INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Comportamentos Relacionais em Futebol Robótico

Análise de Incerteza

João Gonçalo Delgado Torres

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Júri

Presidente:	Professor Francisco Garcia
Orientador:	Professor Pedro U. Lima
Co-Orientador:	Engenheiro Rodrigo Ventura
Vogais:	Professor Paulo Oliveira

Outubro de 2007

Agradecimentos

Agora são muitos os momentos que me vêm à cabeça: momentos de trabalho ou divertimento, de alegria, de frustração, de desespero, de euforia, de orgulho, de dúvida e de tantos outros sentimentos que preenchem qualquer caminhada. Cada uma dessas recordações trazem-me à lembrança pessoas. É a essas pessoas que quero agradecer terem-me acompanhado nesta longa caminhada.

Começo por agradecer à minha família, em especial aos meus pais e ao meu irmão, por todo o amor, carinho, paciência e orgulho que demonstram por mim. À minha Mãe por ser aquela pessoa que, sem perceber quase nada do que eu estudei nos últimos cinco anos, viveu cada momento como se fosse dela, compreendendo a razão de cada alegria e encorajando-me nos momentos de maior dificuldade. Ao meu Pai porque, por já ter feito um percurso idêntico, se tornou o meu farol, obrigado por todos os conselhos. Por fim, ao Filipe pela amizade incondicional. Agradeço à Ana por ser o meu porto de abrigo, pela compreensão quando estava absorto em trabalho e me mostrava mais ausente, pela paciência quando me faltavam horas de sono, pelo amor. Um enorme obrigado também porque um caminho torna-se mais fácil quando é percorrido por dois e porque esta etapa que agora termino não é mais do que parte desse caminho.

Obrigado a todos com quem me cruzei no IST e no projecto SocRob. Obrigado pela ajuda e por me tirarem todas as dúvidas quando precisei. Um obrigado especial aos amigos que fiz. Por todos os momentos que partilhámos, pelas noitadas de trabalho que se tornavam divertidas pela vossa presença. Aqui devo ainda fazer um agradecimento especial ao Tiago, porque é fácil trabalhar contigo, porque a máxima *duas cabeças pensam melhor que uma* se adapta perfeitamente ao nosso caso, pelos momentos de diversão quando o trabalho parecia não ter fim e pela amizade. Agradeço a todos os que, não fazendo directamente parte deste meu percurso, me lançaram palavras de apoio.

Finalmente, um agradecimento ao Engenheiro Rodrigo Ventura e um obrigado especial ao Professor Pedro Lima por toda a atenção, disponibilidade e empenho dedicado ao trabalho.

João

Resumo

O objectivo desta dissertação passa pela criação, demonstração e análise do funcionamento de comportamentos relacionais, desenvolvidos no âmbito do projecto SocRob, para situações de jogo estáticas. A modelação dos comportamentos foi feita utilizando Redes de Petri e a sua implementação seguiu um algoritmo criado a partir do trabalho desenvolvido anteriormente. Como requisito essencial a um comportamento relacional, foram implementados mecanismos de estabelecimento e gestão do compromisso bem como de sincronização. Os comportamentos desenvolvidos foram testados num simulador e nos robots reais, utilizando o mesmo código. Finalmente, foi realizada uma análise quantitativa das Redes de Petri desenvolvidas.

Palavras Chave

Futebol Robótico, Comportamentos Relacionais, Compromisso, Sincronização, Redes de Petri, Análise Quantitativa

Abstract

The goals of this Thesis are the development, demonstration and analysis of relational behaviors, developed in the scope of SocRob project, for static situations of game, for instance the goal kick. The Behaviors were design using Petri Nets and its implementation followed an algorithm modified from previous work. As an essential requisite to a relational behavior, mechanisms of commitment establishment and management as well as of synchronization were implemented. Using the same code, the behaviors were tested in the simulator and in the real robots. Finally, a quantitative analysis of the developed Petri Nets was carried through.

Keywords

Relational Behaviors, Commitment, Synchronization, Petri Nets, Robotic Soccer, Quantitative Analysis

Índice

Lista de Figuras

Lista de Tabelas

Listagem de Código

Lista de Acrónimos

4LL	Four-Legged League
BC	BehaviorCoordinator
BE	BehaviorExecutor
ETPN	Exponential Timed Petri Nets
FIFA	Fédération Internationale de Football Association
GSPN	Generalized Stochastic Petri Nets
IST	Instituto Superior Técnico
MeRMaID	Multiple-Robot Middleware for Intelligent Decision-making
MSL	Middle Size League
MPNP	Multi - Petri Net Plans
PNML	Petri Net Markup Language
PNP	Petri Net Plans
RdP	Rede de Petri
RoboCup	Robot World Cup
SocRob	Sociedade de Robots / Soccer Robots
SPN	Stochastic Petri Nets
SPQR	Soccer Player Quadruped Robots
STA	Sense-Think-Act
TO	TeamOrganizer
XML	Extensible Markup Language

"Precisando, portanto, um príncipe, de saber utilizar bem o animal, deve tomar como exemplo a raposa e o leão: pois o leão não é capaz de se defender das armadilhas, assim como a raposa não sabe defender-se dos lobos."

Niccolo Maquiavel

Capítulo 1

Introdução

O objectivo desta dissertação passa pela criação, demonstração e análise do funcionamento de comportamentos relacionais, desenvolvidos no âmbito do projecto SocRob, para situações de jogo estáticas e utilizando Redes de Petri.

O trabalho está organizado em sete capítulos. Neste capitulo é apresentada a motivação do trabalho, o projecto SocRob no âmbito do qual a tese foi desenvolvida, a especificação do problema e o seu enquadramento cientifico, o ambiente de desenvolvimento e, finalmente, o estado da arte, focando a utilização da Redes de Petri em trabalhos deste género. No Capítulo 2 são introduzidos os fundamentos do trabalho, enquanto que no Capítulo 3 são apresentados os conceitos relacionados com os comportamentos relacionais. Nos Capítulos 4 e 5 é apresentado o trabalho desenvolvido, concentrando-se o primeiro na implementação e o segundo nos comportamentos desenhados. O Capítulo 6 apresenta os resultados obtidos e uma análise quantitativa às Redes de Petri desenvolvidas. Por fim o Capítulo 7 conclui o trabalho referindo possíveis trabalhos futuros nesta área. Em anexo, no Apêndice A são apresentadas todas as RdP dos comportamentos desenvolvidos.

