimento é representado como uma hierarquia de operadores, da qual cada nível equivale, sucessivamente, a uma representação mais específica do comportamento de um agente. Os operadores no topo da hierarquia representam os objetivos do agente. Os operadores no segundo nível da hierarquia descrevem táticas de alto nível que o agente utiliza para atingir objetivos do nível superior. Finalmente, operadores de mais baixo nível são os passos e subpassos, chamados comportamentos, usados pelo agente para implementar as táticas. A cada ciclo de decisão, Soar pode selecionar um operador ativo em cada nível da hierarquia. Se o contexto altera-se subitamente, os operadores podem tornar-se inapropriados, sendo, então, imediatamente substituídos por outros, mais convenientes à nova situação. Assim, um agente Soar pode trocar de comportamento em um único ciclo de decisão. A Figura 2.2 ilustra o uso da hierarquia de operadores em um jogo de ação. Neste exemplo, um agente pode ter o objetivo de “Atacar”, o qual tem disponíveis várias táticas no segundo nível da hierarquia. Os comportamentos que implementam cada tática de ataque preenchem os níveis inferiores.

A representação hierárquica de operadores da arquitetura Soar suporta múltiplas táticas para atingir cada objetivo, além de múltiplos comportamentos para implementar cada tática. Cada operador é descrito por um conjunto de condições de seleção, testes sobre

informação sensorial e memória interna, e um conjunto de ações condicionais. Quando é preciso selecionar um operador em determinado nível da hierarquia, são levados em conta todos aqueles que possuem condições de seleção correspondentes ao contexto, e a escolha é feita com base em um controle de prioridades. Enquanto um operador permanece selecionado, as ações que ele contém podem ser executadas, caso suas condições sejam satisfeitas. Táticas ou comportamentos múltiplos podem ser implementados através da criação de operadores múltiplos contendo condições de seleção similares, mas ações diferentes, as quais geram o mesmo resultado.

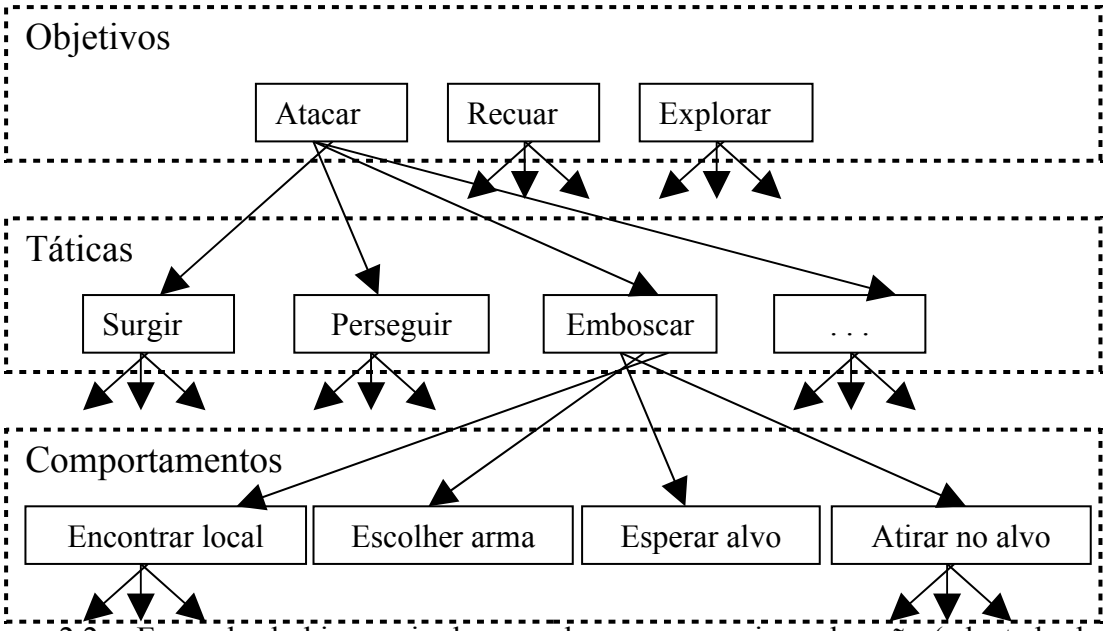


Figura 2.2 – Exemplo de hierarquia de operadores para um jogo de ação (adaptada de [LAI 99])

2.4.2.2. Modelo BDI

O modelo BDI oferece um tipo particular de agente cognitivo cujo estado é definido em termos de atitudes mentais, particularmente *crenças*, *desejos* e *intenções*. Intuitivamente, *crenças* representam a informação que o agente possui sobre o ambiente (o aspecto informativo do estado). *Desejos* são opções de estados futuros disponíveis ao agente (o aspecto motivacional). *Intenções* são estados futuros (ou cursos de ação conduzindo a estados futuros) que o agente escolheu e que comprometeu-se a atingir (ou seguir); este é o aspecto deliberativo do estado mental de um agente. O raciocínio prático do agente envolve repetidamente atualizar as crenças a partir da percepção do ambiente, decidir quais opções estão disponíveis, "filtrar" estas opções para determinar novas intenções e agir baseado nestas intenções [RAO 95].

Os componentes básicos da arquitetura BDI são as estruturas de dados que representam as *crenças*, *desejos* e *intenções* do agente, além de funções que determinam sua deliberação e raciocínio. A Figura 2.3 ilustra o processo de raciocínio prático de um agente BDI, apresentando seus principais componentes:

- *Crenças* é o conjunto de crenças do agente, representando sua informação sobre o ambiente. O agente constantemente percebe as mudanças do ambiente e pode alterar suas crenças através de uma função de revisão de crenças (FRC);

- *Desejos* é o conjunto de opções do agente (seus desejos) e representa os caminhos de ação disponíveis. Uma função de geração de opções (*Gera Opções*) constantemente determina as opções do agente, baseada em suas crenças e intenções correntes;

- *Intenções* é o conjunto de intenções do agente, representando seus focos de atenção, ou seja, os estados do mundo que ele comprometeu-se a tentar estabelecer. Uma função de filtro (*Filtro*) representa o processo deliberativo do agente, determinando suas intenções a partir de suas crenças, desejos e intenções correntes.

Uma função de seleção de ação (*Ação*) determina uma ação a ser executada pelo agente com base em suas intenções e nas prioridades associadas a elas.

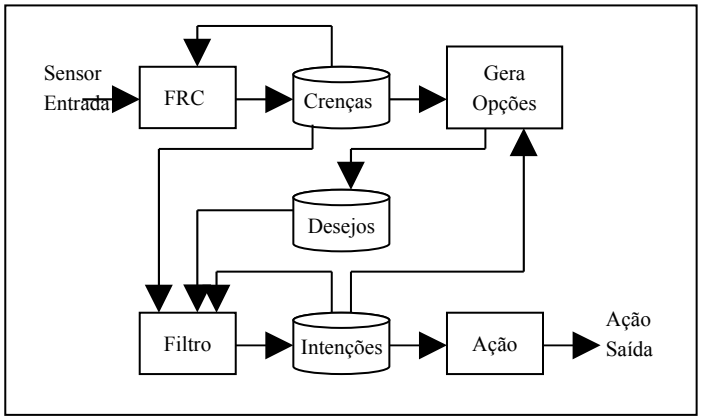


Figura 2.3 – Arquitetura BDI (adaptada de [WOO 99])

Objetivamente, o processo de decisão de um agente BDI começa a partir do conhecimento das opções de cursos de ação disponíveis. Ele precisa escolher entre elas e comprometer-se com uma ou mais, gerando então intenções. Estas, por sua vez, determinam as ações executadas pelo agente e são levadas em conta durante seu raciocínio prático futuro. Intenções têm um papel fundamental no modelo BDI, pois o agente age sempre no sentido de satisfazê-las, e isto fornece estabilidade à sua tomada de decisões. Uma vez adotada uma intenção, o agente é compelido a persistir com ela até que ocorra uma mudança de crença (acredite que a satisfez ou que ela não é mais viável) ou de desejo (seu propósito não existe mais). Se um curso de ação não funcionar, outras opções devem ser tentadas, desde que não sejam inconsistentes com a intenção [WOO 99].

Rao e Georgeff [RAO 95] salientam que as crenças devem ser distinguidas da noção de conhecimento, pois, diferentemente deste, podem ser falsas. A informação fornecida por elas ao agente refere-se ao estado provável do ambiente. Além disso, percepção falha ou ambientes não transparentes podem provocar crenças incorretas ou incompletas. Os desejos, por sua vez, fornecem ao agente informação sobre os objetivos correntes e suas respectivas prioridades de satisfação. Os autores definem objetivos como um subconjunto consistente dos desejos, uma vez que estes podem ser contraditórios entre si.

Grande parte do trabalho de pesquisa nas áreas de agentes autônomos e sistemas multiagentes tem sido desenvolvido com o objetivo de tratar da implementação de personagens virtuais com maior credibilidade. A contribuição dessas áreas compreende tanto

o raciocínio e planejamento de ações dos atores quanto processos cognitivos como emoções e personalidade, sendo que, normalmente, esses aspectos devem ser considerados em conjunto.

2.5. Credibilidade e Planejamento

O projeto *Oz* da Universidade Carnegie Mellon [BAT 92] introduziu o termo *agentes credíveis* (*believable agents*) na área de agentes como ele é usado na arte, significando que um observador pode ter a sensação de que um personagem fictício é real. Pesquisadores do projeto salientam o sucesso dos personagens da Disney e a compreensão que seus animadores tiveram de que a ilusão de vida depende da capacidade dos personagens para administrar e expressar personalidade, emoções e comportamento social [LOY 97]. O que faz uma pessoa gostar de um personagem ("esquecendo" que ele não é real) é a percepção de que ele realmente tem desejos e importa-se com o que acontece em seu ambiente [BAT 94].

Loyall [LOY 97] define *personalidade* como sendo o conjunto de detalhes particulares, especialmente de comportamento, emoção e pensamento, que definem um indivíduo. Uma definição semelhante é adotada por Rousseau e Hayes-Roth [ROU 97], para os quais *personalidade* é um conjunto de traços psicológicos que distinguem um indivíduo dos outros e que caracterizam seu comportamento. Acrescentam ainda que as pessoas podem ser facilmente descritas por seus traços de personalidade, uma vez que eles são facilmente reconhecíveis. Espera-se, então, que o comportamento (individual e social) e a expressão de emoções de um agente credível sejam sempre coerentes com sua personalidade.

2.5.1. Projeto OZ

A arquitetura *Tok*, criada no projeto OZ, permite implementar agentes cujo comportamento varia conforme sua personalidade e seu estado emocional. Nesta arquitetura, um agente embutido em um ambiente virtual repete um ciclo de sentir-pensar-agir durante sua execução. Em cada ciclo, o agente primeiro recebe dados do ambiente, os quais são armazenados e integrados em um modelo interno de mundo. Baseado no modelo do mundo e no estado interno do agente, um mecanismo cognitivo gera emoções e as transmite aos outros componentes da arquitetura, inclusive à sua parte encarregada de controlar os relacionamentos interpessoais do agente. A seguir, um mecanismo reativo encarrega-se da seleção da ação a ser executada. Esta seleção baseia-se nos objetivos ativos do agente, na importância desses objetivos, no estado do ambiente, no estado emocional do agente e em seus traços de personalidade [REI 92]. Não há um planejamento explícito das ações, e os objetivos não caracterizam estados do mundo a serem atingidos [BAT 92].

Durante o processo de seleção de ações, o mecanismo reativo pode criar novos objetivos ou descobrir que objetivos antigos tiveram sucesso ou falharam. Quando isso

ocorre, o mecanismo que gera emoções é avisado e pode mudar seu estado, influenciando, por sua vez, as futuras ações do agente. Esta coordenação fornece um retorno emocional imediato ao sistema. Com o objetivo de testar e apresentar seu trabalho em agentes credíveis, e contando com a colaboração do grupo de computação gráfica da Universidade Carnegie Mellon, o grupo do projeto Oz criou um sistema chamado *Edge of Intention* [LOY 97]. O modelo é composto por um pequeno mundo simulado contendo algumas criaturas interativas (*Woggles*), animadas com base nos princípios da animação tradicional de personagens (Figura 2.4). Alguns destes princípios, definidos pelos mais famosos animadores da Disney, são os seguintes:

- estado emocional do personagem precisa estar claramente definido;
- processo de pensamento revela o sentimento;
- emoção deve ser acentuada. O tempo deve ser usado sabiamente para estabelecer a emoção, para transmiti-la aos observadores e para deixá-los saborear a situação.

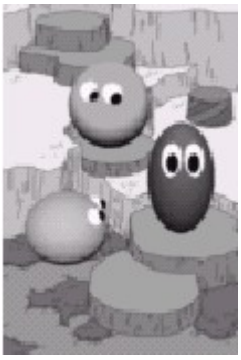


Figura 2.4 – *Woggles* em *Edge of Intention* [LOY 97]

2.5.2. Projeto Virtual Theater

O projeto *Virtual Theater* [ROU 97] desenvolve agentes que atuam como atores inteligentes, improvisando seu comportamento sem planejamento detalhado. Os atores interagem entre si e com participantes reais representados por avatares. Cada performance dos mesmos atores apresenta diferenças, uma vez que sua atuação é improvisada. De acordo com Rousseau e Hayes-Roth, um ator sintético possui uma "mente" e um "corpo". A primeira é implementada através de um agente, o qual atualiza o conhecimento do ator baseado em sua percepção, controla suas decisões e ordena ao corpo que execute ações para mudar o mundo virtual. O corpo, por sua vez, corresponde à representação na tela das ações executadas pelo ator. Como não faz parte do enfoque principal do projeto, esta representação pode ser feita tanto através de texto como através de animação. A Figura 2.5 ilustra este modelo de ator sintético.

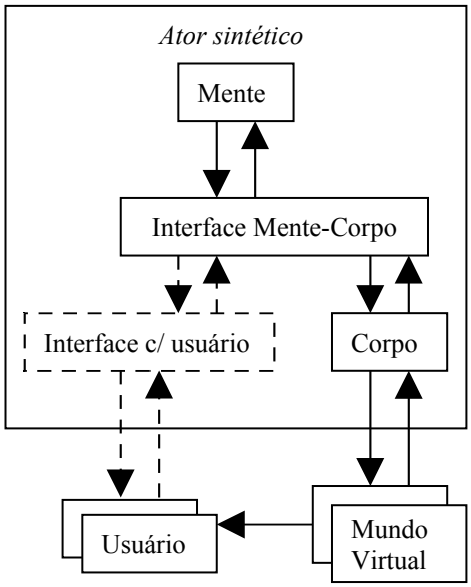


Figura 2.5 – Arquitetura de um ator sintético (adaptada de [ROU 97])

Os personagens em uma aplicação do *Virtual Theater* são essencialmente indivíduos sociais com emoções. O modelo sócio-fisiológico proposto permite criar personalidades flexíveis para os personagem, influenciadas pelo humor e pelas relações sociais. Os três principais elementos do modelo são:

Traços de personalidade - são padrões de comportamento e modos de pensar que determinam a adaptação pessoal ao ambiente. Alteram-se pouco e de forma lenta;

Humores - correspondem a emoções ou a sensações de necessidades físicas. São bastante variáveis no tempo e podem ser divididos entre os que são dirigidos a outros personagens (raiva, reprovação, etc.) e os que são auto dirigidos (alegria, orgulho, sede, etc.);

Atitudes - caracterizam um relacionamento interpessoal (Exemplos: importância social, confiança, nível de simpatia). Dependendo da natureza do relacionamento, podem variar ao longo do tempo ou podem ser bastante estáveis.

2.5.3. Modelo Centrado em Personalidade

Um grupo de pesquisadores da Universidade Federal de Pernambuco (UFPE) desenvolveu um projeto utilizando atores sintéticos como personagens para jogos de aventura ou de estratégia [SIL 99]. O objetivo do projeto é fazer os atores parecerem participantes reais. Para dar ilusão de vida, os personagens precisam exibir emoções próprias de acordo com as ocorrências do jogo. A expressão de sentimentos facilita a percepção pelo usuário da reação do agente e de seus traços de personalidade. De acordo com o grupo, os modelos de atores sintéticos diferem principalmente no modo como a personalidade influencia (ou é influenciada) os outros componentes e na forma como todos eles participam do processo de tomada de decisão (seleção de ações). Tanto o modelo do projeto Oz quanto o do *Virtual Theater* possibilitam que os traços de personalidade sejam alterados abruptamente, a partir de

mudanças de humor e de atitude social. Esta característica não é desejável em jogos de longa duração, pois estes exigem personagens que apresentem, além de reações emocionais, personalidade estável. Assim, visando este tipo de jogos, o modelo proposto pelo grupo da UFPE é centrado em personalidade.

2.5.4. Modelo de Gratch e Marsella

Em [GRJ 2001], é apresentado um modelo de personagens sintéticos que atuam em ambientes virtuais, tomam decisões, e comportam-se de acordo com seu estado emocional. Com o objetivo de implementar tais personagens, são reunidas, em um só sistema, duas abordagens complementares de modelagem emocional. O sistema *Émile* [GRJ 2000] concentra-se no surgimento de emoções a partir da avaliação de eventos do ambiente relacionados aos planos e objetivos de um agente. Já o sistema *IPD* [MAR 2000] foca-se no impacto de emoções sobre o comportamento, representado através da escolha adequada de gestos e linguagem corporal. Ambos são aplicados sobre *Steve* [RIC 99], uma arquitetura baseada em planos para criar agentes pedagógicos, a qual, por sua vez, foi integrada com um modelo aperfeiçoado de animação de corpos.

O modelo OCC [ORT 88] serve de base ao sistema descrito acima e a outros projetos que visam modelar emoções em humanos virtuais. A teoria deste modelo considera que emoções surgem a partir do relacionamento entre os objetivos de um agente e eventos externos. Gratch e Marsella [MAR 2002] integram ao modelo OCC um aspecto do comportamento emocional que tem sido ignorado pelos modelos computacionais. Pessoas lidam com eventos de forma diferente, dependendo de como eles são avaliados. Eventos considerados indesejáveis, mas controláveis, levam à execução de planos para resolver a situação. Entretanto, eventos avaliados como incontroláveis levam as pessoas ao escapismo ou à resignação. Os sistemas computacionais normalmente consideram somente o primeiro tipo de resposta. O modelo proposto por Gratch e Marsella, por sua vez, permite que os agentes reajam a emoções fortes tanto através de ações externas no ambiente, como também através de ações internas para alterar suas crenças.

2.5.5. Sistema Improv

Perlin e Goldberg [PER 96] desenvolveram o sistema *Improv*, o qual fornece ferramentas para criar atores que interagem entre si e com usuários em tempo real, com personalidades e humores consistentes com os objetivos do autor de uma aplicação. O sistema é composto por dois mecanismos, um de animação e outro de comportamento. O primeiro permite criar movimentos e transições suaves entre eles, e o segundo é usado para dirigir a comunicação e a tomada de decisão dos atores através de *scripts*. Os scripts são conjuntos de regras definidas pelo autor da aplicação, sendo que o sistema utiliza uma linguagem de

especificação de alto nível para permitir que não programadores possam criar aplicações interativas. O mecanismo de animação que controla o "corpo" dos atores, possibilita implementar humanos, animais, objetos animados ou criaturas de fantasia. O modelo pode ser deformável, o que pode ser útil para simular flexões musculares ou expressões faciais.

2.5.6. Modelo Cognitivo de Funge

Funge et al. [FUN 99] propõem um modelo cognitivo que pode ser integrado a sistemas de animação. A abordagem proposta vai além do modelo comportamental, permitindo controlar o conhecimento de um personagem, a forma como este conhecimento é adquirido e como pode ser usado para planejar ações. O modelo é dividido em duas subtarefas: especificação de conhecimento do domínio, que envolve a administração de conhecimento ao personagem sobre seu mundo e sobre como ele pode mudar; e controle do personagem, relacionado à instrução do mesmo para comportar-se de uma determinada maneira em seu ambiente, procurando atingir objetivos específicos. Foi empregado um formalismo de inteligência artificial conhecido como *situation calculus*. Nesta abordagem, uma *situação* é um instantâneo do estado do mundo. Não é possível descrever completamente uma situação, mas apenas parcialmente. A partir de fatos sobre situações, além de regras gerais sobre os efeitos de ações e outros eventos, é possível inferir alguma coisa sobre situações futuras. *Situation calculus* foi discutido pela primeira vez por McCarthy [MCC 63], mas o primeiro artigo mais difundido a respeito foi publicado em 1969 [MCC 69].

Para ajudar a construir modelos cognitivos, foi desenvolvida a linguagem CML (*Cognitive Modeling Language*), a qual constitui um nível intermediário entre programação lógica e imperativa. CML permite especificar o conhecimento do personagem sobre seu domínio em termos de ações, suas pré-condições e seus efeitos, e então dirigir o comportamento do personagem em termos de objetivos. Os detalhes de comportamento não fornecidos são preenchidos em tempo de execução pelo raciocínio do personagem, buscando atingir os objetivos especificados. Um sistema de percepção integrado ao modelo habilita os personagens autônomos a criarem planos de ação, mesmo em mundos virtuais dinâmicos complexos.

2.5.7. Personagens Persistentes

Um projeto da Trinity College Dublin integra um sistema de animação de humanos virtuais, desenvolvido pelo Grupo de Síntese de Imagem (*Image Synthesis Group*), com técnicas para implementação de agentes inteligentes, utilizadas pelo Grupo de Inteligência Artificial (*Artificial Intelligence Group*), para criar ambientes virtuais dinâmicos habitados por humanos virtuais inteligentes [MAB 2002]. O principal objetivo do sistema de animação é permitir que a animação e renderização ocorram em tempo real. Para isso, o nível de detalhe

da representação dos modelos varia conforme o foco de atenção, aproveitando as limitações do sistema visual humano. O sistema de inteligência artificial, por sua vez, introduz o conceito de *agentes proativos persistentes* (*PPA*, do inglês *Proactive Persistent Agents*) [MAB 2001].

A motivação para a criação dos PPAs foi a necessidade de criar personagens mais interessantes para jogos do tipo *Adventure* e *RPG (Role Playing Games)*. A persistência, característica diferencial desses agentes, indica que eles mantêm-se atuando de forma autônoma, independente da posição do jogador no universo do jogo. Personagens virtuais tradicionais, tanto reativos como cognitivos, somente entram em ação em áreas nas quais o jogador está ativo. Para implementar PPAs, são usadas as técnicas de *nível de detalhe* [CAR 97] e de *passagem de papel* (*role passing*) [HOR 99]. A primeira, de forma similar ao que ocorre em sistemas de animação, envolve o controle do nível de sofisticação do raciocínio do personagem com base em sua posição, relativa ao jogador, no ambiente virtual. A segunda consiste em determinar um papel a um agente básico (o qual é capaz de executar apenas comportamentos simples), com a finalidade de ensiná-lo a comportar-se em uma situação particular. Esta abordagem simplifica a tarefa de povoar um mundo virtual com agentes. Acrescentar agentes em uma nova situação envolve apenas definir um novo papel.

Como parte do projeto, o grupo apresenta ainda um sistema para permitir que as relações sociais dos agentes sejam baseadas em sua personalidade, humor e relacionamentos, os quais são capturados usando modelos psicológicos quantitativos. O comportamento social de um agente é determinado por uma rede neural, a qual é treinada para escolher uma interação apropriada baseada em valores particulares dos modelos usados [MAB 2003].

2.5.8. Arquitetura PAR

Badler et al. [BAD 2000] desenvolveram a arquitetura PAR (*Parameterized Action Representation*) para a implementação em tempo real de agentes que situam-se, comunicam-se e atuam em um mundo virtual tri-dimensional. A representação das ações é parametrizada, pois os detalhes de execução das ações dependem dos seus participantes (agentes e objetos). Uma PAR inclui um conjunto de condições que precisam ser satisfeitas antes que a ação seja executada. A ação é finalizada quando suas condições de término são satisfeitas. Além das condições de aplicabilidade e término, a representação das ações permite incluir informações espaço-temporais, como o caminho a percorrer e advérbios que definem a maneira de agir (rápido, com cuidado, etc).

2.5.9. Projeto Soar/Games

O projeto Soar/Games estabelece uma interface entre a arquitetura de inteligência artificial Soar e um conjunto de jogos, sendo dois deles comerciais (Quake II e Descent 3). Agentes complexos, possuindo a capacidade de planejar e aprender, foram desenvolvidos para os jogos não-comerciais, ao passo que, para os comerciais, foram criados agentes mais simples. O mecanismo de inteligência artificial desenvolvido, além das características dos agentes implementados, é apresentado por Laird e Van Lent [LAI 99]. Jogos interativos em computador são propostos como o tipo de aplicação mais capaz de desenvolver a área de agentes inteligentes, devido à complexidade de inteligência artificial que exigem [LAI 2001]. Em outro trabalho [LAI 2000], é apresentado um agente desenvolvido na arquitetura Soar para atuar como um participante virtual inteligente no jogo Quake II. O artigo descreve como é estabelecida a comunicação, através de *sockets*, entre o raciocínio do agente e o jogo, e também explica como o agente é dotado da capacidade de antecipar as ações dos oponentes.

