Marcos Santiago Ribeiro

Prototipação de Agentes com Arquitetura em Camadas e Baseada em Componentes para Desenvolvimento do Time de Futebol de Robôs da UESB

Vitória da Conquista – BA
2010

Marcos Santiago Ribeiro

Prototipação de Agentes com Arquitetura em Camadas e Baseada em Componentes para Desenvolvimento do Time de Futebol de Robôs da UESB

Monografia de conclusão de curso apresentada à Universidade Estadual do Sudoeste da Bahia, como requisito parcial à obtenção do título de Bacharel em Ciências da Computação.
Orientador: Prof. Fábio Moura Pereira

Vitória da Conquista – BA
2010

Marcos Santiago Ribeiro

Prototipação de Agentes com Arquitetura em Camadas e Baseada em Componentes para Desenvolvimento do Time de Futebol de Robôs da UESB

Monografia de conclusão de curso apresentada à Universidade Estadual do Sudoeste da Bahia, como requisito parcial à obtenção do título de Bacharel em Ciências da Computação.
Orientador: Prof. Fábio Moura Pereira

Vitória da Conquista, 3 de setembro de 2010.

Banca Examinadora

Orientador:

Doutor, Fábio Moura Pereira

Membros:

Mestre, Marcelo Barbosa de Almeida

Mestre, Cláudio Rodolfo Sousa de Oliveira

AGRADECIMENTOS

Aos meus pais, pelo amor, ensinamentos, incentivos, que me guiaram e me deram determinação para traçar o caminho até a faculdade e me tornar a pessoa que sou.

Aos meus irmãos que sempre me apoiaram e foram um exemplo na minha vida.

A Fernanda, meu amor, que sempre esteve ao meu lado ajudando a superar os momentos mais difíceis nesta jornada.

A meus amigos pela atenção, ajuda, e por estarem presentes apesar da distância.

A meu orientador, Fábio Moura Pereira pela paciência e auxilio à concretização dessa monografia.

Ao pessoal da turma de computação que foram grandes companheiros nesses cinco anos de curso.

A todos que de alguma forma contribuíram para realização desse trabalho.

RESUMO

O desejo do homem de construir máquinas inteligentes vem de muitos anos. Com a evolução computacional esse ideal ganhou mais força. A RoboCup "*Robot World Cup"*, que é uma iniciativa de um grupo internacional de pesquisadores em Inteligência Artificial (IA) e Robótica, por meio de suas competições em nível mundial, tem incentivado e despertado o interesse nessa área para estudantes e pesquisadores em todo o mundo. Esse trabalho é focado na liga de futebol simulado em duas dimensões da RoboCup. Esta propõe que os agentes joguem uma partida, seguindo regras e atuando em um ambiente dinâmico com complexidade do mundo real em suas simulações. Este trabalho apresenta o processo de modelagem para o protótipo de agente jogador, seguindo notações do Desenvolvimento Baseado em Componentes (CBD). A proposta para o desenvolvimento foi feita sob uma arquitetura em camadas vertical que combina elementos reativos e deliberativos, também conhecida como arquitetura híbrida, e para implementação do agente foi utilizada a linguagem de alto nível Python.

Palavras-chave: *RoboCup, Sistemas Multi-agente e, Desenvolvimento Baseado em Componentes*.

ABSTRACT

Man's desire to build intelligent machines comes from many years. With the evolution of computing that ideal has gained more strength. The RoboCup "*Robot World Cup*", which is an initiative of an international group of researchers in Artificial Intelligence (AI) and Robotics, through its contests worldwide, has encouraged and stimulated interest in this area for students and researchers around the world. This paper is focused on soccer simulated league in two dimensions of RoboCup. This suggests that agents play a game, following rules and acting in a dynamic environment with real-world complexity in their simulations. This paper presents the modeling process for the prototype player agent, following Component based Development (CBD) notations. The proposal for the development was done under a vertical layered architecture that combines deliberative and reactive elements, also known as hybrid architecture, and for agent implementation was used the high-level language Python.

Keywords: *RoboCup, Multi-agent Systems, Component Based Development.*

SUMÁRIO

LISTA DE FIGURAS

1. Introdução

O desejo do homem de construir máquinas com capacidade para reproduzir a forma humana de pensar e agir vem de muitos anos. Com a evolução computacional esse ideal ganhou mais força, levando ao surgimento de uma área de pesquisa da ciência da computação: a inteligência artificial. Essa se dispõe a desenvolver mecanismos e dispositivos que simulem a capacidade do ser humano de resolver problemas, ou seja, de ser inteligente.

O desenvolvimento de agentes inteligentes é um ramo de pesquisa da inteligência artificial que busca a criação de entidades, sejam elas físicas ou simuladas, para desempenharem funções de forma autônoma em um ambiente restrito. Segundo a IBM RESEARCH (1998), um agente é uma entidade que executa um conjunto de operações que lhes foram incumbidas por um usuário ou outro programa, com algum grau de independência ou autonomia e, executando estas operações, emprega algum conhecimento dos objetivos ou desejos do usuário.

Nos modelos de agentes inteligentes os elementos autônomos são dotados de alguma forma de inteligência pré-definida que juntamente com sensores capazes de perceber seu ambiente e as mudanças ocorridas através da intervenção humana ou de outros agentes, é capaz de tomar decisões e agir por intermédio de atuadores. Em ambientes constituídos por mais de um agente é importante que uma linguagem comum seja utilizada a fim de manter o cooperativismo.

De acordo com Russel & Norvig (2004) os Sistemas Multi-Agente (SMA) são constituídos por uma comunidade de agentes inteligentes com existência própria, independente da existência dos demais, mas que tendem a atuar em conjunto visando um objetivo comum. Esse sistema tem como idéia principal a saída de um comportamento individual inteligente para chegar a um comportamento global inteligente que envolveria coordenação, leis sociais, planejamento, conhecimento do ambiente e das interações entre agentes.

Um bom exemplo para múltiplos agentes seria um time de futebol onde parte dos jogadores são membros de uma mesma sociedade cuja meta é a vitória no jogo. Para isso, eles agem em conjunto com intenção de fazer gols utilizando táticas de equipe e avançam sobre os membros adversários tentando retardar suas jogadas.

Para se medir a superioridade de um time deve ser levado em conta alguns fatores, tais como: a qualidade das estratégias utilizadas; a velocidade de resposta a percepções do ambiente; a interação com os companheiros e o roteiro de prioridades escolhidas, sendo que

este pode ser modificado de acordo com cada situação específica.

A RoboCup, *"Robot World Cup"*, é uma iniciativa de um grupo internacional de pesquisadores em Inteligência Artificial (IA) e Robótica. Através de competições em nível mundial, essa iniciativa oferece a integração de diversas áreas na tentativa de solucionar problemas, de forma que novas metodologias sejam testadas e, principalmente comparadas (ROBOCUP, s.d.). O foco desse trabalho se dá na liga de simulação para duas dimensões, onde agentes jogadores se enfrentam em uma partida de futebol.

1.1 Motivação

A RoboCup tem tomado grandes proporções e sido divulgada amplamente em todo o mundo através das suas competições anuais. As competições da liga de simulação em duas dimensões tem apresentado agentes jogadores com um alto nível de sofisticação. O desenvolvimento ou aperfeiçoamento desses times motiva o estudo de novas técnicas para inteligência artificial na área de SMA.

O problema proposto por essa liga, é que agentes joguem uma partida de futebol, seguindo regras e atuando em um ambiente dinâmico que adiciona complexidade do mundo real em suas simulações, visto que as percepções são limitadas. Nessa partida cada agente deve assumir uma função (ou papel) específica, uns devem representar o ataque, outros à defesa e meio de campo, além do goleiro. Esses são alguns dos muitos problemas impostos aos agentes.

Os desafios que envolvem as soluções para tais problemas estimularam o estudo e a realização de pesquisas com o intuito de criar um protótipo de um time de agentes jogadores. O desejo que este seja a base para um time competitivo da Universidade Estadual do Sudoeste da Bahia (UESB) foi o estimulo inicial para realização desse trabalho. Contudo, a possibilidade de trabalhar com SMA foi a maior motivação para tal, uma vez que esse é um tema promissor e vem sendo muito abordado. Outro aspecto de interesse foi o emprego de técnicas avançadas de engenharia de software na modelagem do agente, por exemplo, o Desenvolvimento Baseado em Componentes (CBD) que segundo Szyperski (1997 *apud* PINHO, 2006) mantém o projeto conciso de forma a separar os grupos de funcionalidades, e assim permitir uma rápida modelagem, bem como uma eficaz manutenção de código.

1.2 Objetivos

Este trabalho tem como objetivo principal o desenvolvimento do protótipo de um time de futebol de robôs simulados para a UESB. Esse tem como finalidade o uso de técnicas de engenharia de software para o desenvolvimento de Sistemas Multi-Agentes.
Os objetivos específicos deste trabalho são:

- Estudo do simulador de futebol de robôs, as regras da partida e também o protocolo de comunicação entre agente e simulador;
- O emprego de técnicas avançadas de engenharia de software e inteligência artificial, como o Desenvolvimento Baseado em Componentes;
- Realizar a simulação do futebol de robôs no ambiente virtual, possibilitando o estudo do comportamento do time e as técnicas de inteligência artificial;
- Possibilitar a formação de um time de futebol de robôs que seja a base para um futuro time competitivo.

1.3 Metodologia

Para o desenvolvimento deste trabalho inicialmente foi realizado um levantamento bibliográfico sobre arquitetura em camadas de agentes inteligentes e sobre o Desenvolvimento Baseado em Componentes, assim como um estudo para aprofundar os conhecimentos da linguagem Python. Com base nos conhecimentos adquiridos através dessas leituras e do estudo do ambiente de simulação do futebol de robôs foi modelado um protótipo de um time e então codificado baseado na linguagem estudada. Em seguida foram realizados testes nas funções de cada componente. Como complemento foi estudado algumas táticas e regras de futebol que servirão para compreender como o time dever atuar no ambiente virtual ou planejar futuras estratégias do time para uma posterior conclusão do protótipo.

1.4 Organização da Monografia

Esta monografia está dividida em seis Capítulos que tratam sobre conhecimentos relacionados a agentes e Sistemas Multi-Agentes necessários para o desenvolvimento de um time de futebol de robôs no ambiente simulado da RoboCup.

O Capítulo 2 trata-se basicamente sobre os agentes, explicitando propriedades, tipos,

arquiteturas, ambientes de atuação e sua integração e comunicação com outros agentes.

Já o Capítulo 3 busca informar sobre o conceito e histórico da RoboCup, os objetivos, as ligas e competições propostas, bem como funcionamento do ambiente de simulação.

No Capítulo 4 é exposto vários trabalhos que se encontram relacionados a Robocup.

O Capítulo 5 descreve os motivos que justificam a escolha da metodologia, arquitetura e linguagem escolhidas para realizar este trabalho, assim como o protótipo do time de agentes.

No ultimo Capítulo as considerações finais são expostas, mostrando quais os objetivos foram alcançados e trazendo sugestões para possíveis trabalhos futuros.

2. Engenharia de software para Agentes Inteligentes

Esse capítulo trata sobre o que é um agente e quais as propriedades gerais dos mesmos. Posteriormente, são expostos os tipos de agentes, suas arquiteturas, os sistemas multi-agente, a comunicação entre agentes, os ambientes de atuação e por fim o processo de desenvolvimento baseado em componentes.

2.1 Agentes

Os significados do termo “agente” vêm sendo discutidos por diversos pesquisadores da área que tentam responder perguntas como “o que é um agente?” ou “o que torna algo um agente?”. Wooldridge & Jennings (1994) consideram que a resposta à questão "o que é um agente?" é tão imprecisa quanto a resposta à questão "o que é inteligência?" visto que, nos dois casos, no cenário computacional, não existe uma definição para o tema aceita universalmente. Franklin & Graesser (1996) reforçam a idéia de que é difícil encontrar uma definição única para a palavra agente até entre pesquisadores que trabalham com agentes inteligentes, pois cada profissional a define com base em seu próprio entendimento.

2.1.1 Definição

Na língua portuguesa o significado da palavra "agente" segundo Aurélio (2003), é definido como “Alguém que atua”, entretanto os pesquisadores em Inteligência Artificial definem agentes de acordo com o seu ponto de vista. Wooldridge & Jennings, (1994) afirmam que agentes são sistemas que apresentam um comportamento determinado por um processo de raciocínio baseado na representação de suas atitudes, tais como crenças, comprometimentos e desejos. Eles acreditam que um sistema pode ser visto como um agente se ele possuir as seguintes propriedades: autonomia, habilidade social, reatividade e pró-atividade. Franklin & Graesser (1996) consideram que um agente autônomo é um sistema que faz parte de um ambiente, onde ele percebe e atua continuamente em busca de sua própria agenda, a fim de aplicar o que percebeu para atender suas próprias metas futuras. Já Russel & Norvig (2004) apresentam uma proposta de que agentes são como uma entidade real ou virtual, que está imersa ou situada em um ambiente físico ou virtual/simulado, que pode perceber o ambiente através de sensores e agir através de atuadores, possui objetivos próprios explícitos ou implícitos e escolhe suas ações em função das suas percepções para atingir seus objetivos.

Uma definição mais específica e que deve ser aceitável pela maioria dos pesquisadores é proposta por (SHOHAM, 1997 *apud* COSTA, 1999). Ele define agente como sendo uma entidade de software funcionando continuamente e de forma autônoma em um ambiente particular, freqüentemente habitado por outros agentes e processos.

Dessa maneira, pode-se notar a existência de diversas propostas para definir o que é um agente. Entretanto, é possível constatar com clareza que todas elas possuem características em comum. Dentre estas deve-se enfatizar propriedades como autonomia, capacidade de responder a determinadas situações, facilidades para comunicação e capacidade para aprender como alcançar seus objetivos.

2.1.2 Propriedades dos agentes

Existem diversas propriedades encontradas em definições designadas por pesquisadores da área que visam agrupar os agentes em classes ou tipologias. Elas são apresentadas nos agentes de acordo com as funcionalidades desejadas para o seu uso.

"Um agente não precisa possuir todas estas características ou atributos, embora suas capacidades estejam diretamente associadas à presença delas" (WOOLDRIDGE & JENNINGS,1994 *apud* COSTA, 1999).

A seguir são apresentadas propriedades que segundo Wooldredge & Jennings (1994) são consideradas importantes para os agentes:

- Autonomia: os agentes podem operar sem a intervenção de humanos ou de outros agentes;
- Sociabilidade: capacidade de interagir através de troca de informações vindas do ambiente ou de outros agentes;
- Reatividade: os agentes são capazes de reagir a mudanças no ambiente;
- Mobilidade: capacidade de um agente de locomover-se entre diferentes ambientes;
- Pró-atividade, iniciativa: os agentes não são apenas entidades que reagem a um estímulo. Tem caráter empreendedor e podem atuar guiados por seus objetivos.

2.2 Tipos de Agentes

De acordo com Russel & Norvig (2004) os agentes podem ser definidos em tipos como: Agente Reativo; Agente Autômato; Agente Cognitivo; Agente Híbrido; Agente

Otimizador; Agente Deliberativo; Agente Adaptativo. Para cada um dos tipos um modelo de arquitetura é definido, tornando o agente apropriado para seguir seus objetivos através das condições impostas pelo ambiente.

2.2.1 Agente Reativo

São geralmente agentes simples que possuem um mapeamento de situações e respostas associadas, ou seja, quando há alterações em um estado ambiental, o agente executa a ação correspondente para satisfazer o novo estado. Tem semelhanças com modelos de organização biológica ou de comportamento animal, tais como a sociedade das abelhas ou formigas.

2.2.2 Agente Autômato ou Reativo com Estado Interno

Composto por percepções anteriores utilizadas na tentativa de construir parte do mundo em que os sensores no momento estão impossibilitados de interpretar. Diferentemente da arquitetura anterior, utiliza tanto as percepções atuais quanto as passadas. Por isso é de fundamental importância que se saiba como o mundo evolui independente do agente, assim como as ações do agente vão afetar a evolução do mundo.

2.2.3 Agente Cognitivo

Este tipo de agente também pode ser chamado de simbólico ou deliberativo e se baseia em um modelo organizacional social humano. Este modelo possui uma memória que armazena um histórico das ações passadas e auxilia no planejamento de ações futuras, embora só seja alterado pelo agente em resposta a novas informações do ambiente. Com base na interpretação, o agente estima quais ações serão necessárias para atingir o objetivo proposto e também quais suas conseqüências e só posteriormente executa então as ações que possibilitarão a sua realização. É considerado flexível e autônomo embora não trate de objetivos conflitantes.

2.2.4 Agente Baseado em Utilidade

Esse tipo de agente se baseia na função utilidade a partir de decisões racionais quando existem objetivos conflitantes ou quando há vários objetivos com possibilidade incerta de

atingi-los. Desta forma se um estado do mundo é mais desejável que outro, então ele terá maior utilidade para o agente.

2.2.5 Agente Otimizador

Permite escolher melhor compromisso entre vários objetivos conflitantes ou vários objetivos com probabilidades diferentes de serem alcançados. Tem como principal vantagem a utilização de ambiente sem restrição. Apresenta como desvantagens a falta de adaptabilidade assim como as abordagens existentes que tendem a ser pouco escaláveis.

2.2.6 Agente Deliberativo

Este tipo de agente realiza uma associação indireta de percepção-ação, mediada pelo modelo atual do ambiente, com objetivo explícito, e pela previsão de estados futuros do ambiente resultante de uma seqüência de ações. Ele encadeia regras para construir um plano multi-passo necessário para atingir o objetivo a partir do modelo atual. Tem como principal vantagem o fato de tomar ações mais relevantes e seguras e como desvantagem o custo da deliberação, que pode ser excessivo em ambientes de tempo real.

2.2.7 Agente Adaptativo (learning)

Tem como vantagem principal a adaptabilidade uma vez que possui a capacidade de aprender e de adaptar-se ao ambiente em que está inserido. Isso se deve à presença de um componente de análise crítica de desempenho associado a um componente de aprendizagem de conhecimento.

2.3 Arquitetura de Agentes

Algumas das arquiteturas que os agentes podem ser classificados são: Arquitetura de Subsunção; Arquitetura Baseada em Lógica; Arquitetura Baseada em Metas e Arquitetura Baseada em Camadas.

2.3.1 Arquitetura de Subsunção

A arquitetura de subsunção ou de Brooks (1986) foi uma das primeiras propostas para agentes, sendo originalmente idealizada para a utilização em robôs moveis autônomos capazes de realizar o processamento de informações complexas em tempo real. Nela destacam-se duas características: a primeira define que a tomada de decisão do agente é realizada através de um conjunto de comportamentos direcionados a tarefas a cumprir. O agente percebe estímulos de entrada e mapeia esses estímulos em uma ação para realizar. Máquinas de estado finito estendidas formam módulos comportamentais, que são associados a alguma tarefa particular, onde apenas são implementadas regras para as quais uma situação corresponde uma ação. Cada uma destas possui um sinal de entrada e saída. Quando a entrada excede um limite pré-determinado, o comportamento da qual a máquina de estados finitos se refere é ativado, ou seja, a saída é ativada. As entradas são obtidas através dos sensores ou através de outras máquinas de estados finitos. A saída é enviada para os atuadores do agente ou para a entrada de outras máquinas de estados finitos. Cada máquina de estados estendida também aceita sinais de supressão e inibição. Um sinal de supressão sobrepõe o sinal normal de entrada e um sinal de inibição inibe completamente a saída. Estes sinais permitem que comportamentos sobreponham de tal maneira que o sistema possa produzir um comportamento coerente (BARBOSA, 2005).

A outra característica define que vários comportamentos podem ser disparados de maneira simultânea, tornando-se necessário um mecanismo de escolha que selecione a ação correspondente. Deste modo, os módulos comportamentais são organizados em camadas em uma hierarquia dita de subsunção onde as camadas mais baixas inibem as mais altas, ou seja, quanto mais baixa a camada mais alta é sua prioridade. Por ser reativa, essa arquitetura tem como desvantagem a necessidade de readaptação do sistema a novos ambientes, já que não é possível obter informações armazenadas, que não estejam disponíveis nos seus sensores, nem de se tirar vantagem de um conhecimento prévio para uma nova tarefa (BARBOSA, 2005).

2.3.2 Arquitetura Baseada em Lógica

Neste tipo de arquitetura, o comportamento inteligente do sistema pode ser gerado a partir da representação simbólica do seu ambiente através de um conjunto de fórmulas lógicas. A proposta de construção de sistemas inteligentes sugere que um comportamento inteligente pode ser construído ao se prover uma representação simbólica do seu ambiente e

do comportamento desejado para o mesmo, além de meios para manipular sintaticamente essa representação. Os agentes baseados em lógica apresentam representação simbólica como um conjunto de fórmulas lógicas e a manipulação sintática como a dedução lógica e/ou prova de teoremas (WEISS, 1999 *apud* BARBOSA, 2005).