1.1 Motivação

Por natureza, qualquer sociedade tende a organizar-se para completar tarefas que seriam impossíveis de realizar por um indivíduo. No nosso dia-a-dia e na própria Natureza encontramos vários exemplos: uma organização empresarial, uma equipa desportiva, um atleta de maratona e o seu treinador, uma colmeia ou uma população de formigas têm em comum a realização de trabalho em equipa para atingir melhores resultados. Seguindo a lógica de imitação do próprio ser humano, a robótica cria populações de robots que ao funcionar como um robot distribuído atinge objectivos que seriam muito mais complicados se a tarefa fosse levada a cabo apenas por um robot. As aplicações deste conceito estão presentes nos mais diversos campos: exploração do espaço, indústria automóvel, robots de procura-e-salvamento, futebol robótico, entre muitos outros.

O futebol robótico, tal como o futebol humano consiste no confronto de duas equipas constituídas por número igual de jogadores, neste caso robots, que cooperam para derrotar a outra equipa. De igual modo uma equipa poderá ter melhores resultados caso os jogadores cooperem de forma bem sucedida. Este projecto centra-se no desenvolvimento de comportamentos relacionais para um equipa de robots futebolistas, ou seja, comportamentos que envolvem mais do que um robot. Este objectivo requer que sejam encontradas soluções para vários problemas. Em qualquer tarefa que envolva vários indivíduos é necessário um bom entendimento entre os intervenientes para que o objectivo seja alcançado, assim como dois jogares de futebol humano comunicam entre si, o mesmo deve acontecer entre os dois robots. Esta necessidade levanta os principais problemas relacionados com a implementação deste tipo de comportamentos. Dois seres humanos estão dotados de funções sensoriais que lhes permitem comunicar de forma implícita (em que o jogador toma decisões pelo que observa do seu companheiro) e explícita (em que o jogador toma decisões pelas acções comunicadas pelo seu companheiro de equipa). No caso dos robots a comunicação implícita ainda não foi devidamente conseguida, como tal toda a comunicação é feita apenas de forma explícita, através de memória globais, passagem de mensagens, entre outros. É através dessa comunicação que se pretende garantir a sincronização entre os dois agentes para o sucesso do comportamento relacional.

1.2 Projecto SocRob

Este trabalho enquadra-se no âmbito de um projecto do Instituto de Sistemas e Robótica (ISR), pólo do Instituto Superior Técnico (IST), denominado SocRob[18] [1]. Esta designação vem do jogo de palavras entre "Society of Robots"e "Soccer Robots". O projecto centra-se em desenvolver uma sociedade de robots, usando como exemplo de aplicação uma equipa de robots futebolistas.

O SocRob é, neste momento, constituído por alunos, doutorandos e doutorados de Engenharia Electrotécnica e de Computadores e por alunos de Engenharia Informática e de Computadores, tanto Portugueses como estrangeiros, sendo liderado por um Professor Associado do Instituto Superior Técnico. Os seus elementos dividem-se em duas equipas: uma equipa de robots Sony Aibo da Four Legged League (4LL) e uma equipa da Middle Size League(MSL) que trabalham conjuntamente, inserindo-se este trabalho no segundo grupo.

O projecto tem vários objectivos: realizar investigação em robótica cooperativa, ensinar Engenharia Electrotécnica a estudantes do ensino superior, divulgar a ciência e tecnologia e representar o IST e Portugal a nível nacional e internacional, respectivamente, sendo presença habitual no Festival Nacional de Robótica e um dos projectos mais antigos na Feder-

[1] http://socrob.isr.ist.utl.pt/

ação Internacional RoboCup.

Sendo o projecto desenvolvido numa área tão prolifera como é a robótica e centrando-se no desenvolvimento de robots para ambientes extremamente dinâmicos e não-determinísticos, são inúmeras as áreas de investigação que abrange. São exemplos o processamento de imagem, a fusão sensorial, a localização, o planeamento de trajectórias com desvio de obstáculos, a tomada de decisões e a execução de tarefas envolvendo um ou mais agentes. É neste último assunto que se insere mais especificamente este trabalho no projecto.

1.3 Especificação do problema

A primeira grande competição internacional de futebol robótico teve lugar em 1997. Desde então as constantes alterações das regras, dimensões do campo e envolvente, aliada aos objectivos dos projectos de investigação em que estão inseridos, obriga as equipas a chegarem cada vez mais longe nas suas implementações. Ao longo deste tempo foram desenvolvidos e implementados diversos comportamentos cuja complexidade vai aumentando. Contudo, o seu desenvolvimento, por ausência de necessidade de outras soluções para conseguir vantagem competitiva e por limitações tecnológicas, tem-se concentrado em comportamentos individuais. Com o aumento do tamanho do campo e a introdução de regras como o impedimento de reposições da bola em campo utilizando apenas um robot, os comportamentos individuais, apesar do estado maturo de desenvolvimento, apresentam limitações. Tal como no futebol humano, no futebol robótico uma equipa que jogue utilizando passes de forma eficaz gozará de uma enorme vantagem sobre o adversário. O desenvolvimento de comportamentos ganhou assim uma nova necessidade: a implementação de comportamentos relacionais, ou seja, comportamentos que envolvam mais do que um agente.

A introdução de comportamentos relacionais levanta novos problemas, dado que tarefas que envolvam cooperação entre dois ou mais agentes só terão sucesso se todos os robots interagirem correctamente e para isso é necessário um compromisso entre eles. É portanto indispensável introduzir meios para estabelecer e gerir esse compromisso e, não menos importante, meios para realizar a sincronização entre os actores, para os quais é necessário uma comunicação muito fiável. Numa equipa de futebol humano estes objectivos são facilmente conseguidos dado a capacidade de análise e de comunicação entre os jogadores mas no caso de futebol robótico é bem mais complicado.

Este trabalho teve como objectivo a concepção, implementação e teste de comportamentos relacionais para robots futebolistas, utilizando redes de Petri, no seguimento do que vem sendo desenvolvido no Projecto SocRob. Foram implementadas relações estáticas(marcação de foras, pontapés de baliza, cantos, (re)início de jogo)para situações em que a equipa at-

aca. Entende-se por uma situação estática de jogo quando não há necessidade da parte da máquina de decisão de optar por um comportamento relacional. As situações referidas são desse género dado que é sabido que será necessário realizar um comportamento relacional para colocar a bola em jogo. Em contrapartida, a decisão de realizar um passe ao invés de driblar para a baliza diz-se uma situação dinâmica.

O trabalho pretendeu responder a perguntas como:

- Quais os robots que deverão intervir no comportamento relacional?
- Qual o protocolo para estabelecer o compromisso?
- Como gerir o compromisso, ou seja, como mantê-lo e, caso necessário, quebrá-lo?
- Como realizar a sincronização necessária aos comportamentos?