2.5.10. Simulação de Grupos e Multidões

Musse [MUS 97, MUS 98] empregou humanos virtuais para simular multidões humanas em situações diversas, tais como simulação de situações de pânico e simulação de eventos de grupos. Os indivíduos são simples; a autonomia e inteligência estão relacionadas aos grupos. Existem três tipos de grupos no modelo de multidões: *grupos guiados* (controle interativo); *grupos programados* (comportamentos pré-definidos usando uma linguagem de script); e grupos *autônomos* (comportamentos baseados em regras). A idéia deste projeto é também fornecer diferentes níveis de autonomia na simulação de multidões. Com esta finalidade, foi definida uma prioridade de comportamentos, de acordo com a qual são sincronizados os diferentes tipos de controle.

Mais recentemente, foi desenvolvido o Massive [MAS ??], um software para a geração de multidões que foi utilizado para implementar as cenas de combate entre exércitos no filme *O Senhor dos Anéis – As Duas Torres*. A interação entre os componentes da multidão é baseada em decisões únicas e imprevisíveis tomadas por personagens individuais ao longo de uma cena. Ao invés de simplesmente duplicar ações mecânicas, Massive fornece a cada personagem um cérebro digital e autonomia para agir, seguindo a idéia de que a credibilidade de um elenco muito grande depende das ações de cada indivíduo. Embora o resultado final de uma batalha seja pré-determinado, as seqüências individuais são espontâneas, e cada movimento emerge de outro imediatamente anterior.

Os personagens gerados pelo Massive funcionam como seres complexos e sujeitos a forças físicas, com características que compreendem aspectos biológicos e comportamentais. A cada personagem é associado um conjunto de ações disponíveis, que podem chegar a um total de 350, cada uma durando cerca de um segundo. A forma como estas ações são combinadas é determinada pelo cérebro do personagem, uma teia contendo de 100 a 8000 nodos lógicos de comportamento, os quais fornecem as regras que permitem que cada

personagem perceba, interprete, decida e aja. Estes nodos agrupam-se em coleções de regras que controlam agressão, estilo de luta, movimento através de terrenos variados, e uma série de outros fatores. Os projetistas utilizam uma interface gráfica especial para criar o cérebro de um agente e as conexões entre seus nodos.

2.6. Aparência e Movimento

A animação do corpo humano é uma tarefa bastante complicada, pois sua forma é irregular e difícil de modelar [MAG 85]. Além disso, a especificação e computação de seus movimentos não é trivial, posto que suas articulações complexas envolvem mais de 200 graus de liberdade [ZEL 82]. Um modelo de humano virtual típico consiste de uma pele geométrica, geralmente modelada com polígonos para otimizar a velocidade da apresentação gráfica, e de um esqueleto articulado. A superfície pode ser rígida ou então deformável durante os movimentos. Esta última opção é mais realista, mas acrescenta carga extra de modelagem e computação. A estrutura do esqueleto é geralmente uma hierarquia de transformações de rotação nas articulações. O corpo é movimentado através da alteração dos ângulos das articulações e da sua posição global [BAD 99].

2.6.1. Representação

O corpo humano pode ser estruturado em três níveis: o esqueleto, a rede de músculos e a superfície envolvente da pele [TOS 88]. A maioria dos personagens virtuais são estruturados como corpos articulados definidos por um esqueleto. Este esqueleto é composto por um conjunto de segmentos, correspondendo a membros, conectados por articulações. Cada articulação pode ter de um a seis graus de liberdade, e seus ângulos determinam a posição de um segmento em relação a outro [THA 96]. Korein e Badler [KOR 82] utilizam uma estrutura de árvore para modelar o esqueleto, na qual os segmentos do corpo correspondem aos nodos da árvore, e as articulações aos seus arcos. Com relação à representação externa, diversos modelos foram propostos: figura "palito", superfície, volume e muli-níveis.

Os modelos de figura "palito" representam o esqueleto do corpo, descrevendo sua conectividade e flexibilidade. O maior problema destes sistemas é a falta de realismo na visualização (Figura 2.6-a). Não são capazes de representar a forma exterior do corpo, conseguindo, no máximo, sugeri-la. Além disso, sua falta de volume torna difícil a percepção de profundidade, e algumas rotações não podem ser visualizadas [BAD 79]. As principais vantagens são as facilidades para armazenar o modelo inteiro em uma estrutura de árvore e para especificar movimentos, bastando fornecer a cada articulação uma matriz de transformação [TOS 88].

(a)

Figura 2.6 – Estágios na construção do modelo de corpo humano [SHE 96]

Nos modelos de superfície, o esqueleto é coberto por uma superfície particionada em porções planares ou curvas (Figuras 2.6-c e 2.6-d). O número de porções é um fator fundamental na relação entre qualidade da imagem e custo computacional. Fetter [FET 82] propõe uma solução na qual vários níveis de complexidade do modelo (diferente número de porções) são disponibilizados, e a escolha depende da proximidade da figura em relação à câmera.

Nos modelos de volume, o corpo é decomposto em instâncias de volumes primitivos, como esferas, elipses ou cilindros (Figura 2.6-b). Uma pequena quantidade de cilindros ou elipses é suficiente para capturar as propriedades das superfícies e dos eixos longitudinais de várias partes do corpo, mas as formas obtidas ficam muito simétricas e artificiais [BAD 79]. Em aplicações que exigem realismo, estes modelos dificilmente conseguem competir com modelos de superfície [TOS 88].

A abordagem multi-nível para criar modelos articulados complexos permite que o modelo simulado seja dividido em níveis, cada um com suas propriedades físicas e geométricas. O animador especifica as várias restrições no relacionamento entre os níveis e pode então controlar o movimento global a partir de um nível alto. Normalmente, o esqueleto suporta um nível intermediário que simula a camada muscular e gordurosa do corpo. Este nível é então envolvido por uma malha renderizada, a qual implementa a camada de pele, fornecendo ao personagem uma aparência realista (Figura 2.6-e). Modelos mais complexos podem ser construídos utilizando outros níveis, tais como roupas, cabelos e pêlos [GIA 2000].

Usando esta abordagem, o animador pode basear-se no nível do esqueleto para controlar o movimento do personagem, bastando aplicar as restrições apropriadas entre os níveis. Entretanto, para obter movimento realista, os níveis externos precisam deformar-se

conforme os níveis vizinhos. Por exemplo, quando um membro dobra-se, o músculo deve inchar e a pele ao redor da articulação deve enrugar [GIA 2000]. Existem várias técnicas de deformação de objetos que podem ser usadas, entre elas *Joint Dependant Local Deformations* (JLD) e *Free Form Deformations* (FFD).

A técnica JLD utiliza, como pele, uma malha poligonal digitalizada e mapeia cada um dos vértices em um ponto particular no esqueleto, usando operadores JLD. Estes operadores controlam a evolução da superfície e dependem da natureza das articulações. Cada operador é aplicado a uma seção específica da superfície, e seu valor é função dos valores angulares do conjunto de articulações que o definem [MAG 90]. Justamente por basear-se em dados de articulações específicas, esta técnica não possibilita a geração de um modelo genérico de personagem. Magnenat-Thalmann et al. [MAG 88] a utilizaram para simular o movimento complexo da mão humana.

Free Form Deformations (FFD) é um método genérico para deformar objetos, o qual fornece um nível de controle mais alto e eficaz do que o ajuste de pontos de controle individuais. Ao invés de aplicar deformações diretamente ao objeto, esta técnica altera sua forma deformando o espaço no qual o mesmo se encontra. Sederberg e Parry [SED 86] introduziram esta técnica, na qual o objeto é embutido em uma estrutura reticulada que apresenta uma geometria padronizada, tal como um cubo ou cilindro. A manipulação de nodos dessa grade provoca deformações no seu espaço interior, as quais transformam as primitivas gráficas que servem de base para formar o objeto. Um exemplo de animação de personagem utilizando o modelo de representação multi-nível e FFD é apresentado por Chadwick et al. [CHA 89].

Scheepers et al. [SCH 97] argumentam que o uso de superfícies deformáveis próximo às juntas é uma aproximação pobre da anatomia humana, pois muitas deformações (como contrações musculares) ocorrem longe das articulações. Em seu modelo, consideram a influência da musculatura na forma da superfície e desenvolvem modelos musculares que reagem automaticamente a mudanças na postura do esqueleto articulado que serve de base. Na mesma linha, Nedel et al. [NED 98] apresentam o sistema *Body Builder Plus*, o qual integra esqueleto e músculos, permitindo também a combinação destes com superfícies implícitas para simular a pele cobrindo todo o corpo.

2.6.2. Técnicas de Controle de Movimento

Humanos virtuais animados tanto podem ser controlados por pessoas reais, sendo chamados avatares, como podem ser figuras puramente sintéticas. No primeiro caso, os ângulos das articulações são capturados por métodos de vídeo, magnéticos ou óticos e convertidos em rotações para o corpo virtual. No segundo caso, é preciso implementar programas para gerar as seqüências e combinações corretas para criar os movimentos desejados. Os procedimentos para alterar ângulos de articulações e posição do corpo são chamados geradores motores ou habilidades motoras. Algumas dessas habilidades típicas são [BAD 99]:

- mudar postura e ajustar equilíbrio;
- alcançar, agarrar e outros gestos do braço;
- caminhar, correr, subir;
- observar;
- expressar expressões faciais;
- pular, cair, balançar;
- combinar um movimento com o seguinte.

Um humano virtual deve ser capaz de executar várias dessas atividades de forma simultânea, assim como fazem seres humanos reais. Badler *et al.* [BAD 99] propõem uma máquina virtual paralela, chamada *Parallel Transition Networks* (PaT-Nets), que anima modelos gráficos. Em geral, nodos da rede representam processos, e arcos contêm predicados, condições, regras ou outras funções que causam transição para outros nodos. Uma PaT-Net fornece um modelo de animação não-linear e possibilita capacidades de reatividade e tomada de decisão.

A animação em nível de tarefa (*task-level*) utiliza o mesmo conceito das habilidades motoras. Os comandos em nível de tarefa precisam ser transformados em instruções de baixo nível para que o movimento correspondente seja executado. Alguns exemplos típicos de tarefas são: caminhar de um ponto A até um ponto B; pegar um objeto na localidade A e movê-lo até a localidade B; falar uma sentença ou produzir uma expressão de alto nível [THA 89].

Os comandos motores devem ser traduzidos para que a animação do personagem possa ser executada efetivamente, através da alteração dos ângulos das articulações de seu esqueleto e de outros parâmetros. Diversas técnicas têm sido usadas para animação de personagens, algumas delas puramente geométricas, como as cinemáticas, outras fisicamente embasadas, como as dinâmicas.

O método mais tradicional de animar uma figura articulada em computação gráfica é o de quadros-chave (*keyframing*). Neste método, o animador cria quadros-chave, especificando valores apropriados para um conjunto de parâmetros, e as imagens intermediárias são geradas automaticamente a partir da interpolação desses parâmetros. Por exemplo, para dobrar um braço, o animador informa o ângulo do cotovelo em diferentes momentos, e o software encontra os ângulos intermediários usando interpolação. O uso de funções cúbicas para interpolação (como *splines*) fornece um movimento suave para os membros do corpo [MAG 90].

As técnicas de cinemática especificam movimento, independentemente das forças que servem de base para produzi-lo. O uso exclusivo de cinemática direta na animação de figuras articuladas é bastante tedioso, posto que essas figuras podem conter mais de duzentos graus de liberdade, e o animador deve controlar todas as suas articulações para definir uma postura. Além disso, esta técnica não permite impor restrições ao movimento, tais como especificar que o pé não deve penetrar no chão durante o movimento de caminhar. Em contrapartida, a cinemática inversa exerce um papel importante na animação de figuras articuladas. Com a assistência da cinemática inversa, o animador apenas especifica a localização desejada de alguns pontos escolhidos do corpo, e o algoritmo automaticamente computa um conjunto de ângulos de articulações que levam a figura à postura determinada, satisfazendo as restrições impostas [TOL 2000, ZHA 94, GIA 2000, THA 96, MUL 99].

A técnica de captura de movimento é bastante popular na indústria de cinema e jogos. O movimento de um ator real é capturado por sensores óticos ou magnéticos colocados em posições chave no seu corpo, gerando os dados geométricos necessários para a animação. Este mecanismo tem potencial para produzir movimentos bastante realistas, embora muito específicos e não influenciados por fatores dinâmicos do ambiente, tais como gravidade e inércia [GIA 2000].

Abordagens dinâmicas simulam a ação das leis da Física Newtoniana para obter animação realista. Usando dinâmica direta, o movimento do corpo é obtido a partir das equações dinâmicas de movimento, as quais são estabelecidas usando forças, torques, restrições e propriedades de massa dos membros. Métodos de dinâmica inversa, por sua vez, calculam as forças e torques necessários para gerar um movimento pré-determinado [GIA 2000, THA 96, MUL 99]. Hodgins et al. [HOD 95] utilizam simulação dinâmica para animar comportamentos atléticos de corrida, salto e pedalada em bicicleta, e aplicam restrições para modelar pontos de contato entre os pés e o chão ou entre os pés e os pedais. Os métodos de simulação dinâmica apresentam, em geral, alto custo computacional.

O movimento de caminhar, assim como agarrar, é uma tarefa essencial para a animação de um ator sintético. De acordo com Zeltzer [ZEL 82], o ciclo do andar é geralmente dividido em duas fases que se alternam, uma delas correspondendo ao tempo em que o pé permanece em contato com o chão, e outra representando o balanço da perna para a frente. Cada braço balança para a frente, acompanhando a perna oposta, e para trás enquanto a mesma está em contato com o chão. Multon et al. [MUL 99] apresentam um estudo sobre diversas abordagens utilizadas para implementar esse movimento, incluindo cinemática inversa, dinâmica, abordagens híbridas, utilização de restrições e captura de movimento. Giang et al. [GIA 2000] salientam que movimentos cíclicos, tais como o caminhar, obtidos pela maioria dos mecanismos de animação, apresentam aparência artificial por natureza, pois não conseguem representar os pequenos defeitos humanos, necessários para transmitir realismo. Funções geradoras de ruído podem ser aplicadas para provocar perturbações, tentando simular esses defeitos.

Com a finalidade de implementar movimentos mais naturais, que sejam consistentes com o estado interno de um ator, o grupo de pesquisa de Badler criou o sistema EMOTE (*Expressive MOTion Engine*) [CHI 2000], baseado em *Laban Movement Analysis* (LMA), que é um modelo para observação de movimento humano. A partir de modificações na execução de um determinado comportamento, alterando a característica dos movimentos que o compõem, EMOTE consegue produzir gestos de maior ou menor amplitude, além de fazê-los parecerem ter maior ou menor energia.

3. Proposta de Modelo de Ator Sintético Cognitivo

3.1. Introdução

Este capítulo descreve o modelo desenvolvido para a simulação de atores sintéticos com raciocínio cognitivo, usando lógica BDI, em ambientes virtuais tridimensionais. A estrutura dos humanos virtuais no modelo proposto é composta por três elementos principais: representação gráfica, raciocínio cognitivo, interface entre ambos. Estes três elementos serão discutidos neste capítulo, juntamente com os outros componentes do sistema: avatares e o ambiente virtual no qual os atores interagem. Detalhes relacionados à implementação dos conceitos aqui apresentados serão abordados no capítulo seguinte.

Os elementos que compõem o modelo proposto são apresentados nas seções a seguir. O ambiente virtual é introduzido na Seção 3.3. Os avatares são tratados na Seção 3.4, e os agentes são descritos na Seção 3.5.

3.2. Arquitetura do Sistema

Cada agente inserido no ambiente é representado graficamente por um corpo articulado que segue o modelo introduzido por Maciel et al. [MAC 2002]. Neste modelo, os corpos são animados através da especificação de rotações nas articulações. Num nível mais alto de abstração, é possível reunir conjuntos de movimentos individuais para formar movimentos compostos (por exemplo, pegar ou caminhar). A partir da percepção de eventos ocorridos no ambiente virtual, os agentes devem decidir a cada momento quais desses movimentos disponíveis devem ser executados. Esta decisão é função do mecanismo de raciocínio cognitivo de cada agente, o qual é implementado através do interpretador para a linguagem AgentSpeak(L) apresentado por Bordini et al. [BOR 2002a]. Esta linguagem de programação orientada a agentes é uma extensão natural da lógica de programação BDI e oferece a possibilidade de especificação de raciocínio cognitivo em alto nível.

Conforme mencionado anteriormente, o enfoque principal deste trabalho é a combinação da estrutura de animação de personagens articulados com o mecanismo de especificação de raciocínio cognitivo usando AgentSpeak(L). Para tanto, é proposta uma interface de controle, encarregada da integração entre os dois sistemas, cuja arquitetura é apresentada na Figura 3.1.

Esta arquitetura é composta basicamente por quatro módulos principais: ambiente, avatar, agente e usuário. O **ambiente** virtual é composto por **objetos**, ***smart objects*** (objetos que, além de possuir representação gráfica, agregam também informação semântica [KAL

98]) e pode comportar tantos habitantes quantos forem definidos na aplicação (a limitação do número de atores simultâneos está relacionada à performance do sistema). Estes habitantes tanto podem ser **agentes** como **avatares**. Ambos são representados por **corpos** articulados, no entanto, apenas agentes possuem um **sistema perceptivo** e um mecanismo de raciocínio (**mente**). Nos avatares, o responsável pela percepção e cognição é o próprio **usuário**.

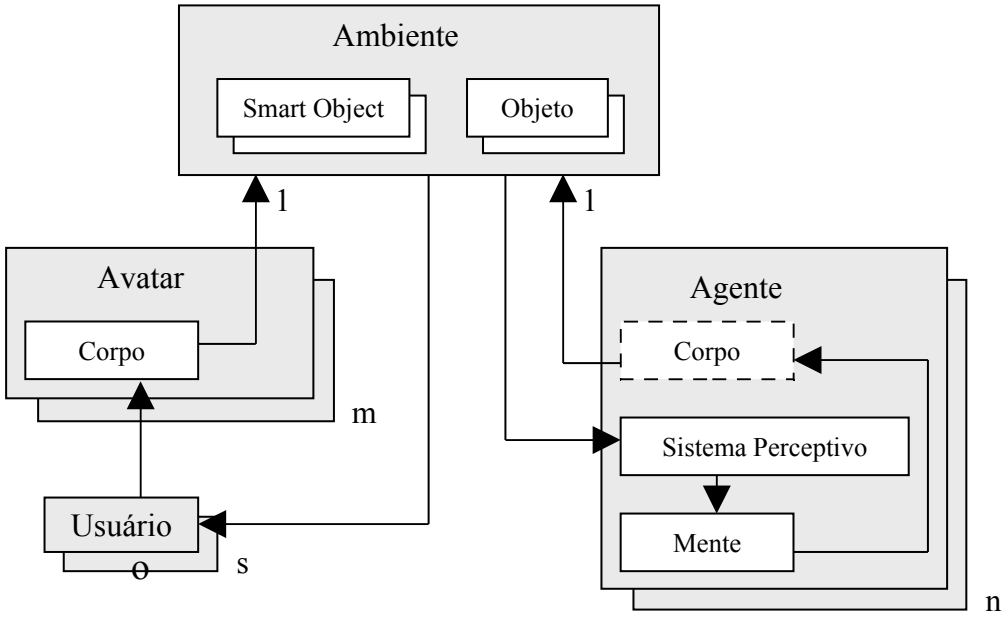


Figura 3.1 – Estrutura da arquitetura proposta

A quantidade de avatares em um ambiente (*m*) não precisa ser igual ao número de usuários (*s*), permitindo assim que um usuário tenha controle sobre mais de um personagem. A quantidade de agentes (*n*) também é independente, e é importante notar que podem haver agentes que não possuem corpo (não são representados no ambiente). Este aspecto possibilita a criação de ambientes virtuais inteligentes, controlados por agentes cognitivos "invisíveis".

3.3. Ambiente Virtual

O ambiente virtual é composto por objetos geométricos e pode ser povoado por avatares e agentes. Os objetos que compõem o ambiente podem ser comuns, possuindo apenas atributos que definem sua apresentação gráfica, ou "inteligentes" (*smart objects*), os quais armazenam também informação semântica que visa facilitar sua interação com outros objetos ou personagens. Por exemplo, considere-se o caso de um agente que precisa colocar um objeto em uma determinada prateleira. É necessário decidir o local exato em que o objeto será posicionado, evitando que haja intersecção do mesmo com a borda da prateleira, ou que uma parte sua fique fora dela. Sendo a prateleira "inteligente", ela pode armazenar a informação de quantos objetos ela contém num dado momento e onde o centro do objeto seguinte deve ser posicionado, facilitando o trabalho do agente.

Tanto os avatares quanto os agentes que são inseridos no ambiente virtual são representados por corpos articulados (lembrando que podem haver agentes sem representação), descritos na Seção 3.4.1. A aparência e estrutura de cada um pode variar de acordo com o número de juntas e graus de liberdade definidos e com os objetos gráficos usados para representar seu corpo. Da mesma forma, os movimentos que são capazes de executar também podem ser individualizados. Enquanto as ações dos avatares são comandadas pelo usuário, o comportamento de cada agente (os movimentos que ele decide executar a cada momento) é definido por seu mecanismo de raciocínio cognitivo.

A partir do momento em que um agente ou avatar é comandado a executar uma determinada ação (definida em alto nível de abstração), ele precisa traduzi-la para um conjunto de movimentos básicos. Este processo de tradução deve levar em conta que as ações estão definidas em um espaço discreto, ao passo que o ambiente virtual é um espaço contínuo ao longo do qual os personagens movimentam-se. Por exemplo, para executar a ação *ir até a prateleira*, o personagem deve virar-se para o local onde esta encontra-se e caminhar até ela. Entretanto, o local enfocado (a região em frente à prateleira) não consiste de um único ponto, mas de um conjunto deles. Assim, para evitar incerteza em relação ao alvo, os personagens precisam de um método para decidirem exatamente qual ponto geométrico do ambiente virtual deve ser atingido em uma determinada ação.

Na arquitetura proposta, o ambiente virtual armazena informação de posição de cada local relevante para a aplicação (são guardadas as coordenadas de um ponto no espaço tri-dimensional). Os pontos relevantes normalmente são locais do ambiente que podem ser alvos de deslocamento dos personagens, como portas, passagens, balcões, prateleiras, etc. Essa informação de posição é usada por agentes e avatares no momento da execução de um comportamento, para calcular, a partir de sua posição de partida, a orientação e o deslocamento necessários para chegarem em seu alvo. Este processo de deslocamento dos personagens no ambiente é melhor ilustrado na Seção 3.4.2. Takenobu et al. [TAK 2003] apresentam um método bem mais complexo para resolver essa questão, o qual utiliza funções de potencial para mapear regiões.

3.4. Avatar

Um avatar é a representação geométrica de um usuário participante. Cada avatar presente no ambiente virtual possui um corpo articulado e tem capacidade de executar um conjunto de comportamentos, interferindo no ambiente. O comportamento do avatar é controlado pelo usuário através de dispositivos de entrada. De forma semelhante aos objetos descritos na seção anterior, avatares também podem ser “inteligentes” (*smart avatars*). Este tipo de avatar, o qual é utilizado na arquitetura proposta, não necessita que o usuário especifique continuamente cada movimento; o usuário apenas informa uma ação a ser executada, e o avatar encarrega-se de produzir os movimentos necessários. As duas próximas seções descrevem o modelo articulado utilizado para representação e movimentação de corpos.

3.4.1. Modelo Articulado

Para modelar graficamente um corpo humano articulado, é necessária uma estrutura de dados hierárquica que represente o conjunto de articulações que compõem o corpo, além de um conjunto de funções usadas para gerar movimento nestas articulações. Cada articulação representa um conjunto de possibilidades de movimento ou, em outras palavras, um conjunto de graus de liberdade (DOFs, abreviação para *Degrees Of Freedom* [MAG 90]). Uma articulação com dois DOFs é capaz de executar dois tipos de movimento (por exemplo, flexão/extensão e adução/abdução).

Para poder construir um corpo humano a partir de um conjunto de articulações, é também necessário especificar a topologia deste conjunto (ou seja, como as articulações relacionam-se). A solução escolhida foi adotar uma estrutura de dados do tipo árvore, a qual aproxima-se da topologia das articulações humanas, estabelecendo uma relação hierárquica entre as articulações através de matrizes de instanciamento local. Deste modo, a posição e orientação de cada articulação é definida (através de sua matriz de instanciamento) em relação ao sistema de referência da articulação que a precede na árvore, com exceção da junta raiz da árvore, a qual é definida com base no Sistema de Referência do Universo.