O processo de tomada de decisão de um agente é modelado a partir de um conjunto de regras de decisão. É necessário que o agente encapsule no seu código tanto as Regras quanto a Base de Conhecimento, de tal forma que, caso uma fórmula seja derivada a partir deles, possa ser concluído que existe um termo que descreve a melhor ação possível para dada situação (WEISS, 1999 *apud* BARBOSA, 2005).

Os agentes que pertencem a essa arquitetura possuem estrutura semântica elegante, simples, clara e eficiente. Mas esse método possui a desvantagem da complexidade computacional inerente da "prova de teoremas", que torna questionável a utilização desse tipo de agentes em ambientes com restrições de tempo de resposta (WEISS, 1999 *apud* BARBOSA, 2005).

2.3.3 Arquitetura Baseada em Metas

Um agente pró-ativo possui um comportamento orientado a metas, onde é prevista uma relação entre uma pré-condição e o efeito da pós-condição. Quando é feita uma chamada a um procedimento e uma pré-condição conduz a uma pós-condição verdadeira, a meta será alcançada (WEISS, 1999).

A meta é um importante conceito em um sistema de agentes inteligentes que pode apresentar dois aspectos: declarativo e procedural. No primeiro é descrito o ambiente em que se deseja alcançar e é notável um raciocínio sobre propriedades fundamentais das metas. Já o aspecto procedural traça um conjunto de planos para que a meta seja alcançada de maneira eficiente em ambiente bastante dinâmico. Resumidamente, o aspecto procedural é fundamental para a aplicação de sistemas em ambientes dinâmicos por oferecer ao agente a capacidade de produzir metas reais e possíveis de se alcançar. Enquanto o aspecto declarativo assegura a habilidade de raciocínio sobre a meta, sem o qual não poderia haver verificação sobre a possibilidade de realizar a meta ou se há interferência entre elas. O comportamento orientado a metas é adequado para ambientes que não se alteram no período de execução do procedimento, mantendo, portanto, a meta válida até que este finalize. Dessa forma, esta arquitetura não é propicia para ambientes dinâmicos (WINIKOFF, 2002 *apud* BARBOSA, 2005).

2.3.4 Arquitetura Baseada em Camadas

Esta abordagem, também conhecida como arquitetura híbrida, combina componentes da arquitetura reativa que pode reagir a eventos ocorridos no ambiente sem se ocupar com raciocínio complexo e da deliberativa que contem o modelo simbólico do mundo, desenvolve planos e toma decisões. Normalmente as ações do componente reativo podem ter precedência sobre o cognitivo, de modo a fornecer uma rápida resposta a algum evento importante do ambiente (WEISS, 1999).

Na arquitetura em camadas os vários subsistemas são organizados em uma hierarquia de camadas interagindo por meio de processos lógicos que dão a cada camada a possibilidade de tomada de decisões sob diferentes níveis de abstração do ambiente (WEISS, 1999).

De acordo com o fluxo de controle dentro das arquiteturas de camadas, a Arquitetura Híbrida pode ser classificada em arquitetura em camadas horizontais ou arquitetura em camadas verticais.

- Horizontal – nas arquiteturas em camadas horizontais, cada camada de software está diretamente conectada ao sensor responsável pela entrada de percepções do mundo para formular suas próprias instruções de saída. Dessa forma, cada camada pode ser considerada um pequeno agente que produz sugestões sobre qual ação executar levando em conta o comportamento programado na camada. Dependendo da necessidade de diferentes comportamentos, mais camadas podem ser acrescentadas ao agente. Para não haver uma competição entra as diferentes propostas de ação, uma camada com função de mediador deve ser inserida. Entretanto, o excesso de camadas pode provocar um gargalo na tomada de decisão do agente e reduzir seu desempenho global (WEISS, 1999 *apud* BARBOSA, 2005).

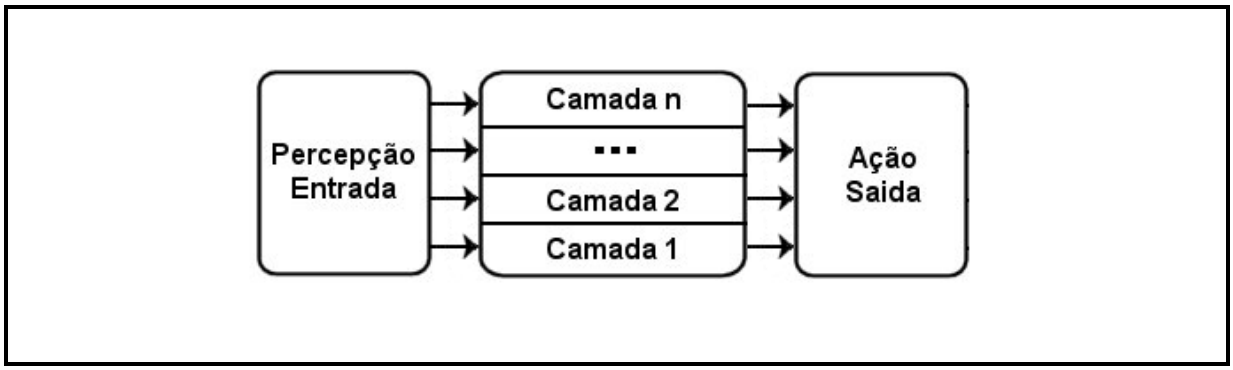

Figura 1: Arquitetura em camadas horizontais (adaptado de WEISS, 1999)

- Vertical – nas arquiteturas em camadas verticais o sensor de percepções fica conectado somente a uma camada de entrada e o fluxo de controle segue sequencialmente pelas demais camadas. Esse fluxo pode ser classificado como de passagem única ou dupla. No fluxo de uma passagem cada camada faz sua interpretação das percepções e avalia sugestões vindas das camadas inferiores até que na camada final é definida a ação mais adequada para ser enviada aos atuadores do agente. Já no fluxo de duas passagens, as informações fluem para os níveis mais altos da organização, e quando alcançam o topo, o controle é novamente passado às camadas inferiores que refinam as possíveis ações para a mais adequada (WEISS, 1999 *apud* BARBOSA, 2005).

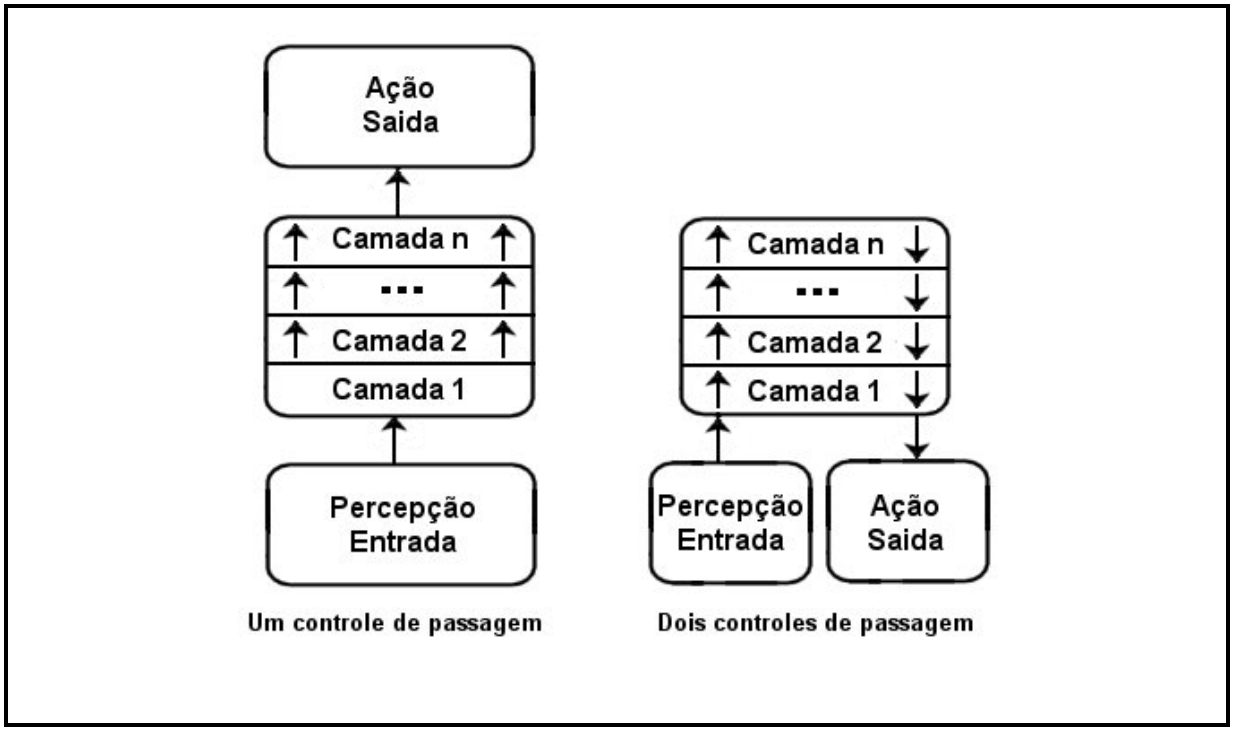

Figura 2: Arquitetura em camadas verticais (adaptado de WEISS, 1999)

2.4 Sistemas Multi-Agentes (SMA)

Os Sistemas Multi-Agentes são considerados um subcampo emergente da IA que visa fornecer os princípios para a construção de sistemas complexos envolvendo múltiplos agentes. O foco principal desses sistemas consiste em coordenar o comportamento inteligente de um conjunto de agentes autônomos, para obter a solução de um problema apresentado que está além da capacidade de resolução de cada um individualmente (OLIVEIRA, 1996 *apud* SOUZA, 1996). O atributo autônomo significa que um agente tem sua existência própria, não dependente dos agentes imersos no ambiente ou mesmo do problema sendo solucionado. No

entanto, para alcançar os seus objetivos ele eventualmente precisará da ajuda de outros.

2.4.1 Inteligência Artificial Distribuída (IAD)

O objetivo do estudo da IAD é construir e desenvolver os sistemas compostos de entidades com graus de autonomia e inteligência variáveis, que cooperem entre si para o aumento do seu desempenho, visando resolver um problema global.

Focando na solução desses problemas, os agentes, sendo entidades que encapsulam conhecimento sobre algum domínio, podem ser agrupados em uma sociedade onde cada agente seja inserido em um subconjunto que suas habilidades sejam requeridas na estratégia de solução do problema. Assim cada agente pode ficar responsável por parte das tarefas a serem cumpridas de acordo com sua disponibilidade de recursos (JUCHEM & BASTOS, 2001).

De acordo com os pesquisadores os sistemas com múltiplos agentes podem distinguir-se em duas principais classes, dependendo da resolução de problemas abordados:

- **Resolução Distribuída de Problemas (RDP)**

Nesses sistemas a coordenação das atividades dos agentes é centralizada. Os múltiplos agentes tem o comportamento definido através da existência de regras já pré-estabelecidas. Inicialmente todos os agentes conhecem a priori os demais agentes da comunidade e todos trabalham de maneira benevolente para atingirem um dado objetivo, existindo desta forma confiança mútua em relação às suas interações.

O planejamento das ações desenvolvidas por cada agente é efetuado inicialmente por decomposição do problema em subproblemas que são entregues aos vários agentes do grupo. Dessa maneira os agentes podem dividir o esforço e partilhar seus conhecimento e resultados.

- **Sistemas Multi-Agentes**

Os agentes podem ter tanto objetivos globais como também objetivos locais. A coordenação do comportamento inteligente dessa comunidade é descentralizada, de maneira que o resultado é obtido com a integração do conhecimento, capacidade, objetivos, e planos dos diferentes agentes autônomos. Os agentes participam mais autonomamente na decisão de realização das suas próprias ações. Não existe um planejamento inicial e os agentes podem ou não ter conhecimento prévio dos demais agentes.

2.4.2 Comunicação entre Agentes

A comunicação fornece a base necessária para a realização da cooperação entre múltiplos agentes. Através da comunicação os agentes podem ter uma visão menos local do problema e, desta forma, adquirir o conhecimento e sincronia necessários, possibilitando resoluções mais rápidas. Deve-se levar em conta, que se a comunicação for excessiva será prejudicial, pois a carga de comunicação poderá ser maior que o trabalho efetivamente realizado (COSTA, 1999).

Os sistemas modernos de computação frequentemente envolvem múltiplos computadores interagindo de forma distribuída. Neste sentido, a habilidade de comunicação é um importante atributo que os agentes inteligentes devem possuir. Ambientes que possuem mais de um agente praticamente exigem o intercâmbio de informações. Dessa forma, torna-se claro a necessidade de uma linguagem de comunicação comum. Mas o problema é definir qual a linguagem ideal. Atualmente, não existe uma linguagem padronizada e aceita mundialmente para a representação de informações trocadas por agentes. A presença de uma linguagem comum para troca de mensagens é uma das características que diferencia um agente de um objeto na programação orientada a objetos. Uma mensagem no contexto dos agentes carrega uma semântica independente do agente, enquanto que uma mensagem no contexto dos objetos pode variar de um objeto para outro (COSTA, 1999).

Finin et al (1993 *apud* COSTA, 1999) estabeleceu a existência de diversos níveis nos quais sistemas baseados em agentes devem interagir:

- Transporte: como agentes enviam e recebem mensagens.
- Linguagem: qual o sentido de mensagens individuais.
- Política: como os agentes estruturam conversações.
- Arquitetura: como conectar sistemas em concordância com protocolos existentes.

2.5 Ambientes de Agentes

Um ambiente provê as condições em que uma entidade (agente ou objeto) existe. Em outras palavras, ele define as propriedades do mundo em que um agente pode realizar suas ações.

Um agente pode estar inserido em diferentes tipos de ambientes, o que afeta diretamente seu projeto e desenvolvimento. De acordo com as dimensões classificadoras do

ambiente, os agentes podem estar inseridos em um universo físico (robôs), de software (softbots) ou de ambientes de simulação em três dimensões.

Os ambientes também podem ser classificados de acordo com as suas características, segundo Russel & Norvig (2004) elas são:

- Totalmente Observável x Parcialmente Observável: se os sensores de um agente são capazes de captar todos os aspectos importantes para que ele possa realizar alguma ação em um ambiente ele é considerado completamente observável. Sua vantagem é que não há necessidade que o agente armazene suas percepções em estados internos para ter o controle do mundo em que atua. Um ambiente pode passar a ser considerado como parcialmente observável se as informações passadas pelos sensores apresentarem ruído ou se o agente tiver sua visão encoberta por qualquer objeto no ambiente.
- Determinista x Estocástico: um ambiente é considerado determinista se o estado atual do agente e as ações executadas por ele determinarão completamente o próximo estado do ambiente, caso contrário ele é considerado estocástico.
- Episódico x Sequencial: quando a experiência de um agente é dividida em episódios atômicos contendo uma seqüência de percepções e ações, a qualidade da ação depende apenas do episódio mais recente. Em ambientes sequenciais a decisão tomada irá afetar as futuras decisões, o que torna o agente mais complicado que o episódico, pois o obriga a ponderar nas consequências das ações.
- Estacionário x Dinâmico: se enquanto um agente estiver raciocinando, o ambiente em que ele se encontra estiver mudando, ele é considerado dinâmico. A grande vantagem do ambiente estático é a falta de preocupação em continuar a observar o mundo enquanto ele estiver pensando porque ele não irá mudar.
- Discreto x Contínuo: um ambiente é discreto se nele é fixado um número finito de ações e percepções em Russell & Norvig (2004) um exemplo de ambiente discreto dado é o jogo de xadrez, e para um ambiente contínuo um motorista de táxi imerso em uma cidade.
- Agente Unitário x Multi-Agente: ambientes onde um agente tenta resolver um problema como um quebra-cabeça são caracterizados como agente unitário. Em ambientes multi-agente existem agentes trabalhando para resolver um problema proposto, sendo que podem estar competindo ou cooperando uns com os outros.

2.5.1 Ambientes de Simulação

Um ambiente de simulação é uma plataforma de desenvolvimento de aplicações e realização de pesquisas em Inteligência Artificial. Estes após serem inicializados, começam seu ciclo de processamento, onde serão geradas as percepções a partir do estado atual do ambiente, e então estas percepções são enviadas aos agentes. O simulador também tem como função receber e processar as ações escolhidas pelos agentes, para que seja possível a atualização do estado do ambiente.

Os ambientes Wumpus e a simulação da RoboCup são dois exemplos de ambientes de simulação mundialmente utilizados por pesquisadores em IA.

A RoboCup propõe o uso de um jogo de futebol como uma plataforma para uma ampla variedade de pesquisas em IA e robótica. O ambiente formado pelos agentes e o simulador é bastante dinâmico, pois enquanto um agente está inferindo em qual ação ele deve tomar, outros podem estar realizando alguma ação, a bola pode estar em movimento e o ambiente, de uma forma geral, está sofrendo influência de todas as ações. Ao contrário do que acontece, por exemplo, com um ambiente formado por uma partida de xadrez, em que enquanto um jogador raciocina qual peça ele deve mover, o ambiente fica estático e o oponente deve ficar aguardando. Este ambiente também é considerado estocástico, pois uma ação realizada pelo agente não será suficiente para inferir no próximo estado do ambiente, impedindo a previsibilidade e tornando o ambiente parcialmente observável porque é comum que um agente consiga encobrir a visão de outros agentes (ROBOCUP, s.d.).

2.6 Desenvolvimento Baseado em Componentes X Engenharia de Agentes Inteligentes

Componentes são unidades de software desenvolvidas e testadas separadamente e que podem ser integradas com outras para construir algo com maior funcionalidade (SZYPERSKI, 1997 *apud* PINHO, 2006). Eles encapsulam objetos, protegendo-os com uma interface, de maneira contratual. O uso de contrato implica que cada uma das funcionalidades do componente é definida em termos de sua assinatura, ou seja, seus tipos de argumentos de entrada e saída. Esses contratos podem ser substituídos por outro que obedeça a um contrato compatível com o publicado por sua interface (PINHO, 2006).

Definições sobre componentes podem variar um pouco de autor para autor, porém, alguns aspectos são sempre ressaltados na literatura: Um componente oferece

funcionalidades, por meio de interfaces bem definidas, para o meio externo; Componentes são substituíveis e modulares, eles ajudam a gerenciar a complexidade e encorajam a reutilização (PINHO, 2006).

2.6.1 O Processo de Desenvolvimento Baseado em Componentes

O processo de desenvolvimento baseado em componentes visa fornecer um conjunto de procedimentos, ferramentas e notações que possibilitem, ao longo do processo de software, a ocorrência tanto da produção de novos componentes quanto a reutilização de componentes existentes. Um processo de desenvolvimento de software é um conjunto de etapas, métodos, técnicas e práticas que empregam pessoas para o desenvolvimento e manutenção de um software e seus artefatos associados (planos, documentos, modelos, código, casos de testes, manuais, etc.). É composto de boas práticas de engenharia de software que conduzem o desenvolvimento do software, reduzindo os riscos e aumentando a confiabilidade dos sistemas (JACOBSON, 1999 *apud* PINHO, 2006). Um exemplo de CBD seria o caso do UML Componentes (CHEESMAN, 2000 *apud* PINHO, 2006) baseado na linguagem UML (Unified Modeling Language), trata do problema de especificar e arquitetar sistemas baseados em componentes. Embora existam muitos processos de desenvolvimento de software, algumas atividades fundamentais estão presentes em todos eles. São elas: levantamento de requisitos, projeto, implementação e testes (PINHO, 2006).

Em Sistemas Multi-Agentes o uso de componentes dá uma maior flexibilidade a construção da arquitetura dos agentes inteligentes, que pode ser decomposta em módulos componentes com funções e tarefas definidas, podendo ser estudados e modificados ou até substituídos com eficiência. O conjunto de módulos e suas interações devem prover uma resposta para a questão de como os sensores de dados e o estado interno corrente do agente determinam suas ações e futuro estado interno (WOOLDRIDGE & JENNINGS, 1994 *apud* SOUZA, 1996).