Posteriormente foi realizada uma avaliação dos resultados tendo como base uma análise estocástica das redes de Petri e a simulação de alguns comportamentos também implementados em robots reais.

1.4 Enquadramento científico

O projecto SocRob enquadra-se numa área da robótica que envolve ambientes extremamente dinâmicos e não-determinísticos. Sendo assim, são investigados diversos temas tais como o processamento de imagem, a fusão sensorial, a localização, o planeamento de trajectórias com desvio de obstáculos, a tomada de decisões e a execução de tarefas envolvendo um ou mais agentes.

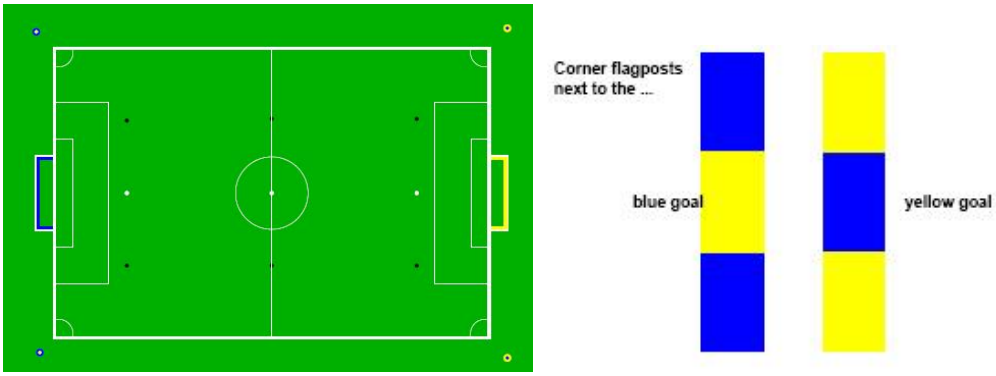
A cooperação multi-agente, área em que se insere o trabalho, tem vindo a ser explorada em diversas áreas da robótica. Exemplo disso é outro projecto do ISR chamado Rescue[20] que utiliza cooperação para busca e salvamento. A cooperação bem sucedida leva a melhores resultados, é um facto comum as sociedades na Natureza e também o será para uma sociedade de robots. Tratando-se de uma equipa de futebol robótico uma das áreas que vem ganhando mais importância e cada vez mais se apresentará fulcral é a implementação de comportamentos relacionais.

O projecto SocRob, cuja equipa utiliza a designação ISocRob nas competições, é presença comum em encontros científicos tanto a nível nacional como a nível internacional. Esses encontros envolvem seminários e competições de futebol robótico. No nosso país o maior e mais importante evento é o Festival Nacional de Robótica, realizado anualmente desde 2001. A nível internacional o projecto faz parte da RoboCup Federation[19] que persegue o objectivo de vencer uma equipa humana em 2050[21] e realiza anualmente um encontro internacional desde 1997, RoboCup, e no qual o ISocRob participa desde 1998.

A constante alterações das regras por parte do RoboCup obrigam a um constante melhoramento dos robots em diversas áreas como é o caso da visão, localização, comportamentos e do próprio hardware.

1.5 Ambiente de desenvolvimento

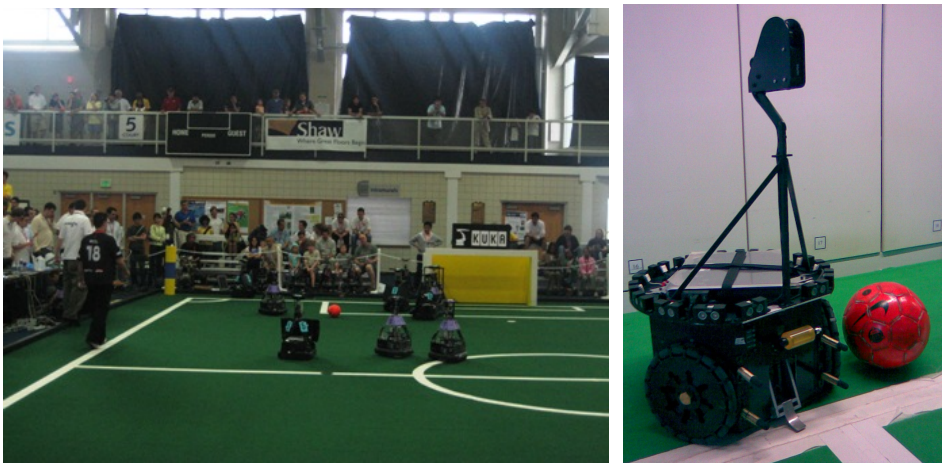
O trabalho desenvolvido utiliza robots da MSL do RoboCup. As regras[22] e dimensões das competições nacionais e internacionais em que o Projecto participa são definidas pelo comité técnico do RoboCup. Neste momento o campo apresenta uma dimensão de 18x12 metros, cor verde, linhas brancas e as balizas distinguem-se pela cor amarela e azul. No cantos existem postes também amarelos e azuis. A bola deve ser laranja. A Figura 1.5.1 representa esta descrição.



(a) Campo(tamanhos não proporcionais). (b) Postes presentes nos cantos.

Figura 1.5.1: Representação do ambiente de desenvolvimento da liga MSL.

A Figura 1.5.2 apresenta o ambiente de desenvolvimento real, neste caso uma fotografia tirada durante um jogo da liga MSL, e os robots desenvolvidos e utilizados no projecto SocRob.



(a) Jogo da liga MSL. (b) Robots do projecto SocRob

Figura 1.5.2: Ambiente real de desenvolvimento da liga MSL.

Durante um jogo de futebol robótico é utilizada uma aplicação designada por *referee box* que envia sinais aos robots indicando o estado do jogo, tal como um árbitro de futebol humano.

Foi desenvolvido no âmbito do projecto SocRob e apoiado no simulador *Webots*[32] um simulador do ambiente descrito acima. Esta aplicação é da maior importância no desenvolvimento de comportamentos porque permite reduzir o tempo desse desenvolvimento. Este simulador permite correr o mesmo código que corre nos robots, o que se apresenta como outra grande vantagem da sua utilização. A Figura 1.5.3 apresenta uma imagem deste simulador.

Figura 1.5.3: Simulador Webots.