As articulações não possuem representação geométrica. Elas simplesmente representam os relacionamentos matemáticos entre as partes do corpo. Assim, outras estruturas são necessárias para representar ossos, músculos e pele. No modelo adotado, estas estruturas são determinadas por objetos gráficos representados por primitivas geométricas ou malhas poligonais que podem ser associadas com qualquer articulação da árvore que representa o corpo. Cada um destes objetos também possui uma matriz de instanciamento que o associa com uma articulação, assim como define sua posição e orientação em relação a essa articulação. Desta forma, as estruturas que são representadas pelos objetos gráficos têm sua posição e/ou orientação alteradas de acordo com a mudança de posição e/ou orientação de cada articulação.

A Figura 3.2 ilustra a relação entre as articulações e os objetos gráficos (neste caso foram usados cilindros) que representam as estruturas do corpo. O objeto 1 corresponde ao braço, e sua posição e orientação dependem de rotações no ombro (junta 1). O antebraço (objeto gráfico 2) está ligado à junta 2 (cotovelo), e seu movimento depende não só desta articulação, mas também da junta 1, indiretamente. Por fim, a mão (objeto 3) é o último membro da hierarquia, e sua posição e orientação variam conforme a composição de rotações das três articulações.

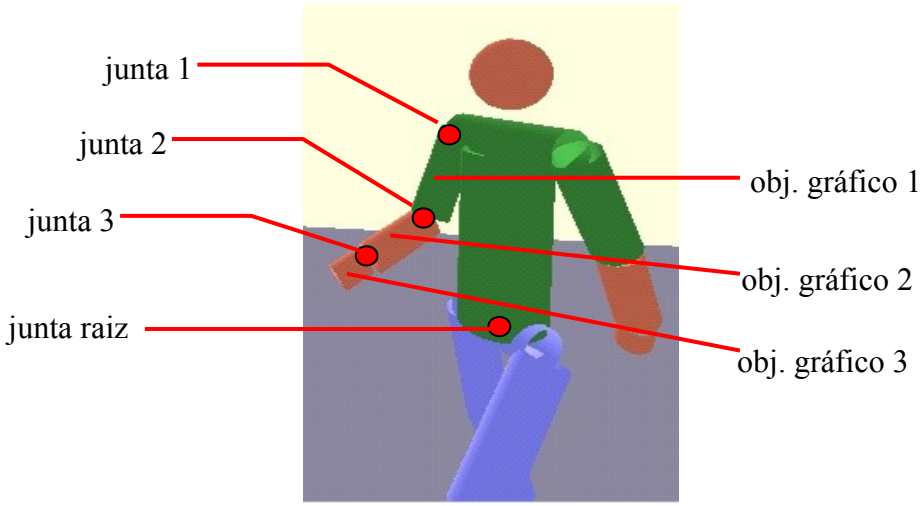


Figura 3.2 – Relação entre objetos gráficos e articulações

Para encontrar a posição final de um determinado objeto gráfico, é necessário multiplicar sua matriz de instanciamento pelas matrizes de instanciamento desde a junta raiz até a junta onde está o objeto. A Figura 3.3 ilustra o cálculo da posição da mão direita no exemplo acima. A matriz que define a posição e orientação da mão em relação ao sistema de referência do Universo (representada por $M_{[OBJ3 \rightarrow SRU]}$) equivale à multiplicação das matrizes de instanciamento (M) do objeto gráfico 3 (OBJ3), da junta 3 (J3), da junta 2 (J2), da junta 1 (J1) e da junta raiz (JR).

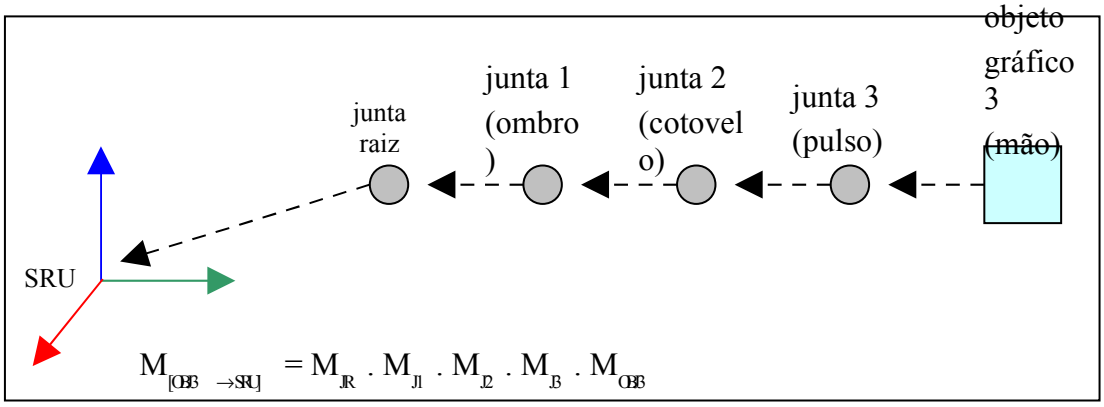


Figura 3.3 – Composição da posição final de um objeto gráfico

O modelo articulado mencionado acima, introduzido por Maciel et al. [MAC 2002], permite representar os diferentes tipos de juntas do corpo humano. Juntas uniaxiais têm apenas um DOF, o qual define rotações em torno de um eixo; juntas biaxiais possuem dois DOFs, cada um deles definindo um eixo de movimento rotacional, não necessariamente ortogonais; juntas poliaxiais possuem um terceiro grau de liberdade, também de rotação; articulações planares possuem seis graus de liberdade: três para rotação e três outros para translação. Uma junta deste tipo é geralmente usada como articulação raiz, e seu movimento de translação determina o deslocamento do corpo ao longo do ambiente no qual ele está situado. Cada DOF é um elemento do modelo e armazena sua posição em seu sistema de referência local, uma matriz de instanciamento que o associa ao sistema de referência da junta da qual faz parte, um vetor que define o seu eixo de movimento, os valores angulares máximo e mínimo que pode atingir, e seu ângulo corrente.

O modelo permite construir corpos articulados compostos por um número variável de juntas e de graus de liberdade. Nos estudos de caso desenvolvidos, foram utilizados humanos virtuais com apenas dezesseis articulações, a maioria delas apresentando somente um grau de liberdade (juntas uniaxiais). A Figura 3.4 ilustra a disposição das articulações no corpo em questão, especificando o número de DOFs em cada uma delas. É importante salientar que o modelo de humano virtual desenvolvido para este trabalho é extremamente simplificado e não tem o intuito de representar a funcionalidade humana. Assim, foram criados graus de liberdade suficientes apenas para permitir os movimentos necessários às animações. Os objetos gráficos adotados para representar os membros do corpo são primitivas gráficas: cilindros e esferas. O arquivo com a descrição deste corpo em XML está disponível no anexo A.

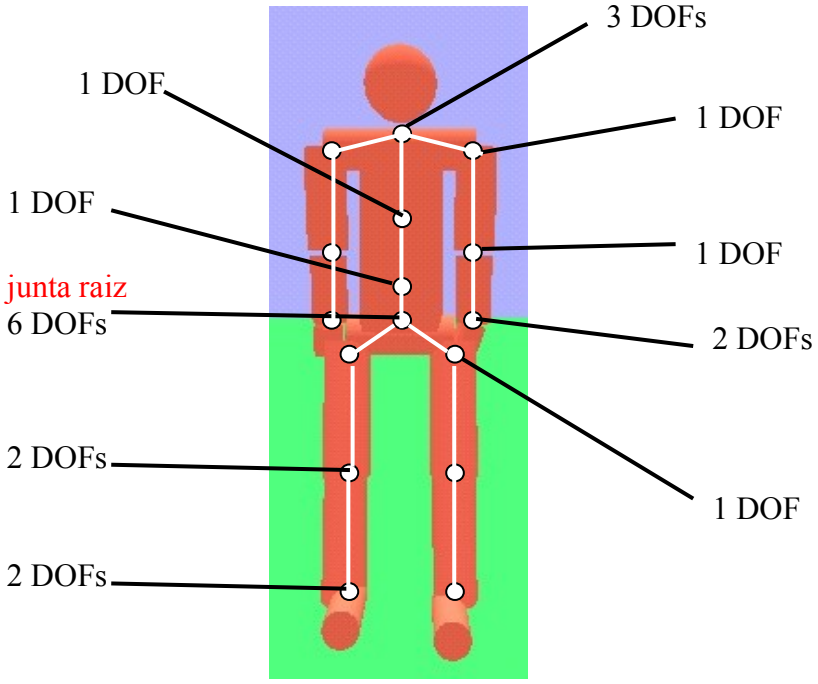


Figura 3.4 – Exemplo de corpo articulado com dezesseis articulações

3.4.2. Composição de Movimentos

O movimento dos atores é implementado em três níveis (Figura 3.5). O mais baixo deles corresponde às transformações em cada articulação do corpo, de forma individual. Como foi descrito na seção anterior, cada articulação no modelo é composta por um conjunto de DOFs, que representam seus graus de liberdade de movimento. Cada DOF de rotação, por sua vez, controla a rotação da junta em torno de um eixo e pode mover-se em um intervalo angular específico (DOFs de translação controlam a translação da junta ao longo de um eixo). O movimento de cada articulação, a partir da raiz, é propagado às suas juntas-filhas, somando-se ao movimento destas.

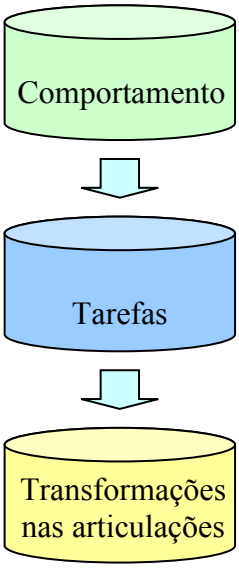


Figura 3.5 – Composição do movimento em níveis

Uma rotação em um DOF é implementada através da especificação de um intervalo de tempo e de um parâmetro angular. Baseado na posição angular corrente, o sistema calcula a rotação necessária para que o DOF atinja o parâmetro estipulado. O intervalo de tempo informado determina a velocidade de execução do movimento. Considere-se, por exemplo, o movimento de um DOF de um ângulo α até um ângulo β. Com um intervalo de tempo muito curto, o movimento será abrupto, ao passo que um intervalo maior gera um movimento mais suave. O movimento de um DOF de translação, por sua vez, também requer dois parâmetros: o intervalo de tempo e o deslocamento. Cada vez que um DOF qualquer é movimentado, sua matriz de instanciamento é atualizada, assim como a matriz da sua articulação, e os objetos gráficos associados a esta têm sua posição alterada.

Determinar o movimento de todo um corpo articulado especificando rotações nas suas juntas é uma tarefa muito árdua. Uma vez obtido o movimento desejado, é conveniente armazená-lo. Assim, ele não precisará ser refeito para que possa ser usado novamente, e também poderá ser combinado com outros para compor movimentos mais complexos. O modelo permite encapsular um conjunto de movimentos individuais de articulações, compondo movimentos completos (por exemplo, dar um passo completo, pegar, sentar). Estes movimentos compostos (os quais serão referidos como *tarefas*) oferecem um nível mais alto de abstração em relação aos movimentos articulares simples.

Uma tarefa é composta por um conjunto de movimentos individuais. Cada um deles é um movimento sobre um DOF de uma junta específica e ocorre em um intervalo de tempo determinado. Assim, durante a execução de uma tarefa, podem ocorrer movimentos em diversas articulações diferentes ao mesmo tempo (refletindo o que ocorre em um corpo real). A duração de uma tarefa é determinada pelo tempo compreendido entre o instante inicial do movimento individual que começa primeiro e o instante final do movimento que termina por último. A Figura 3.6 ilustra uma tarefa composta pelos movimentos *m1*, *m2* e *m3* (executados sobre DOFs diferentes), cada um deles iniciando num instante *i* e terminando em *f*. O instante inicial da tarefa coincide com o início de *m2* (o movimento que começa primeiro), e o instante final coincide com o fim de *m3* (o movimento que termina por último). Observe que *m1* compreende, na verdade, dois movimentos sobre o mesmo DOF, em instantes diferentes.

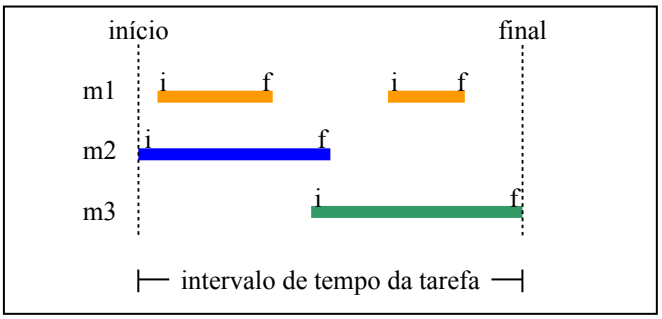


Figura 3.6 – Tarefa composta por três movimentos

Das tarefas atualmente disponíveis para agentes e avatares, *caminhar* é a única que envolve deslocamento no ambiente virtual. Além disso, é uma tarefa cíclica, já que a posição do corpo ao final de um passo coincide com sua posição inicial no passo seguinte. Sendo assim, para criar esta tarefa é suficiente compor um único passo (reunindo os movimentos individuais necessários), já que os demais são iguais. É preciso definir também o deslocamento no ambiente que este passo provoca. Para isso, deve-se levar em conta o tamanho do ambiente e as distâncias a serem percorridas. O passo não deve ser muito curto, para não tornar o deslocamento muito lento, nem muito longo, a fim de permitir deslocamentos pequenos. Mesmo utilizando este critério, em algumas situações necessita-se apenas de uma fração do deslocamento (como o tamanho do passo é definido previamente e mantém-se fixo, os deslocamentos que não forem múltiplos desse tamanho podem precisar de pequenos complementos). Assim, é conveniente dispor também de passos curtos, nos quais o ciclo de movimento é o mesmo dos passos normais, mas com menor deslocamento. Com esta finalidade, foram criadas três tarefas: *dar um passo*, *dar meio passo* e *dar um quarto de passo*.

O nível mais alto de abstração do movimento é responsável pela combinação de tarefas disponíveis para gerar *comportamentos*. Diferente das tarefas, que são previamente definidas e permanecem armazenadas em memória, um comportamento é estabelecido dinamicamente sempre que é necessário executar uma ação no ambiente virtual (se o ator for um avatar, suas ações são comandadas pelo usuário através de um dispositivo de entrada; caso seja um agente, elas são determinadas por seu mecanismo cognitivo). A definição do conjunto de tarefas que compõem o comportamento é dependente do contexto, levando em conta propriedades do ambiente e do próprio ator. Comportamentos que exigem deslocamento (por exemplo, visitar algum local do ambiente) baseiam-se na posição e orientação do ator e na distância até seu alvo, para calcular a direção para a qual ele deve virar-se e a distância a ser percorrida para chegar lá. Já aqueles sem deslocamento, normalmente são mais simples, podendo até consistir em uma única tarefa. Ainda assim, precisam levar em conta o contexto, caso contrário, parecerão muito artificiais. Para pegar um objeto, por exemplo, pode ser necessário apenas fazer um movimento com o braço; se este objeto estiver no chão, contudo, o comportamento correto será primeiro abaixar-se e depois pegá-lo.

Considere-se, por exemplo, um ator capaz de realizar as seguintes tarefas: d*ar um passo* e *sentar*. A partir destes movimentos, é possível compor diversos comportamentos diferentes, como os listados a seguir:

- sentar-se numa cadeira localizada a cinco passos de distância. Para obter este comportamento, a tarefa *dar um passo* é realizada cinco vezes em seqüência e seguida, então, pela tarefa *sentar*;

- sentar-se numa cadeira localizada a um passo de distância. Neste caso, é executada uma vez a tarefa *dar um passo* e depois a tarefa *sentar*;

- sentar-se. Este comportamento é composto somente por uma ocorrência da tarefa *sentar*.

A Figura 3.7 ilustra a formação de um comportamento de um ator. A figura apresenta a vista superior de um ambiente, no qual o ator é representado pelo quadrado pequeno, localizado no ponto *Pi*, e o local que ele deve atingir corresponde ao quadrado maior, localizado no ponto *Pf*. A orientação do ator no plano do ambiente é relativa ao sistema de referência indicado no canto inferior esquerdo. As tarefas que o ator é capaz de executar são: *dar um passo*, *dar meio passo*, *dar um quarto de passo*, *parar* e *girar* (estas tarefas foram realmente implementadas nos estudos de caso). O comportamento desejado do ator é *visitar (quadrado)*. A Figura 3.7 (a) mostra que, antes de executar o comportamento, a orientação do ator era de α graus positivos. Usando o Teorema de Pitágoras, é possível calcular tanto a orientação correta para apontar para o local alvo, como também a distância entre os pontos *Pi* e *Pf*. Na Figura 3.7 (b), a orientação ideal corresponde a β graus positivos, e a distância entre os pontos está representada por *d*. A primeira tarefa que compõe o comportamento é *girar(β)* (observe que, neste caso, o ator vai girar, de fato, $\beta - \alpha$ graus para a esquerda). A seguir, a quantidade de tarefas *dar um passo* incluídas depende da divisão de *d* pelo tamanho de um passo do ator. O resto desta divisão determina se, no final do conjunto de tarefas, ainda será adicionada alguma tarefa do tipo *dar meio passo* ou *dar um quarto de passo*. Para finalizar o comportamento, é inserida a tarefa *parar*.

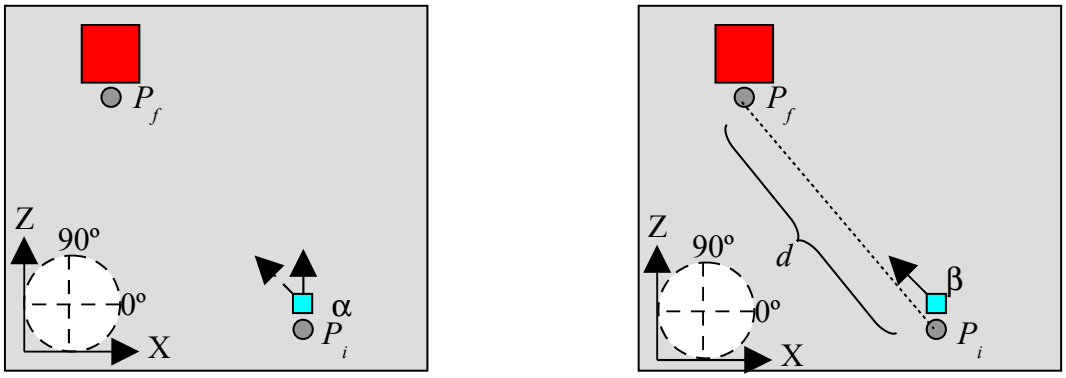


Figura 3.7 – Representação da direção e deslocamento de um agente: (a) posição inicial; (b) rotação = $\beta - \alpha$

3.5. Agente

Cada agente presente no ambiente virtual possui um sistema perceptivo e um mecanismo de raciocínio cognitivo. Além disso, pode estar representado por um corpo articulado, cujo modelo é o mesmo utilizado pelos avatares e descrito nas Seções 3.4.1 e 3.4.2. O comportamento que deve ser adotado pelo agente a cada momento é determinado por seu mecanismo cognitivo. O sistema perceptivo, por sua vez, é responsável pela transmissão das informações percebidas pelo agente ao seu mecanismo cognitivo, o qual é executado em um processo independente. A função conjunta do sistema perceptivo e do mecanismo

cognitivo equivale à de um usuário que controla um avatar, capacitando o agente a planejar suas ações e tomar decisões inteligentes, reagindo aos eventos que ocorrem no ambiente.

A cada instante de uma animação, o modelo permite que os agentes executem um ciclo em seu estado corrente. Para que isto seja possível, o estado de cada agente é armazenado e pode variar entre quatro opções (Figura 3.8): *envio*, quando o sistema perceptivo do agente está transmitindo informações ao seu mecanismo de raciocínio; *recebimento*, se o resultado de um ciclo de raciocínio está sendo recebido pelo corpo; *comportamento*, quando o agente está iniciando a execução de uma das tarefas de um comportamento; *tarefa*, se o agente está executando um dos passos de uma tarefa. Os agentes podem encontrar-se em diferentes contextos a cada momento. Por exemplo, enquanto um agente recebe informação de percepção, outro pode estar enviando uma requisição de ação, outros podem estar executando ações, e tudo isso de forma concorrente.

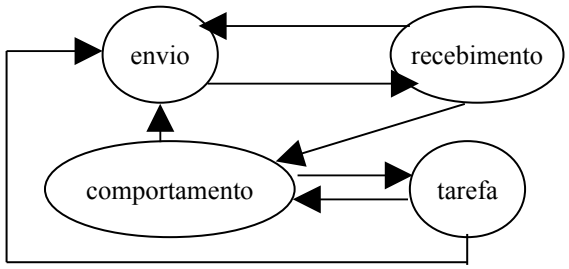


Figura 3.8 – Máquina de estados dos agentes

A tarefa de comunicação entre o modelo articulado e o mecanismo de raciocínio pode ser resumida em quatro passos conceituais (Figura 3.9):

1) o estado do ambiente é percebido pelo agente através do seu sistema perceptivo;

2) esta informação é traduzida para a sintaxe apropriada e transferida para o seu processo cognitivo;

3) o resultado do raciocínio é recebido, e o agente decide qual comportamento deve ser adotado;

4) ao executar este comportamento, o agente age no ambiente gerando eventos. A partir de nova percepção, o ciclo é repetido.

No caso de agentes que não possuem corpo, os passos 3 e 4 são unificados, sendo que as decisões tomadas pelo mecanismo de raciocínio geram eventos no ambiente. As Seções 3.5.1 e 3.5.2 descrevem, respectivamente, o sistema perceptivo e o mecanismo de raciocínio cognitivo dos agentes.

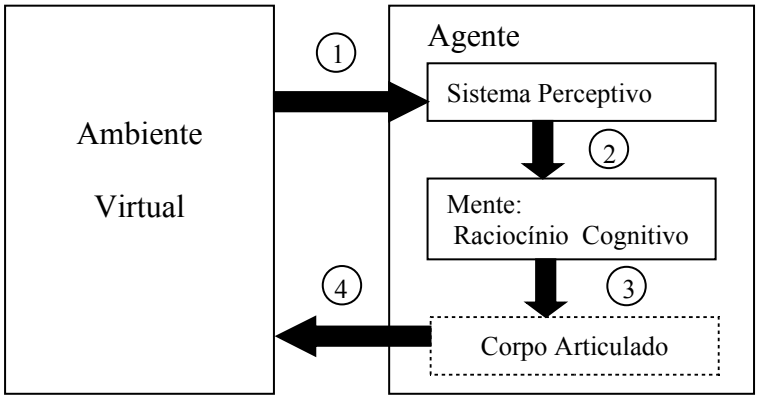


Figura 3.9 – Ciclo de ação dos agentes

3.5.1. Sistema Perceptivo

O mecanismo de raciocínio do modelo requer que esteja disponível informação sensorial sobre o ambiente (considerando eventualmente dados incorretos que poderiam resultar de falhas nos sensores). A forma mais realista de fornecer esta informação aos agentes seria dotá-los de visão e audição sintéticas, como proposto por Noser e Thalmann [NOS 96]. A arquitetura proposta, contudo, ainda utiliza sensores virtuais simplificados, mas que são suficientes para passar a cada agente somente dados parciais e condizentes com seu contexto.

A maneira utilizada para implementar o sistema de percepção foi determinar previamente, para cada aplicação, o conjunto completo de percepções que podem ser relevantes para cada agente. Para representar este conjunto, são usadas duas listas cujas posições correspondem às percepções. A primeira destas listas armazena expressões (*strings*) que correspondem às percepções, por exemplo: "*box(blue)*"; "*request(red)*"; "*full(2)*" (estas percepções são utilizadas no estudo de caso apresentado na Seção 5.2. Na Seção 5.2.1 são listadas e descritas todas as percepções dessa aplicação). A outra lista armazena valores inteiros, os quais têm a função de informar, a cada instante da execução da aplicação, quais percepções estão ativas e quais estão inativas (o valor 0 indica percepção inativa, ao passo que valores positivos correspondem a percepções ativas). O conjunto de expressões que correspondem às percepções ativas são reunidas em uma única expressão geral, a qual é transmitida ao mecanismo de raciocínio. A Figura 3.10 demonstra este processo.