Segundo Nissen (1995 *apud* SOUZA, 1996) cinco componentes são necessários para a infra-estrutura de atuação dos agentes inteligentes:

- Recursos de execução – consistem de aplicações e equipamentos necessários para executar programas agentes em ambientes de agentes;
- Recursos de comunicação - é um conjunto de protocolos padrão que permite o processo de comunicação;

- Recursos de transporte - preocupa-se com a maneira como os agentes são transportados e a maneira como se comunicam, sendo responsável pela movimentação destes entre os ambientes de execução;
- Recursos de empacotamento - tem como objetivo prover um método padrão de encapsulamento de agentes com suas respectivas informações (e.g. estado, autenticação, objetivo, capacidades do agente, método ou plano)
- Segurança integrada - A infra-estrutura do agente deve possuir métodos de segurança integrados, aplicáveis para todos os seus recursos.

Já a arquitetura IRMA (Intelligent Resource-bounded Machine Architecture) baseada em crenças, desejos e intenções, apresentada por Bratman et al. (1988 *apud* SOUZA, 1996) é composta pelos componentes:

- Motor de Inferência - para resolução sobre o mundo;
- Analisador Meios-Fins - para determinar que plano deve ser utilizado para atingir as intenções do agente;
- Analisador de Oportunidades - para monitorar o ambiente e determinar mais opções para o agente;
- Processo de Filtragem - para determinar o subconjunto dos cursos de ação consistentes com as intenções do agente;
- Processo de Deliberação - para fazer a escolha final entre as opções.

No processo de CBD para agentes inteligentes, os componentes podem ser organizados de acordo com a arquitetura proposta. No entanto algumas características adicionais aos processos convencionais devem ser tratadas. Segundo Jacobson et. al. (1999 *apud* PINHO, 2006), um processo de desenvolvimento de software baseado em componentes inclui fases como:

- Separação de contextos a partir do modelo de domínios;
- Particionamento do sistema em unidades independentes (componentes);
- Identificação do comportamento interno dos componentes;
- Identificação das interfaces dos componentes;
- Definição de um kit de arquitetura, que inclua princípios e elementos que facilitem a conexão de componentes;
- Manutenção de um repositório de componentes.

3. A Robocup

RoboCup é uma competição mundial que atua na construção de um time de futebol de robôs e envolve a integração de diversas tecnologias tais como: projeto de agentes autônomos, cooperação em Sistemas Multi-Agentes, estratégias de aquisição de conhecimento, engenharia de sistemas de tempo real, sistemas distribuídos, reconhecimento de padrões, aprendizagem, controle de processos, entre outros (ROBOCUP, s.d.).

Utilizando o futebol de robôs como tema central, o projeto almeja inovações que possam ser utilizadas para resolverem problemas sociais e relacionados à indústria. Seu objetivo final é desenvolver, até o ano de 2050, um time de futebol composto por robôs humanóides, completamente autônomos, que possam vencer a equipe campeã mundial do período (ROBOCUP, s.d.).

3.1. Histórico

Segundo RoboCup (s.d. *apud* RIOS, 2006), a primeira vez que foi citada a possibilidade de se fazer pesquisas e promover a ciência e tecnologia usando futebol com robôs, foi em 1992 pelo professor Alan Mackworth, mas de forma independente, uma equipe de pesquisadores do Japão, que participavam de um *WorkShop* sobre Inteligência Artificial em Tókio, levantaram sérias discussões a este respeito. Um estudo foi feito para analisar o impacto social, tecnológico e financeiro que esta pesquisa poderia causar, e o resultado deste estudo foi que isso seria não apenas viável, mas também desejável.

Em Junho do ano seguinte, foi disputada então a Robot J-League, que quer dizer Liga Profissional Japonesa de Futebol de Robôs. Este campeonato despertou o interesse de pesquisadores do mundo inteiro, que começaram a participar e por esse motivo o nome do torneio passou a se chamar RoboCup que significa Robot World Cup Initiative, ou seja, Iniciativa da Copa do Mundo de Robôs. O futebol de robôs passou a ser um dos principais domínios de estudo, e em 1993 no ETL (ElectroTechnical Laboratory), um centro de pesquisas do governo japonês, Itsuki Noda, escreveu um simulador dedicado de partida de futebol, e que mais tarde, com a colaboração de outros desenvolvedores, veio a se tornar o servidor oficial da RoboCup.

Em 1996 durante uma Conferência Internacional sobre Inteligência Artificial foi anunciado a primeira Copa do Mundo de Futebol de Robôs, que dariam aos competidores um tempo de aproximadamente de dois anos para que eles pudessem construir seus times e

compreender as dificuldades da construção dos robôs e dos times simulados.

Um evento conhecido como pré-RoboCup aconteceu em 1996 e contou com a participação de oito equipes na categoria de futebol simulado e demonstração de Robôs reais de tamanho pequeno e médio, para as categorias SmallSize e MiddleSze, respectivamente. Então em 1997 aconteceu a primeira Copa Mundial Oficial de Futebol de Robô, que contou com a participação de mais de 40 (quarenta) times, entre simulados e reais, e mais de 5000 (cinco mil) torcedores.

3.2 Categorias

As competições da Robocup são divididas em três grupos (*RobocupSoccer*, *RobocupRescue* e *RobocupJunior*), sendo cada um composto por várias ligas (ROBOCUP, s.d.).

3.2.1 RobocupSoccer

A *RobocupSoccer* foi, inicialmente, a grande motivação para a *RoboCup*. O futebol, além de ser um esporte bastante popular é também um jogo com problemas individuais e coletivos. Essa liga é subdividida em competições com agentes físicos e simulados (ROBOCUP, s.d.).

- ***Middle size league***

Jogada por duas equipes de cinco robôs móveis autônomos com rodas com uma altura que varia entre os 50cm e os 90cm. Toda a informação relativa ao ambiente é obtida pelos seus sensores e são capazes de comunicar-se entre eles via rede *wireless*. Tirando a introdução e remoção de robôs do terreno de jogo e recolocação da bola em campo, não é permitida qualquer intervenção humana durante o jogo (ROBOCUP, s.d.).

- ***Small size league***

Nesta liga os robôs são bem menores e o ambiente é muito mais controlado. Embora sejam independentes uns dos outros, todo o processamento é efetuado num computador central, baseado em informação (e.g. posição dos jogadores, da bola, etc.) vinda de uma câmera de vídeo colocada por cima do campo de jogo, sendo que os comandos são enviados para os robôs via *wireless*. A intervenção humana resume-se à introdução e remoção dos robôs no campo (ROBOCUP, s.d.).

- ***Four-legged league***

Como o nome da liga indica, esta é jogada por robôs de quatro patas. São equipes de cinco robôs, que assentam na plataforma desenvolvida pela Sony, o cão robótico AIBO, programado pelas equipes de forma a obter os resultados pretendidos. Estes robôs tem os sensores necessários incorporados e comunicam-se com os restantes elementos via *wireless* (ROBOCUP, s.d.).

- ***Simulation league***

Nesta liga não existem robôs/agentes físicos, em todo o ambiente os agentes são simulados. Um simulador fornece aos agentes todos os dados que seriam obtidos na realidade através dos seus sensores e calcula o resultado das ações de cada agente. Cada agente é um processo separado e são no total 11 de cada equipe. Existe simulação tanto 2D quanto 3D (ROBOCUP, s.d.).

- ***Humanoid league***

Esta competição baseia-se ainda em aspectos básicos para jogar futebol, não existindo atualmente jogos de futebol como nas outras ligas. É a liga mais atrativa, mas também é aquela que apresenta os maiores desafios e com mais complexidade. A intervenção humana nesta liga é permitida, já que em muitos casos, os robôs são tele-operados (ROBOCUP, s.d.).

- ***E-league***

Esta é a mais recente liga do Robocup e foi introduzida para permitir que equipes com menos recursos ou experiência também possam participar. Centra-se nos problemas de alto-nível, usando plataformas simples e tecnologias generalizadas quanto à visão e comunicação (ROBOCUP, s.d.).

3.2.2 RobocupRescue

A *RobocupRescue* consiste na utilização de robôs para busca e salvamento em cenários de catástrofes artificiais e é subdividido em agentes físicos e simulados (ROBOCUP, s.d.).

- ***Robot league***

Ocorre quando um ou vários robôs movem-se dentro de uma zona de testes que recria um edifício parcialmente destruído onde existem vítimas que esperam ajuda exterior, com zonas de diferentes níveis de dificuldade. O objetivo é procurar as vítimas e construir um mapa do ambiente e transmiti-lo ou trazê-lo até a equipe humana no exterior. Os robôs podem ser autônomos ou operados remotamente (ROBOCUP, s.d.).

- ***Simulation league***

Neste ambiente, a catástrofe é mais generalizada. Um conjunto de simuladores simula as consequências de um sismo numa cidade. Edifícios desabam, estradas são obstruídas, incêndios são originados e começam a propagar-se, etc. Os agentes desenvolvidos pelas equipes irão atuar neste ambiente. Três conjuntos de agentes (bombeiros, polícias e ambulâncias, com as respectivas centrais) irão fazer tudo para salvar o maior número possível de vítimas e evitar que os incêndios existentes se propaguem e consumam toda a cidade, minimizando os estragos na cidade. Este ambiente obriga as equipes a pesquisar e apresentar soluções em diversas áreas (planejamento multi-agente, coordenação de agentes heterogêneos**,** planejamento robusto, etc.) (ROBOCUP, s.d.).

3.2.3 RobocupJunior

Por fim, a *RobocupJunior* visa iniciar jovens estudantes do ensino primário e secundário nas áreas de robótica e inteligência artificial. Esse grupo é composto por diversas competições que procuram promover as capacidades dos participantes em trabalhar e resolver problemas para atingir um objetivo comum (ROBOCUP, s.d.).

- ***Soccer Challenge***

São formadas equipes de dois robôs e estas competem num pequeno campo de futebol pintado em tons de cinzento, sendo mais escuro de um lado, evoluindo gradualmente para mais claro no outro lado. Esta variação da cor do chão serve para que os robôs possam definir sua posição no ambiente. A bola contém no seu interior um transmissor, o que facilita a sua detecção por parte dos robôs (ROBOCUP, s.d.).

- ***Rescue Challenge***

Pequenos robôs correm para salvar vítimas em cenários de desastres artificiais. Pode ser simplesmente seguir uma linha desenhada ou procurar caminhos através de obstáculos no terreno (ROBOCUP, s.d.).

- ***Dance Challenge***

Um ou vários robôs apresentam uma dança ao som de uma música, tendo particular relevância nesta competição o movimento, coreografia e aparência visual dos robôs por si (ROBOCUP, s.d.).

3.2.4 Robocup@Home

RoboCup@Home tem seu foco principal nas aplicações em mundo real e na interação homem - máquina com robôs autônomos. O objetivo é promover o desenvolvimento de robôs que irão auxiliar os humanos no seu dia-a-dia. O cenário envolve a casa e por isso é dado ao participante ambientes que envolvem a cozinha, a sala de estar etc. Desse modo, o concorrente procura demonstrar as habilidades dos robôs deles neste ambiente. A primeira demonstração foi realizada em 2007 (ROBOCUP, s.d.).

3.3 O futebol de robôs simulado

O futebol de robôs é um sistema que permite agentes autônomos, feitos por variadas linguagem de programação, jogarem uma partida de futebol virtual uns contra os outros. Essa simulação é viabilizada pelo uso de dois programas, um monitor (*soccermonitor*) que exibe a partida um campo virtual com medidas de 105 metros de comprimento por 68 metros de largura e um servidor (*soccerserver*) que administra a comunicação com os agentes assim como simula as regras da partida, movimentos da bola e jogadores (CHEN *et al.*, 2003).

Um terceiro programa também utilizado é o *logplayer*, esse pode exibir uma partida através registro de atividades gravado em cada jogo (CHEN *et al.*, 2003).

3.3.1 O Monitor

O *soccermonitor* é uma ferramenta para exibir o que esta ocorrendo no servidor durante as partidas em tempo real. Através dele é possível visualizar o estado da partida, a posição dos jogadores e da bola, a pontuação de cada time e seus respectivos nomes. Outras informações sobre os jogadores podem ser ativadas, tais como exibir o cone de visão, suas velocidades e a energia restante. O monitor também pode ser utilizado como uma interface de interação com o *soccerserver*, podendo enviar comandos para dar inicio à partida ou alterar o estado do jogo (CHEN *et al.*, 2003). A Figura 3 mostra uma partida vista através do *soccermonitor*, onde jogadores então posicionados aguardando o início do jogo.

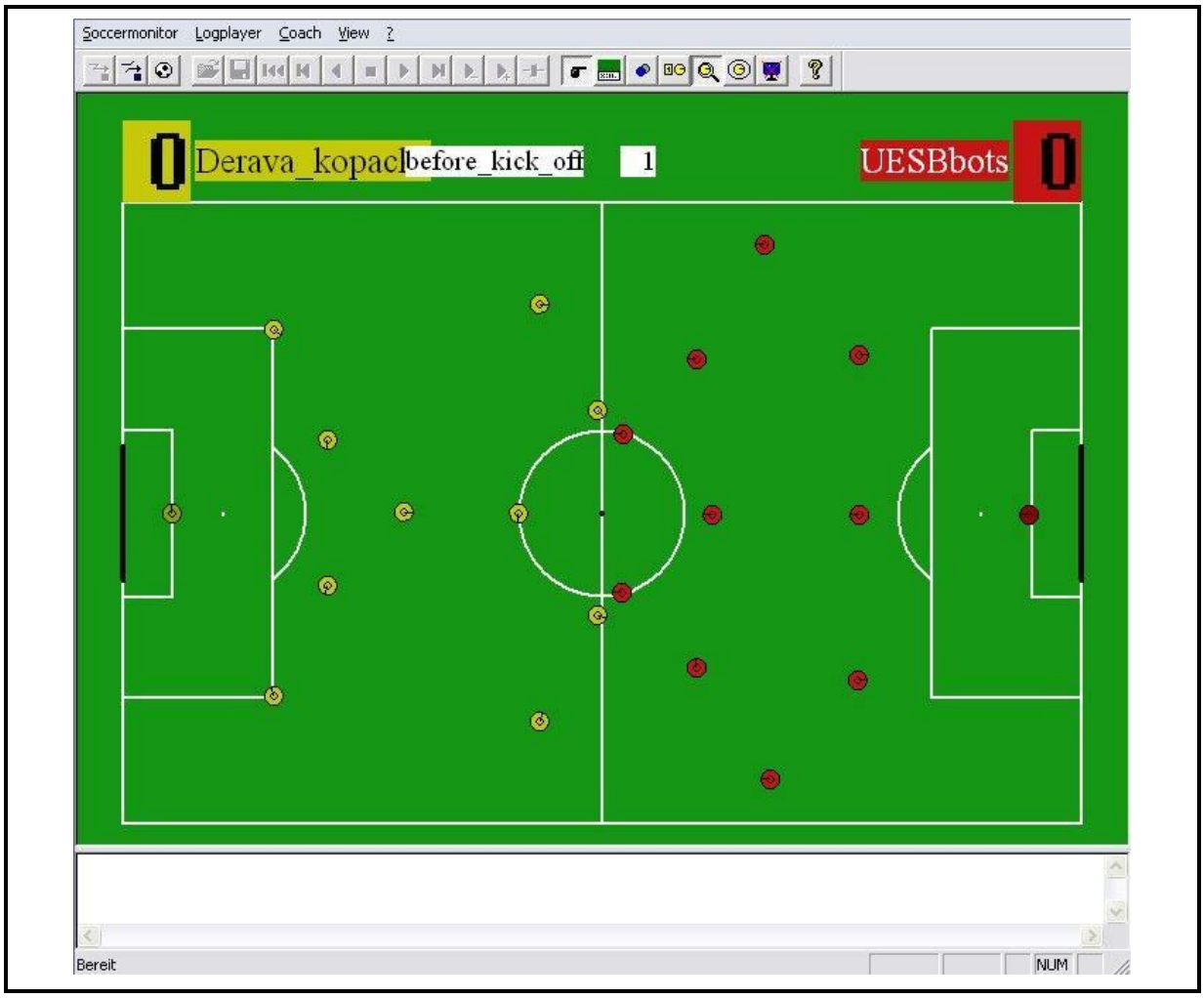

Figura 3: SoccerMonitor

A comunicação entre servidor e monitor é feito através de mensagens User Datagram Protocol (UDP) e seus módulos são executados em processos distintos possibilitando que diversos monitores sejam conectados ao mesmo servidor. Dessa forma em cada processo a partida poderia ser exibida com ampliações em locais diferentes ou expondo mais informações de um determinado jogador. Também é possível que o monitor seja executado em qualquer outro computador que possa acessar o *soccerserver* através de uma conexão de rede (CHEN *et al.*, 2003).

Com a variedade de informações que o *soccermonitor* provê, este se tornou uma ferramenta de grande utilidade para os desenvolvedores de times.

3.3.2 O Servidor

O servidor é um sistema que permite Multi-Agentes competirem em um jogo de futebol através da simulação de todos os movimentos da bola e jogadores em um campo

virtual. É baseado no estilo cliente-servidor, e não impõe qualquer restrição na arquitetura dos times, basta que a comunicação entre cliente-servidor tenha suporte a UDP e obedeça aos protocolos de comunicação utilizados (CHEN *et al.*, 2003).

As regras de formação das equipes seguem as regras do futebol, onde até doze agentes compõem cada time, sendo que um é o treinador (coach). Cada cliente é um processo distinto e controla os movimentos de um agente. Os jogadores podem enviar comandos para o servidor requisitando ações que eles desejam tomar (e.g. chutar a bola, virar-se, correr, etc.). Então o servidor recebe essas mensagens, interpreta as requisições, atualiza o ambiente de acordo e depois retorna informação sensorial (e.g. informação visual da posição dos objetos no campo ou informação sobre a velocidade e vigor dos jogadores). A comunicação entre os agentes pode ser feita somente através de informações auditivas com os comandos SAY e HEAR. Em outras palavras, os clientes que representam os jogadores não podem trocar informações diretamente (CHEN *et al.*, 2003).

É importante mencionar que o servidor é um sistema em tempo real que trabalha com intervalos discretos (ciclos). Cada ciclo tem uma duração específica e ações precisam ser efetuadas em um dado ciclo devem chegar até o servidor no intervalo certo. Caso contrário, o jogador perderá oportunidades de jogadas e causará uma baixa no desempenho para todo o time (CHEN *et al.*, 2003).

Percepções sensoriais

As percepções sensoriais são divididas em informação visual, auditiva e corporal, cada uma tem o seu respectivo formato de mensagem.

Informação Visual

O campo virtual utilizado pelo servidor possui as dimensões de 105x68m e contem objetos estáticos que servem para ajudar o jogador a se localizar no campo. As informações visuais desses objetos são enviadas para os jogadores a cada 150ms e chegam com formato abaixo (CHEN *et al.*, 2003):

> (see Time ObjName Distance Direction DistChange DirChange BodyDir HeadDir)
> Onde:
> ObjName = (p "TeamName" UniformNumber [goalie]) ou objetos como bandeiras e linhas do campo;
> *TeamName* nome da equipe em formato de texto;

UniformNumber número do uniforme que varia de 1 a 11;

[goalie] aparece quando o jogador é um goleiro;

Time é a simulação de ciclos do soccerserver;

Distance distância do jogador até o objeto observado;

Direction ângulo formado com o jogador varia de -180 a 180 graus;

DistChange mudança de distância;

DirChange mudança de direção;

HeadFaceDir direção da cabeça do jogador -180 a180 graus;

BodyFaceDir direção do corpo -180 a 180 graus.

DistChange e DirChange representam a mudança da distância e direção desde a ultima visão do objeto, se o objeto é algo imóvel como gol, bandeiras ou linhas, esses valores serão omitidos (CHEN *et al.*, 2003).

No campo virtual do soccerserver, visto na Figura 4, existem objetos posicionados em diversos locais e são utilizados como referencia para que o agente possa se localizar no campo. Esses objetos são classificados como bandeiras, linhas e gol (CHEN *et al.*, 2003).