1.6 Estado da arte

1.6.1 Redes de Petri

As redes de Petri foram introduzidas por Carl Adam Petri. Apresentam-se como uma ferramenta de modelação matemática e gráfica que tem tomado um papel cada vez mais central no desenho de sistemas em robótica. Tadao Murata[9] faz uma introdução às redes de Petri apresentando-as como *"uma ferramenta para descrever e estudar sistemas de processamento que são caracterizados como sendo concorrentes, assíncronos, distribuídos, paralelos, não deterministas e/ou estocásticos"*. Esta flexibilidade confere a este tipo de redes o poder de representação de sistemas em que é necessário a realização de tarefas de forma paralela, assim como a tomada de um decisão em caso de conflitos, ambos necessários à implementação de comportamentos relacionais. Ainda neste texto o autor apresenta as propriedades que caracterizam uma rede de Petri (alcance(*reachability*), limitação(*boundedness*), vivacidade(*liveness*), reversibilidade(*reversibility*), cobertura(*coverability*), persistência (*persistence*), distância síncrona (*synchronic distance*)) e métodos de análise disponíveis (*coverability tree* e *incidence matrix and state equation*). Na Secção 2.1 são apresentados os conceitos base das redes de Petri.

1.6.2 Redes de Petri em robótica

Tal como referido acima, as redes de Petri surgem cada vez mais como uma ferramenta utilizada para descrever comportamentos de robots ou de uma população de robots. Em [7] e [11] são apresentados alguns formalismos que permitem fazer a ponte entre a ferramenta apresentada por Petri e a sua utilização na modelação desses comportamentos. Lima e Milutinovic[11] construíram um modelo de tarefas robóticas baseados em redes de Petri (RTM) envolvendo uma *framework* que permite uma avaliação quantitativa e qualitativa da tarefa pretendida. Um RTM é apresentado como um conjunto de três tuplos: recursos (robots e tarefas primitivas), eventos e acções primitivas. É assim associado a cada lugar de uma rede de Petri um recurso e a cada transição condições lógicas que definem o evento. Cada tarefa primitiva realiza-se através da execução de acções primitivas. Esta representação permite a definição de macros, ou seja, o próprio lugar de uma rede de Petri pode representar outra rede de Petri que representa a tarefa de mais baixo-nível. A *framework*, tal como referido, apresenta ainda métodos de análise qualitativa e qualitativa. No primeiro caso através da análise das propriedades das redes de Petri, no segundo caso estabelecendo uma função de custo para cada lugar de forma a descobrir qual o caminho ideal para completar uma tarefa, através minimização do custo. Em [7] Ziparo e Iocchi apresentam outra *framework* designada Petri Net Plans (PNP) utilizada para desenvolver redes de Petri que descrevem comportamentos de robots ou multi-robot. Esta ferramenta tem sido utilizada em equipas de futebol robótico e de procura-e-salvamento, como é o caso da equipa SPQR[2]. O PNP estabelece um formalismo para a representação de acções não instantâneas, acções condicionais, falhas, concorrência, interrupções e sincronização num contexto multi-agente, utilizando redes de Petri. Essa definição permite representar os comportamentos em ambientes dinâmicos, parcialmente observáveis e imprevisíveis, como é o caso do futebol robótico. A conjugação de diversos PNP referentes a cada robot, juntamente com a introdução de sincronização, permite o estabelecimento de um *Multi-Agent* PNP (MA-PNP) que represente um comportamento relacional. Através de primitivas de comunicação é possível dividir um MA-PNP atribuindo a cada robot o seu PNP sem perder a consistência de um sistema distribuído, promovendo a análise individual de cada rede.

1.6.3 Redes de Petri no Projecto SocRob

Sendo as tarefas realizadas por robots um assunto cada vez mais presente são diversos, a nível mundial, os projectos nesta área. Os comportamentos relacionais estão muitas vezes associados a robots de procura-e-salvamento e a robots futebolistas, como é este o caso. Como referido, o desenvolvimento desta dissertação insere-se no projecto SocRob que compreende a implementação de uma equipa de futebol robótico. Algumas das soluções presentes nesse projecto são apresentadas em [8] por Lima e Custódio. A equipa SocRob tem a sua equipa

[2] http://spqr.dis.uniroma.1.it

assente numa arquitectura próprio e que será explicada de forma mais pormenorizada na Secção 4.1.

Actualmente grande parte dos comportamentos, com excepção do guarda-redes que utiliza lógica fuzzy, utilizam Redes de Petri para gerir as acções primitivas. Estão disponíveis três tipos de objectos para o seu desenho: acções primitivas, predicados e eventos. As acções primitivas representam tarefas robóticas básicas (andar, chutar, entre outros), os predicados representam circunstâncias que ocorrem ou não(tem a bola, está em posição de remate, entre outros) e os eventos representam acontecimentos despoletados pelo jogo ou pela *referee box*, neste último caso pela *referee box*. Nas redes de Petri as acções e o predicados são representados por lugares e os eventos por transições.

Para a utilização de Redes de Petri foi desenvolvida no âmbito do projecto SocRob uma *framework* para executar as redes de Petri desenhadas. A *framework* recebe ficheiros no formato PNML(Petri Net Markup Language)[29] que descrevem a rede respectiva. Posteriormente a rede vai sendo percorrida à medida que os eventos vão ocorrendo e as acções vão sendo executadas, adicionando e removendo tokens aos locais respectivos.

1.6.4 Comportamentos Relacionais para Robots Futebolistas

No que se refere ao desenvolvimento de comportamentos relacionais foram desenvolvidos no passado vários trabalhos: Vecht[12], Milhinhos e Moura Vaz[13] e Orey[14]. Os dois primeiros utilizando robots MSL e o último utilizando robots da 4LL. Estes três trabalhos têm como característica comum a utilização de sincronização e de comunicação explicita. Os três trabalhos foram desenvolvidos seguindo uma abordagem lógica e tendo como base a Teoria dos Compromissos Conjuntos[6] que é apresentada com mais detalhe na Secção 2.2. Esta teoria exige o estabelecimento inicial de um compromisso que ao ser quebrado necessita de um aviso por parte do robot que vai terminá-lo, ou seja, quando um comportamento relacional está a ser executado todos os robots envolvidos sabem que o estão a executar e quando termina todos os robots sabem que terminou. De outra forma poderia acontecer que um robot estivesse à espera de receber um passe apesar do comportamento já ter terminado para o robot que iria passar a bola, o que não seria desejável. Os dois últimos trabalhos foram desenvolvidos utilizando redes de Petri, enquanto o primeiro utilizou máquinas de estado.

Em [12], foram desenvolvidos dois tipos de comportamentos: passe para um local definido e intercepção de bola, com bastante sucesso. Para a implementação dos comportamentos relacionais foi definido um método assente em três fases: *setup, loop* e *end*. Cada fase é divida em vários estados, obtendo-se sincronização ao garantir que todos os robots se encontram no mesmo estado.