Figura 3.10 – Formação da expressão geral contendo percepções ativas de um agente

3.5.2. Mecanismo Cognitivo (Mente)

O mecanismo de raciocínio dos agentes foi planejado visando capacitá-los a interagir com avatares e com outros agentes (apesar de não se comunicarem diretamente com nenhum deles) em um ambiente virtual, reagindo apropriadamente a circunstâncias inesperadas. Para tanto, foi escolhida a arquitetura BDI, a qual tem sido largamente usada em sistemas complexos e dinâmicos, nos quais os agentes precisam agir com base em informação (que pode ser incompleta e incorreta) sobre outros agentes e sobre o ambiente no qual estão inseridos. O modelo BDI foi descrito na Seção 2.4.2.2. A seguir, será apresentada uma breve descrição da linguagem de programação orientada a agentes AgentSpeak(L), a qual é baseada na teoria BDI e foi utilizada neste trabalho.

3.5.2.1. AgentSpeak(L)

Rao e Georgeff [RAO 95] propõem uma arquitetura abstrata que segue a teoria do modelo BDI. Salientam, contudo, que a teoria BDI, em sua concepção original, não oferece um sistema prático para agentes racionais, uma vez que não há indicação de como tornar os procedimentos de geração de opções e deliberação rápidos o suficiente para aplicações em tempo real. O tempo de raciocínio é potencialmente ilimitado, destruindo assim a reatividade que é essencial para o agente.

Para resolver este problema, os autores usam representações que limitam o poder do modelo, mas que o tornam mais prático para agentes racionais. Entre outras medidas, as informações sobre os meios para atingir estados futuros do ambiente (objetivos), além das opções disponíveis ao agente, podem ser representadas por *planos*. O corpo de um plano descreve as ações primitivas que o agente deve executar ou outros objetivos que deve atingir para que o plano tenha sucesso. A satisfação de uma pré-condição e a ocorrência de um evento *disparador* são as condições para que um plano seja escolhido. Baseada nestas simplificações práticas da teoria, a linguagem AgentSpeak(L) [RAO 96] foi desenvolvida para a implementação de agentes BDI.

AgentSpeak(L) é uma extensão natural da programação lógica para a arquitetura de agentes BDI [RAO 95]. Desde que Rao a propôs como uma linguagem de programação abstrata [RAO 96], vários trabalhos têm sido realizados em vários aspectos da linguagem. Algumas extensões para AgentSpeak(L) foram propostas [BOR 2002a], e um interpretador para esta linguagem estendida foi introduzido. As extensões visam oferecer uma linguagem de programação mais prática; a linguagem estendida permite também a especificação de relações entre planos e critérios quantitativos para sua execução.

Bordini e Moreira [BOR 2002b] deram uma semântica operacional a AgentSpeak(L), a qual usaram na especificação de uma estrutura (*framework*) para comprovar propriedades BDI de agentes AgentSpeak(L). Neste artigo também foi mostrada a combinação particular de princípios relativos à tese de assimetria [RAO 98] satisfeitos por qualquer agente

AgentSpeak(L). Isto é relevante para assegurar a racionalidade de agentes programados em AgentSpeak(L), e também representou um passo na direção de fundamentar formalmente uma versão simplificada da lógica BDI [RAO 98] em termos de processos computáveis (expressados como programas AgentSpeak(L)).

3.5.2.1.1. Sintaxe

Um agente AgentSpeak(L) é criado a partir da especificação de um conjunto de crenças base e de um conjunto de planos. Um átomo de crença é simplesmente um predicado de primeira ordem na notação usual, e átomos de crença ou sua negação formam literais de crença. Um conjunto inicial de crenças é uma coleção de átomos de crença.

AgentSpeak(L) distingue dois tipos de objetivos: *objetivos de realização* (*achievement goals*) e *objetivos de teste* (*test goals*). Objetivos de realização são predicados (como para crenças) prefixados com o operador '!', enquanto que objetivos de teste são prefixados com o operador '?'. Objetivos de realização declaram que o agente deseja atingir um estado do mundo no qual o predicado associado é verdadeiro (na prática, estes objetivos iniciam a execução de planos). Um objetivo de teste, por sua vez, declara que o agente deseja testar se o predicado associado é uma crença (se pode ser unificado com as crenças base desse agente).

A seguir, é introduzida a noção de *evento disparador* (*triggering event*). Este é um conceito muito importante nesta linguagem, uma vez que eventos disparadores definem quais eventos podem iniciar a execução de planos. Existem dois tipos de eventos disparadores: relacionados à adição ('+') ou à remoção ('–') de atitudes mentais (crenças ou objetivos).

Planos referem-se às ações básicas que um agente é capaz de executar em seu ambiente. Tais ações também são definidas como predicados de primeira ordem, mas com símbolos de predicado especiais (chamados símbolos de ação) usados para distingui-los. A sintaxe corrente de programas AgentSpeak(L) é baseada na definição de planos apresentada abaixo. É importante lembrar que o projetista de um agente AgentSpeak(L) especifica somente um conjunto de crenças e um conjunto de planos.

Se *e* é um evento disparador, *b1, ..., bm* são literais de crença, e *h1, ..., hn* são objetivos ou ações, então "*e : b1 & ... & bm <– h1; ...; hn*." é um plano. Um plano AgentSpeak(L) possui um cabeçalho (a expressão à esquerda da seta), o qual é formado por um evento disparador (denotando o propósito desse plano) e por uma conjunção de literais de crença representando um contexto (separada do evento disparador por ':'). A conjunção de literais no contexto precisa ser satisfeita para que o plano seja executado (ou seja, o contexto precisa ser uma conseqüência lógica das crenças correntes do agente em questão). Um plano possui também um corpo (a expressão à direita da seta), o qual é uma seqüência de ações básicas ou (sub)objetivos que o agente possui para atingir (ou testar) quando o plano é disparado.

Para facilitar a compreensão das intuições relacionadas a planos AgentSpeak(L), vamos considerar os seguintes exemplos:

```
+concert(A,V) : likes(A)
  <- !book_tickets(A,V).

+!book_tickets(A,V) : not(busy(phone))
  <- call(V);
     ... ;
     !choose_seats(A,V).
```

O primeiro plano estabelece que, quando um concerto do artista A no local V é anunciado (de modo que, por percepção do ambiente, uma crença *concert(A,V)* é adicionada), então, se o agente gosta do artista A, ele passará a ter o novo objetivo de reservar entradas para o concerto. O segundo plano determina que, quando o agente adota o objetivo de reservar entradas para a apresentação de A em V, se o telefone não estiver ocupado, então ele poderá executar a ação básica *call(V)* (assumindo que "fazer uma chamada telefônica" é uma ação atômica que o agente pode executar), seguida por um certo protocolo para reservar entradas (indicado por '…') e depois pela execução de um plano para escolher os assentos para a apresentação A no local V.

3.5.2.1.2. Semântica Informal

A seguir, são discutidas algumas noções relacionadas à interpretação de programas AgentSpeak(L). *Intenções* são cursos de ação particulares aos quais um agente comprometeu-se visando atingir um determinado objetivo; cada intenção é uma pilha de *planos parcialmente instanciados*, ou seja, planos nos quais apenas algumas das variáveis foram instanciadas. Um *evento*, o qual pode disparar a execução de um plano, pode ser *externo*, quando originado a partir de percepção do ambiente, ou *interno*, quando gerado pela execução de um plano pelo próprio agente (por exemplo, um objetivo de realização localizado no corpo de um plano é um evento de adição de objetivo que pode tornar-se um evento disparador. Considerando o exemplo da seção anterior, se o agente passar a ter o objetivo *!book_tickets*, será gerado um evento interno *+!book_tickets*, o qual funcionará como evento disparador do segundo plano). Formalmente, um evento é um par $<te,i>$, onde *te* é um evento disparador e *i* é uma intenção. Para eventos internos, *i* é a intenção que gerou o evento, e para eventos externos *i* é T (a *intenção verdadeira*).

Um agente AgentSpeak(L) é formalmente definido por uma tupla $<E, B, P, I, A, S_E, S_O, S_I>$, na qual E é um conjunto de eventos, B é um conjunto de crenças base, P é um conjunto de planos, I é um conjunto de intenções, e A é um conjunto de ações (no qual são inseridas as ações que o agente decide executar). A função de seleção S_E seleciona um evento do conjunto E; a função de seleção S_O seleciona uma opção ou um plano aplicável de um conjunto de planos aplicáveis; e S_I seleciona uma intenção do conjunto I (a intenção escolhida é então executada).

A cada ciclo de interpretação do programa de um agente, AgentSpeak(L) atualiza uma lista de eventos, os quais podem ser gerados a partir de percepção do ambiente ou da execução de intenções (quando subobjetivos são especificados no corpo do plano). Uma função de revisão de crenças é requerida na arquitetura global do agente (esta função é implícita na definição do interpretador de Rao, mas normalmente tornada explícita na arquitetura BDI genérica). É assumido que crenças são atualizadas a partir de percepção, e

sempre que ocorrem modificações nas crenças do agente, isto implica na inserção de um evento no conjunto de eventos.

As funções de seleção S_E, S_O, e S_I fazem parte da definição de um agente AgentSpeak(L). Trabalhos anteriores em AgentSpeak(L) não abordaram como os usuários especificam tais funções, mas elas são assumidas como específicas do agente. Após S_E selecionar um evento, AgentSpeak(L) precisa unificá-lo com eventos disparadores nos cabeçalhos de planos. Isto gera um conjunto de todos os *planos relevantes*. Quando unifica a parte de contexto dos cabeçalhos de planos neste conjunto com a base de crenças do agente, AgentSpeak(L) determina um conjunto de *planos aplicáveis* (planos que podem realmente ser usados para lidar com o evento escolhido). Então, S_O seleciona um único plano aplicável deste conjunto e, ou empilha esse plano no topo de uma intenção existente (se o evento era interno), ou então cria uma nova intenção no conjunto de intenções (se o evento era externo). Cada intenção de um agente é, desta forma, uma pilha de planos parcialmente instanciados.

Tudo o que permanece para ser feito neste estágio é selecionar uma única intenção para ser executada no ciclo corrente. É importante observar que cada evento externo para o qual existe um plano aplicável gera uma pilha independente de planos parcialmente instanciados dentro do conjunto de intenções. A função S_I seleciona uma das intenções do agente (ou seja, uma das pilhas de planos independentes do conjunto de intenções). No topo dessa intenção existe um plano, e a fórmula no início do seu corpo é tomada para execução. Isto implica em uma de três possibilidades: uma ação básica é executada pelo agente em seu ambiente; um evento interno é gerado (caso o subobjetivo seja um objetivo de realização); ou um objetivo de teste é executado (o que significa que o conjunto de crenças precisa ser consultado). Se a intenção é uma ação básica ou um objetivo de teste, o conjunto de intenções deve ser atualizado. No caso de objetivo de teste, ocorre instanciação adicional de variáveis no plano parcialmente instanciado que continha o objetivo (e o mesmo é removido da intenção da qual ele foi retirado). No caso em que uma ação básica é selecionada, a atualização necessária no conjunto de intenções é simplesmente remover essa ação da intenção. Quando uma fórmula removida marca o final do corpo de um subplano, o subobjetivo que o gerou é também removido da intenção (caso contrário, a intenção inteira é terminada). Isto encerra um ciclo de execução, e AgentSpeak(L) inicia tudo novamente, checando o estado do ambiente após a atuação dos agentes sobre ele, gerando eventos e assim por diante.

4. Implementação do Modelo de Ator Sintético Cognitivo

4.1. Introdução

Este capítulo descreve a implementação do modelo proposto neste trabalho e apresentado no capítulo anterior. Considerando os elementos apresentados, o ambiente virtual é implementado pela classe **Environment**; cada avatar é representado por uma instância da classe **Avatar**; e cada agente corresponde a uma instância de **Agent**. O diagrama da Figura 4.1 apresenta o relacionamento destes elementos com as classes que representam seus componentes. Os corpos dos agentes e avatares são instâncias da classe **VPBody**, e os movimentos sobre estes corpos são implementados pelas classes **VPJointMotion**, **Task** e **Behaviour**, correspondendo aos três níveis descritos na Seção 3.4.2. Cada instância de **VPJointMotion** contém um movimento individual numa articulação, **Task** armazena um conjunto destes movimentos, compondo uma tarefa, e **Behaviour** reúne um conjunto de tarefas que podem ser usadas num comportamento. É importante salientar que o mecanismo de raciocínio de cada agente é implementado em um processo cliente separado e não está representado no diagrama. Os objetos que compõem o ambiente e os membros dos corpos são representados pela classe abstrata **VPGraphicObj**. As classes cujos nomes começam com "VP" fazem parte do *framework* VPat (*Virtual Patients*), o qual será apresentado na próxima seção.

A Seção 4.3 descreve a implementação do modelo de corpo articulado utilizado por avatares e agentes. Na Seção 4.4, é abordada a utilização do interpretador AgentSpeak(L) como mecanismo de raciocínio cognitivo dos agentes. A Seção 4.5 explica a implementação do ambiente virtual, e a Seção 4.6 descreve o funcionamento dos avatares e, principalmente, dos agentes. Na Seção 4.7 são apresentados detalhes referentes ao controle e visualização das animações. Finalmente, a Seção 4.8 descreve alguns detalhes de implementação do modelo, especialmente com relação à linguagem, sistema operacional e comunicação entre processos via *sockets*.

4.2. *Framework* VPat

VPat é um *framework* orientado a objetos que fornece classes básicas para auxiliar na implementação de aplicações de computação gráfica na área médica. As classes comuns do projeto podem ser compartilhadas e estendidas, facilitando o desenvolvimento de classes mais especializadas de acordo com a necessidade da aplicação. O conjunto de classes básicas do VPat compreende desde primitivas gráficas até elementos de mais alto nível, como objetos gráficos ou cenas [FRE 2003].

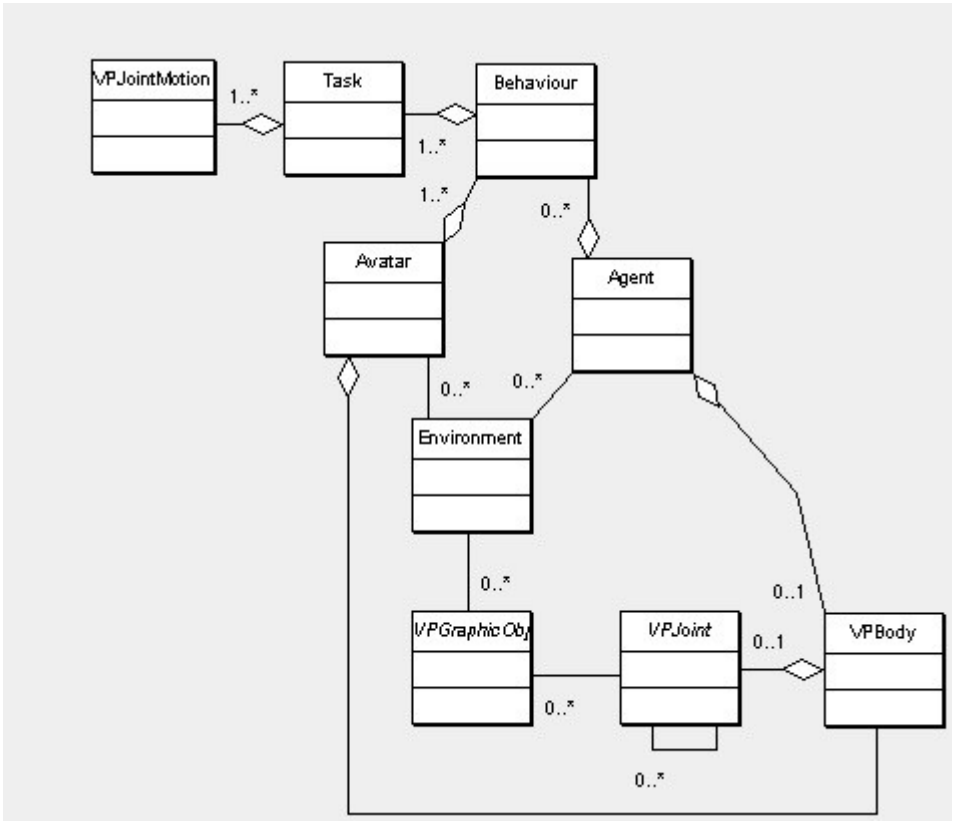


Figura 4.1 - Relacionamento das classes do modelo com o *Framework* VPat

A Figura 4.2 apresenta o diagrama com as classes básicas do *framework* VPat. Pontos podem ser representados nos espaços bidimensional e tridimensional pelas classes **VPPoint2D** e **VPPoint3D**, as quais são extensões da classe abstrata **VPPoint**. As classes **VPVector3D** e **VPVertex3D** também herdam funcionalidade de **VPPoint**; **VPVector3D** acrescenta métodos que implementam operações sobre vetores e pontos, e **VPVertex3D** possui um vetor normal associado ao ponto. É importante observar que essas classes aparecem isoladas no diagrama porque seu relacionamento com o restante do *framework* ocorre por intermédio de outras classes que não são básicas. A classe **VPMatrix** implementa matrizes 4x4 de valores reais, as quais podem ser usadas, por exemplo, para representar transformações entre sistemas de referência. As classes **VPSpotLight**, **VPPointLight** e **VPDirectionalLight**, extensões da classe **VPLight**, implementam as luzes utilizadas para calcular a iluminação de cenas, enquanto que a classe **VPCamera** representa câmeras virtuais, utilizadas para obter a imagem de uma cena virtual correspondente ao ponto de vista do observador.

Os elementos visíveis que fazem parte de uma cena virtual são *objetos gráficos* implementados a partir da classe abstrata **VPGraphicObj**. Extensões desta classe permitem a criação de primitivas geométricas, como cubos ou cilindros, malhas de polígonos e superfícies curvas. Objetos gráficos tanto podem estar estáticos em uma cena, como móveis, fazendo parte de uma estrutura articulada.

Uma cena é uma instância da classe **VPScene**, e é formada por um conjunto de luzes (objetos da classe **VPLight**), um conjunto de câmeras (objetos da classe **VPCamera**) e um conjunto de objetos gráficos (objetos da classe **VPGraphicObj**). A classe abstrata **VPView** é encarregada de apresentar o modelo, representado pela classe **VPScene**, ao usuário. Além disso, também é responsável pela interpretação dos eventos gerados pelos dispositivos de

interação. Portanto, o modelo definido em um objeto **VPScene** é independente do modelo de interação especificado nas subclasses da **VPView**, as quais são dependentes de uma plataforma específica.

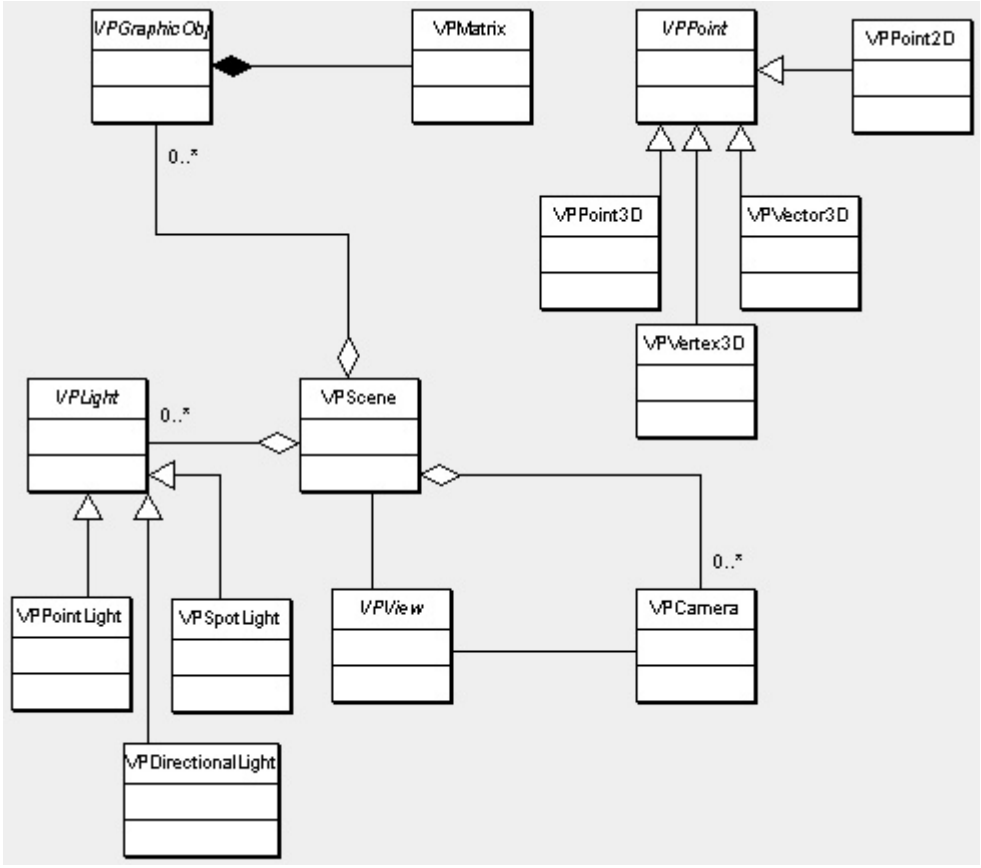


Figura 4.2 – Classes básicas do *Framework* VPat

4.3. Modelo Articulado

4.3.1. Representação

O modelo articulado descrito na Seção 3.3.1 permite criar corpos com diferentes estruturas, dependendo do número de articulações utilizadas, de sua topologia e da quantidade de graus de liberdade de movimento. Também a aparência dos corpos produzidos é variável, de acordo com os objetos gráficos associados a cada junta. Para fazer a descrição de um corpo neste modelo, utiliza-se um arquivo em formato XML [W3C 2003], padrão mundial para troca de dados estruturados. A estrutura do formato XML, a qual não é objeto desta dissertação, facilita a especificação de dados estruturados hierarquicamente, como é o caso do esqueleto humano. O modelo foi implementado como parte do *framework* VPat, usando o

paradigma de orientação a objetos, o que facilita sua extensão através da criação de novas classes.

Os corpos articulados que fazem parte de uma animação são instanciados no momento em que são inseridos no sistema. Cada um deles é definido por um arquivo XML, sendo que um único arquivo pode ser utilizado para criar vários corpos, caso sejam compostos pelas mesmas articulações e objetos gráficos. Os atributos de cada corpo são armazenados em uma instância da classe **VPBody**, e todos os parâmetros que definem suas articulações, graus de liberdade (DOFs) e objetos gráficos associados são armazenados, respectivamente, em instâncias das classes **VPJoint**, **VPDof** e **VPGraphicObj**. A relação entre estas classes é ilustrada na Figura 4.3.

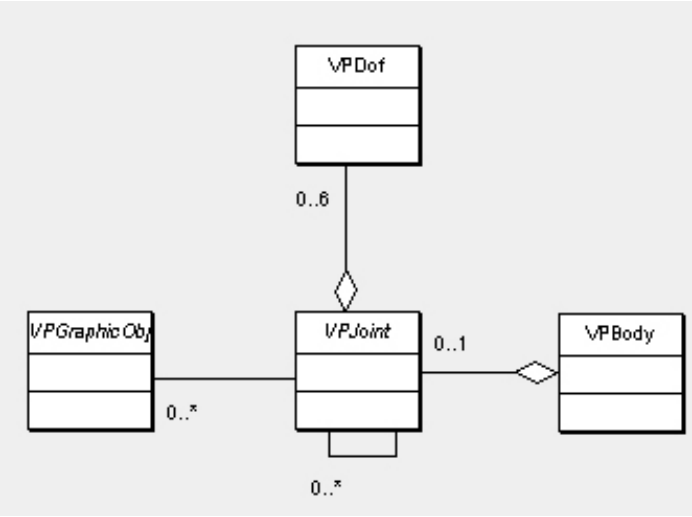


Figura 4.3 – Diagrama de classes usadas para representação de corpos articulados

Um objeto da classe **VPBody** possui três atributos principais. O primeiro deles, *rootJoint*, indica um conjunto hierárquico de articulações, representado por uma junta raiz, a partir da qual pode-se obter toda a hierarquia. Conceitualmente, a junta raiz não pode ser considerada uma articulação, embora seja implementada como tal. Sua função é servir de referência para a hierarquia, além de determinar a orientação do corpo em relação ao sistema de referência global (através de três DOFs de rotação). Os outros atributos importantes do corpo são *position* e *orientation*, os quais determinam sua posição no ambiente e seu ângulo de giro horizontal (em torno do eixo perpendicular à sua base). O atributo de posição é do tipo **VPPoint3D**, definido por três valores reais, enquanto que a orientação é definida, em graus, por um valor real variando entre –180.0 e 180.0.