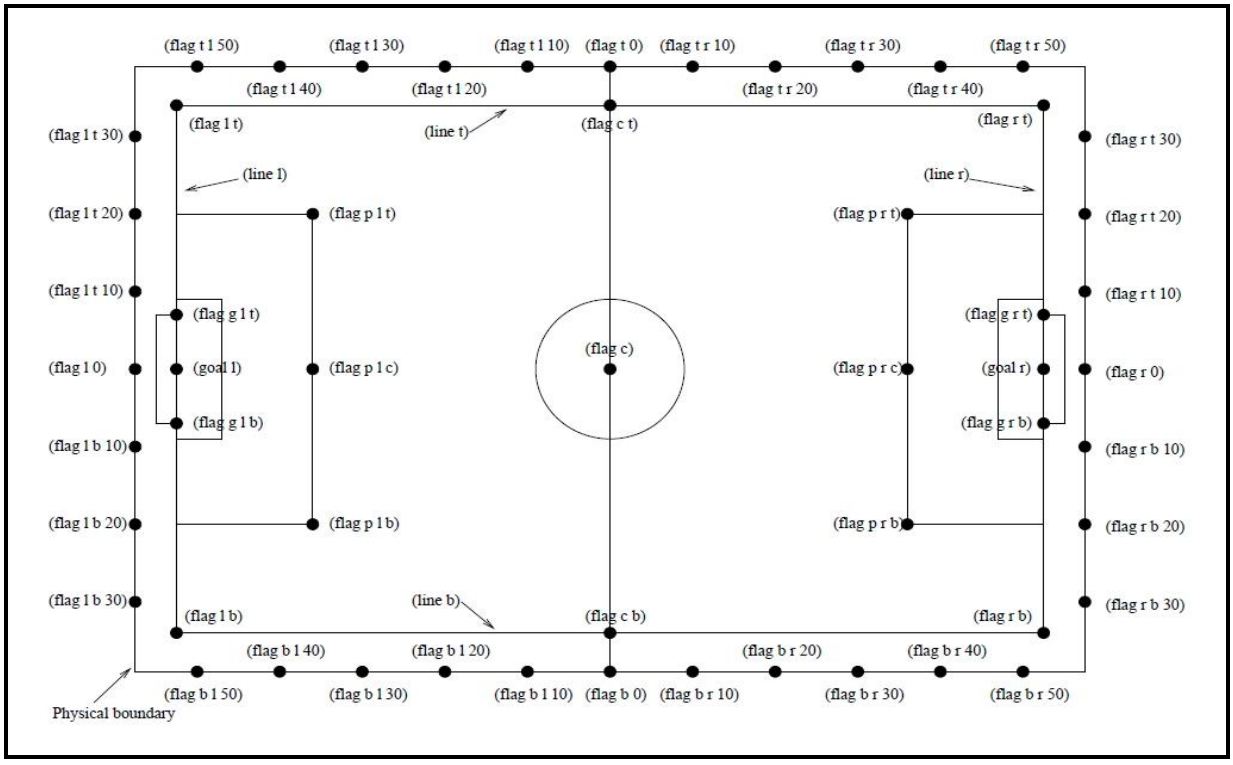

Figura 4: Objetos delimitadores do campo virtual (CHEN *et al.*, 2003)

O objeto estático *"flag c"* é uma bandeira que serve para representar o centro do campo, como indicado pela letra *"c"*. As linhas laterais e de fundo de cada gol é representada

pelo objeto "*line l|r|t|b"*. As bandeiras "*goal l|r"* representam os gols de cada lada, esquerdo e direito respectivamente. As bandeiras *"flag r|l t|b"* equivalem às bandeiras de escanteio de um campo real. Enquanto que "*flag p l|r t|b"* representam as linhas que demarcam a área de pênaltis ou em comparação com o campo real, com a *Grande área*, como é popularmente conhecida. As bandeiras *"flag g l|r t|b"* são associadas às traves do gol de cada lado do campo. As outras bandeiras estão localizadas a 5 metros fora do campo, como por exemplo, a bandeira "*flag t l 30"* que está localizada a 5 metros da linha superior *(line t)* e a 30 metros da linha central (CHEN *et al.*, 2003 *apud* RIOS, 2006).

A informação visual enviada para os agentes corresponde aos objetos identificados pelo robô, no setor visível do jogador. Esse setor visível pode assumir três diferentes ângulos: normal [-45, 45], amplo [-90, 90], direcionado [-22.5, 22.5]. Cada um dos ângulos de visão pode assumir o valor alto ou baixo para a qualidade da informação visual (CORTEM *et al.*, 1999 *apud* COSTA & BITTENCOURT, 1999).

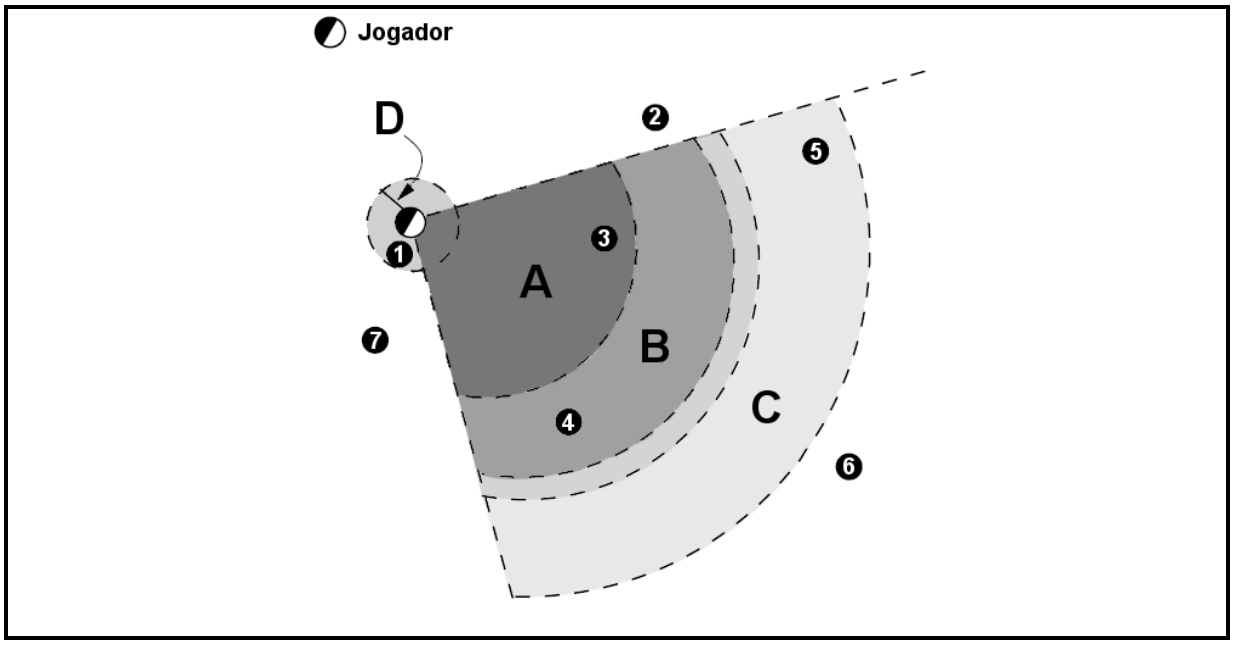

Figura 5: Cone de visão do agente (adaptado de CHEN *et al.*, 2003)

Na Figura 5 os objetos correspondentes aos números 2 e 7 são os únicos fora da visão do jogador, os demais se encontram em situações de exposição diferentes e, a depender do quão longe estão, a precisão na identificação é afetada. De acordo com Chen *et al.* (2003) os valores de distancias são:

- Área "A" - Corresponde a 20 metros de distância do jogador, dentro desse distancia é possível identificar tanto o número do uniforme como o nome da equipe de outro

jogador;

- Área “B” - Corresponde a 40 metros de distância do jogador, até essa distância é possível identificar o número do uniforme de outro jogador com uma probabilidade de 50%, acima desse valor o número do uniforme não é visível;
- Interseção entre a área “B” e “C” - Ainda é possível identificar o nome da equipe de um jogador;
- Área “C” - corresponde a 60 metros de distância do jogador, até essa distância é possível identificar o nome da equipe de outro jogador com uma probabilidade de 50%, acima desse valor o nome da equipe não é visível;
- Área “D” - Corresponde a um raio de 3 metros do centro do jogador, tal que quando o objeto nela se encontra, é identificado mesmo estando fora do setor visível.

Informação Auditiva

Alem da informação visual, o agente recebe do servidor, mensagens enviadas pelo juiz informando alterações no estado do jogo (e.g. ocorrência de um gol) que não tem limite de distancia e mensagens enviadas por outros jogadores em até 50 metros de raio. O jogador poderá processar apenas uma mensagem de cada time a cada dois ciclos de simulação. Essas mensagens são ordenadas de acordo com a ordem de chegada. Existe ainda uma limitação quanto ao comprimento da mensagem, que não poderá ultrapassar 512 caracteres (CORTEM *et al.*, 1999 *apud* COSTA & BITTENCOURT, 1999).

O formato da mensagem auditiva é:

(hear Time Sender "Message")

Time é a simulação de ciclos do soccerserver;
Sender pode ser uma direção quando a mensagem foi originada de outro jogador ou então:
self quando é do próprio jogador;
referee quando vem do juiz do jogo;
online_coach_left/right quando vem de um dos técnicos dos times;
Message é a mensagem recebida no formato de texto.

Uma das intenções do soccerserver é a evolução dos Sistemas Multi-Agentes, e a eficiência da comunicação entre os agentes é um dos critérios principais. Os usuários devem

realizar o controle dos múltiplos agentes com restrição na comunicação (CHEN *et al.*, 2003).

Informação Corporal

O sensor de corpo reporta o atual estado físico do corpo de um jogador. Essas informações são enviadas automaticamente a cada 100 milisegundos (ms) e são úteis para o agente tomar decisões de acordo com os dados relativos ao seu corpo ou determinar se um comando foi aceito ou não pelo servidor (CHEN *et al.*, 2003).

O formato da mensagem corporal é:

(sense_body Time (view_mode ViewQuality ViewWidth) (stamina Stamina Effort) (speed AmountOfSpeed DirectionOfSpeed) (head angle HeadAngle) (kick KickCount) (dash DashCount) (turn TurnCount) (say SayCount) (turn neck TurnNeckCount) (catch CatchCount) (move MoveCount) (change_view ChangeViewCount))

Time é a simulação de ciclos do soccerserver;
ViewQuality pode ser *high* ou *low*;
ViewWidth pode ser *narrow*, *normal*, ou *wide*;
Stamina representa a resistência do jogador, varia de 0 a 4000 por padrão;
Effort representa o esforço feito pelo jogador;
AmountOfSpeed é a soma da velocidade do jogador;
DirectionOfSpeed é a direção para onde o se movimenta, varia entre -180 a 180 graus;
HeadAngle é a direção relativa da cabeça do jogador, varia entre -180 a 180 graus;

As demais variáveis representam a contagem da quantidade de vezes que cada respectivo comando que foi executado pelo servidor.

Estados e Regras da partida

O juiz da partida é o responsável por atribuir estados e regras de controle de acordo com as situações que o jogo se encontra, essas atribuições serão interpretadas pelos agentes e utilizadas para selecionar as estratégias de jogo que sejam mais adequadas. Para Chen *et al.* (2003) os possíveis estados da partida são definidos por:

- *before_kick_off* - A partida encontra-se parada aguardando seu início. Este estado acontece no início do jogo antes do botão *kick_off* do SoccerServer ser acionado ou durante os cinco segundos de intervalo dados pelo juiz após a ocorrência de um gol.

- *time_over* - Fim de jogo.
- *play_on* - Quando a bola está em jogo.
- *kick_off_l* ou *kick_off_r* - Saída de bola para o time da esquerda e da direita.
- *kick_in_l* ou *kick_in-r* - Reposição de bola em jogo para o time da esquerda ou para o da direita.
- *free_kick_l* ou *free_kick_r* - Chute livre para o time da esquerda ou para o da direita.
- *corner_kick_l* ou *corner_kick_r* - Cobrança de escanteio para o time da esquerda ou para o da direita.
- *goal_kick_l* ou *goal_kick_r* - Cobrança de tiro de meta para o time da esquerda e da direita.
- *goal_l* ou *goal_r* - Ocorrência de um gol em favor do time da esquerda ou para da direita.
- *offside_l ou offside_r* - Todos os jogadores atacantes são posicionados num local em que não estejam fora de jogo. São considerados atacantes os jogadores que estão fora de jogo e os que estão a menos de 9.15 metros da bola.

Alem dos estados da partida os agente jogadores devem conhecer algumas regras de controle que segundo Chen *et al*. (2003) são descritas como:

- *Gol* - Quando a bola se encontra dentro do gol, o juiz difunde para todos os agentes uma mensagem informando o acontecimento de um gol. Além disso, atualiza o placar, suspende a execução da partida por cinco segundos, move a bola para o centro do campo e muda o estado do jogo para *kick_off*, reiniciando a partida. Durante os cinco segundos de interrupção da partida todos os agentes devem voltar para seus lados do campo.
- *Saída de Bola* - Durante a saída de bola, inicio ou reinicio do jogo, todos os jogadores (agentes) devem estar em seu campo defensivo. Caso um jogador se encontre no campo adversário, este será movido para uma posição aleatoriamente escolhida em seu campo defensivo.
- *Bola Fora de Jogo* - Quando a bola se encontra fora das dimensões do campo, o juiz move a bola para a posição apropriada (e.g. lateral do campo, marca de escanteio ou pequena área) e muda o estado do jogo para *kick_in* (reposição de bola), *corner_kick* (escanteio) ou *goal_kick* (tiro de meta).
- *Desobstrução* - Apos o goleiro segurar a bola, ou quando a partida se encontra em um

dos seguintes modos, *kick_off, corner_kick, goal_kick* ou *offside*, o juiz remove qualquer jogador adversário raio de 9.15 metros da bola para o perímetro do circulo de mesmo raio centrado na bola.

- *Controle do modo de jogo* - Quando o jogo se encontra em um dos seguintes estados *kick_off, corner_kick* ou *gool_kick*, o juiz muda o estado do jogo para *play_on* apos a bola ter sido colocada em jogo, por um dos jogadores do time que detém a posse da bola.
- *Final de Primeiro Tempo* e *Fim de Jogo* - O juiz suspende o jogo quando o relógio do soccerserver atingir 3000 ciclos de simulação (5 minutos) decretando o final do primeiro tempo. Difundirá então uma mensagem informando o novo status do jogo. De forma similar, o juiz terminara o jogo ao atingir 6000 ciclos de simulação.

Ações executadas pelos agentes (Comandos)

Em resposta as informações visuais e auditivas, recebidas pelo agente, devem ser enviadas ao Soccer Server comandos que serão responsáveis pela ação do jogador. O Soccer Server aceita um novo comando a cada 20ms, entretanto existem limitações para utilização destes comandos (CHEN *et al.*, 2003).

De acordo com Chen *et al.* (2003) os comandos disponíveis e suas respectivas limitações são:

- (*turn moment*) - Permite que o jogador execute um giro de [-180, 180], graus em torno de seu próprio eixo. A este comando é associado um argumento *moment* (i.e. momento de inércia) que considera a inércia do objeto;
- (*turn_neck angulo*) - Permite que o jogador gire [-180, 180] graus em torno de seu próprio eixo, a parte superior do robô onde se encontra a câmera;
- (*dash potência*) - incrementa a velocidade do jogador na direção atual com a potência especificada no argumento. A potência pode assumir um valor inteiro no intervalo [-100, 100];
- (*kick potência direção*) - Chuta a bola com a potência de -100 a 100 na direção -180 a 180 graus. Este comando só terá efeito se a bola estiver na margem de chute especificada pelo Soccer Server;
- (*move X Y*) - Movimenta o jogador para a posição determinada. Os valores de X variam de -52.5 a 52.5 e os de Y variam de -34 a 34. Este comando é somente

utilizado nos modos de partida *before_kick_off*, *goal_r* e *goal_l*;

- (*catch direção*) - Tenta agarrar a bola na direção especificada. A possibilidade de sucesso é definida no parâmetro do Soccer Server *catch_possibility*. A bola deve se encontrar em uma área definida como *goalie_cacthable_area* um retângulo de 2x1 metros. Esse comando somente poderá ser utilizado pelo goleiro;
- (*say mensagem*) - Difunde uma mensagem para todos os jogadores num raio de 50 metros;
- (*change_view ângulo qualidade*) - Modifica o Angulo do setor visível para a *normal* [-45, 45], *narrow* [22.5, 22.5], *wide* [-90, -90] e a qualidade da informação visual para *low ou high*;
- (*sense_body*) – O servidor retorna valores do estado físico atual do jogador, como os valores do modo de visão e o vigor do jogador;
- (*score*) - O servidor retorna a pontuação dos dois times.

Também existem comandos que o agente utiliza para iniciar a conexão com o servidor, isto é, avisar qual o nome do time e quando entrou em campo. Esses comandos são:

- (*init nomeTime [(versãoSimulador)] [(goalie)]*) - O primeiro comando que deve ser enviado para iniciar a conexão com o simulador, é necessário informar o nome do time e caso o jogador seja o goleiro;
- (*reconnect nomeTime númeroJogador*) – Este comando é útil para mudar o cliente em campo para o determinando numero do jogador;
- (*bye*) – Esse comando faz com que o simulador retire o jogador de campo e não mais aceite comandos do mesmo.

Chen *et al.* (2003) afirma que o soccerserver aceita um novo comando a cada 20ms, entretanto apenas um dos comandos *turn, turn_neck, kick, move, dash, catch,* é executado a cada ciclo de simulação, que é de 100ms.

3.3.3 O Logplayer

É uma ferramenta que foi adicionada na terceira versão do simulador lançado em Fevereiro de 1997. Sua finalidade é reproduzir partidas que foram previamente salvas em arquivos de extensão “rcg”. O principal objetivo da utilização desta ferramenta é poder

compreender e analisar estratégias montadas pelas equipes (CHEN *et al.*, 2003 *apud* RIOS, 2006). Os registros de varias partidas disputadas pela RoboCup podem ser encontrado na internet.

4. Trabalhos Relacionados

Nesta seção serão expostas as características e metodologias de alguns times utilizados como fonte de pesquisa para o desenvolvimento deste trabalho monográfico.

4.1 MecaTeam

O MecaTeam é o time de futebol de robôs simulados criado na Universidade Federal da Bahia (UFBA). Ele apresenta uma evolução na arquitetura do agente UFSC-Team-98 e utiliza a biblioteca orientada a objetos Expert-Coop++ como um Sistema Baseado em Conhecimento que auxilia no planejamento cognitivo de agente. O time MecaTeam 2006 conseguiu o segundo lugar no campeonato brasileiro da liga simulada do mesmo ano.

4.1.1 Metodologia

Segundo Teixeira *et al.* (2008), a tomada de decisão na arquitetura do agente do MecaTeam baseia-se num modelo híbrido de agente que compartilha a rápida resposta aos estímulos do ambiente dos agentes reativos com um planejamento bem elaborado apresentado pelos agentes cognitivos. Apresenta ainda uma arquitetura em camadas que explora o paradigma de programação concorrente.

O raciocínio reativo do agente e implementado num Sistema Baseado em Regras de Produção, que é composto por um analisador léxico e sintático de regras para o Agente MecaTeam. Este analisador é responsável por ler o arquivo de regras em modo texto, extrair estas regras e colocá-las em estruturas de dados, que são manipuladas pelo Sistema Baseado em Conhecimento.

A biblioteca orientada a objetos Expert-Coop++ é encarregada de dar suporte a inteligência do agente MecaTeam 2006. Ela apresenta uma arquitetura de três módulos:

- A base de regras que é responsável pela leitura do arquivo de regras, previamente definido pela extração destas regras e pelo seu armazenamento em uma lista e contém as reações aos possíveis estados do ambiente;
- O motor de inferência que é o principal componente do sistema, ele é o mecanismo de controle do sistema que avalia e aplica as regras de acordo com as informações na base de fatos;

- A base de fatos que contém uma descrição dos estados do ambiente. Ela é construída, tendo árvores binárias como base, onde cada "nó" possui uma representação de conhecimento.

O MecaTeam 2009 possui o Nível cognitivo simples, implementado utilizando programação *multi-threading* juntamente com técnicas de sincronização para garantir um bom funcionamento.

4.2 Pet Soccer 2D

O Pet Soccer 2D foi desenvolvido pelos alunos de Engenharia de Computação da Universidade Federal do Espírito Santo (UFES). O time conseguiu o terceiro lugar no campeonato brasileiro da liga simulada de 2006.

4.2.1 Metodologia

Segundo Santos *et al*. (2009), o time Pet Soccer 2D de 2009 implementa o uso de lógica fuzzy para diferente tipos de habilidades dos jogadores. Como a classificação das áreas de atuação dos jogadores ou definição do grau de risco que um atacante pode submeter a um zagueiro.

Na categoria RoboCup Soccer Simulation 2D, cada time recebe diferentes tipos de jogadores, variando basicamente em: aceleração máxima, velocidade máxima, área de chute, taxa de descanso e erro do chute, etc. Uma boa escolha das áreas de atuação dos jogadores, de acordo com as suas funções, pode influenciar no resultado do jogo. Sendo assim, cada jogador é submetido para avaliação por um conjunto de funções de calculo fuzzy e como resultado tem-se a qualidade do jogador. Assim, um jogador que é bom para ser atacante pode não ser um bom zagueiro ou lateral.