Orey[14] desenvolveu, seguindo o mesmo tipo de implementação, um comportamento individual (*sweep*) e vários comportamentos relacionais (*Attacking Kickoff, Defensive Kickoff, Attacking Throwin*), para robots Aibo, utilizados na 4LL. Apesar da implementação ser difer-

ente da realizada nos outros dois casos, devido às diferenças dos robots, o desenho das redes de Petri tem muito em comum com o que se pretendeu realizar.

Mais uma vez seguindo a mesma implementação, Milhinhos e Moura Vaz[13] apresentam a criação e demonstração de vários comportamentos relacionais utilizados quando a equipa está atacar, tanto estáticos (lançamento, pontapé de baliza, inicio de jogo) como dinâmicos ("tabelinha"). Foi ainda definido um algoritmo que permite definir de forma sistemática a implementação de um novo comportamento. Esse algoritmo engloba o desenho da rede de Petri, a identificação dos vários estados, a determinação das condições para a realização do comportamento e mudança de estado, a definição da sincronização necessária e as condições necessárias à execução de uma acção primitiva presente no comportamento relacional.

Capítulo 2

Fundamentos

Neste capítulo são apresentados os fundamentos teóricos que sustentam este trabalho. Na Secção 2.1 é introduzida a representação por Redes de Petri utilizada na descrição dos comportamentos relacionais, dando-se especial ênfase às Redes de Petri estocásticas. Na Secção 2.2 é apresentada a Teoria dos Compromissos Conjuntos, desenvolvida por Cohen e Levesque[6], que organizou e uniformizou várias ideias no que diz respeito ao trabalho em equipa.

2.1 Redes de Petri

Um sistema de eventos discreto[2] pode ser definido como um sistema dinâmico com espaço de estados discretos cuja evolução é definida pela ocorrência de eventos assíncronos que alteram o seu estado. O estudo deste tipo de sistemas tornou-se extremamente importante nos últimos anos, nomeadamente na área da robótica. As redes de Petri ([9] e [13])apresentam-se como uma ferramenta útil na descrição deste tipo de sistemas.

Uma rede de Petri é constituída por lugares, transições e arcos direccionados que unem um lugar a uma transição e vice-versa. Os lugares representam-se por círculos e correspondem a acções, predicados ou tarefas. Cada lugar contém um número maior ou igual a zero de tokens, representados por pontos. A presença de um token num lugar indica que aquele recurso está disponível ou aquela tarefa está a ser executada. As transições representam-se por rectângulos ou rectângulos, para RdP ordinárias ou estocásticas, respectivamente, e correspondem aos eventos. A cada arco está associado um peso maior ou igual a um. A Figura 2.1.1 apresenta uma rede de Petri muito simples.

Em [9], Tadao Murata faz uma introdução às redes de Petri e apresenta as propriedades as caracterizam.

2.1.1 Definição matemática

Formalmente uma rede de Petri é definida como um quíntuplo $PN = (P, T, F, W, M_0)$ onde:

- $P = \{p_1, p_2, ..., p_n\}$ é um conjunto finito de lugares,

- $T = \{t_1, t_2, ..., t_m\}$ é um conjunto finito de transições,
- $F \subseteq (P \times T) \bigcup (TxP)$ é o conjunto de arcos,
- $W : F \longrightarrow \{1, 2, 3, 4, ...\}$ é a função que define os pesos dos arcos,
- $M_0 : P \longrightarrow \{0, 1, 2, 3, ...\}$ define o número de tokens inicialmente em cada lugar (marcação inicial),
- $P \bigcap T = \emptyset e P \bigcup T \neq \emptyset$.

Uma Rede de Petri sem uma marcação inicial é definida por N=(P,T,F,W), definindo-se a correspondente RdP marcada por *(N,* M_0*).*

2.1.2 Regra de Transição

A regra de transição de estados para uma Rede de Petri é dada por:

- Uma transição *t* diz-se activa se em cada lugar de entrada *p* existem pelo menos *w(p,t)* tokens, sendo *w(p,t)* o peso do arco que une *p* a *t*.
- Uma transição activa pode ou não disparar (dependendo se o evento correspondente ocorreu ou não).
- Se uma transição *t* disparar remove *w(*p_i*,t)* tokens de cada um dos lugares de entrada e adiciona *w(t,*p_o*)* tokens a cada lugar de saída.

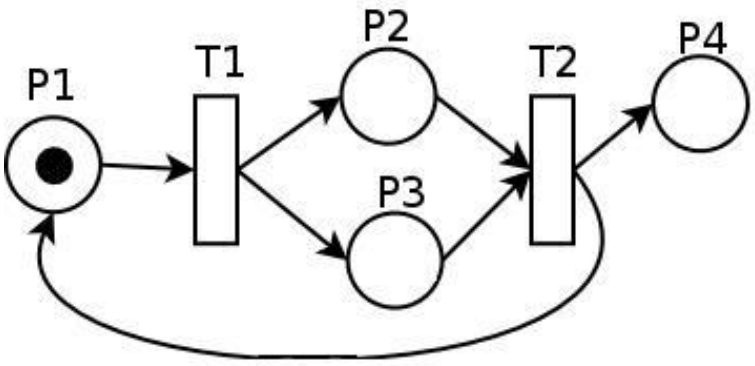


Figura 2.1.1: Exemplo de uma Rede de Petri.

2.1.3 Redes de Petri estocásticas

As redes de Petri na sua forma clássica, apresentada anteriormente, são de extrema utilidade para uma análise qualitativa e/ou lógica de forma a excluir a existência de exclusões mútuas, deadlocks, entre outros. Contudo, a análise a que este trabalho se propõe é de cariz quantitativo o que torna indispensável a introdução de tempo nas redes. Este elemento será importantíssimo em aspectos como, por exemplo, a modelação da transmissão de mensagens

entre os robots. Desta forma a redes analisadas são Redes de Petri Estocásticas Generalizadas (Generalized Stochastic Petri Nets - GSPN), que são uma versão alargada das Redes de Petri Estocásticas. Estes tipos de redes são apresentado de seguida.

Definição

Numa Rede de Petri Estocástica (SPN) - note-se que se tratam de redes ligeiramente diferentes das GSPN, tal como será explicado - o tempo é naturalmente associado com as transições, apesar de existirem na literatura opções em que se associa tempo aos lugares. Uma SPN define-se[5] formalmente como um sêxtuplo $PN = (P, T, F, W, M_0, F)$ onde (P, T, F, W, M_0) é uma rede de Petri marcada e F é uma função com domínio $(R[M_0] \times T)$ que associa a cada transição em cada marcação alcançável uma variável aleatória. As SPN utilizadas são Redes de Petri de Tempo Exponencial, ou seja, redes em que os tempos de disparo das transições são distribuídos exponencialmente.