Cada articulação do corpo, representada pela classe **VPJoint**, possui um atributo *childList*, contendo referências a todas suas juntas-filhas. Uma lista de objetos gráficos associados à articulação é armazenada no atributo *shapeList*, enquanto *dofList* recebe o conjunto de DOFs que representam seus graus de liberdade de movimento. O atributo *lim*, do tipo **VPMatrix**, guarda a matriz de instanciamento local (LIM, abreviação de *Local Instance Matrix*) da junta. Além de todos esses dados, a articulação possui uma descrição, armazenada como uma *string* no atributo *description*. Esta descrição da articulação é importante para a especificação dos movimentos sobre ela. A classe **VPJoint** representa uma articulação genérica, contendo as propriedades que são comuns a todas as juntas humanas. Para que se possa criar instâncias de juntas é necessário utilizar uma das subclasses de **VPJoint**, de acordo com o tipo de junta que se deseja criar. Os tipos implementados são os seguintes:

a) **VPUniaxialJoint**

Representa as juntas do modelo que têm apenas um grau de liberdade de movimento. Esta classe possui um DOF, que define um eixo de movimento rotacional. Ela disponibiliza um método para movimentação em torno desse eixo chamado *vpSetFlexionTo*, que recebe como parâmetro um valor inteiro, entre 0 e 1, que define a posição angular do eixo.

b) **VPBiaxialJoint**

Representa as juntas com dois graus de liberdade de movimento. Esta classe possui dois DOFs, cada um deles definindo um eixo de movimento rotacional, não necessariamente ortogonais. Além do método *vpSetFlexionTo* usado para movimentação em torno do primeiro eixo, disponibiliza o método *vpSetAdductionTo* para atualizar a posição angular do segundo eixo.

c) **VPPolyaxialJoint**

Semelhante ao tipo anterior, apresenta ainda um terceiro grau de liberdade, também representado por um DOF de rotação. Ao seu conjunto de métodos se adiciona o *vpSetTwistTo*, utilizado para movimentação em torno do terceiro eixo de movimento.

d) **VPPlaneJoint**

Além dos três eixos já apresentados, sua lista de DOFs inclui a presença de outros três, estes para permitir os movimentos de translação. Portanto, também são apresentados três novo métodos: *vpSetXSlideTo*, *vpSetYSlideTo* e *vpSetZSlideTo*.

Objetos da classe DOF possuem uma posição e um eixo que define seu movimento. O atributo *position* é uma instância de **VPPoint3D**, e *axis* é um vetor do tipo **VPVector3D**. A matriz LIM que associa o DOF à sua junta é representada no atributo *lim*, do tipo **VPMatrix**. Os valores angulares mínimo e máximo que o DOF pode atingir são armazenados em radianos nos atributos *minAngle* e *maxAngle*. O ângulo corrente de um DOF é especificado pelo parâmetro *currentPosition*, o qual pode variar entre 0 e 1 (0 corresponde ao limite angular mínimo, e 1 equivale ao ângulo máximo alcançável). A classe **VPDof** possui ainda outros atributos que permitem implementar movimentos mais precisos e realistas. Tais movimentos, contudo, não fazem parte dos objetivos desta dissertação. Assim, esses atributos não serão descritos, com a finalidade de evitar desvio de foco. Mais detalhes podem ser encontrados em Maciel *et al.* [MAC 2002].

Quando um método do tipo *vpSet...To* (por exemplo, *vpSetXSlideTo*) da junta que possui um DOF é invocado, o que ocorre é uma chamada para o método *vpMoveTo* do DOF em questão. Após a movimentação do DOF e a conseqüente modificação de sua LIM, a junta recompõe a sua LIM a partir dos seus DOFs, passando a refletir nos objetos geométricos de sua lista, a modificação feita no DOF.

A Figura 4.4 ilustra um trecho de um arquivo XML contendo a descrição da articulação do ombro esquerdo, identificada como *lshoulder* e contendo um grau de liberdade (definido entre o par de marcas <*dof*> e <*/dof*>) e um objeto gráfico associado (especificado entre as marcas <*shape*> e <*/shape*>). Neste exemplo, os ângulos mínimo e máximo do DOF são, respectivamente, –0,5 e 0,8 radianos (valores de *min* e *max*), e o ângulo inicial do DOF, o qual é obtido a partir do valor de *rest* (que é armazenado como o valor inicial de

currentPosition), é 0,15 radianos (como *rest* = 0,5, o ângulo inicial é exatamente a média entre os ângulos mínimo e máximo).

```
<joint description="lshoulder" parent="torax" type="uniaxial">
      <dof description="flexionS">
            <position x="0.00" y="-10.0" z="0.00"/>
            <axis x="1.00" y="0.00" z="0.00"/>
            <range min="-0.5" max="0.8"/>
            <comfort_range min="-0.5" max="0.8" rest="0.5"/>
      </dof>
      <shape description="lhumero" file_name = "lhumerus.iv">
            <position x="0.0" y = "-10.0" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
</joint>
```

Figura 4.4 – Descrição da articulação do ombro esquerdo em formato XML

O uso de arquivos XML para a especificação dos corpos articulados oferece flexibilidade ao processo. Entretanto, a necessidade de especificar a topologia das articulações, as posições de todos os elementos que compõem os membros do corpo, e as posições e intervalos angulares de cada DOF requer bastante tempo e trabalho. Para simplificar a criação de corpos neste modelo, está sendo desenvolvido, em paralelo a este trabalho, um modelador interativo mais amigável.

4.3.2. Movimento

Como descrito na Seção 3.4.2, o movimento dos corpos articulados é dividido em três níveis: movimentos em articulações individuais, tarefas e comportamentos. Destes, os dois primeiros são especificados através de arquivos XML e carregados, a exemplo da estrutura dos corpos, quando os atores são inicializados, permanecendo disponíveis em memória durante a animação. Já os comportamentos, que correspondem ao nível mais alto de movimento, não ficam armazenados na memória, pois dependem do contexto para serem compostos.

Um movimento em uma determinada articulação é representado pela classe **VPJointMotion**. Como as articulações podem ter mais de um grau de liberdade, é preciso especificar qual de seus DOFs será movimentado, e esta informação é armazenada no atributo *motionType*. Movimentar um DOF significa alterar o valor de seu atributo *currentPosition*. Para fazer isso, **VPJointMotion** possui os atributos *parameter*, *t0*, *tf* e *deltaS*. O primeiro deles informa a posição assumida pelo DOF no fim do movimento; sua mudança de posição, contudo, não deve ocorrer num único instante. Para tornar o movimento suave, os demais parâmetros determinam seu instante inicial e final (*t0* e *tf*, respectivamente), além da fração do movimento a ser aplicada em cada intervalo de tempo (*deltaS*).

Um objeto da classe **Task** representa uma tarefa e possui como parâmetros o tempo atual (*time*) e o tempo final (*tFinal*) do movimento, o intervalo de tempo entre os quadros da animação (*deltaT*), e uma lista de objetos da classe **VPJointMotion** (*motionTimeline*). Durante a execução da tarefa, o tempo é contado em intervalos de tamanho definido no atributo *deltaT*. Toda vez que o atributo *time* atinge o instante inicial de um objeto da **VPJointMotion**, o estado da junta correspondente a este objeto passa a ser modificado a cada intervalo de tempo, até que *time* seja igual ao instante final do objeto. Para ilustrar essas definições, a Figura 4.5 apresenta um arquivo XML que contém a definição da tarefa *pegar*, a qual exige movimentos nas articulações da coluna (*spine*) e dos ombros (*lshoulder* e *rshoulder*) e cotovelos (*lelbow* e *relbow*) de ambos os braços. Neste exemplo, o par de marcas <*task*> e <*task*> define uma tarefa, e cada marca <*motion*> especifica um movimento em uma junta determinada (*joint_name* informa a junta, *t0* e *tf* determinam os instantes inicial e final do movimento e *type* indica o DOF envolvido).

```
<task delta_t="0.05">
  <motion joint_name="spine" t0="0.0" tf="0.3" type="FLEX" parameter="0.25"/>
  <motion joint_name="spine" t0="0.35" tf="0.5" type="FLEX" parameter="0.0"/>
  <motion joint_name="rshoulder" t0="0.05" tf="0.35" type="FLEX" parameter="0.0"/>
  <motion joint_name="rshoulder" t0="0.35" tf="0.5" type="FLEX" parameter="0.3"/>
  <motion joint_name="lshoulder" t0="0.05" tf="0.35" type="FLEX" parameter="0.0"/>
  <motion joint_name="lshoulder" t0="0.35" tf="0.5" type="FLEX" parameter="0.3"/>
  <motion joint_name="relbow" t0="0.1" tf="0.25" type="FLEX" parameter="0.0"/>
  <motion joint_name="relbow" t0="0.25" tf="0.35" type="FLEX" parameter="0.2"/>
  <motion joint_name="relbow" t0="0.35" tf="0.5" type="FLEX" parameter="0.5"/>
  <motion joint_name="lelbow" t0="0.1" tf="0.25" type="FLEX" parameter="0.0"/>
  <motion joint_name="lelbow" t0="0.25" tf="0.35" type="FLEX" parameter="0.2"/>
  <motion joint_name="lelbow" t0="0.35" tf="0.5" type="FLEX" parameter="0.5"/>
</task>
```

Figura 4.5 – Arquivo XML com especificação de movimentos para compor uma tarefa

Os comportamentos são representados pela classe **Behaviour**. Os atributos de um objeto desta classe são *owner*, que indica o agente ou avatar ao qual o comportamento é associado, e *task*, contendo o conjunto de tarefas que este ator é capaz de executar. O método *compose* é encarregado de estabelecer a seqüência de tarefas que compõem um comportamento. Dois parâmetros são passados como argumentos deste método: *movement type* e *target*. O primeiro deles serve para identificar as tarefas envolvidas, enquanto que *target* permite analisar se há necessidade de deslocamento e, neste caso, calcular a distância a ser percorrida pelo ator e sua orientação ideal para chegar ao seu objetivo em linha reta. Conhecendo a distância até o alvo e o tamanho de um passo do ator, o método pode estabelecer a quantidade de passos necessários. De forma semelhante, o ângulo de orientação calculado determina quantos graus o ator deve girar para apontar para o alvo. É importante observar, neste momento, que ainda não foi implementado no modelo nenhum controle de colisão. Assim, as aplicações apresentadas no capítulo seguinte envolvem ambientes pouco congestionados, visando permitir deslocamentos em linha reta.

4.4. Modelo Cognitivo

A sintaxe e a semântica da linguagem AgentSpeak(L) foram descritas no capítulo anterior. Nesta seção, serão abordados aspectos relativos à execução de programas AgentSpeak(L) no interpretador. Como já foi dito, o interpretador AgentSpeak(L) é executado em um processo cliente e necessita, por conseguinte, de um servidor aberto a conexões via *socket*. Cada cliente rodando o interpretador equivale a um agente, cujo raciocínio é estipulado em um arquivo texto que é passado como parâmetro de entrada. Este arquivo armazena um programa AgentSpeak(L) e contém as crenças iniciais e os planos do agente.

Como está explicado no capítulo anterior, os planos definidos para um agente dependem da ocorrência de eventos para serem ativados. Assim, o interpretador mantém-se em estado de espera até receber alguma percepção externa. O processo servidor é quem implementa o ambiente no qual os agentes estão inseridos, e encarrega-se de enviar-lhes informação sensorial do ambiente. Esta informação é recebida pelo interpretador e usada na sua função de revisão de crenças, a qual gera eventos (externos) sempre que adicionar ou remover crenças da base de crenças em função da informação de percepção recebida. Um evento pode disparar a execução de um plano, dando início ao processo de raciocínio do agente. A partir de então, novos eventos podem ser gerados, a partir de novas percepções (eventos externos) ou resultando da execução de planos (eventos internos).

Para transmitir informação de percepção ao interpretador, o servidor dispõe de dois comandos. O primeiro deles, *@set(world(. . .))*, é usado para transmitir a percepção propriamente dita, a qual é representada, dentro dos parênteses, por um conjunto de predicados que seguem a sintaxe de AgentSpeak(L) e representam a informação sobre o ambiente, sendo separados entre si por vírgula. Para informar ao interpretador, por exemplo, que há um concerto do artista *Alaor* no *Teatro da OSPA* e que o telefone está ocupado (aproveitando o código apresentado na Seção 3.5.2.1.1), deve ser enviado o comando a seguir (incluindo o ponto e vírgula): *@set (world (concert (Alaor, OSPA), busy (phone)));*.

O conjunto de predicados recebido representa para o agente toda a sua percepção num dado instante. Para acrescentar alguma outra informação de percepção, não basta ao servidor enviar um novo comando contendo somente o predicado adicional, pois o comando *@set(world(...))* não é cumulativo; é necessário enviar novamente o conjunto completo de percepções. Por esta razão, a classe **Agent**, descrita na Seção 4.6, precisa manter controle contínuo das percepções ativas através de um *array* de controle, conforme mencionado na Seção 3.5.1.

O comando *@set(world(...))* apenas informa a percepção ao interpretador. Para fazê-lo executar um ciclo de raciocínio, é necessário enviar-lhe um comando *@step;*. A Seção 3.5.2.1.2 descreve o que ocorre durante um ciclo de interpretação do programa de um agente. Ao final deste ciclo, o interpretador envia ao servidor um *string* contendo o resultado, que pode consistir em uma ação básica, a qual deverá ser aplicada no ambiente, ou apenas na palavra *true*, indicando que possivelmente há algum plano em andamento (pode estar sendo executado um objetivo de teste, ou pode ter sido gerado um evento interno). Neste caso, é necessário enviar mais comandos *@step;*, mesmo que não exista nova situação de percepção a ser informada. Sempre que houver novos eventos perceptíveis, normalmente gerados por

ações básicas dos agentes, devem ser transmitidos ao interpretador através do par de comandos *@set(world(...));* e *@step;*.

4.5. Ambiente Virtual

O ambiente virtual é representado por um objeto da classe **Environment**, cujos atributos são um conjunto de objetos gráficos, uma lista de atores e outra de avatares, um objeto *socket* e um conjunto de lugares (pontos relevantes no ambiente). O *socket*, representado pelo atributo *servSock*, é um objeto da classe **TCPServSock**, criado no método construtor de **Environment**. Este objeto estabelece um servidor que aceita conexões de processos clientes na porta 10001. Os demais atributos são dependentes da aplicação que estabelece o ambiente, e armazenam dados que são carregados de arquivos texto.

Os objetos gráficos, representados por extensões da classe **VPGraphicObj**, são carregados no método *loadObjects* e armazenados no atributo *object*. O método *loadPlaces* armazena, no atributo *place*, um *array* de pontos (instâncias de **VPPoint3D**) que representam os locais relevantes no ambiente. Estes locais não precisam corresponder necessariamente a objetos; sua função é servir como referência para os movimentos dos atores. Em uma determinada aplicação, se for importante que os atores possam mover-se até o centro do ambiente, por exemplo, então o ponto que representa o centro é relevante e precisa ficar disponível a eles (que o usarão para cálculo de distância e orientação). Os atributos *agent* e *avatar* são *arrays* que armazenam, respectivamente, objetos das classes **Agent** e **Avatar**. A identificação de cada avatar e de cada agente, inseridos no ambiente através do método *loadActors*, corresponde à sua própria posição no *array* correspondente.

4.6. Agentes e Avatares

Avatares e agentes possuem, como atributos comuns, um corpo articulado, um conjunto de comportamentos e a identificação de seu estado corrente. Além destes, os agentes têm ainda um conjunto de atributos encarregados de controlar a comunicação com seu mecanismo cognitivo e as suas percepções do ambiente. As informações correspondentes a cada ator inserido no ambiente virtual são carregadas a partir de um arquivo texto.

No método construtor das classes **Agent** e **Avatar**, são inicializados os atributos *mode* e *body*. O primeiro determina o estado corrente do ator, é do tipo inteiro e pode variar entre 1 e 4, no caso dos agentes, e entre 2 e 4 para os avatares. O valor 1 representa o estado *envio*, que somente faz sentido para os agentes; o estado 2 é *recebimento*, indicando que um comando de ação será recebido (no caso dos agentes, proveniente do interpretador AgentSpeak, via socket; para os avatares, estipulado através de um dispositivo de entrada); os estados 3 e 4 são *comportamento* e *tarefa*. Os agentes sempre iniciam no estado *envio*, e os avatares no estado *recebimento*. O atributo *body*, por sua vez, representa o corpo articulado do

ator. A posição e orientação iniciais do corpo no ambiente, além do nome do arquivo XML que define sua estrutura, são carregados do arquivo texto que descreve o ator. Os comportamentos de um ator são carregados através do método *loadBehaviour* e armazenados no atributo *behaviour*.

O método construtor da classe **Agent** inicializa, além dos atributos descritos acima, um *array* que controla quais percepções individuais do agente estão ativas. Este *array*, representado pelo atributo *perceptions*, inicia com o valor 0 em todas as suas posições, indicando que não há nenhuma percepção ativa. Durante a execução da animação, ele é atualizado através dos métodos *addPerception* e *deletePerception*. O primeiro deles incrementa o valor de uma posição individual do *array* (ou seja, acrescenta uma percepção), e o segundo decrementa o valor de uma posição (retira uma percepção). O método *loadOutStrings* é responsável por armazenar, no *array outString*, as *strings* que representam as percepções individuais. De forma similar, o método *loadInStrings* carrega, no *array inString*, as *strings* que representam os possíveis comandos de ação retornados pelo interpretador (é importante lembrar que tanto as percepções como as ações disponíveis são definidas por quem programa a aplicação e carregadas na inicialização da mesma). Sempre que o agente precisar enviar informação de percepção ao interpretador (ou seja, quando o estado corrente do agente for *envio*), o método *composeOutString* é responsável por analisar o *array perceptions* e compor uma *string* contendo todas as percepções ativas, a qual será enviada através do *socket*.

O mecanismo cognitivo de um agente é implementado em um processo cliente rodando o interpretador AgentSpeak(L). Para estabelecer sua comunicação com o servidor, a classe **Agent** possui os atributos *clntSock*, do tipo **TCPSocket**, e *threadID*, do tipo **pthread_t** (este tipo está implementado na biblioteca pthread, que deve ser incluída no projeto). Cada cliente é executado em uma *thread* própria, com o objetivo de facilitar a programação dos *sockets*. O método *create_thread* tem a função de estabelecer o *socket* para depois utilizá-lo como parâmetro na criação de uma *thread*. O *socket* é gerado a partir de uma chamada do método *accept* da classe **TCPServerSocket**, e a *thread* é criada através da função *pthread_create*. A transmissão de *strings* entre o servidor e o cliente é possibilitada pelos métodos *send* e *recv* da classe **TCPSocket**.

4.7. Motor de Animação e Rendering

Uma aplicação é representada por uma classe derivada da classe abstrata **VPView**. Essa subclasse, que recebe o nome da própria aplicação, é encarregada de estabelecer as janelas necessárias para a visualização da animação, além de controlar a interação com dispositivos de entrada e o laço principal do programa. Tudo isto é implementado com auxílio das funções da biblioteca GLUT.

A criação das janelas ocorre no método construtor da classe da aplicação. Neste mesmo método, são especificados os nomes das funções utilizadas pela GLUT para tratamento dos eventos. Nas aplicações desenvolvidas, a função *workProc* é um *timer* que

controla o laço principal; *keyboard* é a função encarregada de tratar eventos do teclado, o qual é usado para controlar a câmera e também os avatares; a renderização da cena é executada na função *renderScene*.

A função *workProc*, executada a cada intervalo de tempo (estipulado na definição da função), é encarregada de controlar a geração de eventos aleatórios que podem ocorrer no ambiente virtual, além de analisar o estado corrente de cada ator e de executar seu próximo passo, atualizando as variáveis que representam propriedades do ambiente sempre que for necessário. Se um agente encontra-se no estado de *envio*, a função compõe sua *string* de percepções correntes (chamando o método *composeOutString* do agente) e a envia ao processo cliente correspondente a este agente, seguida por um comando "*@step;*". Para um ator (agente ou avatar) em estado de *recebimento*, a função recebe uma *string* com um comando de ação proveniente do interpretador ou do teclado (dependendo de ser um agente ou um avatar). A partir desta *string*, é decidido qual será o comportamento adotado pelo ator. Quando um ator encontra-se no estado de *comportamento*, é executado o método *getTask* (da classe **Behaviour**) para obter a tarefa seguinte do comportamento corrente, seguido pelo método *setTime* da classe **Task**, o qual define o instante de tempo inicial da tarefa. Se o estado corrente de um ator for *tarefa*, então *workProc* chama a função *motionKernel*, a qual executa um ciclo de movimento no seu corpo articulado. Este processo é ilustrado na Figura 4.6.

A função *motionKernel* é responsável pela execução de uma tarefa. A cada chamada da função, um ciclo do movimento é realizado, e, depois, o instante de tempo corrente (representado pelo atributo *time* da classe **Task**) é incrementado em um intervalo de tempo (definido pelo atributo *deltaT* de **Task**). Cada DOF com movimento previsto no ciclo corrente tem sua posição atualizada através do método *vpMoveTo*, o qual é invocado por um método do tipo *vpSet...To* da junta que possui este DOF.

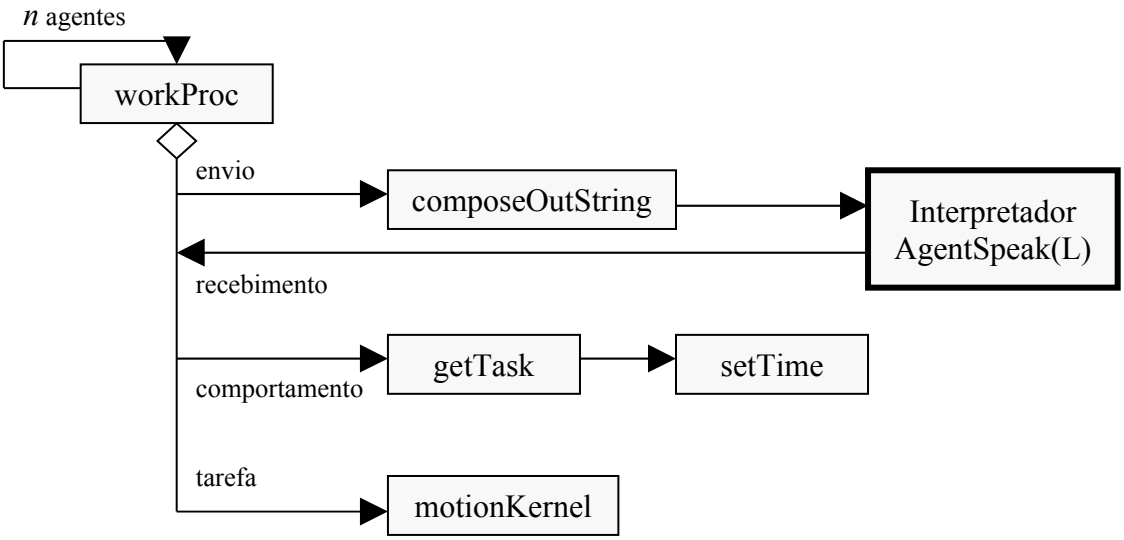


Figura 4.6 – Fluxo de controle dos personagens de uma aplicação

4.8. Detalhes de Implementação

O interpretador AgentSpeak(L), responsável pelo raciocínio cognitivo dos agentes, foi incorporado no sistema como um módulo funcional pronto, com parâmetros de entrada e de saída definidos. Ele foi programado em linguagem C e executa em plataforma Linux. Os outros elementos da arquitetura foram implementados utilizando a linguagem C++ e a biblioteca gráfica OpenGL, e seguem o paradigma de orientação a objetos, visando facilitar futuras extensões do sistema.

Todas as classes envolvidas com a especificação do ambiente virtual, com a representação e animação dos corpos articulados, e com a comunicação entre os corpos e os mecanismos cognitivos, são portáveis para Linux ou Windows sem necessidade de qualquer alteração. Para obter independência de plataforma na utilização de *sockets*, foi utilizada uma biblioteca de classes desenvolvida para uso pedagógico e disponibilizada na Internet em [DON 2002]. Esta biblioteca é brevemente descrita no anexo B.

5. Estudos de Caso

5.1. Introdução

Neste capítulo serão apresentados dois estudos de caso. O primeiro deles (Armazém Químico) é o mais importante, pois foi desenvolvido por último e ilustra a arquitetura implementada. O segundo (Senha Secreta), por sua vez, foi desenvolvido quando a arquitetura proposta ainda estava incompleta. A comunicação entre o interpretador AgentSpeak(L) e o corpo articulado dos agentes estava sendo testada, e ainda não havia sido definida a melhor forma de implementar o sistema perceptivo (nesta fase, havia necessidade de estudar melhor o funcionamento de programas AgentSpeak(L), com a finalidade de encontrar a forma de transmissão dos dados ao interpretador que melhor se adaptasse às necessidades da linguagem).