Para a marcação dos adversários foi utilizada a lógica fuzzy para o cálculo de alguns parâmetros envolvidos nesse problema. A primeira preocupação para a marcação dos jogadores é quem marcar. Durante o ataque do time adversário, o zagueiro terá que escolher qual dos jogadores adversários marcar. Para tomar essa decisão, o zagueiro avalia primeiramente o risco que cada jogador adversário oferece caso ele receba a bola. Para tanto é usado um módulo fuzzy que tem como entradas a distância e o ângulo do jogador adversário até o gol. A saída é um valor entre 0 a 100, sendo que o número zero significa que o jogador

adversário não oferece nenhum risco para o time e o número cem que o jogador adversário oferece risco máximo.

Sabendo do risco de cada jogador, o zagueiro calcula a distancia de marcação para o jogador. Essa distancia é calculada por outro módulo fuzzy, que tem como entradas a distancia e o angulo do jogador até a bola e o risco que o jogador oferece. Com a distancia de marcação a determinado oponente, o zagueiro verifica se já tem algum outro zagueiro marcando o oponente. Dentre os oponentes desmarcados, o zagueiro escolhe aquele que oferece o maior risco ao time e que o zagueiro esteja mais próximo do que outro zagueiro que não esteja marcando ninguém.

4.3 FEI CDU-2006

Este time utiliza regras predefinidas para determinar as estratégias do goleiro, dos defensores (zagueiros) e dos atacantes. E teve seu sistema implementado com a linguagem C++ em Linux. O time FEI CDU-2006 conseguiu o primeiro lugar no campeonato brasileiro da liga simulada de 2006.

4.3.1 Metodologia

Segundo Oliveira (2006), o time CDU-2006 é subdividido em três tipos básicos de jogadores: Goleiro, Zagueiro e Atacante, cujas funções se diferenciam devido à necessidade de especialização em cada setor do campo. Cada tipo de jogador segue sua própria estratégia e se comunicam através dos comandos específicos do simulador. Cada tipo de jogador reage de uma forma para cada situação do jogo, essa reação é definida pela estratégia, que são regras de comportamento.

O comportamento do goleiro limita-se a um eixo vertical posicionado a frente do gol, o goleiro deve mover-se nesse eixo sempre considerando a posição da bola. Este segue regras para quatro estados. O estado (E1) determina a posição do goleiro no início do jogo e o agente é posicionado no centro do gol. No estado (E2) o goleiro observa a bola e se posiciona estrategicamente. No estado (E3) é assumido que a bola esteja na área de defesa do time e o goleiro é posicionado na intersecção entre uma reta vertical obtida a partir do centro do goleiro e uma reta que traça a posição atual e anterior da bola. No estado (E4) o goleiro irá interceptar a bola e chutar para o companheiro sem marcação, caso a bola esteja muito próxima.

O defensor deve estar sempre rastreando a bola, os jogadores de seu time e do time adversário, porém sua atuação está restringida a área defensiva. Sua função se resume ao impedimento do time adversário chegar perto do goleiro e fazer gol. Ele segue regras para dois estados. No estado (E1) o defensor esta sem a bola e deverá a localizar para iniciar a interceptação quando o adversário vem em sentido ao seu gol. No estado (E2) quando em posse da bola, o defensor deve procurar pelas bandeiras ao seu redor e direcionar-se ao gol adversário com o objetivo de passar a bola para algum atacante, ou seguir com a bola quando impossibilitado de dar passe.

O agente atacante tem a movimentação limitada pela sua área de atuação, não podendo voltar para o campo de defesa. Sua habilidade é conduzir a bola até o gol adversário e interceptar a bola. Segue regras para quatro estados. No estado (E1) o atacante posiciona-se atrás da bola e modifica seu campo de visão para a direção do gol adversário. No estado (E2) ele leva a bola dando pequenos chutes em direção ao gol evitando adversários. No estado (E3) quando não é possível caminhar com a bola, o agente dar um passe para o companheiro visível mais próximo e desmarcado. No estado (E4) o atacante sem a bola tenta uma interceptação quando possível.

5. O Time de Futebol de Robôs da UESB

O presente capítulo será iniciado com uma breve contextualização sobre as características de desenvolvimento do agente, em seguida, serão apresentados os passos da modelagem dos agentes o modelo do mundo onde o agente está imerso e funcionamento dos sensores, atuadores e complementares, também será feita uma pequena abordagem sobre o mecanismo de inteligência.

5.1 Contextualização

Nesse projeto será apresentado o processo de modelagem de um protótipo base para um time de agentes jogadores com o objetivo de operar na liga de futebol simulado em duas dimensões da RoboCup, onde o ambiente é classificado como dinâmico, contínuo, inacessível e não determinístico. A prioridade foi desenvolver os mecanismos que correspondem à primeira camada da arquitetura, sendo esta responsável pela tradução dos dados e a manutenção do modelo de mundo. As camadas que utilizam analisadores inteligentes para solução dos problemas serão apenas descritas com o propósito de uma futura modelagem e implementação.

O plano de construção do projeto foi determinado por meio de um estudo sobre arquiteturas de agentes inteligentes, a arquitetura em camadas mostrou ser a mais adequada para os agentes que representam jogadores de um time de futebol, por ser uma arquitetura hibrida, ela pode ser composta por camadas reativa e cognitiva, onde problemas individuais do agente (e.g. chute a gol, interceptação da bola) que exigem respostas rápidas podem ser tratados pela camada reativa. Já problemas mais complexos tanto coletivos como individuais (e.g. estratégias de ataque e defesa, passes, dribles estratégicos, etc.) são avaliados detalhadamente pelas camadas cognitivas. Dessa maneira o agente pode agir em um ambiente de tempo real, onde agentes individuais buscam cooperar entre si para atingir um objetivo comum, enquanto agem autonomamente.

A metodologia para o desenvolvimento dos agentes será baseada em componentes por possuir maior flexibilidade para o estudo e manutenção dos códigos, também apresenta vantagens na organização dos objetos que compõem o agente, além de empregar métodos e técnicas que são boas práticas de engenharia de software. Os componentes referentes ao agente serão definidos de acordo com suas funcionalidades e necessidades de atuação para cada camada.

Para a implementação do agente jogador, foi escolhida a linguagem de programação de alto nível Python que dá suporte ao protocolo UDP, esse sendo o requisito básico para a construção de um time para a liga simulada, a linguagem também dá suporte a orientação a objetos e interfaces necessários para a criação dos componentes, bem como o suporte a *multi-threads.*

Python é uma linguagem interpretada de uso geral que pode ser empregada em vários tipos de problemas, possui uma licença livre aprovada pela OSI (*Open Source Initiative*) e compatível com a GPL (*General Public License*). O fato de Python ser uma linguagem interpretada possibilita que os códigos sejam feitos e testados com agilidade ou estudados no ambiente acadêmico por grupos com intenção de melhorar o código base do time, além disso, os módulos são pré compilados em *ByteCode* assim como na linguagem Java, dessa forma existe um ganho no desempenho. Também existe uma opção gerar executáveis a partir dos códigos feitos em Python, a fim de melhorar a velocidade de execução.

5.2 Modelagem

O agente jogador proposto terá uma arquitetura híbrida com atributos reativos e cognitivos que será dividida em camadas verticais. Essas camadas são representações abstratas para grupos de funcionalidades que no agente jogador foram divididas em componentes.

No esboço do modelo em camadas verticais, representado na Figura 6, é possível analisar o fluxo de passagem das informações, que começa com alguma percepção sensorial recebida. Em seguida ela é interpretada e refinada pela camada mais baixa, então ela é utilizada para atualizar o modelo de mundo e por fim determinar um comportamento correspondente. Feito isso, o fluxo de informações volta para a camada mais baixa e é convertido em comandos aceitáveis pelo simulador.

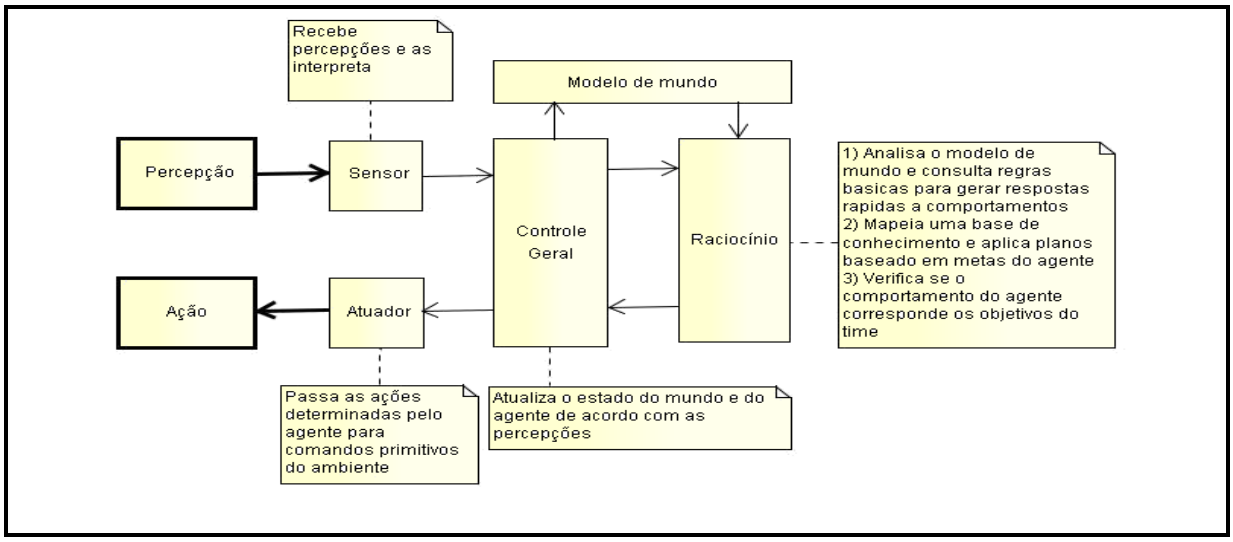

Figura 6: Arquitetura em camadas verticais de duas passagens do agente jogador

Uma das prioridades dessa arquitetura é possibilitar ao agente reavaliar constantemente e rapidamente suas tarefas no campo de futebol, adaptando-se às mudanças do mundo conforme necessário.

Para implementar essa arquitetura utilizando o modelo de desenvolvimento baseado em componentes, inicialmente foram especificados sete componentes encapsulados por outro denominado *Agente*. O relacionamento entre eles é feito por meio de interfaces onde são publicadas as funcionalidades que o mesmo pode realizar. Dessa forma, é sempre mantido o isolamento interno, a fim de facilitar a manutenção ou substituição.

A Figura 7 apresenta os componentes encapsulados do ponto de vista interno do *Agente*, onde são vistos como as partes que o compõem. No entanto, do ponto de vista externo, o *Agente* é visto como um único componente. Assim como os demais, ele também possui uma interface e relações de uso.

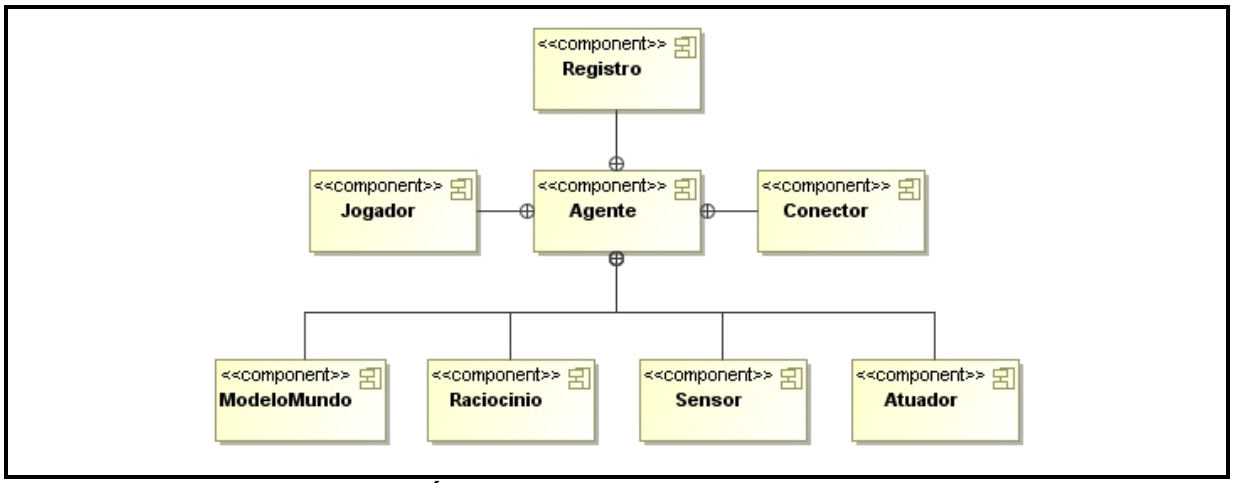

Figura 7: Árvore de Componentes do Agente Jogador

As funções publicadas em *IAgente* (Figura 8) são utilizadas para definir o endereço do servidor soccerserver configurando o componente *Conector* e dar inicio ao sistema do jogador. Quando o sistema é iniciado, o *Agente* cria um novo *Thread* (i.e. processamentos concorrentes em uma mesma aplicação) para o *Sensor*.

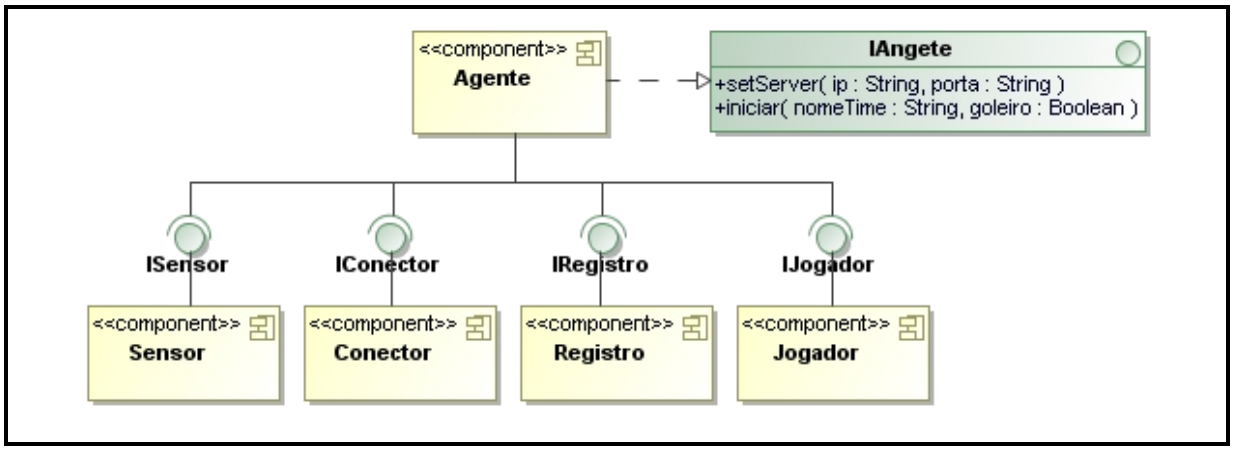

Figura 8: Modelagem do componente principal (Agente)

O fluxo de atividade dos *Threads* (Figura 9) é dividido em dois processos concorrentes. No primeiro o *Sensor* mantém a execução contínua de funções que recebem informações do simulador. Já o segundo corresponde à execução dos demais componentes referentes a tomada de decisões e execução de ações.

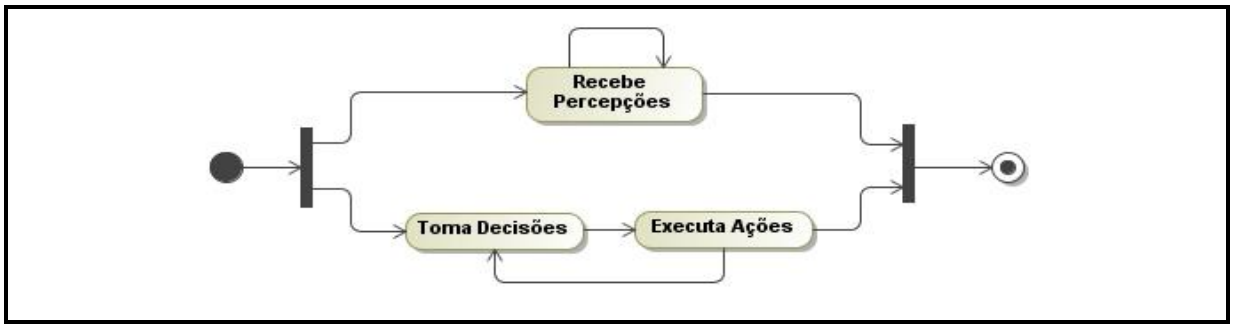

Figura 9: Fluxo de Threads do sistema

Para auxiliar na modelagem do agente foram utilizadas classes e enumerações, algumas delas são compartilhadas e podem ser aproveitadas pelos componentes internos ao Agente. Esses recursos são apresentados na Figura 10.

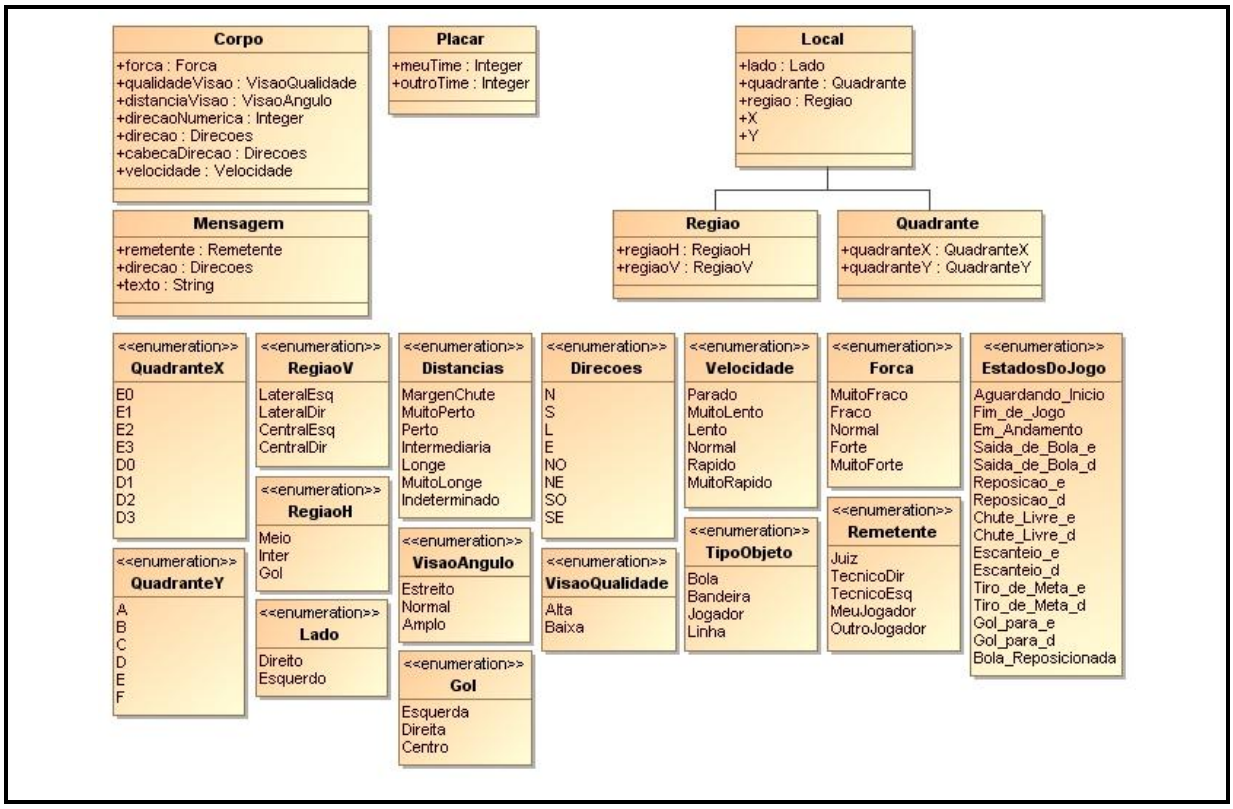

Figura 10: Classes e enumerações utilizadas pelo sistema

As enumerações são tipos de dados restritos aos valores definidos em seu modelo. Os parâmetros das funções publicadas nas interfaces dos componentes fazem uso desses tipos para uma melhor autonomia do agente, isto é, como as enumerações dão menos possibilidades de valores e evitando grandes quantidades de números, o mecanismo de raciocínio faz as escolhas mais rapidamente.