Variável Aleatória de Distribuição Exponencial

Uma variável aleatória de distribuição exponencial[25] na teoria de probabilidade e estatística representa uma classe de distribuição de probabilidade contínua. É frequentemente utilizada para modelar o tempo entre eventos independentes que acontecem a uma taxa média constante. A sua função de densidade de probabilidade apresenta a fórmula:

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{2.1}$$

onde λ > 0 é o parâmetro da distribuição, também designado por taxa da distribuição. A distribuição estende-se no intervalo [0, ∞]. Uma variável aleatória X exponencial é representada por X $\sim$ Exponencial(λ).

O valor esperado de uma variável aleatória exponencial é dado por:

$$\mathrm{E}[X] = \frac{1}{\lambda}. \tag{2.2}$$

Fazendo a ligação a uma aplicação real: se um telefone tem uma taxa de recepção de telefonemas de 2 por hora então pode-se afirmar que se espera receber em média um telefonema a cada meia hora. A sua variância apresenta o seguinte valor:

$$\mathrm{Var}[X] = \frac{1}{\lambda^2}. \tag{2.3}$$

Este tipo de variável aleatória não guarda memória, ou seja,

$$P(T > s + t \mid T > s) = P(T > t) \text{ para todos } s, t \geq 0. \tag{2.4}$$

Mais concretamente, a probabilidade de se ter de esperar mais 10 segundos pelo próximo evento é independente do tempo que já tenha passado:

$$P(T > 35 \mid T > 25) = P(T > 10). \tag{2.5}$$

Redes de Petri de Tempo Exponencial

Uma classe importante das SPN são as Redes de Petri de Tempo Exponencial (ETPN) em que todos os tempos de disparo das transições são distribuídos exponencialmente. Uma ETPN[5] é um sêxtuplo $PN = (P, T, F, W, M_0, F)$ onde (P, T, F, W, M_0) é uma rede de Petri e $F : (R[M_0] \times T) \rightarrow R$ associa a cada transição *t* ϵ T em cada marcação alcançável uma variável exponencial de taxa $F(M, t)$.

Seja M_i uma marcação alcançável e T_i o conjunto de transições activas nessa marcação. Neste caso, e para este tipo de redes, a probabilidade de $t_j \epsilon T_i$ disparar é dada por:

$$\frac{F(M_i, t_j)}{\sum_{t_k \epsilon T_i} F(M_i, t_k)} \tag{2.6}$$

Pode-se confirmar aquilo que seria retirado intuitivamente, uma transição cuja variável aleatória apresenta uma maior taxa terá uma maior probabilidade de disparar.

Redes de Petri Estocásticas Generalizadas

As Redes de Petri Estocásticas Generalizadas (GSPN)[5] constituem a classe mais extensa de SPNs. São redes de Petri cujas transições pertencem a uma de duas classes: transições imediatas ou transições exponenciais. Desta forma evita-se que a análise associe tempos a transições que não precisem.

Uma GSPN é definida formalmente como um oito-tuplo (P, T, F, W, M_0, F, S) onde:

- (P, T, F, W, M_0) é uma rede de Petri,
- T está dividido em dois conjuntos: T_I de transições imediatas e T_E de transições exponenciais,
- $F : (R[M_0] \times T_E) \rightarrow R$ é uma função de disparo que associa a cada $t \epsilon T_E$ em cada $M \epsilon R[M_0]$ uma variável aleatória com a taxa $F(M, t)$,
- Cada $t \epsilon T_I$ tem tempo de disparo zero em todas as marcações alcançáveis,
- S é um conjunto, possivelmente vazio, de elementos denominados *random switches* que associam probabilidades a subconjuntos de transições imediatas que possam entrar em conflito.

Uma transição imediata é representada por uma linha ao passo que uma transição exponencial é representada por um rectângulo.

A regra de disparo de transições, alcançada uma marcação M_i com o conjunto T_i de transições activas, é dada por:

- caso T_i apenas contenha transições exponenciais a transição $t_j \epsilon T_i$ dispara com a probabilidade indicada em (2.6), dado tratar-se de uma situação idêntica a uma ETPN. Neste caso está-se perante uma marcação tangível.

- caso T_i contenha exactamente uma transição imediata então disparará essa transição,
- caso T_i seja um conjunto contendo duas ou mais transições imediatas a probabilidade de cada transição disparar é dada pelo *random switch* correspondente. Note-se que tanto neste caso como no anterior T_i pode conter transições exponenciais em conjunto com transições imediatas, sendo as primeiras preteridas em função das últimas. Está-se perante casos de transições voláteis dado que ocorrem imediatamente.

2.2 Teoria dos Compromissos Conjuntos

O trabalho desenvolvido por Cohen e Levesque[6] serve de base a muitos trabalhos nesta área([12], [13] e [14]). Organizou e uniformizou várias ideias no que diz respeito ao trabalho em equipa, apoiando o desenvolvimento de comportamentos relacionais. Uma importante ideia é a distinção entre sincronismo e trabalho em equipa. Apesar do trabalho em equipa exigir sincronização, a existência de sincronização não garante que estamos perante tarefas cooperativas. Um exemplo dado é o caso do trânsito numa cidade. Apesar de ser síncrono não representa trabalho em equipa entre os condutores. Distingue-se assim cooperação de coordenação. A Teoria dos Compromissos Conjuntos é baseada na ideia de que agentes com um mesmo objectivo estabelecem um compromisso. Posteriormente, através de comunicação, informam os outros agentes do seu estado, desta forma é obtido o sincronismo. São apresentadas de seguida as definições em que assenta a teoria[13].