Assim, o código AgentSpeak(L) deste segundo estudo de caso não corresponde à solução mais apropriada para a aplicação proposta (a própria aplicação não é muito favorável ao modelo BDI e poderia ser desenvolvida com técnicas mais simples), mas foi importante para testar a comunicação entre mais de um corpo articulado e o mecanismo cognitivo correspondente a cada um deles. Além disso, este código serviu como base para a análise e definição de como deveria ser o sistema perceptivo dos agentes, uma vez que não se encontravam exemplos de aplicações usando AgentSpeak(L) na literatura ou Internet.

AgentSpeak(L) permite ao programador abstrair os fatores externos e concentrar-se no raciocínio do agente. Entretanto, os interpretadores AgentSpeak(L) assumem que a informação de percepção estará continuamente disponível. Desta forma, a interface entre o sistema articulado e o mecanismo cognitivo precisa controlar as percepções do ambiente. Um inconveniente desta abordagem é a necessidade de uma interface diferente para cada aplicação.

As aplicações desenvolvidas como estudos de caso são apresentadas nas seções a seguir. Naturalmente, é dada maior ênfase ao primeiro estudo de caso, o qual é tratado na Seção 5.2. A Seção 5.3, por sua vez, trata da aplicação experimental (detalhes de implementação não interessam neste caso, pois a arquitetura estava incompleta). É interessante observar como, neste estudo de caso, o código AgentSpeak(L) é mais confuso, pois encarrega-se também do controle das percepções do agente, além de seu raciocínio que visa à tomada de decisões (a própria localização do agente no ambiente é controlada por intermédio de crenças). Esta aplicação foi inserida na dissertação justamente com a finalidade de ilustrar a evolução da arquitetura e a necessidade que programas AgentSpeak(L) têm de ser abastecidos por um sistema perceptivo externo.

5.2. Armazém Químico

O cenário usado para ilustrar a arquitetura proposta é o de um robô em um armazém. O robô é encarregado de guardar *caixas* que chegam ao armazém, provenientes da *linha de produção*. As caixas podem ser percebidas pelo robô ao aparecerem em uma esteira rolante que conecta o armazém à linha de produção. As caixas são coloridas e precisam ser armazenadas em *prateleiras*. Entretanto, em uma única prateleira, somente caixas da mesma cor podem ser colocadas. Além disso, se uma prateleira já foi usada para guardar caixas de uma certa cor, somente caixas desta cor podem ser colocadas nela, mesmo depois da prateleira ser esvaziada. Isto é relevante, por exemplo, na armazenagem de produtos químicos ou em outras aplicações nas quais segurança é um fator crítico. A Figura 5.1 apresenta os elementos que compõem o cenário, salientando os lugares (pontos relevantes) do ambiente (numerados de 1 a 11; o décimo segundo ponto relevante é o balcão de atendimento, ilustrado na Figura 5.2)

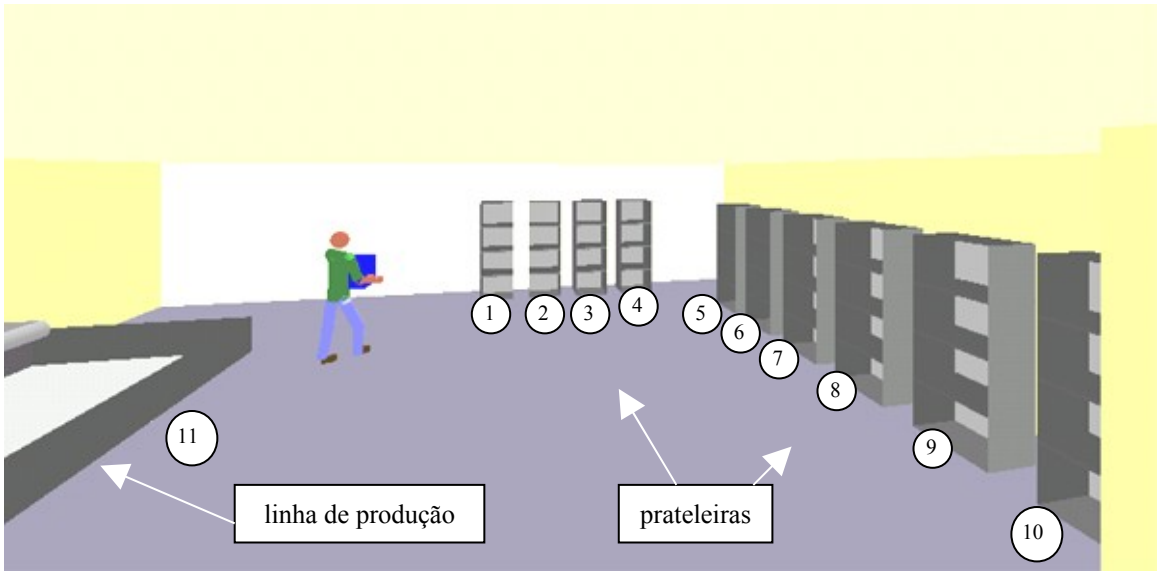


Figura 5.1 – O robô no armazém

Além do processo de armazenamento mencionado acima, o robô também precisa entregar caixas. Membros do quadro de funcionários, os quais podem ser representados por outros agentes ou por avatares, podem entrar no armazém e requerer uma caixa de uma determinada cor (Figura 5.2). O robô sempre dá prioridade a atender essas requisições em detrimento de guardar novas caixas chegando da linha de produção (ou seja, quando ele termina uma tarefa, se existem novas caixas e uma requisição foi percebida, então o robô atenderá primeiro a requisição). Uma caixa de uma cor particular é requerida pressionando um de quatro botões coloridos no balcão que fica na entrada do armazém. Somente um funcionário pode entrar no armazém por vez, e apenas um dos botões coloridos pode ser pressionado por ele. O botão é automaticamente liberado assim que o robô entrega a caixa ao funcionário.

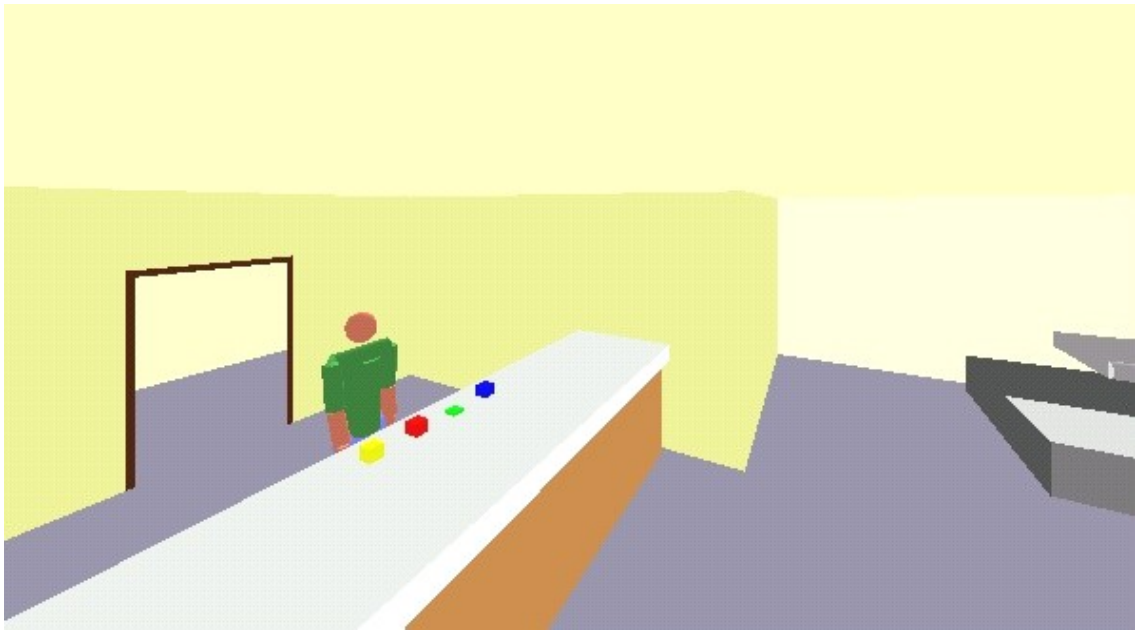


Figura 5.2 – Requisição de uma caixa verde (botão pressionado) por um funcionário

5.2.1 Percepções e Ações

O sistema perceptivo necessita transmitir os eventos do ambiente (gerados aleatoriamente ou através da ação dos atores) ao interpretador AgentSpeak(L) seguindo a sintaxe desta linguagem. Para isso, traduz os eventos em expressões com o formato de predicados e as envia. Este mecanismo precisa oferecer um resultado similar ao de um sensor visual, ou seja, todas as informações que se tornam ou se mantêm visíveis num dado momento devem fazer parte do conjunto de crenças transmitido. Isto é possível, pois a arquitetura oferece um controle sobre todas as percepções relevantes possíveis em uma aplicação. A seguir, são listadas, de forma resumida, as trinta e oito expressões (seguindo a sintaxe de AgentSpeak(L)) que representam todas as percepções do agente inserido neste estudo de caso (as quais foram determinadas a partir da descrição da aplicação):

- "box(*colour*)", onde *colour* pode ser *blue*, *green*, *red* e *yellow* – uma percepção deste tipo indica a presença de uma caixa da cor especificada proveniente da linha de produção (e que necessita ser armazenada);

- "request(colour)", onde colour pode repetir os valores acima – este tipo de percepção determina a necessidade de entregar no balcão uma caixa da cor especificada;

- "location(number)", onde number pode variar de 1 a 10, sendo que cada número representa uma prateleira – esta percepção determina que o agente está localizado em frente a uma prateleira específica (obs.: para este código AgentSpeak(L), não há necessidade de manter controle da localização do agente quando ele está no balcão ou na linha de produção);

- "empty(number)", onde number varia conforme descrito acima – serve para informar que a prateleira especificada está vazia (serve somente para as prateleiras que já foram usadas durante a aplicação; aquelas que nunca receberam nenhuma caixa não precisam deste controle);

- "full(number)", onde number varia conforme descrito acima – informa que a prateleira especificada encontra-se cheia.

O resultado de cada ciclo de raciocínio do interpretador AgentSpeak(L) é retornado ao servidor como uma expressão, a qual determina o comportamento que deve ser adotado pelo agente. A lista abaixo mostra, de forma resumida, as trinta e três expressões que representam todos os resultados possíveis neste estudo de caso (também determinados a partir da descrição da aplicação):

"go_to(counter)\n" – caminhar até o balcão;

"go_to(production)\n" – caminhar até a linha de produção;

"go_to(*number*)\n", onde *number* pode variar de 1 a 10 – caminhar até uma prateleira específica;

"go_to(*colour*)\n" – caminhar até a primeira prateleira usada para a cor especificada que não estiver cheia;

"go_to(empty_shelf)\n" – caminhar até a primeira prateleira livre (que ainda não foi utilizada para armazenar nenhuma caixa);

"pick_box(*colour*)\n" – pegar caixa da cor especificada na linha de produção;

"stack(*colour*)\n" – armazenar caixa da cor especificada na prateleira;

"unstack(*colour*)\n" – pegar caixa da cor especificada na prateleira;

"hand_in(*colour*)\n" – entregar caixa da cor especificada no balcão;

5.2.2 Raciocínio

O código AgentSpeak(L) especificando o raciocínio (prático) do robô é mostrado abaixo (Figura 5.3). Observe que o agente não possui crenças iniciais, somente um conjunto de planos, os quais são explicados abaixo (os planos estão rotulados no código para que seja possível fazer referência a planos individuais no texto).

```
+box(Colour) :                          [p1]
 not(busy) & not(request(AnyColour)) <-
   +busy;
   !handle_box(Colour).

+!handle_box(Colour) :                  [p2]
 true <-
   go_to(production);
   pick_box(Colour);
   !find_shelf(Colour);
   !store(Colour).

+!find_shelf(Colour) :                  [p3]
 last(Colour,Shelf) <-
   go_to(Shelf);
   !check_full(Colour,Shelf).

+!find_shelf(Colour) :                  [p4]
 not(last(Colour,AnyShelf))<-
   go_to(empty_shelf);
   ?location(NewShelf);
   +last(Colour,NewShelf).

+!check_full(Colour,Shelf) :            [p5]
 full(Shelf) <-
   go_to(empty_shelf);
   -last(Colour,Shelf);
   ?location(NewShelf);
   +last(Colour,NewShelf).

+!check_full(Colour,Shelf) :            [p6]
 not(full(Shelf)) <-
   true.

+!store(Colour) :                       [p7]
 true <-
   stack(Colour);
   -busy;
   !check_request.

+request(Colour) :                      [p8]
 not(busy) <-
                                  +busy;
   !handle_request(Colour).

+!check_request :                       [p9]
 request(Colour) <-
   +busy;
   !handle_request(Colour).

+!check_request :                       [p10]
 not(request(AnyColour)) <-
   !check_boxes.

+!handle_request(Colour) :              [p11]
 true <-
   ?last(Colour,Shelf);
   go_to(Shelf);
   !check_empty(Colour,Shelf);
   !deliver(Colour).

+!check_empty(Colour,Shelf) :           [p12]
 empty(Shelf) <-
   go_to(Colour);
   -last(Colour,Shelf);
   ?location(NewShelf);
   +last(Colour,NewShelf).

+!check_empty(Colour,Shelf) :           [p13]
        not(empty(Shelf)) <-
        true.

+!deliver(Colour) :                     [p14]
 true <-
   unstack(Colour);
   go_to(counter);
   hand_in(Colour);
   -busy;
   !check_request.

+!check_boxes :                         [p15]
 box(Colour) <-
   +busy;
   !handle_box(Colour).
```

Figura 5.3 – Código AgentSpeak(L) que especifica os planos do agente

O plano **p1** pode ser disparado sempre que uma nova caixa for percebida chegando da linha de produção. Na crença adicionada *box(Colour)* (por revisão de crenças e por percepção do ambiente), a variável *Colour* será instanciada com uma única referência àquela caixa particular, mas tal referência também indica qual é a cor da caixa (a informação sobre cor é tudo o que interessa quando o robô está guardando a caixa). Este plano é aplicável somente se o robô já não está ocupado fazendo alguma outra coisa, e também se não há requisições de funcionários (lembrando que é dada

prioridade a estas). Se uma instância do plano passa a fazer parte do conjunto de intenções, o robô precisa "lembrar a ele mesmo" que agora está ocupado, lidando com o armazenamento de uma caixa (adicionando uma crença *busy*), e então ele terá o objetivo de lidar com essa caixa particular.

Neste ponto, é importante fazer algumas observações a respeito do mecanismo de percepção do robô. Uma caixa é percebida pelos sensores do robô até ser armazenada em segurança em uma prateleira (de caixas da mesma cor). Quando uma crença sobre uma caixa percebida é adicionada, e o robô está ocupado fazendo alguma outra coisa, nenhum plano para tratar a caixa passa a fazer parte do conjunto de intenções imediatamente. Entretanto, assim que o robô terminar qualquer uma de suas tarefas, ele terá o objetivo de checar se há qualquer requisição pendente e depois se há caixas novas pendentes. Aquelas crenças sobre novas caixas, que foram ignoradas no momento em que foram adicionadas, ainda estarão na base de crenças do agente, uma vez que as caixas ainda são perceptíveis. Dessa forma, o robô compensará os eventos previamente ignorados. Ainda sobre percepção, é importante dizer que o agente pode perceber quando uma prateleira está vazia, quando ela está cheia, e qual é a cor das caixas já armazenadas em uma prateleira (ou previamente armazenadas se a prateleira foi esvaziada, mas já usada para uma cor). O robô recebe, continuamente, informação simbólica a respeito de sua localização no armazém. Finalmente, o robô percebe qual é a cor de um botão que foi pressionado requerendo uma caixa; a requisição é removida da lista de percepções que são passadas ao robô assim que uma caixa da cor apropriada é entregue no balcão.

A seguir, o plano **p2** é usado para tratar do aparecimento (no armazém) de uma nova caixa de uma certa cor. O robô executa uma ação básica *go_to(production)* — isto assume que o robô é capaz de mover-se em seu ambiente sem qualquer interferência de seu raciocínio de alto nível. O ciclo de raciocínio seguinte acontece somente quando o robô atinge o local do armazém no qual chegam caixas provenientes da linha de produção. Uma vez lá, o robô executa *pick_box*, outra ação entre aquelas que ele é capaz de executar "fisicamente". Depois disso, ele passa a ter o objetivo de encontrar uma prateleira apropriada para guardar a caixa e, finalmente, de armazenar esta caixa na prateleira escolhida (o primeiro objetivo já posiciona o robô no local certo).

Para tratar o objetivo de encontrar uma prateleira para uma determinada cor, será selecionado o plano **p3** ou **p4**. O plano **p3** é usado quando o agente "lembra" da última prateleira na qual ele guardou uma caixa com a mesma cor da que ele deve armazenar agora. Neste caso, ele vai até essa prateleira e, uma vez lá, precisa checar se ela já está cheia. Isto não é necessário caso **p4** tenha sido selecionado (quando nenhuma prateleira é conhecida para aquela cor). Neste caso, o robô vai até uma prateleira vazia (uma prateleira não foi usada para nenhuma cor), e a "anotação" interna anterior que o agente tinha sobre a última prateleira usada para aquela cor precisa ser atualizada. Observe que *go_to(empty_shelf)* é, de novo, uma habilidade física do robô; após executá-la, ele estará localizado na próxima prateleira vazia (e "limpa"). O agente pode checar suas crenças correntes (usando um objetivo de teste como em *? location(NewShelf)*) para obter a informação sobre sua localização presente (isto terá sido atualizado por percepção do ambiente).

A checagem se uma determinada prateleira já está cheia (por conseguinte, indisponível para armazenar uma nova caixa) é realizada pelos planos **p5** e **p6**. Caso a prateleira esteja realmente cheia (informação recebida por percepção do ambiente), **p5** é

usado; este plano instrui o robô para ir até a próxima prateleira vazia (usando uma ação básica para isso), atualizando então a informação sobre a última prateleira usada para uma certa cor. A ação para encontrar uma prateleira vazia leva o robô a uma prateleira limpa vazia ou a uma prateleira vazia que foi previamente usada para caixas da mesma cor da que ele está presentemente segurando. O plano **p6** simplesmente estabelece que o objetivo de certificar-se de que a prateleira não está cheia já está atingido se o robô está localizado em uma prateleira que não é percebida como cheia.

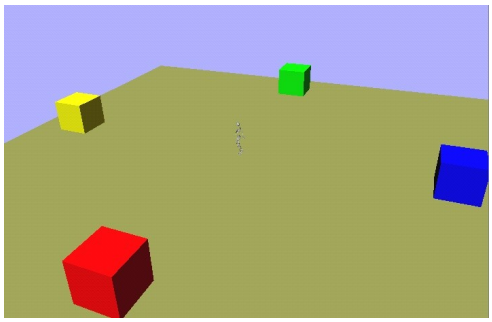
O plano **p7** é usado para efetivamente armazenar uma caixa em uma prateleira. O agente, neste caso, executa a ação básica de empilhar uma caixa no espaço livre mais baixo da prateleira (novamente uma ação física) e remove sua anotação interna de que estava ocupado (removendo a crença busy); por fim, ele precisa ter o objetivo de checar se há alguma requisição pendente, a qual pode ter sido adicionada (e ignorada) enquanto ele estava ocupado.

O plano **p8** é disparado quando uma nova requisição de um funcionário é emitida (um dos botões no balcão frontal é pressionado). Novamente, este plano somente torna-se parte do conjunto de intenções se o agente não encontra-se ocupado, da mesma forma que acontece com a adição de crenças sobre novas caixas (plano **p1**). Os planos **p9** e **p10** checam se existe alguma requisição pendente ou não, passando a ter, respectivamente, o objetivo de tratá-la (através do plano **p11**) ou, caso contrário, de checar as caixas. Tratar uma requisição, como especificado no plano **p11**, significa mover-se até a prateleira na qual foi guardada pela última vez uma caixa de uma cor particular, certificar-se de que essa prateleira não está vazia (movendo-se até uma prateleira apropriada nesse caso), e então entregar a caixa. Os planos **p12**, **p13**, e **p14** são suficientemente análogos a **p5**, **p6**, e **p7** para serem dispensados de maior explicação. É importante observar que outra ação que o robô deve ser capaz de executar é mover-se para uma prateleira previamente preenchida com caixas de uma certa cor (dada como parâmetro). Finalmente, o plano **p15** certifica-se de que novas caixas, que podem ter sido ignoradas quando apareceram, são adequadamente tratadas. Este plano é referenciado pelo plano **p10**.

Com este programa AgentSpeak(L), o agente nunca terá múltiplos focos de atenção (ou seja, ele termina uma tarefa antes de preocupar-se com a seguinte). O cenário foi definido desta forma para não haver preocupação com uma função de seleção de intenção. A idéia era ter um cenário simples que fosse adequado para ilustrar de forma breve a abordagem proposta. Além disso, planos que são usados para lidar com falhas na execução de outros planos (aqueles com um evento disparador *-!p*, usados quando um plano para *+!p* falha) não foram considerados acima. Eles seriam necessários para tratar situações tais como todas as prateleiras estarem cheias ou impróprias para a cor de uma caixa nova, ou o robô tentar entregar uma caixa de uma cor quando não há nenhuma caixa dessa cor previamente armazenada.

5.3. Senha Secreta

Neste segundo exemplo, dois agentes são inseridos em um ambiente com quatro cubos coloridos: um azul, um vermelho, um verde e outro amarelo (Figura 5.4). Uma seqüência de três cores (dentre as mesmas utilizadas nos cubos), não repetidas e com posições determinadas, é gerada aleatoriamente pelo sistema, e o objetivo dos agentes é descobrir, no menor número de tentativas possível, qual é a seqüência correta



(a qual será referida como *senha*). Para isso, devem escolher três dos quatro cubos e visitá-los em determinada ordem (uma visita a três cubos em seqüência será referida como uma *rodada*). Esta ordem define a seqüência de cores estabelecida por cada agente e é comparada pelo sistema com a senha gerada. O sistema informa então aos agentes quantas cores estão corretas e, destas, quantas estão na posição correta (sem identificar, contudo, quais são as corretas). A partir desta informação, e visando descobrir a senha, o mecanismo cognitivo de cada agente decide qual deve ser a nova seqüência de cubos coloridos visitados.

Figura 5.4 – Ambiente virtual com cubos coloridos

O sistema controla o resultado de uma rodada de cada agente através de duas variáveis: *black*, que indica a quantidade de cores certas e na posição correta; e *white*, que especifica o número de cores certas e com posição errada. Nesta aplicação, na qual existem quatro cores possíveis, e três cores formam a seqüência, sem haver repetição de cores, são possíveis seis resultados distintos:

- 3 pretos (*black*) e 0 brancos (*white*) – este resultado indica que a seqüência foi descoberta, encerrando a aplicação;
- 2 brancos e 1 preto;
- 3 brancos;
- 2 pretos;
- 2 brancos;
- 1 preto e 1 branco.

5.3.1 Percepções e Ações

Como foi mencionado anteriormente, a maioria das percepções neste estudo de caso são tratadas pelo próprio código AgentSpeak(L). Ainda assim, algumas percepções precisam ser transmitidas ao interpretador. As expressões que representam essas percepções estão listadas a seguir:

"arrived(*Cube*)", onde *Cube* representa uma cor, podendo variar entre *blue*, *green*, *red* e *yellow* – indica que o agente chegou ao cubo da cor especificada;

"new_result(*Black*, *White*)", onde *Black* e *White* são números que podem variar entre 0 e 3 – indica o número de acertos do agente em uma rodada, de acordo com a definição apresentada acima.

Abaixo, são listadas as expressões que podem ser retornadas pelo interpretador, representando as ações que os agentes devem adotar:

"take_result()" – indica que o agente terminou uma rodada de visitas aos cubos e força o sistema a retornar o resultado;

"visit(Cube)" – caminhar até o cubo com a cor especificada.