A atividade realizada pelo sistema interno do agente jogador requer que cada componente tenha suas responsabilidades. O *Conector* visto na Figura 11, realiza a conexão UDP com o soccerserver e serve de ponte para o *Sensor* e *Atuador*, sendo estes encarregados das traduções dos dados primitivos. A interface *IConector* conta com funções para enviar e receber dados, e também para dar inicio a conexão do jogador, informando ao simulador que time o agente faz parte e se ele representa o goleiro. As funções '*conectar'*, '*reconectar'*, *desconectar* utilizam os comandos primitivos '*init'*, '*reconnect'* e '*bye'*.

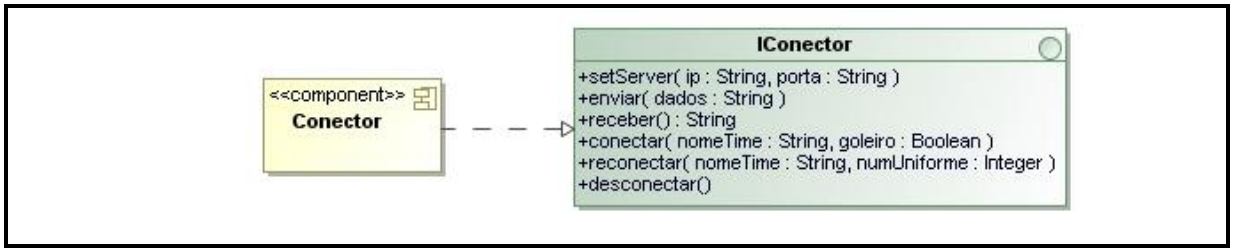

Figura 11: Modelagem do componente Conector

Embora o *Sensor* e *Atuador* estejam conectados ao servidor, eles são apenas os meios de contato do agente com o ambiente e não tem autonomia. Portanto, para fazer o controle desses componentes é necessário que exista um modulo principal capaz de unir os demais. Essa função foi atribuída ao componente *Jogador* (Figura 12).

Dentre as funcionalidades do *Jogador,* as mais relevantes são:

- Atualizar o estado do jogo mantido pelo *ModeloMundo*;
- Recorrer ao componente *Raciocínio* para tomada de decisões;
- Enviar as ações tomadas para do componente *Atuador*;
- Enviar requisições para o componente *Sensor*;
- Manter informações do jogo como o placar, nome do seu time, tempo de jogo em ciclos, mensagem do árbitro para os estados do jogo, etc.

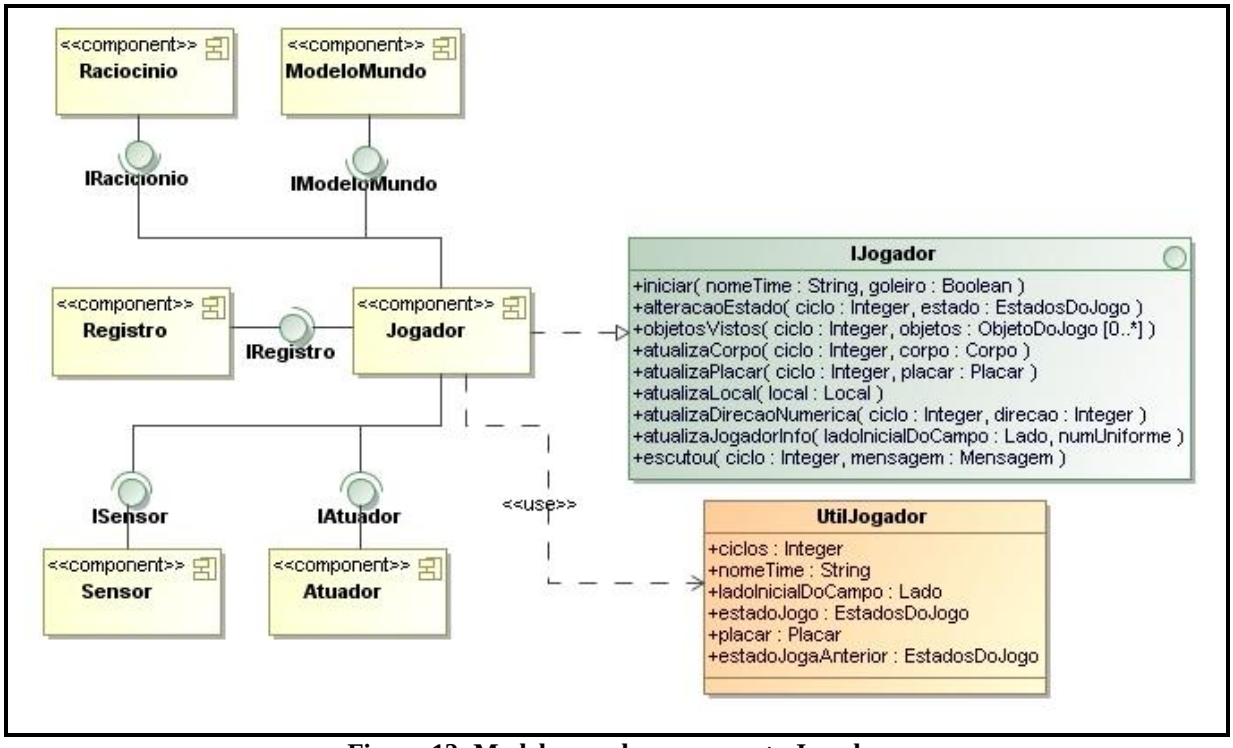

Figura 12: Modelagem do componente Jogador

A funções disponibilizadas na interface *IJogador* servem para atualizar as informações que o *Jogador* é encarregado de lidar. Algumas são de grande importância para o funcionamento do agente, como: a alteração de estados do jogo que envolve a mudança de comportamento de acordo com cada estado, ou os grupos de objetos vistos tendo efeito diretamente sobre o raciocínio. O uso dessa interface é normalmente feito pelo *Sensor* que converte os dados primitivos para o padrão aceito.

As características das funções são descritas abaixo:

- (*iniciar*) Executa funções iniciais do componente, em seguida armazena o nome do time e se o jogador tem o papel de goleiro;
- (*alteracaoEstado*) A alteração de estado é utilizada para informar ao jogador, o estado do jogo determinado pelo juiz da partida, como: iniciada, não iniciada, finalizada, cobrança de escanteio, etc. Está função armazena o estado atual no atributo *estadoJogo* e o passado em *estadoJogaAnterior*, ambos na classe *UtilJogador* (Figura 12);
- (*objetosVistos*) Recebe grupos de objetos visto pelo agente, ou objetos específicos que foram requisitados ao *Sensor*;
- (*atualizaCorpo*) Recebe os dados do corporais do agente listados na classe *Corpo* (Figura 10);
- (*atualizaPlacar*) Atualiza o placar do jogo, quando um gol é feito;
- (*atualizaLocal*) Atualiza o local atual do agente;
- (*atualizaDireçãoNumerica*) Atualiza a direção do corpo do jogador, que é necessária para execução de quase todos os comandos do *Atuador*;
- (*atualizaJogadorInfo*) Atualiza informação do jogador como o lado inicial que o time entrou em campo e o número que o jogador recebeu;
- (*escutou*) Recebe dados de comunicação dos agentes contendo uma mensagem de tamanho máximo de 512 caracteres, o seu remetente e sua direção em relação ao ouvinte caso seja um jogador.

As funções relacionadas a percepções enviam como parâmetro o ciclo que a mesma foi recebida para que esse valor seja salvo e utilizado como referência.

5.3 Modelo de Mundo

Estabelecido em um campo virtual com dimensões de 115 por 78 metros e caracterizado em um ambiente dinâmico, contínuo, inacessível e não determinístico. O modelo de mundo atua como uma memória para que o agente possa relembrar estados anteriores do jogo ou realizar de previsões de próximos estados. Assim sendo uma ferramenta essencial para agentes cognitivos decidirem que ações serão mais apropriadas em determinado estado do ambiente. No conjunto de componentes do agente o mundo é representado pelo

ModeloMundo. Esse permite que cada agente jogador possa construir internamente o seu próprio modelo do mundo se baseando em percepções visuais, auditivas e corporais.

Utilizando uma estrutura interna de classes (Figura 13), o *ModeloMundo* é apto a conservar os objetos do jogo referentes a elementos do ambiente, onde estes são classificados pelo seu tipo (e.g. Jogador, Bola, Bandeira, etc.). Os objetos do jogo são classes que através de generalização, herdam os atributos de suas antecessoras na hierarquia.

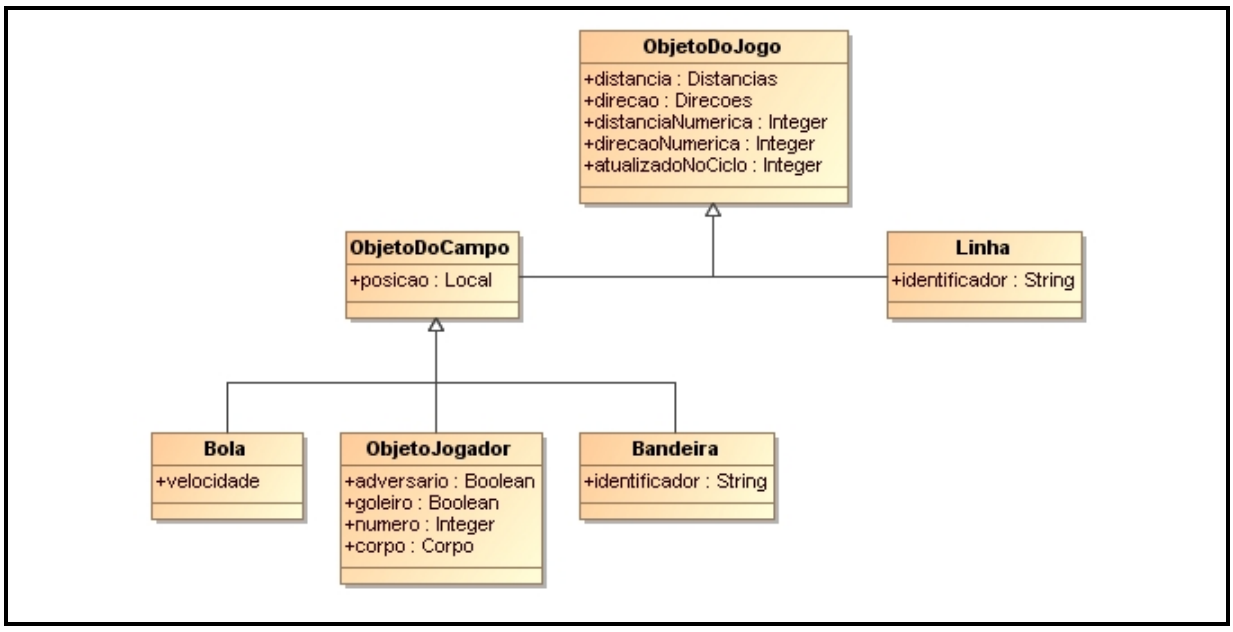

Figura 13: Hierarquia de classes referentes a elementos do jogo

Alem de manter um modelo dos objetos do jogo, o *ModeloMundo* também mantém informações do próprio agente jogador, tais como: energia, velocidade e quantidades de ações realizadas. Todas essas informações contidas, são ordenadas de forma a melhorar o desempenho das funções busca publicadas na interface *IModeloMund*o (Figura 14).

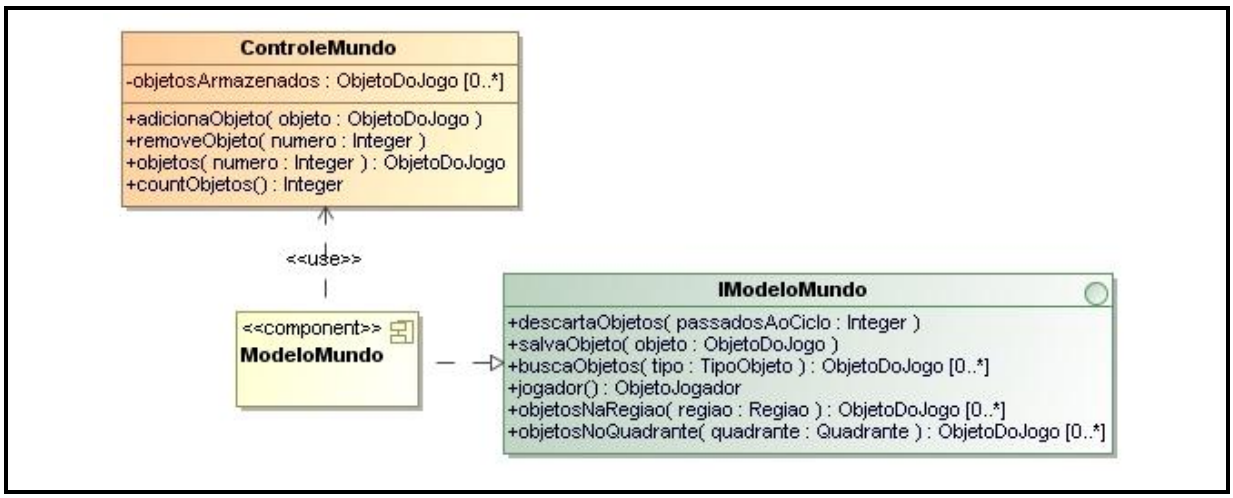

Figura 14: Modelagem do componente ModeloMundo

As descrições de cada função publicada nessa interface estão listadas abaixo:

- (*descartaObjetos*) Remove todos os objetos com a última atualização anterior ao ciclo informado no parâmetro;
- (*salvaObjeto*) Utilizada para salvar um objeto é necessário que uma instância de um objeto do jogo na estrutura interna;
- (*buscaObjetos*) A função retorna uma lista de objetos do tipo requisitado no parâmetro;
- (*jogador*) Como os dados do próprio agente são requeridos em todas as decisões. A função *jogador*, que retornar a instancia do objeto referente ao agente, foi criada com o intuito de otimizar essas atividades;
- (*objetosNaRegiao*) Busca entre todos os objetos inseridos no modelo de mundo, os que se encontram na região especificada no parâmetro. Então retorna a lista dos elementos encontrados;
- (*objetosNoQuadrante*) Idêntica a função *objetosNaRegiao*, mas limita a extensão para somente um quadrante.

Existem dois componentes que mantém relações com o *ModeloMundo*. São eles: o *Jogador* responsável por inserir informações do mundo selecionadas entre as percepções do agente; e o componente *Raciocínio* que utiliza essas informações por meio das funções de busca.

5.3.1 Divisões do Campo Virtual

O campo foi dividido em regiões e quadrantes onde as regiões servem para delimitar áreas onde o jogador devera atuar, ou seja, nessas áreas o jogador devera fiscalizar e marcar os adversários, sempre buscar a localização da bola, a fim de receber passes dos companheiros, etc. Já os quadrantes são utilizados para movimentação do jogador, e tem o objetivo de facilitar as decisões do sistema de inteligência, de forma que não será necessário trabalhar com valores numéricos, pois tornariam o raciocínio pouco eficiente.

Quando um comando de movimento é requisitado. O *Atuador* devera posicionar o agente no local mais adequado nos limites do quadrante especificado, então algum comando como ‘*interceptarBola*’ pode ser utilizado.

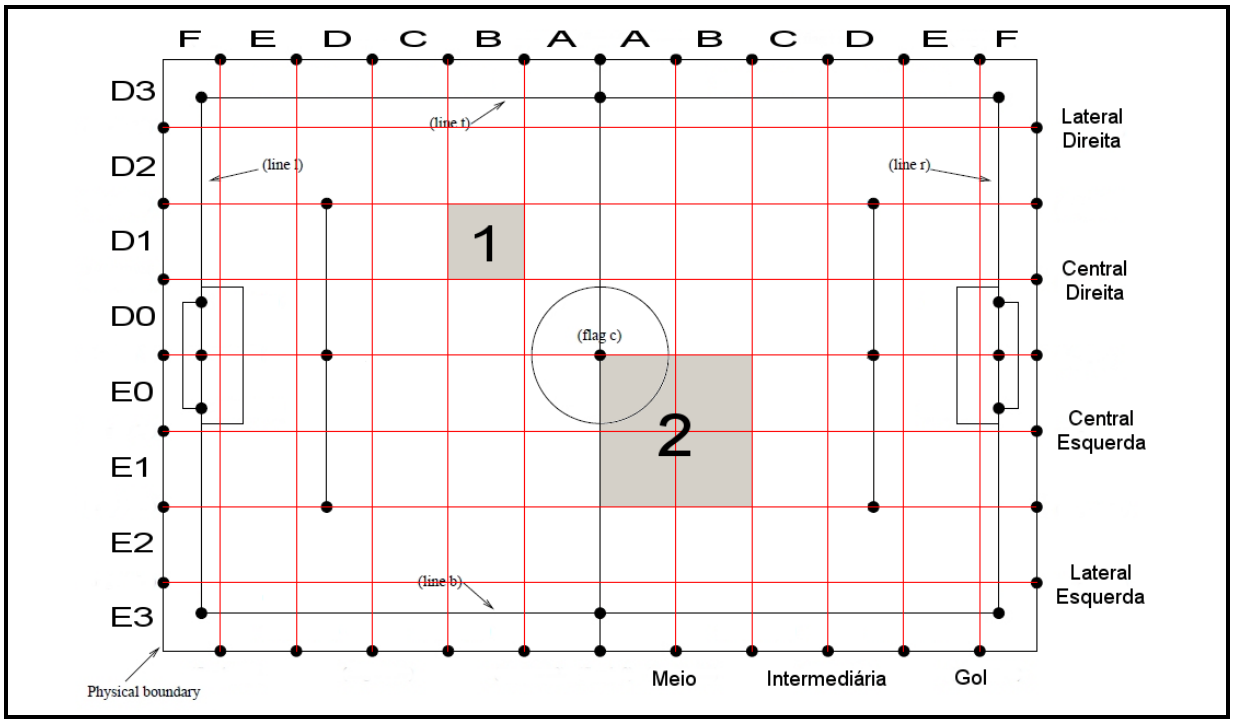

Figura 15: Quadrantes e regiões do campo virtual (adaptado de CHEN *et al.*, 2003)

Na Figura 15 os valores F, E, D, C, B e A são correspondentes a abscissa e os valores de D3-D0 e E0-E3 a ordenada do campo virtual. A combinação deles representa quadrantes para cada lado do campo. Um exemplo de quadrante é a área marcada com o número um, ela representa o quadrante (D1,B) em relação ao gol posicionado ao lado esquerdo do campo virtual.

As regiões tem uma área equivalente a de quatro quadrantes, definidas com as combinações de: meio, intermediário e gol na horizontal e na vertical por: central esquerda/direita e lateral esquerda/direita. A área marcada com o número dois na Figura 15 representa a região (Meio,Central-Esquerda) em relação ao gol posicionado ao lado direito do campo virtual.

As classes e enumerações referentes as regiões e quadrantes utilizadas pelo sistema podem ser vistas na Figura 10.

5.4 Sensores

Representando uma interface de entrada, o componente *Sensor* é responsável por traduzir os dados das percepções recebidas para informações de alto nível compostas por objetos que o agente seja capaz de lidar. Através dessas informações o agente pode saber das condições de seu corpo e utilizando suas percepções visuais seja capaz de construir e atualizar

um modelo do estado do jogo. Esses conhecimentos são essenciais para que o agente possa criar planos ou tomar decisões.

A Figura 16 é a representação da modelagem do *Sensor* onde são apresentadas as classes utilitárias e os comandos publicados em sua interface. Assim como as relações desse componente com a interface de outros.