2.2.1 Definições

- Objectivo persistente : um agente tem um objectivo persistente relativo a q para atingir p, se as condições seguintes se verificarem:
  - O agente acredita que p é momentaneamente falso;
  - O agente quer que p venha a ser verdadeiro;
  - Assume-se como verdade (e o agente sabe disso) que a segunda condição se irá manter até que este venha a acreditar que p é verdadeiro, ou que nunca virá a ser verdade ou que q é falso.
- Objectivo de baixa prioridade : um agente terá um objectivo de baixa prioridade relativo a q e respeitante a uma equipa que acredita em p se as condições seguintes se verificarem:
  - O agente tem um objectivo normal de modo a acreditar p, isto é, o agente ainda não acredita que p é verdadeiro mas tem p como um objectivo que se virá a tornar verdadeiro;
  - O agente acredita que p é verdadeiro, nunca será verdadeiro, ou é irrelevante (ou seja, q é falso), mas tem um objectivo que lhe diz que o estado de p é acreditado conjuntamente por todos os membros da equipa;

- Objectivo persistente relacional : Uma equipa de agentes tem um objectivo persistente relacional relativo a q para atingir p no caso de :
  - Todos acreditarem que p é momentaneamente falso;
  - Todos querem que p venha a ser verdadeiro;
  - É verdade (e é conhecimento comum) que até todos concordarem que p é verdadeiro, ou que p nunca vai ser verdadeiro, ou que q é falso, todos os agentes irão continuar a acreditar que cada um deles tem um objectivo de baixa prioridade relativo a q e respeitante a toda a equipa;

2.2.2 Aplicação

O estabelecimento de um compromisso que visa um objectivo começa com a aquisição de um mesmo objectivo persistente por parte de cada agente envolvido. Mas esta ocorrência não basta para o compromisso ser estabelecido. A simples existência de um objectivo persistente comum apenas garante que cada robot irá perseguir a sua meta até a ter atingido ou perceber que é inatingível, não garantindo que no caso do objectivo se tornar inválido o agente informe os seus companheiros de equipa. Nasce assim a necessidade de se criar um outro tipo de objectivo, designado por objectivo de baixa prioridade. Neste caso os agentes só deixam o compromisso se todos estiverem de acordo quanto ao sucesso ou insucesso da tarefa, comprometendo-se a ratificar os companheiros de tal decisão. Cria-se assim o conceito de objectivo persistente global.

Capítulo 3

Comportamentos Relacionais

Este capítulo apresenta os conceitos base relacionados com comportamentos relacionais. Na Secção 3.1 faz-se a introdução ao tema apresentando a sua definição. A Secção 3.2 explica a importância da existência de comunicação neste tipo de aplicações, apresentando as diferentes opções existentes. As Secções 3.3 e 3.4 explicam como é estabelecido e gerido o compromisso necessário a comportamentos relacionais, respectivamente. Por fim, na Secção 3.5 é apresentada a razão pela qual é fulcral implementar-se sincronização em comportamentos envolvendo dois ou mais robots.

3.1 Definição

Um comportamento é um conjunto de acções que realizadas em conjunto, de forma paralela ou em série, têm como objectivo a execução de uma determinada tarefa. A sequência de acções depende de condições que se referem ao robot e à sua envolvente. A tarefa de levar a bola até à baliza é um exemplo de um comportamento, contendo acções como apanhar a bola, driblar ou rematar. Conforme anteriormente referido, o momento de execução de cada acção é definido pelas condições envolventes: o robot não poderá chutar a bola de esta não estiver na sua posse, por exemplo.

Os comportamentos podem-se dividir em três classes:

- **Comportamentos Organizacionais**: este tipo de comportamentos promove a organização da equipa em campo e dizem respeito ao papel que cada jogador tem na táctica da equipa (p.e., defesa, atacante, guarda-redes).
- **Comportamentos Relacionais**: são comportamentos em que estão envolvidos dois ou mais robots para realizarem uma tarefa em conjunto e que executam as acções de forma paralela e sincronizada(p.e., passar a bola, não irem simultaneamente à bola).
- **Comportamentos Individuais**: são comportamentos que envolvem apenas um actor (p.e., defender a baliza, apoiar ataque ,atacar).

Muitas vezes os comportamentos relacionais e organizacionais são composições de comportamentos individuais que, em conjunto com os comportamentos do parceiro de equipa, ganham significado como um todo. A escolha de qual tipo de comportamento deve ser realizado em cada momento depende da arquitectura de software da equipa, nomeadamente da forma como foi implementada a sua máquina de decisão. No Capítulo 4 será abordado esse assunto no que diz respeito ao projecto SocRob.

Os comportamentos relacionais apresentam novos requisitos. Ao não se tratar um robot de forma isolada remete-se para a necessidade de se realizar um compromisso entre os seus intervenientes e de se utilizar sincronização, ambas apoiados numa forma de comunicação que deverá ser eficaz. Estes temas são abordados de seguida.

3.2 Comunicação

A comunicação é um ponto importantíssimo quando se trata de comportamentos que envolvam mais que dois robots. Um comportamento, por mais bem desenhado que esteja, na ausência de uma comunicação eficaz será conduzido ao insucesso. Imagine-se duas pessoas a transportar uma placa de madeira: mesmo sendo perfeitamente capazes de lhe pegar, senão comunicarem durante o transporte será muito complicado que o consigam fazer com sucesso.

Distinguem-se dois tipos de comunicação[26]: implícita e explícita.

O primeiro género é, por agora, uma meta ainda relativamente longe de alcançar no futebol robótico. Comunicando de forma implícita os intervenientes retiram mensagens e decidem com base no que observam. Por exemplo, no caso de um passe o robot que vai receber a bola começaria a desmarcar-se quando visse que o companheiro de equipa ia chutar, tal como acontece no futebol humano. Este tipo de comunicação apresenta a vantagem de ser simples e robusta, se funcional, e não torna o robot dependente dos suportes físicos de comunicação. Contudo, o robot só terá conhecimento do que consegue "sentir".

Pelos limites actuais referidos a opção do tipo de comunicação utilizado no futebol robótico pende para o lado da comunicação explicita. Neste caso, a comunicação é feita usando um qualquer suporte físico, normalmente uma rede wireless, através da transmissão de mensagens. Ao robot é indicado explicitamente o que se passa, daí a designação, não fazendo parte das suas tarefas retirar mensagens da acções que vê o seu companheiro de equipa realizar. Recorrendo de novo ao exemplo do passe, o robot que vai passar a bola enviaria uma mensagem ao seu companheiro quando chutasse a bola, desta forma o robot encarregue da recepção estaria pronto para a realizar. Como grande vantagem a comunicação explícita apresenta uma implementação relativamente fácil, apresentado a desvantagem de se fazer depender do suporte físico de comunicações o que põe em causa a sua robustez.

Para representar a comunicação entre os dois robots utilizando uma rede de Petri são criado dois lugares que representam a acção de enviar e receber a mensagem, respectivamente, e uma transição representando o evento correspondente ao envio da mensagem propriamente dito. A Figura 3.4.2 apresenta um exemplo da utilização deste formalismo. Esta implementação permite que posteriormente cada rede de Petri referente a cada robot seja facilmente identificável, bastando para isso retirar essa transição[15]. Por exemplo, no caso da Figura 3.4.2, se se eliminar as transições identificadas como *msg_sent* ficamos com duas RdP independentes. A da esquerda corresponderia à RdP executada pelo robot que faz o passe e a da direita à RdP executado pelo robot que recebe a bola.