5.3.2 Raciocínio

O código AgentSpeak(L) contendo os planos e as crenças iniciais dos agentes é apresentado abaixo (Figura 5.5). Os dois agentes envolvidos utilizam o mesmo conjunto de planos, mas possuem crenças iniciais diferentes a respeito da ordem inicial de visitação dos cubos. As crenças e planos estão rotulados e serão explicados a seguir.

```
next_number(1,2).          [c1]
next_number(2,3).          [c2]
next_number(3,4).          [c3]
next_number(4,5).          [c4]
next_number(5,6).          [c5]
next_number(6,7).          [c6]

visit_place(1).            [c7]

result_number(1).          [c8]

next_cube(yellow,red).     [c9]
next_cube(red,blue).       [c10]
next_cube(blue,green).     [c11]
next_cube(green,yellow).   [c12]
next_cube(center,blue).    [c13]

location(center).          [c14]

+start:                    [p1]
      true<-
      !manage.

+!manage:                  [p2]
      visit_place(4)<-
      -visit_place(4);
      +visit_place(1);
      take_result.

+!manage:                  [p3]
```

```
      true<-
      ?location(Loc);
      ?next_cube(Loc,Cub);
      visit(Cub).

+arrived(Cube):            [p4]
      true<-
      ?location(Loc);
      -location(Loc);
      +location(Cube);
      ?visit_place(Pla);
      +colour(Pla,Cube);
      ?next_number(Pla,NextPla);
      -visit_place(Pla);
      +visit_place(NextPla);
      !manage.

+new_result(Black,White):  [p5]
      true<-
      ?result_number(Num);
      ?next_number(Num,NextNum);
      +result(Num,Black,White);
      -result_number(Num);
      +result_number(NextNum);
      !deliberate(Num).

+!deliberate(ResNum):      [p6]
      result(ResNum,3,0)<-
      end.

+!deliberate(ResNum):      [p7]
      result(ResNum,0,3)<-
      ?colour(1,Col1);
      ?colour(2,Col2);
      ?colour(3,Col3);
      -next_cube(center,Col1);
      +next_cube(center,Col2);
      ?next_cube(Col3,Col4);
      -next_cube(Col3,Col4);
      +next_cube(Col3,Col1);
      !again.

+!deliberate(ResNum):      [p8]
      result(ResNum,1,2)<-
      ?colour(1,Col1);
      ?colour(2,Col2);
      ?colour(3,Col3);
      -next_cube(Col1,Col2);
      +next_cube(Col1,Col3);
      ?next_cube(Col3,Col4);
      -next_cube(Col3,Col4);
      +next_cube(Col3,Col2);
      !again.

+!deliberate(ResNum):      [p9]
      result(ResNum,2,0) &
result_number(2)<-
      ?colour(1,Col1);
      -next_cube(center,Col1);
      ?next_cube(Col4,Col1);
      +next_cube(center,Col4);
      -next_cube(Col4,Col1);
      ?colour(2,Col2);
      +next_cube(Col4,Col2);
      ?colour(3,Col3);
      -next_cube(Col3,Col4);
      +next_cube(Col3,Col1);
      -next_cube(Col1,Col2);
      +next_cube(Col1,Col4);
      !again.

+!deliberate(ResNum):      [p10]
      result(ResNum,2,0)<-
      ?colour(2,Col2);
      ?colour(1,Col1);
      ?colour(3,Col3);
      ?next_cube(Col3,Col4);
      -next_cube(Col1,Col2);
      +next_cube(Col1,Col4);
      -next_cube(Col2,Col3);
      +next_cube(Col4,Col3);
      -next_cube(Col3,Col4);
      +next_cube(Col3,Col2);
      -next_cube(Col4,Col1);
      +next_cube(Col2,Col1);
      !again.

+!deliberate(ResNum):      [p11]
      result(ResNum,0,2) &
result_number(2)<-
      ?colour(1,Col1);
      -next_cube(center,Col1);
      ?next_cube(Col4,Col1);
      +next_cube(center,Col4);
      -next_cube(Col4,Col1);
      ?colour(2,Col2);
      ?colour(3,Col3);
      +next_cube(Col4,Col3);
      -next_cube(Col3,Col4);
      +next_cube(Col3,Col2);
      -next_cube(Col2,Col3);
      +next_cube(Col2,Col1);
      -next_cube(Col1,Col2);
      +next_cube(Col1,Col4);
      !again.

+!deliberate(ResNum):      [p12]
      result(ResNum,0,2)<-
      ?colour(1,Col1);
      ?colour(3,Col3);
      -next_cube(center,Col1);
      +next_cube(center,Col3);
      !again.

+!deliberate(ResNum):      [p13]
      result(ResNum,1,1) &
result_number(2)<-
      ?colour(1,Col1);
      -next_cube(center,Col1);
      ?next_cube(Col4,Col1);
      +next_cube(center,Col4);
      -next_cube(Col4,Col1);
```

```
?colour(2,Col2);
+next_cube(Col4,Col2);          +!again:                    [p14]
?colour(3,Col3);                      true<-
-next_cube(Col3,Col4);                ?location(Loc);
+next_cube(Col3,Col1);                -location(Loc);
-next_cube(Col1,Col2);                +location(center);      !
+next_cube(Col1,Col4);          manage.
!again.
```

Figura 5.5 – Código AgentSpeak(L)

As crenças **c1** a **c6** são usadas para implementar contadores, os quais são usados para controlar a rodada corrente e a quantidade de cubos visitados na rodada (nesta aplicação, de 1 a 7 é suficiente). A crença **c7** serve para controlar, a cada rodada, se os três cubos já foram visitados pelo agente – quando isto ocorre, o interpretador deve solicitar o resultado ao sistema (através da ação *take_result*). A crença **c8** representa o número da rodada corrente. As crenças **c9** a **c13**, por sua vez, determinam a seqüência corrente de visitação dos cubos, considerando que *center* representa sempre a posição inicial. A cada nova rodada, esta ordem é alterada de acordo com o resultado obtido. Por fim, a crença **c14** é usada pelo agente para controlar sua localização corrente no ambiente.

O plano **p1** inicia o programa. Dependendo da quantidade de cubos já visitados na rodada (representada pela crença *visit_place*), será ativado o plano **p2** ou **p3**. O primeiro deles entra em ação somente quando a rodada está completa, ou seja, três cubos já foram visitados. Neste caso, a crença *visit_place* é reiniciada (para preparar a próxima rodada) e o resultado é solicitado ao sistema através da ação *take_result*. Já o plano **p3** verifica qual é o cubo seguinte a ser visitado e envia ao sistema a ação *visit*, a qual provocará a execução de um comportamento por parte do agente.

O plano **p4** é executado quando o agente chega em um cubo (o sistema informa a adição da crença *arrived* ao interpretador). Ele atualiza algumas crenças e retorna ao objetivo *manage* (**p2** ou **p3**). O plano **p5** é ativado pela adição da crença *new_result*, a qual é enviada ao interpretador pelo sistema contendo o resultado solicitado. Este plano atualiza algumas crenças e ativa o objetivo *deliberate*. Os planos para tratar este objetivo compreendem desde **p6** até **p13,** sendo escolhido aquele cujo contexto atender ao resultado obtido. Nestes planos, a partir da análise do resultado obtido, é gerada uma nova ordem de visitação aos cubos, e o objetivo *again* é ativado (caso ainda não tenha sido encontrada a senha). No plano **p14**, que trata este objetivo, a localização do agente é atualizada para *center* (indicando que uma nova rodada inicia, embora graficamente ele não esteja mais no centro do ambiente), a fim de que o agente percorra uma nova seqüência através do objetivo *manage*.

6. Conclusões e Trabalhos Futuros

O principal objetivo deste trabalho foi desenvolver um modelo para produzir atores sintéticos articulados com raciocínio cognitivo usando lógica BDI. Para isso, foi proposta a combinação do modelo articulado de Maciel et al. [MAC 2002] com o interpretador para a linguagem AgentSpeak(L) apresentado em Bordini et al. [BOR 2002a]. O primeiro desafio deste projeto foi estabelecer a comunicação entre o modelo de corpo articulado e o interpretador de forma eficiente. Uma vez atingido este primeiro estágio, o foco passou a ser oferecer uma estrutura capaz de adaptar-se ao alto nível de abstração de AgentSpeak(L), permitindo assim um uso eficiente da linguagem. Programas AgentSpeak(L) devem concentrar-se somente nos planos dos agentes, recebendo prontas todas as informações a respeito de fatores externos. Esse segundo objetivo levou à criação de um sistema perceptivo que controla quais dados externos precisam ser transmitidos ao interpretador.

Para implementar este modelo, foi criada uma arquitetura modular que oferece bastante flexibilidade. O ambiente virtual é estabelecido em um processo servidor e fica aberto a conexões de clientes. Para acrescentar novos habitantes, basta descrever os seus componentes. No caso de avatares, tudo o que é preciso é definir a representação do corpo e os movimentos associados a ele em um arquivo XML, além de informar em um arquivo texto as teclas pelas quais o usuário o controlará. Para os agentes, além da descrição do corpo e movimentos em formato XML, é necessário listar todas as suas percepções e ações possíveis em um arquivo texto, além, é claro, de especificar seu raciocínio em um programa AgentSpeak(L). Estas características permitem à arquitetura oferecer uma série de possibilidades adicionais:

- o mesmo usuário pode controlar vários avatares, bastando para isto conhecer as teclas que os controlam;

- são aceitos também agentes sem representação (sem corpo), os quais podem ser muito úteis para comandar os eventos do ambiente de forma cognitiva, tornando-o "inteligente";

- se houver necessidade, é possível transformar, em tempo de execução, agentes em avatares e vice-versa, bastando alterar os arquivos contendo as descrições dos componentes. Por exemplo, para transformar um avatar em agente, é necessário fornecer o programa AgentSpeak(L) do agente e trocar o arquivo texto com a lista de teclas de controle por outro com as percepções e ações possíveis;

- em algumas aplicações, pode ser conveniente dispor de personagens híbridos, os quais são dirigidos em determinadas situações por seu mecanismo cognitivo e em outras pelo usuário. Um ator deste tipo não é nada mais do que um agente, cujo código AgentSpeak(L) prevê apenas um subconjunto dos eventos possíveis, acrescido de um arquivo texto contendo as teclas de controle para as situações não tratadas pelo mecanismo cognitivo.

Não houve preocupação em assegurar a pertinência do modelo BDI, pois este é um modelo bem estabelecido na área de sistemas multiagentes. Os estudos de caso apresentados foram criados visando apenas ilustrar a arquitetura, sem a intenção de comprovar o poder e a necessidade de usar BDI. Nem todas as aplicações precisam deste modelo, pois podem ser

implementadas de formas mais simples. O modelo BDI é apropriado para ambientes complexos e dinâmicos, pois seu poder está na capacidade de abstrair tais ambientes, nos quais são usadas estratégias complexas. A funcionalidade de humanos virtuais, por exemplo, principalmente em jogos, exige o uso de raciocínio complexo.

A arquitetura implementada oferece suporte à criação de agentes articulados com as capacidades cognitivas definidas no modelo BDI. O fato de o modelo básico não ter sido alterado ao longo dos anos, não fornecendo, por exemplo, capacidade de aprendizado de forma explícita [GEO 98], pode ser facilmente contornado através da adição de novos mecanismos à arquitetura. Um dos objetivos futuros, visando oferecer maior credibilidade aos agentes, é acoplar à arquitetura um modelo de emoções como o de Marsella e Gratch [MAR 2002], apresentado na Seção 2.5.4.

Daqui para frente, para que este trabalho apresente resultados mais práticos e efetivos, a prioridade será a pesquisa e implementação de aplicações capazes de justificar a utilidade do modelo BDI para melhorar a credibilidade de uma animação comportamental. Especificamente com relação a jogos, espera-se dar um passo a frente no sentido de atingir a possibilidade de alterar o curso de execução a partir da ação dos personagens. Com esta finalidade, a idéia é implementar agentes que mantenham-se utilizando estratégias complexas de ação, influindo no ambiente virtual e nos outros habitantes, mesmo quando não estão no foco principal do jogo nem interagindo com o usuário. Como conseqüência natural do desenvolvimento de novas aplicações, será possível avaliar qual é a quantidade máxima de agentes com raciocínio mais complexo que a arquitetura suporta em tempo real.

Até agora, este trabalho somente testou o uso da lógica BDI no nível mais alto da animação comportamental: a escolha de ações. O modelo BDI poderá ser testado também num nível intermediário, com a finalidade de coordenar os canais de expressão de movimento com o estado interno dos agentes, como é feito no sistema EMOTE [CHI 2000]. Por exemplo, um movimento de caminhar pode variar de acordo com o estado de espírito do personagem, transmitindo a impressão de cansaço, pressa, alegria, etc. Um dos maiores desafios será assegurar um tempo de resposta adequado para aplicações em tempo real.

Para finalizar, estão sendo desenvolvidas ferramentas para simplificar a especificação dos corpos articulados e a geração de movimentos em nível de tarefa. Até este momento, todos os parâmetros de definição utilizados nos arquivos XML têm sido obtidos através de tentativa, partindo de valores numéricos. As ferramentas permitirão que as posições das juntas e objetos gráficos, além dos movimentos sobre as articulações, sejam determinados de forma interativa, facilitando a criação de corpos com maior qualidade gráfica e movimentos mais realistas.

Anexo A – Descrição de um Corpo Articulado em XML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE vpat SYSTEM "vpat.dtd">
<vpat>
<body description="agente0" file_name="agente0.xml">
      <joint description="root" parent="null" type="plane">
            <dof description="flexionB">
                  <position x="0.0" y="0.0" z="0.00"/>
                  <axis x="1.00" y="0.00" z="0.00"/>
                  <range min="0.0" max="0.0"/>
                  <comfort_range min="0.0" max="0.0" rest="0.0"/>
            </dof>
            <dof description="adductionB">
                  <position x="0.0" y="0.0" z="0.00"/>
                  <axis x="0.00" y="1.00" z="0.00"/>
                  <range min="-3.14159" max="3.14159"/>
                  <comfort_range min="-3.14159" max="3.14159" rest="0.5"/>
            </dof>
            <dof description="twistB">
                  <position x="0.0" y="0.0" z="0.00"/>
                  <axis x="0.00" y="0.00" z="1.00"/>
                  <range min="0.0" max="1.0"/>
                  <comfort_range min="0.0" max="1.0" rest="0.0"/>
            </dof>
            <dof description="xslideB">
                  <position x="0.0" y="0.0" z="0.00"/>
                  <axis x="1.00" y="0.00" z="0.00"/>
                  <range min="0.0" max="0.0"/>
                  <comfort_range min="0.0" max="0.0" rest="0.0"/>
            </dof>
            <dof description="yslideB">
                  <position x="0.0" y="0.0" z="0.00"/>
                  <axis x="0.00" y="1.00" z="0.00"/>
                  <range min="0.0" max="0.0"/>
                  <comfort_range min="0.0" max="0.0" rest="0.0"/>
            </dof>
            <dof description="zslideB">
                  <position x="0.0" y="0.0" z="0.00"/>
                  <axis x="0.00" y="0.00" z="1.00"/>
                  <range min="0.0" max="0.0"/>
                  <comfort_range min="0.0" max="0.0" rest="0.0"/>
            </dof>
            <shape description="bacia" file_name="sacrum.iv">
                  <position x="0.00" y="0.0" z="0.0"/>
                  <iaxis x="1.0" y="0.0" z="0.0"/>
                  <jaxis x="0.0" y="1.0" z="0.0"/>
                  <kaxis x="0.0" y="0.0" z="1.0"/>
            </shape>
            <shape description="iliaco" file_name="pelvis.iv">
                  <position x="0.00" y="0.0" z="0.0"/>
                  <iaxis x="1.0" y="0.0" z="0.0"/>
                  <jaxis x="0.0" y="1.0" z="0.0"/>
                  <kaxis x="0.0" y="0.0" z="1.0"/>
            </shape>
      </joint>
      <joint description="coluna" parent="root" type="uniaxial">
            <dof description="flexionC">
                  <position x="0.0" y="0.0" z="0.00"/>
```

```
            <axis x="1.00" y="0.00" z="0.00"/>
            <range min="0.0" max="0.5"/>
            <comfort_range min="0.0" max="0.5" rest="0.0"/>
      </dof>
      <shape description="esterno" file_name = "sternum.iv">
            <position x="0.00" y = "0.0" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
</joint>
<joint description="cabeca" parent="coluna" type="polyaxial">
      <dof description="flexionR">
            <position x="0.00" y="0.00" z="0.00"/>
            <axis x="1.00" y="0.00" z="0.00"/>
            <range min="0.0" max="1.0"/>
            <comfort_range min="0.0" max="0.0" rest="0.0"/>
      </dof>
      <dof description="adductionR">
            <position x="0.00" y="0.00" z="0.00"/>
            <axis x="0.00" y="1.00" z="0.00"/>
            <range min="0.0" max="1.0"/>
            <comfort_range min="0.0" max="0.0" rest="0.0"/>
      </dof>
      <dof description="twistR">
            <position x="0.00" y="0.00" z="0.00"/>
            <axis x="0.00" y="0.00" z="1.00"/>
            <range min="0.0" max="0.0"/>
            <comfort_range min="0.0" max="0.0" rest="0.0"/>
      </dof>
      <shape description="skull" file_name = "skull.iv">
            <position x="0.00" y = "0.0" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
</joint>
<joint description="torax" parent="coluna" type="uniaxial">
      <dof description="flexionR">
            <position x="0.00" y="0.00" z="0.00"/>
            <axis x="1.00" y="0.00" z="0.00"/>
            <range min="0.0" max="0.0"/>
            <comfort_range min="0.0" max="0.0" rest="0.0"/>
      </dof>
      <shape description="lclav" file_name = "lclavicl.iv">
            <position x="0.00" y = "0.00" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
      <shape description="lescap" file_name = "lscapula.iv">
            <position x="0.00" y = "0.00" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
      <shape description="rclav" file_name = "rclavicl.iv">
            <position x="0.00" y = "0.00" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
```

```
      </shape>
      <shape description="rescap" file_name = "rscapula.iv">
            <position x="0.00" y = "0.00" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
</joint>
<joint description="lshoulder" parent="torax" type="uniaxial">
      <dof description="flexionS">
            <position x="0.00" y="-10.0" z="0.00"/>
            <axis x="1.00" y="0.00" z="0.00"/>
            <range min="-0.5" max="0.8"/>
            <comfort_range min="-0.5" max="0.8" rest="0.5"/>
      </dof>
      <shape description="lhumero" file_name = "lhumerus.iv">
            <position x="0.0" y = "-10.0" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
</joint>
<joint description="lelbow" parent="lshoulder" type="uniaxial">
      <dof description="flexionE">
            <position x="0.00" y="10.00" z="0.00"/>
            <axis x="1.00" y="0.00" z="0.00"/>
            <range min="-0.50" max="0.50"/>
            <comfort_range min="-0.5" max="0.5" rest="0.5"/>
      </dof>
      <shape description="lradio" file_name = "lradius.iv">
            <position x="0.0" y = "0.0" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
</joint>
<joint description="lwrist" parent="lelbow" type="biaxial">
      <dof description="flexionW">
            <position x="0.00" y="10.00" z="10.00"/>
            <axis x="1.00" y="0.00" z="0.00"/>
            <range min="0.0" max="0.0"/>
            <comfort_range min="0.0" max="0.0" rest="0.0"/>
      </dof>
      <dof description="twistW">
            <position x="0.00" y="0.00" z="0.00"/>
            <axis x="0.00" y="0.00" z="1.00"/>
            <range min="-1.0" max="0.7"/>
            <comfort_range min="-1.0" max="0.7" rest="0.4"/>
      </dof>
      <shape description="lmao" file_name = "lhand.iv">
            <position x="0.0" y = "0.0" z = "0.0"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
</joint>
<joint description="rshoulder" parent="torax" type="uniaxial">
      <dof description="flexionS">
            <position x="0.00" y="-10.0" z="0.00"/>
            <axis x="1.00" y="0.00" z="0.00"/>
            <range min="-0.5" max="0.8 "/>
```

```
                <comfort_range min="-0.5" max="0.8" rest="0.5"/>
            </dof>
            <shape description="rhumero" file_name = "rhumerus.iv">
                <position x="0.0" y = "-10.0" z = "0.0"/>
                <iaxis x="1.0" y="0.0" z="0.0"/>
                <jaxis x="0.0" y="1.0" z="0.0"/>
                <kaxis x="0.0" y="0.0" z="1.0"/>
            </shape>
        </joint>
        <joint description="relbow" parent="rshoulder" type="uniaxial">
            <dof description="flexionE">
                <position x="0.00" y="10.00" z="0.00"/>
                <axis x="1.00" y="0.00" z="0.00"/>
                <range min="-0.5" max="0.5"/>
                <comfort_range min="-0.5" max="0.5" rest="0.5"/>
            </dof>
            <shape description="rradio" file_name = "rradius.iv">
                <position x="0.0" y = "0.0" z = "0.0"/>
                <iaxis x="1.0" y="0.0" z="0.0"/>
                <jaxis x="0.0" y="1.0" z="0.0"/>
                <kaxis x="0.0" y="0.0" z="1.0"/>
            </shape>
        </joint>
        <joint description="rwrist" parent="relbow" type="biaxial">
            <dof description="flexionW">
                <position x="0.8" y="0.00" z="0.0"/>
                <axis x="1.00" y="0.00" z="0.00"/>
                <range min="0.0" max="0.0"/>
                <comfort_range min="0.0" max="0.0" rest="0.0"/>
            </dof>
            <dof description="twistW">
                <position x="0.00" y="0.00" z="0.00"/>
                <axis x="0.00" y="0.00" z="1.00"/>
                <range min="0.0" max="1.0"/>
                <comfort_range min="0.0" max="1.0" rest="0.0"/>
            </dof>
            <shape description="rmao" file_name = "rhand.iv">
                <position x="0.0" y="0.0" z="0.0"/>
                <iaxis x="1.0" y="0.0" z="0.0"/>
                <jaxis x="0.0" y="1.0" z="0.0"/>
                <kaxis x="0.0" y="0.0" z="1.0"/>
            </shape>
        </joint>
        <joint description="lhip" parent="root" type="uniaxial">
        <dof description="flexionH">
            <position x="0.0" y="6.5" z="5.0"/>
            <axis x="1.0" y="0.0" z="0.0"/>
            <range min="-1.0" max="1.0"/>
            <comfort_range min="-1.0" max="1.0" rest="0.45"/>
        </dof>
        <shape description="lFemur" file_name="lfemur.iv">
            <position x="-3.15" y="8.2" z="0.7"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
        </shape>
    </joint>
    <joint description="lknee" parent="lhip" type="biaxial">
            <dof description="flexionK">
                <position x="-2.48" y="17.05" z="0.0"/>
                <axis x="1.0" y="0.0" z="0.0"/>
```

```
         <range min="0.0" max="2.2"/>
         <comfort_range min="-0.3" max="0.4" rest="0.0"/>

      </dof>
      <dof description="twistK">
         <position x="0.0" y="0.0" z="0.0"/>
         <axis x="0.0" y="1.0" z="0.0"/>
         <range min="0.0" max="0.28"/>
         <comfort_range min="-0.8" max="1.0" rest="0.0"/>

      </dof>
      <shape description="lTibia" file_name="ltibia.iv">
         <position x="-3.15" y="25.25" z="3.18"/>
         <iaxis x="1.0" y="0.0" z="0.0"/>
         <jaxis x="0.0" y="1.0" z="0.0"/>
         <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
      <shape description="lPatela" file_name="lpatella.iv">
         <position x="-3.15" y="25.25" z="3.18"/>
         <iaxis x="1.0" y="0.0" z="0.0"/>
         <jaxis x="0.0" y="1.0" z="0.0"/>
         <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
      <shape description="lFibula" file_name="lfibula.iv">
         <position x="-3.15" y="25.25" z="3.18"/>
         <iaxis x="1.0" y="0.0" z="0.0"/>
         <jaxis x="0.0" y="1.0" z="0.0"/>
         <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
</joint>
<joint description="lankle" parent="lknee" type="biaxial">
      <dof description="flexionA">
         <position x="-1.0" y="15.25" z="-0.15"/>
         <axis x="1.0" y="0.0" z="0.0"/>
         <range min="0.0" max="1.0"/>
         <comfort_range min="-0.3" max="0.4" rest="0.0"/>
      </dof>
      <dof description="twistA">
         <position x="0.0" y="0.0" z="0.0"/>
         <axis x="0.0" y="0.0" z="1.0"/>
         <range min="-0.5" max="0.2"/>
         <comfort_range min="-0.8" max="1.0" rest="0.0"/>
      </dof>
      <shape description="lFoot" file_name="lfoot.iv">
         <position x="-3.0" y="40.5" z="4.18"/>
         <iaxis x="1.0" y="0.0" z="0.0"/>
         <jaxis x="0.0" y="1.0" z="0.0"/>
         <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
</joint>
<joint description="rhip" parent="root" type="uniaxial">
   <dof description="flexionH">
      <position x="0.0" y="6.5" z="2.0"/>
      <axis x="1.0" y="0.0" z="0.0"/>
      <range min="-1.0" max="1.0"/>
      <comfort_range min="-1.0" max="1.0" rest="0.45"/>
   </dof>
   <shape description="rFemur" file_name="rfemur.iv">
      <position x="-3.15" y="8.2" z="0.7"/>
      <iaxis x="1.0" y="0.0" z="0.0"/>
      <jaxis x="0.0" y="1.0" z="0.0"/>
```

```
         <kaxis x="0.0" y="0.0" z="1.0"/>
      </shape>
   </joint>
      <joint description="rknee" parent="rhip" type="biaxial">
         <dof description="flexionK">
            <position x="-2.48" y="17.05" z="0.0"/>
            <axis x="1.0" y="0.0" z="0.0"/>
            <range min="0.0" max="2.2"/>
            <comfort_range min="-0.3" max="0.4" rest="0.0"/>