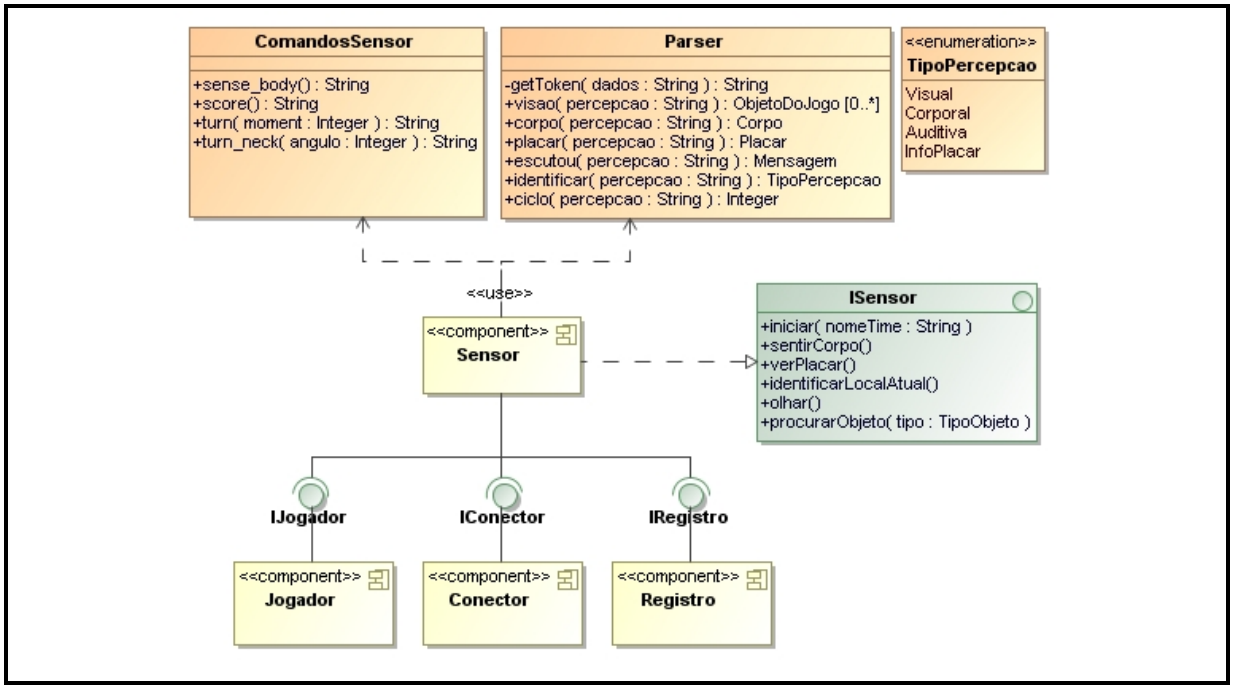

Figura 16: Modelagem do componente Sensor

Quando o agente é iniciado, o *Sensor* tem sua execução separada dos demais componentes, e passa a ser processado em um novo *Thread*. Isso é feito levando em conta as restrições de tempo impostas pelo soccerserver, pois as partes de raciocínio e atuação não devem estar inativas enquanto as percepções estão sendo aguardadas. Sendo assim, o *Sensor* pode iniciar uma execução continua da função para receber dados, publicada na interface *IConector* (Figura 11), e tem a liberdade para tratar as informações.

Assim que informação é recebida, vem a necessidade de interpretar essas informações, então a classe *Parser* implementa as funções de interpretação necessárias. Para iniciar o processo a função '*identificar*' é chamada e determina de qual tipo é a percepção, em seguida, as informações são passadas para o interpretador apropriado que decompõe as informações utilizando um autômato (i.e. um reconhecedor de uma determinada linguagem) implementado na função '*getToken*'. Essa por sua vez recebe uma string como entrada e então retorna somente um *token*. Isto é, uma parte do texto que pode ser somente um colchete ou o que estiver delimitado por espaços ou colchetes. Por fim, o *Sensor* utiliza a função '*ciclo*' para

identificar o valor ciclo contido na percepção. Cada uma das funções de interpretação é capaz de lidar com a quantidade de *tokens* para suas respectivas percepções.

A classe *ComandosSensor* é utilizada como auxiliar e converte os comandos para o formato de texto utilizado no protocolo do simulador, para que em seguida possam ser enviados ao soccerserver por meio do *Conector.*

5.4.1 Funções do Sensor

Na Figura 16 onde é apresentada a modelagem do componente *Sensor* é possível verificar que as funções publicadas na interface *ISensor* não utilizam tipo de retorno, pois os comandos do sensor não implementam resposta imediata, isto é, quando uma função é executada em alguns dos casos o componente deverá pedir informações para o simulador e então quando obtiver uma resposta, os dados poderão ser retornados através das funções publicadas na interface *IJogador* (Figura 12). Esse processo pode ser quase instantâneo ou demorar poucos milisegundos, a depender do tempo em que o soccerserver envia as novas percepções.

Segue abaixo uma breve descrição de cada função publicada na interface do *Sensor*:

- (*iniciar*) O componente executa comandos iniciais como a criação do *Thread* e salva o nome do time, necessário para identificar os jogadores;
- (*sentirCorpo*) Pede informações ao simulador sobre o corpo do jogador. Essa função devolve a resposta em até 20ms, caso nenhum erro ocorra;
- (*verPlacar*) Pede ao simulador o placar atual do jogo enviando o comando '*score*'. A resposta pode ser adquirida em ate 20ms;
- (*identificarLocalAtual*) Utilizada para fazer uma requisição da localização atual do jogador no campo;
- (*olhar*) Utilizada para fazer uma requisição de objetos vistos no momento e posição atual. A resposta é instantânea, pois o sensor mantém salvo a última percepção visual e o jogador as recebe a todo instante;
- (*procurarObjeto*) O *Sensor* modifica a direção do pescoço e se possível a do corpo do jogador para tentar localizar o objeto requerido. Algumas tentativas na mudança da direção podem ser realizadas até a função ser finalizada. Os comandos '*turn*' e '*turn_neck*' são utilizados para compor essa função.

5.4.2 Cálculo da Posição Absoluta, Quadrantes e Regiões

Após a tradução das percepções visuais o *Sensor* deve calcular a posição atual do jogador e com isso determinar a posição da bola ou de outros jogadores que podem estar incluídos na percepção. Isso é necessário, pois o simulador não informa a posição exata dos objetos vistos no campo, e sim sua distância e direção em relação a visão do jogador. Para realizar essas operações é fundamental o uso de expressões trigonométricas, mais especificamente as propriedades do triângulo retângulo e também que a posição de alguns objetos estáticos como bandeiras e linhas do campo sejam previamente conhecidas. Com isso e as informações de distância e direção originadas na percepção, é possível calcular os valores das distâncias X e Y do jogador em relação a um determinado objeto, e assim definir os valores absolutos das coordenadas do jogador no campo. É importante lembrar que o simulador arredonda as distâncias causando algumas diferenças nos cálculos. Entretanto, isso pode ser amenizado com o uso de médias entre os cálculos de posições para uma mesma percepção.

A Figura 17 mostra dois exemplos reais, onde o jogador recebe uma percepção visual e deve calcular sua atual posição com base em objetos estáticos do campo. Os pontos em preto demarcam alguns dos objetos estáticos vistos em cada percepção. E as medidas 105m x 68 representam o área interna às linhas laterais. Lembrando que ainda existe uma área de 5m após cada linha lateral, onde são posicionadas as bandeiras numeradas.

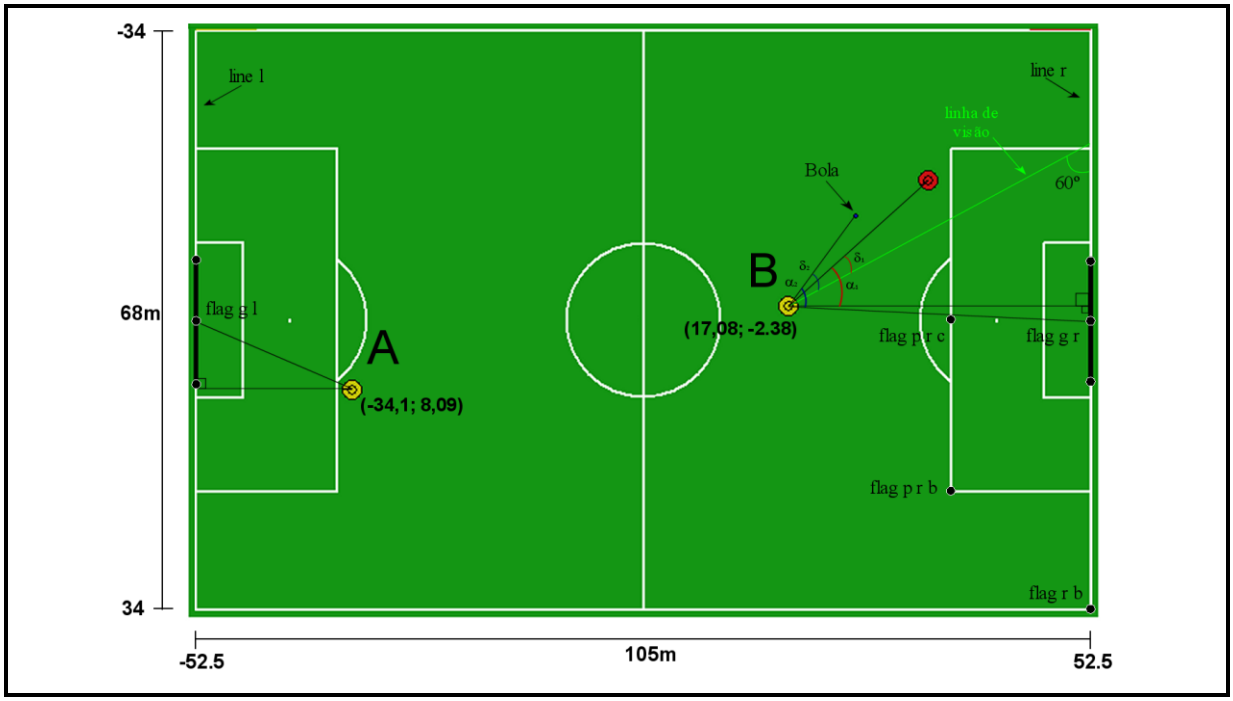

Figura 17: Objetos visualizados pelos jogadores A e B

Para iniciar o cálculo, o *Sensor* deve buscar o objeto linha visto na percepção, e dependendo de qual linha for visualizada será possível calcular uma das componentes da distância, ou seja, quando o jogador ver uma das linhas verticais, o cálculo é iniciado por sua posição X e o contrario para as linhas horizontais. A fim de facilitar essa etapa, o simulador sempre disponibiliza as informações de uma das linhas em cada percepção. Outra característica importante das percepções é que o ângulo que representa direção dos objetos vistos no lado esquerdo da linha de visão do jogador recebe o valor negativo e do lado direito recebe valor positivo. Essa linha de visão é uma reta imaginaria que segue da cabeça do jogador para a direção que o mesmo estiver olhando.

- **Cálculo do exemplo A, o jogador recebe a percepção visual abaixo:**

(see 2589 ((f g l b) 18.4 4) ***((g l) 20.1 24)*** *((f g l t) 23.8 40) ((f l 0) 24.8 19) ((f l t 10) 29.7 38) ((f l b 10) 23.3 -4) ((f l b 20) 26 -27) ((f l b 30) 31.8 -43)* ***((l l) 18.4 90))***

É possível verificar que a linha visualizada é '*line left*', com distância 18.4 metros e apresenta um ângulo de 90º formado com direção da visão do jogador. Através desses valores pode ser definido o valor da componente X, utilizando a distância para a linha como hipotenusa do triângulo retângulo e então calculando o cateto oposto ao ângulo formado com a linha. A fórmula para o cálculo é:

$$cateto\ oposto = sen(\theta) * hipotenusa$$

O resultado da componente X será de 18,4 metros. A componente Y pode ser calculada utilizando o teorema de Pitágoras:

$$hipotenusa^2 = ladoA^2 + ladoB^2$$

Para esse cálculo o objeto '*goal left*' foi escolhido como referência, ele está a 20.1 metros de distância do jogador. A distância será utilizada novamente como hipotenusa e a componente X calculada acima, será utilizada como um dos lados do triângulo retângulo. Substituindo os valores no teorema, temos: $(20.1)^2 = (18,4)^2 + (componenteY)^2$

Logo, o valor da componente Y é 8,09 e é positivo, pois o jogador ver o objeto '*goal left*' com ângulo positivo, isso indica que esse objeto esta acima da sua linha de visão.

Para definir a posição absoluta no campo basta subtrair valor encontrado para componente X da distância da linha central que é de 52,5. O valor total será de -34,1 para coordenada absoluta X, esse valor passa a ser negativo, pois esta no lado esquerdo do campo.

Nesse exemplo a componente Y mantém o mesmo valor, visto que a bandeira de referência se encontra sobre linha horizontal imaginaria que cruza o centro do campo. Então o valor do ponto absoluto do jogador A é ***P(x;y) = (-34,1; 8,09)***.

- **No exemplo B, o jogador recebe a percepção visual abaixo:**

(see 2130 ((f r t) 47.9 -13) ((f g r b) 36.2 43) ***((g r) 35.5 32)*** *((f g r t) 35.9 21) ((f p r c) 18.9 34 0 0) ((f p r t) 26.6 -15 0 0) ((f t r 30) 39.6 -41) ((f t r 40) 43.8 -29) ((f t r 50) 49.9 -19) ((f r 0) 40.4 32) ((f r t 10) 41.3 18) ((f r t 20) 44.3 5) ((f r t 30) 49.4 -5)* ***((b) 13.5 -24 0 0)*** *((p "TTeste") 22.2 -12)* ***((l r) 40.9 -60))***

O jogador vê a linha direita ao campo '*line right*' a 40.9 metros de distância e fazendo um ângulo de 60º com a direção da visão do jogador. O objeto '*goal right*' foi escolhido como referência, ele está a 35,5 metros de distancia.

Os valores para componente X e Y para esse exemplo são respectivamente 35,42 e 2,38 metros, no entanto, a direção da componente Y não pode ser determinada, uma vez que o agente pode estar tanto acima como abaixo da linha horizontal imaginaria que cruza o centro do campo. Essa solução envolve outros fatores não mencionados no exemplo anterior.

Para fazer a dedução é necessário calcular o ângulo de incidência (i.e. ângulo formato pela direção da cabeça do jogador e uma linha horizontal imaginaria que passa por ele) entre o jogador e a linha esquerda do campo. Esse ângulo é determinado utilizando umas das regras do triângulo retângulo onde a soma dos ângulos internos é igual a *180°*. Logo: *180° – 90° – 60° = 30°*.

Com o ângulo de incidência em mãos, basta fazer a normalização (i.e. modificar o ângulo para estar relacionado com a linha horizontal imaginaria que passa pelo jogador) da direção do objeto '*goal left*' que tem direção 32°, a nova direção será de 2º. Agora é possível determinar que o objeto esta abaixo do jogador, logo o valor da componente Y é negativo, pois o jogador esta acima da linha horizontal que cruza o centro do campo. O valor do ponto absoluto do jogador B é ***P(x;y) = (17,08; -2.38)***.

Para calcular as posições do da bola e do outro jogador é necessário observar e entender a Figura 18.

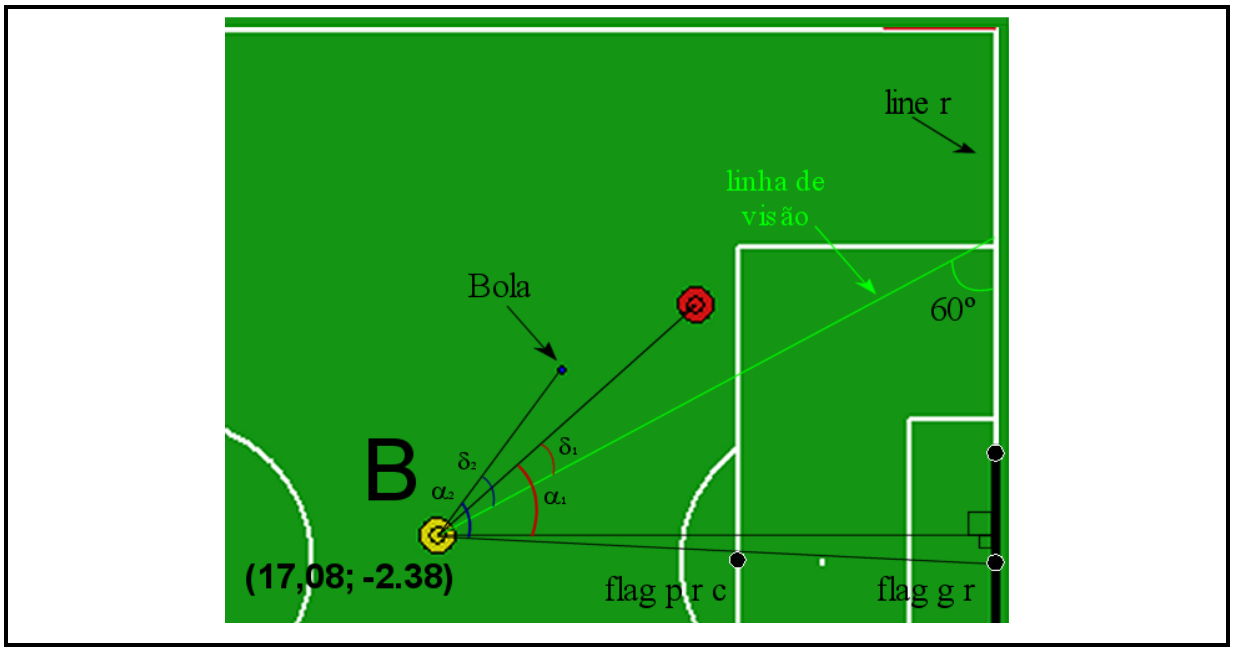

Figura 18: Exemplo jogador B ampliado

Os ângulos de incidência *δ1* e *δ2* devem ser normalizados para *α1* e *α2* que são formados com a linha horizontal imaginaria que passa pelo jogador, então os cálculos das componentes serão feitos com base nos triângulos retângulos formando com essa mesma linha. Como a bola se encontra com distância 13,5 metros e direção -24°. Então os cálculos para as componentes da bola serão:

*componenteX = cos(θ) * hipotenuza*; e *componenteY = sen(θ) * hipotenuza*
Logo: *cos(24°) * 13,5 = 12,33*; e *sen(24º) * 13,5 = 5,49*

Utilizando o jogador B como referência para a bola, a posição absoluta encontrada será ***P(x;y) = (29,41; -7,77)***. O mesmo cálculo pode ser feito para o jogador adversário.

- **Exemplo para o cálculo dos quadrantes**

As expressões utilizadas para determinar os quadrantes são definidas pela divisão sem resto por constantes preestabelecidas. São elas:

Px / 9,58 para a abscissa e Py / 9,75 para ordenada

Onde a constante 9,58 vem da divisão do comprimento do campo pelo número de quadrantes horizontais e 9,75 da divisão da largura do campo pelo número de quadrantes verticais. A parte inteira para os resultados das divisões representa o número do quadrante. Por exemplo em Py, 0 para E0, 1 para E1, -2 para D2, etc. Em Px, 0 para A, 2 para B, etc. Para definir a região o *Sensor* verifica em uma tabela para o quadrante corresponder a região.

No exemplo A da Figura 17, o jogador está localizado no quadrante (E0,D) e na região (Intermediária,Central-esquerda).

5.5 Atuadores

O atuador ou interface de saída é o mecanismo que traduz as ações de alto nível definidas pelo agente para comandos de baixo nível que são enviados para o servidor sob forma de texto, seguindo as normas do protocolo do soccerserver discutidas no capitulo três. As ações denominadas de alto nível são compostas por um ou mais comandos desse protocolo, e utilizam parâmetros simplificados (i.e. enumeração de valores onde só serão aceitos os tipos definidos).

O componente *Atuador* disponibiliza, em sua interface *IAtuador*, ações que representam as habilidades disponíveis a cada agente jogador. A Figura 19 apresenta a modelagem desse componente e sua interface.