         </dof>
         <dof description="twistK">
            <position x="0.0" y="0.0" z="0.0"/>
            <axis x="0.0" y="1.0" z="0.0"/>
            <range min="0.0" max="0.28"/>
            <comfort_range min="-0.8" max="1.0" rest="0.0"/>

         </dof>
         <shape description="rTibia" file_name="rtibia.iv">
            <position x="-3.15" y="25.25" z="3.18"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
         </shape>
         <shape description="rPatela" file_name="rpatella.iv">
            <position x="-3.15" y="25.25" z="3.18"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
         </shape>
         <shape description="rFibula" file_name="rfibula.iv">
            <position x="-3.15" y="25.25" z="3.18"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
         </shape>
   </joint>
   <joint description="rankle" parent="rknee" type="biaxial">
         <dof description="flexionA">
            <position x="-1.0" y="15.25" z="-0.15"/>
            <axis x="1.0" y="0.0" z="0.0"/>
            <range min="0.0" max="1.0"/>
            <comfort_range min="-0.3" max="0.4" rest="0.0"/>
         </dof>
         <dof description="twistA">
            <position x="0.0" y="0.0" z="0.0"/>
            <axis x="0.0" y="0.0" z="1.0"/>
            <range min="-0.5" max="0.2"/>
            <comfort_range min="-0.8" max="1.0" rest="0.0"/>
         </dof>
         <shape description="rFoot" file_name="rfoot.iv">
            <position x="-2.0" y="37.5" z="4.18"/>
            <iaxis x="1.0" y="0.0" z="0.0"/>
            <jaxis x="0.0" y="1.0" z="0.0"/>
            <kaxis x="0.0" y="0.0" z="1.0"/>
         </shape>
   </joint>
</body>
</vpat>
```

Anexo B - Sockets

A hierarquia de classes disponíveis para a implementação de *sockets* está ilustrada na Figura B.1. As classes **CommunicatingSocket** e **TCPServerSocket** são subclasses da classe **Socket** e representam, respectivamente, *sockets* que são capazes de conectar, enviar e receber, e *sockets* que estabelecem servidores TCP e aceitam conexões. As classes **TCPSocket** e **UDPSocket** estendem a classe **CommunicatingSocket** para representar *sockets* TCP e UDP. Desse conjunto de classes, a arquitetura proposta utiliza uma instância de **TCPServerSocket** para implementar o *socket* do servidor e um objeto da classe **TCPSocket** para cada *socket* cliente. Os métodos necessários estão descritos abaixo:

- TCPServerSocket(unsigned short localPort, int queueLen = 5) throw(SocketException)

Este é o método construtor da classe **TCPServerSocket**. Cria um *socket* TCP para uso com um servidor, aceitando conexões na porta especificada pelo parâmetro *localPort*. O parâmetro *queueLen* determina o comprimento máximo da fila de requisições de conexão pendentes, e seu valor *default* é 5. Se houver falha na criação do *socket*, o método gera uma exceção.

- TCPSocket* accept() throw(SocketException)

Este método da classe **TCPServerSocket** bloqueia a execução do processo corrente até que uma nova conexão seja estabelecida no *socket* ou ocorra uma falha. Retorna um novo *socket* de conexão (objeto da classe **TCPSocket**), ou gera uma exceção, em caso de falha na conexão.

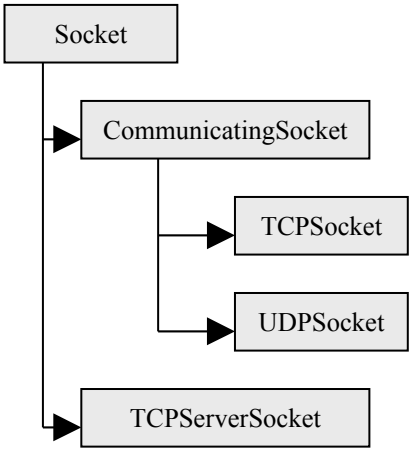


Figura B.1 – Hierarquia de classes para implementação de s*ockets*

- void send(const void* buffer, int bufferLen) throw(SocketException)

Este método, herdado da classe **CommunicatingSocket** por **TCPSocket**, transmite o conteúdo do parâmetro *buffer* ao *socket*, gerando uma exceção em caso de falha. O parâmetro *bufferLen* indica o número de bytes do buffer a serem enviados.

- int recv(void* buffer, int bufferLen) throw(SocketException)

Este método, também herdado de **CommunicationSocket** por **TCPSocket**, lê, a partir do *socket*, um número de bytes indicado por *bufferLen*, gerando uma exceção em caso de falha.

A documentação completa das classes, contendo a descrição de todos os seus métodos, além dos arquivos fonte correspondentes estão disponíveis em [DON 2002].

Bibliografia

[ALL 2000] ALLBECK, J. M.; BADLER, N. I. Towards Behavioral Consistency in Animated Agents. Deformable Avatars, IFIP TC5/WG5.10 *DEFORM'2000 Workshop*, 2000, Genebra, Suíça and *AVATARS'2000 Workshop*, November 30-December 1, Lausanne, Switzerland / edited Nadia Magnenat-Thalmann, Daniel Thalmann. p. 191–205.

[ALV 97] ALVARES, L. O.; SICHMAN, J. S. Introdução aos Sistemas Multiagentes. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA (JAI 97), XVII CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. Brasília, 1997.

[BAD 79] BADLER, N.; SMOLIAR, S. W. Digital Representations of Human Movement. **ACM Computing Surveys**, Vol. 11, Nº 1, março de 1979.

[BAD 99] BADLER, N.; et al. Real time virtual humans. In: INTERNATIONAL CONFERENCE ON DIGITAL MEDIA FUTURES. Bradford, UK, 1999.

[BAD 2000] BADLER, N.; et al. Parameterized Action Representation for Virtual Human Agents. **Embodied Conversational Characters**, Cambridge, MA, MITPress, 2000.

[BAT 92] BATES, J.; LOYALL, A. B.; REILLY, W. S. An Architecture for Action, Emotion and Social Behavior, **Technical Report CMU-CS-92-144**, School of Computer Science, Carnegie Mellon University, 1992.

[BAT 94] BATES, J. The Role of Emotion in Believable Agents. **Communications of the ACM**, 37(7), julho de 1994.

[BEC 98] BÉCHEIRAZ, P.; THALMANN, D. A Behavioral Animation System for Autonomous Actors personified by Emotions. In: FIRST WORKSHOP ON EMBODIED CONVERSATIONAL CHARACTERS (WECC '98). **Proceedings**..., Lake Tahoe, CA, 1998.

[BOR 2002a] BORDINI, R. H.; et al. AgentSpeak(XL): Efficient Intention Selection in BDI Agents via Decision-Theoretic Task Scheduling. In: FIRST INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS (AAMAS 2002, featuring 6th AGENTS, 5th ICMAS and 9th ATAL). **Proceedings**..., Bolonha, Itália, 2002.

[BOR 2002b] BORDINI, R. H.; MOREIRA Á. F. Proving the asymmetry thesis principles for a BDI agent-oriented programming language. In: THIRD INTERNATIONAL WORKSHOP ON COMPUTATIONAL LOGIC IN MULTI-AGENT SYSTEMS (CLIMA-02). **Proceedings**..., held with FLoC'02, Copenhagen, Denmark. Electronic Notes in Theoretical Computer Science, 70(5). 2002. Disponível em <http://www.elsevier.nl/locate/entcs/volume70.html>

[BRA 87] BRATMAN, M. E. **Intentions, Plans, and Practical Reason**. Harvard University Press: Cambridge, MA, 1987.

[BRO 86] BROOKS, R. A. A robust layered control system for a mobile robot. **IEEE Journal of Robotics and Automation**, RA-2, 1986. p. 14-23.

[CAR 97] CARLSON, D. A.; HODGINS, J. K. Simulation levels of detail for real-time animation. In: GRAPHICS INTERFACE '97. **Proceedings**..., 1997, p. 1-8.

[CAS 98] CASTELFRANCHI, C. Through the Minds of the Agents. **Journal of Artificial Societies and Social Simulation**, vol. 1, no. 1, notas 4 e 5, 1998. Disponível em <http://www.soc.surrey.ac.uk/JASSS/1/1/5.html>.

[CHA 89] CHADWICK, J.; HAUMANN, D.; PARENT, R. Layered Construction for Deformable Animated Characters. **Computer Graphics**, Vol. 23, Nº 4, p. 243-252, 1989 (Proc. SIGGRAPH '89).

[CHI 2000] CHI, D.; et al. The emote model for effort and shape. In: SIGGRAPH 2000. **Proceedings**..., 2000. p. 173-182.

[DON 2002] DONAHOO, M.; CALVERT, K. **TCP/IP Sockets in C Practical Guide for Programmers**. Morgan Kaufmann, 2002. 130p.

[FET 82] FETTER, W. A progression of human Figures Simulated by computer graphics. **IEEE Comput. Graph. Appl.**, novembro de 1982.

[FOS 91] FOSTER, S.; WENZEL, E. Virtual Acoustic Environments: The Convolvotron. In: SIGGRAPH'91 Virtual reality and hypermedia show. 1991.

[FRE 2003] FREITAS, C. M. D. S.; et al.. Framework para Construção de Pacientes Virtuais: Uma Aplicação em Laparoscopia Virtual. In: SVR 2003 - SBC SYMPOSIUM ON VIRTUAL REALITY, 2003, Ribeirão Preto. **Proceedings** of SVR 2003 - VI Symposium on Virtual Reality. Porto Alegre: SBC - Brazilian Computer Society, 2003. v. 6, p. 283-294.

[FUN 99] FUNGE, J.; TU, X.; TERZOPOULOS, D. Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. In: SIGGRAPH 99. **Proceedings**..., Los Angeles, CA, 1999.

[GEO 98] GEORGEFF, M.; et al. The belief-desire-intention model of agency. In: FIFTH INTERNATIONAL WORKSHOP ON AGENT THEORIES, ARCHITECTURES AND LANGUAGES. **Proceedings...** Springer Lecture Notes in Artificial Intelligence. Springer, volume 155, 1998.

[GIA 2000] GIANG, T.; et al. Real-Time Character Animation Techniques. Image Synthesis Group Trinity College Dublin**, Technical Report TCD-CS-2000-06**, fevereiro de 2000.

[GRA 97] GRAND, S.; CLIFF, D.; MALHOTRA, A. Creatures: Artificial Life

Autonomous Software Agents for Home Entertainment. In: AGENTS '97. **Proceedings**..., Marina Del Ray, California, 1997.

[GRJ 2000] GRATCH, J. Émile: marshalling passions in thraining and education. In: FOURTH INTERNATIONAL CONFERENCED ON INTELLIGENT AGENTS. **Proceedings**..., Barcelona, Espanha, 2000.

[GRJ 2001] GRATCH, J.; MARSELLA, S. Tears and Fears: Modeling Emotions and Emotional Behaviors in Synthetic Agents. In: 5TH INT'L CONF. AUTONOMOUS AGENTS. **Proceedings**..., ACM Press, Nova Iorque, 2001. p. 278–285.

[GRJ 2002] GRATCH, J.; et al. Creating Interactive Virtual Humans: Some Assembly Required. **IEEE Intelligent Systems**, vol. Julho/Agosto, 2002. p. 54-61.

[HAU 88] HAUMANN, D. R.; PARENT, R. E. The Behavioral Test-bed: Obtaining Complex Behavior from Simple Rules. **The Visual Computer**, Vol. 4, Nº 6, p. 332-347, 1988.

[HIR 94] HIROSE, M.; KOMORI, S.; NAGUMO, T. A Study on the Synthesis of Environmental Sounds. In: VIRTUAL REALITY SOFTWARE AND TECHNOLOGY CONFERENCE (VRST'94). **Proceedings**..., Ed.: G. Singh, S. K. Feiner e D. Thalmann, World Scientific, Londres, 1994, p. 185-199.

[HOD 95] HODGINS, J. K.; et al. Animating Human Athletics. In: SIGGRAPH '95. **Proceedings**..., Los Angeles, CA, 1995.

[HOR 99] HORSWILL, I. D.; ZUBEK, R. Robot Architectures for Believable Game Agents. In: 1999 AAAI SPRING SYMPOSIUM ON ARTIFICIAL INTELLIGENCE AND COMPUTER GAMES. **Proceedings**..., AAAI Technical Report SS-99-02, 1999.

[KAL 98] KALLMANN, M.; THALMANN, D. Modeling Objects for Interaction Tasks. In: EUROGRAPHICS WORKSHOP ON ANIMATION AND SIMULATION. **Proceedings**..., 1998, p. 73-86.

[KAY 2000] KAYE, J. The Olfactory Display of Abstract Information. MIT Media Lab, 2000.

[KOR 82] KOREIN, V.; BADLER, N. Techniques for Generating the Goal-Directed Motions of Articulated Structures. **IEEE Computer Graphics Applications**, novembro de 1982.

[LAI 99] LAIRD, J. E.; VAN LENT, M. Developing an Artificial Intelligence Engine. In: GAME DEVELOPERS' CONFERENCE. **Proceedings**..., San Jose, CA, 1999, p. 577-588.

[LAI 2000] LAIRD, J. E. It Knows What You're Going To Do: Adding Anticipation to a Quakebot. In: AAAI 2000 SPRING SYMPOSIUM ON

ARTIFICIAL INTELLIGENCE AND INTERACTIVE ENTERTAINMENT. **Technical Report SS-00-02**, AAAI Press, 2000, p. 41-50.

[LAI 2001] LAIRD, J. E.; VAN LENT, M. Human-level AI's Killer Application: Interactive Computer Games. **AI Magazine**, 22(2), p. 15-26, 2001.

[LOY 97] LOYALL, A. B. Believable Agents: Building Interactive Personalities. **Technical Report CMU-CS-97-123**, School of Computer Science, Carnegie Mellon University, maio de 1997.

[MAC 2002] MACIEL, A.; NEDEL, L. P.; FREITAS, C.M.D.S. Anatomy Based Joint Models for Virtual Humans Skeletons. In: **IEEE Computer Animation**, Genebra, 2002.

[MAB 2001] MAC NAMEE, B.; CUNNINGHAM, P. Proposal for an Agent Architecture for Proactive Persistent Non Player Characters. In: TWELFTH IRISH CONFERENCE ON ARTIFICIAL INTELLIGENCE AND COGNITIVE SCIENCE. **Proceedings**..., 2001. p. 221 – 232.

[MAB 2002] MAC NAMEE, B.; et al. Men Behaving Appropriately: Integrating the Role Passing Technique into the ALOHA System. In: AISB'02 SYMPOSIUM: ANIMATING EXPRESSIVE CHARACTERS FOR SOCIAL INTERACTIONS. **Proceedings**...(short paper), 2002. p. 59-62.

[MAB 2003] MAC NAMEE, B.; CUNNINGHAM, P. Enhancing Non Player Characters in Computer Games using Psychological Models. **ERCIM News,** n. 53, abr. 2003.
Disponível em:
<http://www.ercim.org/publication/Ercim_News/enw53/cunningham.html>

[MAG 85] MAGNENAT-THALMANN, N.; THALMANN, D. Computer Animation: Theory and Practice. **Computer Science Workbench**, Springer Verlag, 1985.

[MAG 88] MAGNENAT-THALMANN, N.; LAPERRIERE, R.; THALMANN, D. Joint-Dependant Local Deformations for Hand Animation and Object Grasping. In: GRAPHICS INTERFACE '88. **Proceedings**..., Edmonton, 1988.

[MAG 90] MAGNENAT-THALMANN, N.; THALMANN, D. Synthetic Actors in Computer-Generated 3D Films. **Computer Science Workbench**, Springer Verlag, 1990.

[MAR 2000] MARSELLA, S.; JOHNSON, W.L.; LABORE, C. Interactive Pedagogical Drama. FOURTH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS. **Proceedings**..., 2000. p 301-308.

[MAR 2002] MARSELLA, S.; GRATCH, J. A Step Toward Irrationality: using Emotion to Change Belief. In: FIRST INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS. **Proceedings**..., Bologna, Itália, 2002.

[MAS ??] MASSIVE. Disponível em <www.massivesoftware.com>

[MCC 63] MCCARTHY, J. Situations, actions and causal laws. **Journal of logic and computation**, 4(5), p. 655-678. 1963.

[MCC 69] MCCARTHY, J.; HAYES, P.J. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: D. Michie (ed), **Machine Intelligence 4**. American Elsevier, Nova Iorque, 1969.

[MUL 99] MULTON, F. L.; et al. Computer animation of human walking: a survey. **The Journal of Visualization and computer Animation**, Vol. 10, p 39-54, 1999.

[MUS 97] MUSSE, S. R.; THALMANN, D. A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis. In: WORKSHOP OF COMPUTER ANIMATION AND SIMULATION OF EUROGRAPHICS'97. **Proceedings**..., Budapeste, Hungria, 1997.

[MUS 98] MUSSE, S. R.; et al. Crowd Modelling in Collaborative Virtual Environments. **ACM VRST**. Taiwan, 1998.

[NED 98] NEDEL, L. P.; THALMANN, D. Modeling and Deformation of the Human Body using an Anatomically-Based Approach. In: COMPUTER ANIMATION'98. **Proceedings**..., Philadelphia, Pennsylvania, USA, 1998. p. 34-40.

[NOS 95a] NOSER, H.; et al. Navigation for Digital Actors based on Synthetic Vision, Memory and Learning. **Computer and Graphics**, Vol. 19, Nº 1, 1995, p. 7-19, Pergammon Press.

[NOS 95b] NOSER, H.; THALMANN, D. Synthetic Vision and Audition for Digital Actors. In: EUROGRAPHICS `95. **Proceedings**..., Maastricht, 1995, p. 325-336.

[NOS 96] NOSER, H.; THALMANN, D. The Animation of Autonomous Actors Based on Production Rules. In: COMPUTER ANIMATION '96. **Proceedings**..., Genebra, Suiça, IEEE Computer Society Press, Los Alamitos, CA, 1996, p. 47-57.

[ORT 88] ORTONY, A.; CLORE, G. L; COLLINS, A. The Cognitive Structure of Emotions. Cambridge: Cambridge University Press, 1988.

[PAR 2001] PARENT, R. **Computer Animation: Algorithms and Techniques**. Morgan-Kaufmann, 2001.

[PER 96] PERLIN, K.; GOLDBERG, A. IMPROV: A System for Scripting Interactive Actors in Virtual Worlds. In: SIGGRAPH 96. **Proceedings**..., 1996, p. 205-216.

[PRI 9?] PRITCHARD, E. Sensation & Perception Lecture Notes. Disponível em < http://www.uwinnipeg.ca/~epritch1/sensperc.htm>.

[PRU 90] PRUSINKIEWICZ, P.; LINDENMAYER, A. **The Algorithmic Beauty of Plants**. New York: Springer Verlag, 1990.

[RAO 95] RAO, A. S.; GEORGEFF, M. P. BDI Agents: From Theory to Practice. In: FIRST INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS (ICMAS'95). **Proceedings**..., São Francisco, CA, 1995, p. 312—319.

[RAO 96] RAO, A. S. AgentSpeak (L): BDI Agents Speak Out in a Logical Computable Language. In: SEVENTH WORKSHOP ON MODELING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD (MAAMAW '96). **Proceedings**..., Eindhoven, Netherlands, Springer Verlag, LNAI 1038, 1996. p. 42–55.

[RAO 98] RAO, A. S.; GEORGEFF, M. P. Decision procedures for BDI logics. **Journal of Logic and Computation**, 8(3), p. 293–343, 1998.

[REI 92] REILLY, W. S.; BATES, J. Building Emotional Agents. **Technical Report CMU-CS-92-143**, School of Computer Science, Carnegie Mellon University, maio de 1992.

[REN 90] RENAULT, O.; THALMANN, N. M.; THALMANN, D. A Vision-based Approach to Behavioral Animation. **The Journal of Visualization and Computer Animation**, Vol. 1, Nº 1, p. 18-21, 1990.

[REY 87] REYNOLDS, C. Flocks, Herds and Schools: A Distributed Behavioral Model. In: SIGGRAPH 1987. **Proceedings**..., Computer Graphics, Vol. 21, Nº 4, 1987, p. 25-34.

[RIC 99] RICKEL, J.; JOHNSON, W. I. Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition and Motor Control. **Applied Artificial Intelligence**. Vol. 13, Nºs 4-5, 1999, p. 343-382.

[ROU 97] ROUSSEAU, D.; HAYES-ROTH, B. Improvisational Synthetic Actors with Flexible Personalities. Knowledge Systems Laboratory, **Report nº KSL 97-10**, Stanford, CA, dezembro de 1997.

[SCH 97] SCHEEPERS, F.; et al. Anatomy-Based Modeling of the Human Musculature. In: CONFERENCE SIGGRAPH, 1997, Los Angeles, CA. **Proceedings**..., New York: ACM, 1997. p. 163-172.

[SED 86] SEDERBERG, T. W.; PARRY, S. R. Free Form Deformations of Solid Geometric Models. In: SIGGRAPH '86. **Proceedings**..., 1986, p. 151-160.

[SHE 96] SHEN, J. Human Body Modeling and Deformations. PhD thesis, École Polytechnique Fédérale de Lausanne, 1996.

[SIL 99] SILVA, D.; et al. Personality-Centered Agents for Virtual Computer Games. In: WORKSHOP ON INTELLIGENT VIRTUAL AGENTS (VIRTUAL AGENTS '99). **Proceedings**..., Salford, 1999.

[STE 90] STEELS, L. Cooperation between Distributed Agents through Self-Organization. In: **Descentralized A. I.** Demazeau, Y. e Muller, J-P. (Eds), North-Holland, Amsterdam, 1990.

[TAK 2003] TAKENOBU, T.; et al. Bridging the Gap between Language and Action. In: IVA 2003. **Proceedings**..., Springer Verlag, LNAI 2792, p. 127-135, 2003.

[TER 97] TERZOPOULOS, D.; RABIE, T. Animat Vision: Active Vision in Artificial Animals. **Journal of Computer Vision Research**, 1(1):2-19, 1997.

[THA 89] THALMANN, D. Motion Control: from Keyframe to Task-Level Animation. In: STATE-OF-THE-ART IN COMPUTER ANIMATION. Springer, Tokio, 1989, p. 3-17.

[THA 95] THALMANN, D. Autonomy and Task-Level Control for Virtual Actors. **Programming and Computer Software**, Nº 4, Moscou, Rússia, 1995.

[THA 96] THALMANN, D. Physical, Behavioral, and Sensor-Based Animation. In: GRAPHICON 96. **Proceedings**..., São Petersburgo, Russia, 1996, p. 214-221.

[THA 99] THALMANN, D.; NOSER, H. Towards Autonomous, Perceptive, and Intelligent Virtual Actors. In: **Artificial Intelligence Today**, Lecture Notes in Artificial Intelligence, Nº 1600, Springer, p. 457-472, 1999.

[TOL 2000] TOLANI, D.; GOSWAMI, A.; BADLER, N. Real-time inverse kinematics techniques for anthropomorphic limbs. **Graphical Models** 62 (5), p. 353-388, setembro de 2000.

[TOS 88] TOST, D.; PUEYO, X. Human body animation: a survey. **The Visual Computer**, vol. 3, p. 254-264, Springer-Verlag, 1988.

[TU 94] TU, X.; TERZOPOULOS, D. Artificial Fishes: Physics, Locomotion, Perception, Behavior. In: SIGGRAPH '94. **Proceedings**..., Computer Graphics, 1994, p. 42-48.

[W3C 2003] W3C. **Extensible Markup Language (XML)**. Disponível em: http://www.w3.org/XML/. Acesso em: 02 dez. 2003.

[WOO 95] WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent Agents: Theory and Practice. **The Knowledge Engineering Review**, 10(2):115-152, 1995.

[WOO 99] WOOLDRIGDGE, M. Intelligent Agents. **A Modern Approach to Distributed Artificial Intelligence**. Editado por G. Weiss, 1999. p. 27-77.

[WRA 94] WRAY, R.; et al. A Survey of Cognitive and Agent Architectures. EECS 547 (Cognitive Architectures) at the University of Michigan, Department of Electrical Engineering and Computer Science, 1994. Disponível em <http://ai.eecs.umich.edu/cogarch0/common/capa/percept.html>

[ZEL 82] ZELTZER, D. Representation of Complex Animated Figures. In: GRAPHICS INTERFACE. Proceedings..., Toronto, Canada, 1982. p.205-211.

[ZHA 94] ZHAO, J.; BADLER, N. Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures. **ACM Transactions on Graphics**, 13(4), outubro de 1994.

[ZYB 99] ZYBURA, M.; ESKELAND, G. A. Olfaction for Virtual Reality. Washington University, 1999.