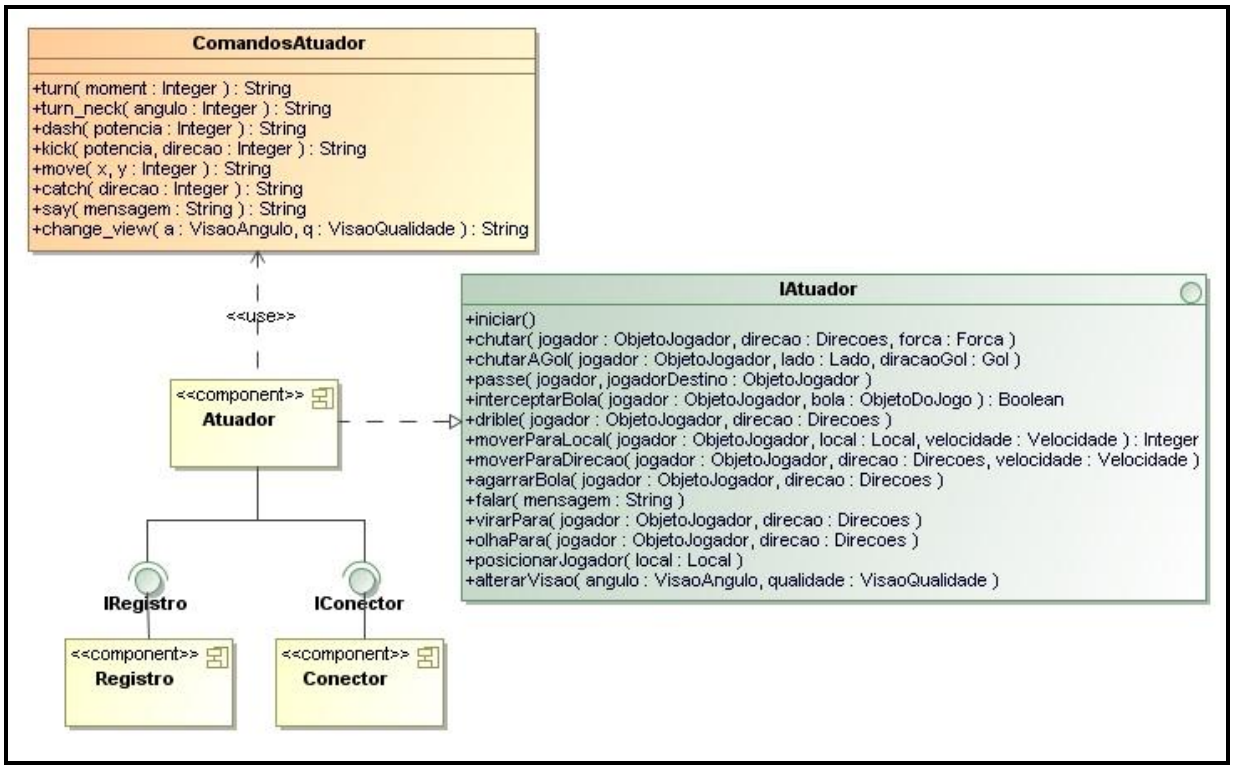

Figura 19: Modelagem do componente Atuador

Na arquitetura proposta, o *Jogador* (Figura 12) terá o papel de executar no *Atuador*, as ações escolhidas pelo componente Raciocínio. Quando uma ação é executada: primeiro os parâmetros de tipos enumerados são convertidos paras suas constantes correspondentes; então após execução interna da função, a classe *ComandosAtuador* fica encarregada da conversão

dos comandos para o formato de texto; para em seguida serem enviados ao soccerserver através do *Conector*.

Alguns dos tipos enumerados compartilhados para todos os componentes do agente são listados na Figura 10.

5.5.1 Ações do Atuador

As ações listadas na interface *IAtuador* (Figura 19) são utilizadas dependendo do estado do mundo onde o agente se encontra. Alguns dos comandos do soccerserver utilizados nessas ações dependem de certas condições para serem aceitos. Sendo assim, é importante mencionar que o *Atuador* somente executa os comandos requeridos a ele, ou seja, uma ação de chute é executada independente de a bola estar perto. O trabalho de verificação dessas condições é feito pelo mecanismo de raciocínio.

Segue abaixo uma breve descrição de cada função publicada na interface do *Atuador*:

- (*iniciar*) Executa os comandos iniciais do *Atuador*;
- (*chutar*) Representa um comando básico do agente que pode ser usado em estratégias de drible complexas. Recebe dois parâmetros com valores enumerados para direção do chute e a força aplicada. Após as conversões o comando '*kick*' é enviado ao simulador;
- *(chutarAGol*) Essa função é uma ação composta. Ela utiliza a informação do ultimo local onde o jogador se moveu para localizar a direção do gol, então chuta a bola em direção ao gol. A ação é composta por apenas um comando de '*kick*';
- (*passe*) Para executar o passe o *Atuador* utiliza a informação de localização do jogador passado no parâmetro, então calcula a direção e força necessária para o passe. A ação é composta por apenas um comando de '*kick*';
- (*interceptarBola*) O jogador segue até o local próximo à bola e então executa o comando '*kick*' com pouca potência. Case o jogador não tenha oportunidade de chutar a bola o comando retorna o valor lógico falso. Esse comando é utilizado para roubar a bola do adversário. A ação é composta por comandos '*turn*', '*dash*' e '*kick*';
- (*drible*) Essa habilidade é uma aplicação da forma mais simples de drible, onde o jogador apenas desvia seu corpo e a bola para a direção informada no parâmetro. Pois o *Atuador* não tem conhecimento da posição do jogador adversário. A ação é composta por um comando '*kick*' seguido de comandos '*turn*' e '*dash*';
- (*moverParaLocal*) O comando mover é de grande importância para o agente, é a ação

mais utilizada da interface de saída. Através do parâmetro do tipo Local o *Atuador* interpreta qual quadrante ou região foi especificada e então move o agente para a posição mais apropriada no local determinado. A ação é composta por comandos '*turn*' e '*dash*';

- (*moverParaDirecao*) Movimenta o jogador assim como o comando *moverParaLocal*. No entanto, anda somente a distância percorrida por um comando '*dash*' de acordo com a velocidade informada no parâmetro. A ação é composta por comandos '*turn*' e '*dash*';
- (*agarrarBola*) Ação de tentar agarrar a bola na direção informada, O uso é exclusivo para o goleiro. A ação é composta por somente um comando '*catch*';
- (*falar*) O jogador utiliza o comando '*say*' para passar informações aos jogadores próximos.
- (*virarPara*) Muda a direção do corpo do jogador através do comando '*turn*';
- (*olharPara*) Muda a direção da cabeça do jogador através do comando '*turn_neck*';
- (*posicionarJogador*) Utiliza o comando '*move*' para posicionar o jogador em determinado local. Esse comando só poderá ser utilizado nos estados: '*before_kick_off*', '*goal_r*' ou '*goal_l*'. Sua maior vantagem é poupar a força dos jogadores, visto que os mesmo não terão necessidade de correr ate o local;
- (*alterarVisao*) Modifica o cone de visão do jogador para uma combinação dos valores das enumerações *VisaoAngulo* e *VisaoQualidade* encontradas na Figura 10.

As funções *moverParaLocal e posicionarJogador,* que utilizam uma instancia da classe *Local* (Figura 10) como parâmetro, tentam inicialmente fazer uso da posição absoluta, caso essa não esteja especificada, o próximo será o quadrante, ou por fim a Região.

Como algumas das ações são compostas por um ou mais comandos do soccerserver, e esse impõe restrições de tempo para a quantidade de comandos. Para comandos geradores de ações no jogador (e.g. *turn, turn_neck, kick, move, dash, catch),* a restrição é de um ciclo equivalente a 100ms, os demais podem ser enviados a cada 20ms. Nessas condições o *Atuador* é obrigado a manter sincronia com o simulador, a fim de respeitar o tempo para cada comando que constitui uma ação, caso contrário, o simulador pode ignorar os comandos e o agente aparentar instabilidade, assim dando oportunidades para o time adversário.

5.6 Raciocínio

O sistema responsável pela inteligência do jogador representado pelo componente *Raciocínio* deverá avaliar a cada momento as condições dos agentes e do ambiente do jogo, mas também saber lidar com as prioridades de comportamentos para os diferentes tipos de jogadores (e.g. goleiro, zagueiro, atacante, etc.), e ainda ser apto a criar, aplicar e modificar estratégias para satisfazer as metas da equipe. Para isso o componente será implementado com duas formas de analisadores. Uma com características reativas, justificado pela necessidade de respostas rápidas. E outro com características cognitivas que deverá corresponder às expectativas de trabalho em conjunto com os demais agentes.

A atividade reativa apresenta um comportamento preestabelecido, as ações correspondem a estímulos recebidos e são determinadas pelo estado atual do ambiente e dos jogadores. Para selecionar os comportamentos, o analisador reativo utiliza um sistema baseado em regras constituído por um conjunto de condições que resultam em algumas possíveis ações.

Os analisadores cognitivos são encarregados do planejamento e focam nas metas em longo prazo. Estão sempre tentando encontrar uma solução para o estado do mundo corrente e chegar ao resultado esperado.

A utilização da comunicação entre os jogadores amplia a capacidade desse componente, e da possibilidade de combinar estratégias, responder a pedidos de passe e a comandos do técnico do time, ou até obter informação sobre o estado de objetos do mundo por outro ponto de visão.

5.7 Registro

O componente de *Registro*, como o nome já diz, registra as atividades dos componentes e um arquivo de texto. Isso é útil para avaliar o funcionamento do agente separadamente ou como um todo. Para registrar as ações realizadas, cada componente que tem ma ligação direta com a interface do *IRegistro*, deve executar um comando informando o que pretende fazer. Assim algum erro ocorrido poderá ser facilmente localizado.

6. Conclusão

Nesse capítulo final serão apresentados e analisados os resultados obtidos e também sugestões de temas para trabalhos futuros, bem como as contribuições realizadas por esse trabalho.

6.1 Contexto

Os torneios de futebol de robôs promovidos pela RoboCup tem despertado o interesse de estudantes e pesquisadores de todo o mundo. Como um exemplo de Sistemas Multi-Agentes que é uma área da inteligência artificial distribuída, essas competições e seus desafios científicos incentiva a pesquisa e o aperfeiçoamento de técnicas de IA. Partindo dessas motivações, o presente trabalho teve como objetivo criar um protótipo base de um time para operar na liga de futebol simulado em duas dimensões, atuando em um ambiente dinâmico, contínuo, inacessível e não determinístico. Para tal, foi necessário entender os conceitos de agente e Sistemas Multi-Agentes, analisar suas arquiteturas e os diferentes tipos de ambientes, pesquisar sobre técnicas de engenharia de software como o desenvolvimento baseado em componentes, estudar o funcionamento simulador Soccerserver e o protocolo de comunicação utilizado pelo mesmo.

O ponto fundamental desse projeto envolveu a proposta para o desenvolvimento de agentes jogadores sob uma arquitetura em camadas vertical que atua com as informações fluindo de baixo para cima e então as decisões retornam de cima para baixo, também envolveu o uso do desenvolvimento baseado em componentes para a organização do sistema. Essa proposta foi limitada à modelagem da primeira camada da arquitetura, onde o agente manipula um modelo de mundo e traduz informações de baixo nível originadas dos dados recebidos e enviados através da comunicação com o simulador. As demais camadas que tratam da inteligência do jogador, onde são implementados os comportamentos mais sofisticados, somente tiveram uma abordagem sobre as obrigações que as mesmas devem lidar.

Com o inicio do desenvolvimento do agente, surgiram algumas dificuldades, ora imposta pelo simulador, ora vinculada à modelagem. Como exemplo temos: as restrições de tempo para execução de ações ou o atraso para observar as mudanças no ambiente; a necessidade que um jogador execute suas ações no tempo disponível, caso contrário a chance de executar uma determinada ação pode ser perdida; os jogadores não sabem os valores

exatos de suas próprias posições, nem de outros objetos moveis; as informações sobre os objetos visualizados são perdidas, quando as distâncias são longas. Algumas dessas dificuldades foram contornadas, e outras foram expostas para que sejam ajustadas no futuro.

A modelagem do jogador foi determinada pelo uso de sete componentes que com seus comportamentos definidos mostraram-se ter os requerimentos necessários para funcionamento do agente, são eles: *Conector* (mantem o contato com o simulador); *Sensor* (responsável por interpretar as percepções); *Atuador* (permite o contato com o ambiente); *Jogador* (controla os demais componentes); *ModeloMundo* (conserva as informações obtidas do ambiente); *Registro* (mantém um histórico das atividades); *Raciocínio* (toma decisões de forma inteligente). As conexões entre estes se dão por meio de suas interfaces e todos são encapsulados pelo componente *Agente*.

Por fim para um melhor desempenho em um ambiente com tempo real, o processamento foi dividido em dois setores concorrentes, um envolve o *Sensor* que trabalha continuamente com os dados recebidos e o outro representa os demais componentes. Cabe salientar que como a modelagem do componente *Raciocínio* não foi aprofundada, até então não foi definido se este devera operar em um terceiro setor concorrente.

6.2 Resultados alcançados

Após a finalização deste projeto, obteve-se uma visão dos bons resultados obtidos. Observa-se que o desenvolvimento do agente foi concebido utilizando a notação de Desenvolvimento Baseado em Componentes pretendida e que foi concluída a modelagem de um sistema básico, onde o agente tem capacidade para entender e atuar em um ambiente de forma coerente com as suas possibilidades.

A implementação do protótipo constituído por todos os métodos das interfaces dos componentes e classes auxiliares foi concluída com sucesso. Para fins de teste, foi implementada uma inteligência reativa, onde o agente procura a bola, tenta tomar a posse e, caso tenha êxito caminha em direção ao gol adversário e chuta ou então realiza um passe caso aviste um companheiro mais próximo do gol.

Após vários testes realizados foi constatado que o jogador é capaz de realizar suas funções correspondendo às expectativas. Contudo ao executar testes com agentes de outros times ficou clara a necessidade de uma inteligência mais apurada.

6.3 Contribuições

Os conhecimentos produzidos a partir deste trabalho representam o ponto de partida para o desenvolvimento de um time de agentes jogadores. Através do estudo dos agentes e suas arquiteturas, juntamente com a pesquisa do funcionamento do simulador, foi possível a modelagem de um jogador protótipo. Essa pesquisa contribuiu em aspectos relevantes para o desenvolvimento de agentes inteligentes e ou jogadores da RoboCup, tais como:

- A pesquisa sobre os agentes e suas arquiteturas, assim como escolha da arquitetura hibrida que é eficaz em ambientes de tempo real por sua rapidez de resposta a estímulos;
- O estudo dos protocolos e comandos do soccerserver possibilitou a criação de uma interface que traduz dados do servidor para alto nível permitindo ao agente trabalhar com eficiência;
- O uso de componentes para a modelagem permite o fácil entendimento do funcionamento agente.

Alem do fato que a UESB ainda não tem um time que possa participar das competições da RoboCup. Portanto, este protótipo de jogador pode servir como base de estudo para interessados na área e contribuir para o desenvolvimento de times competitivos.

6.4 Trabalhos futuros

Com o objetivo de dar continuidade no presente trabalho de pesquisa, torna-se necessário aprofundar o tema no aspecto da autonomia agente, que prevê a incorporação e otimização de um componente de raciocínio capaz de tomar decisões reativas a mudanças no ambiente do jogo, como também fazer inferências a partir de dados passados e atuais do mundo, possibilitando a tomada de decisões mais complexas e a criação de planos de ação.

Outros aspectos que poderão ser desenvolvidos futuramente são:

- Utilizar alguma forma de aprendizagem para que o agente possa aprender com suas ações do passado ou dos adversários;
- Implementar outra camada que atue como moderadora para o agente trabalhar em conjunto com os demais do time;

- Atribuir estratégias defensivas e ofensivas;
- Incluir um protocolo para comunicação entre os agentes. Para que possa combinar estratégias e também enviar e receber informações visuais, de maneira que o jogador amplie o estado do mundo atual e possa tomar decisões baseadas em diferentes perspectivas;
- Aperfeiçoar interface dos componentes existentes e implementar mais ações para o componente atuador, dando maior autonomia para jogador completar suas metas.

7. Referências

BARBOSA, L. R. C. Um sistema multiagente para monitoramento atmosférico. 2005. Dissertação (Mestrado em redes de computadores), Universidade Salvador, 2005. Disponível em: http://tede.unifacs.br/tde_arquivos/2/TDE-2007-01-09T125233Z-54/Publico/>. Acessado em 07 de setembro de 2010.

CHEN, M. *et al*. RoboCup Soccer Server: Users Manual. 2003.

CHEESMAN, J.; DANIELS, J. UML Components. Addison-Wesley, 2000.

CORTEN, E et al. Soccerserver manual. Technicel report, RoboCup Federation, 1999.

COSTA, A. C. P. L.; BITTENCOURT, G. Soccer Server: um simulador para o futebol de robôs da Robocup Federation Tutorial - 3. Encontro Nacional de Inteligencia Artificial: Florianópolis, 1999. Disponível em: <https://intranet.dcc.ufba.br/pastas/mecateam/publicacoes/Tutorial_ENIA99.pdf>. Acessado em 07 de setembro de 2010.

COSTA, M. T. C. Uma Arquitetura Baseada em Agentes para Suporte ao Ensino à Distância. 1999. Tese (Doutorado em Engenharia de Produção), Universidade Federal de Santa Catarina, Florianópolis, 1999. Disponível em: <http://www.eps.ufsc.br/teses99/thiry/>. Acessado em 07 de setembro de 2010.

FRANKLIN, S; GRAESSER, A. Is it an Agent, or just a Program? 1996. Disponível em: <http://www.msci.memphis.edu/~franklin/AgentProg.html>. Acessado em 20 de julho de 2010.

FININ, T.; WEBER, J. et al. DRAFT: Specification of KQML Agent Communication Language. The DARPA Knowledge Sharing Initiative External Interfaces Working Group, Junho 1993.

IBM RESEARCH. Disponível em: <http://www.research.ibm.com/iagents> Acessado em 20 de Julho de 2010.

JACOBSON, I.; RUMBAUGH, J.; BOOCH, G.. Unified Software Development Process – Addison Wesley, Reading, 1999.

JUCHEM, M.; BASTOS, R. Engenharia de Sistemas Multiagentes: Uma Investigação sobre o Estado da Arte. 2001. Disponível em: <http://www3.pucrs.br/pucrs/files/uni/poa/facin/pos/relatoriostec/tr014.pdf>. Acessado em 07 de setembro de 2010.

NISSEN, M. Intelligent Agents: A Technology and Business Application Analysis. 1995.

OLIVEIRA, E. C.; FRACCAROLI, E. S.; BIANCHI, R. A. C. O time FEI CDU-2006 da categoria RoboCup Soccer Simulation 2D. FEI, 2006. Disponível em: <http://www.fei.edu.br/~rbianchi/publications/EnRI2006-CDU.pdf>. Acessado em 07 de setembro de 2010.

OLIVEIRA, F. Inteligência Artificial Distribuída. In: IV Escola Regional de Informática, SBC, SC, 1996.

PINHO, H. S. RIGEL – Um repositório com suporte para Desenvolvimento Baseado em Componentes. Dissertação (Mestrado em computação), Universidade Estadual de Campinas, 2006. Disponível em: <http://cutter.unicamp.br/document/?code=vtls000388249>. Acessado em 07 de setembro de 2010.

REIS, L. P. Introdução aos sistemas multi-agente. Porto, 2002.

RIOS, A. R. UESBTeam: Utilização de técnicas de Engenharia de Software e Inteligência Artificial para construção de um time de futebol de robôs. 2006. Monografia (Graduação em Ciência da Computação), Universidade Estadual do Sudoeste da Bahia, 2006.

RoboCupCIn. Disponível em: <http://www.cin.ufpe.br/~robocupcin> Acessado em 20 de Julho de 2010.

RUSSELL, S., NORVIG, P. Inteligência Artificial, Editora Campus, 2004.

SANTOS, G. A.; FAVORETO, R. C.; SILVA, E. N.; QUIRINO, G. K. L.; FLORIANO, A. S. P. Uso da lógica fuzzy no Pet Soccer 2D usado na categoria de simulação. Espírito Santo, 2009. Disponível em: <http://www.cbr09.fei.edu.br/inscricoes/TDP/PETSoccerUFES.pdf>. Acessado em 07 de setembro de 2010.

SHOHAM, Y. An Overview of Agent-Oriented Program. In Software Agents, ed. J. M. Bradshaw, AAAI Press, 1997.

SOUZA, E. M. Uma estrutura de agentes para assessoria na internet. 1996. Dissertação (Mestrado em engenharia), Universidade Federal de Santa Catarina, 1996. Disponível em: <http://www.eps.ufsc.br/disserta96/eliane/index/index.htm>. Acessado em 07 de setembro de 2010.

STONE, P.; VELOSO, M. Multiagent Systems: A Survey from a Machine Learning Perspective. Pittsburgh: Carnegie Mellon University, 2000. Disponível em: <http://www.societyofrobots.com/robottheory/Multiagent_Sys_Survey_Machine_Learning_Perspective.pdf>. Acessado em 07 de setembro de 2010.

SZYPERSKI, C. Component Software – Beyond Object Oriented Programming, Addison-Wesley, 1997.

TEIXEIRA, C. B.; SANTANA, O. V.; COSTA, A. L. TDP MecaTeam 2009. 2009. Disponível em: <http://www.cbr09.fei.edu.br/inscricoes/TDP/MecaTeam.pdf>. Acessado em 07 de setembro de 2010.

WEISS, G. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press, 1999.

WINIKOFF, M.; PADGHAM, L.; HARLAND, J.; THANGARAJAH, J. Declarative and procedural goals in intelligent agent systems, In: The proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning. Toulouse, France, 2002 Disponível em: <http://www.cs.rmit.edu.au/agents/Papers/KR2002.pdf>. Acessado em 07 de setembro de 2010.

WOOLDRIDGE, M.; JENNINGS, N. Intelligent Agents: Theory and Practice. 10ª ed. Manchester: Cambridge University Press, 1995.