A comunicação é um elemento fundamental do estabelecimento e gestão do compromisso, bem como da sincronização, partes integrantes de qualquer comportamento relacional e assuntos abordados nas secções seguintes.

3.3 Estabelecimento do compromisso

Seguindo a Teoria dos Compromissos Conjuntos desenvolvida por Cohen e Levesque[6] e tendo em conta que um comportamento relacional envolve dois ou mais robots na perseguição de um objectivo, torna-se indispensável o estabelecimento de um compromisso. O estabelecimento do compromisso permite que desta forma cada robot saiba que é parte integrante de um comportamento que envolve mais companheiros de equipa, ou seja, que está comprometido.

O estabelecimento do compromisso atravessa várias etapas até que os robots estejam comprometidos. De seguida é apresentado esse processo tendo em conta que R1 representa o robot que inicia o processo e R2 o robot que recebe o pedido para estabelecer um compromisso com R1:

1. **R1 - Pedido para estabelecimento do compromisso:** o robot R1 envia um convite aos companheiros de equipa para estabelecer um compromisso para um qualquer comportamento relacional.

2. **R2 - Análise e resposta ao pedido em 1:** os robots recebem o pedido e verificam se estão em condições de entrar no compromisso, enviando uma mensagem de volta em caso afirmativo.

3. **R1 - Análise e respostas às candidaturas em 2:** o robot R1 recebe as candidaturas de resposta ao seu pedido, analisa-as e responde com uma aceitação ou rejeição, conforme o caso. Neste caso considera-se que o robot R2 foi escolhido.

4. **R2 - Recepção da resposta à candidatura em 2:** o robot recebe a sua resposta que, em caso positivo, significará que vai entrar no compromisso com R1.

5. **R1 e R2 consideram-se comprometidos:** esta fase dependerá da implementação e do equilíbrio que se procura nessa mesma implementação entre atraso e robustez. Os robots poderão assumir, no caso de R1 quando envia a resposta e no caso de R2 quando a recebe, que estão comprometidos ou poderão ser trocadas mais mensagens para garantir o sucesso da processo. No primeiro caso promove-se a rapidez do estabelecimento do compromisso, enquanto que no segundo se promove a robustez.

A Figura 3.3.1 representa a sequência descrita acima. As reticências na figura representam a existência de várias implementações possíveis a partir desse momento, tal como indicado no último ponto da enumeração anterior.

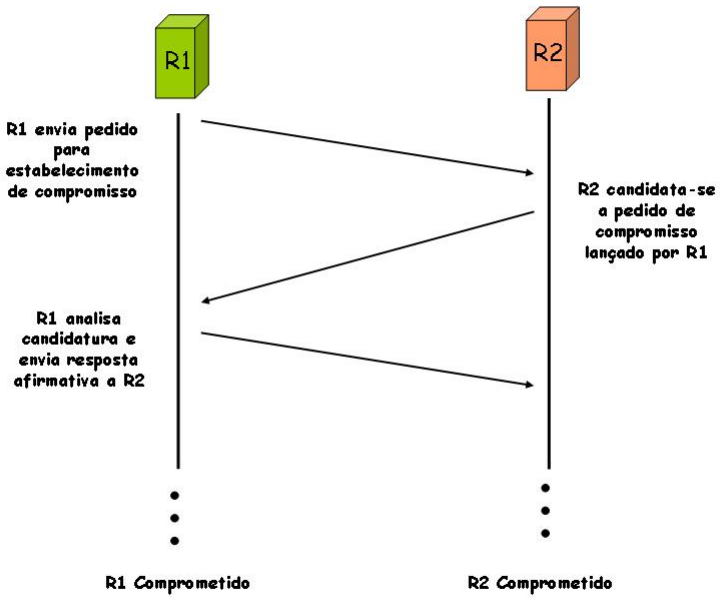


Figura 3.3.1: Etapas presentes no estabelecimento de compromisso.

Existem inúmeros protocolos que permitem implementar o processo de estabelecimento de um compromisso. São exemplo, o two-phase commit protocol[27] e o three-phase commit protocol[28]. Contudo, estes protocolos não satisfazem todos os requisitos necessários a este trabalho. No primeiro caso estamos perante um protocolo bloqueante, enquanto que no segundo caso não é possível o compromisso ser efectuado se algum dos elementos, mesmo os elementos não escolhidos, abortar durante o processo de estabelecimento. Sendo assim, optou-se por uma implementação própria, descrita na Secção 4.2.1.

3.4 Gestão do compromisso

Da execução de um comportamento relacional podem resultar varias consequências para o jogo. Contudo, do ponto de vista do objectivo do comportamento considerem-se as duas situações elementares: o sucesso ou insucesso do comportamento. Tanto num caso como no outro os dois robots intervenientes no comportamento devem estar de acordo quanto ao estado do compromisso, ou seja, se algum dos robots quebrar o compromisso, quer por sucesso ou insucesso na execução da tarefa corrente, deve informar o outro. Desta forma nenhum dos robots ficará bloqueado num comportamento inexistente, residindo neste ponto a necessidade da existência de um compromisso.

A Figura 3.4.2 apresenta um exemplo de uma Rede de Petri para um comportamento relacional enquanto a Figura 3.4.1 representa a respectiva gestão do compromisso[15]. Os estados azuis e verdes da Figura 3.4.2 correspondem aos estados da mesma cor da Figura 3.4.1, ou seja, cada acção primitiva está ligada a uma rede de gestão do compromisso própria.

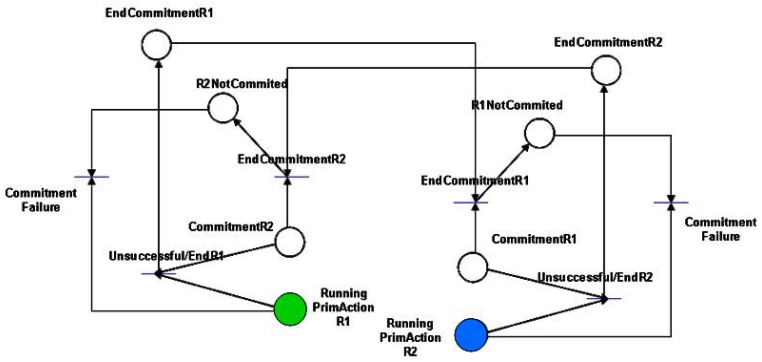


Figura 3.4.1: Rede de Petri que gere compromisso[15].

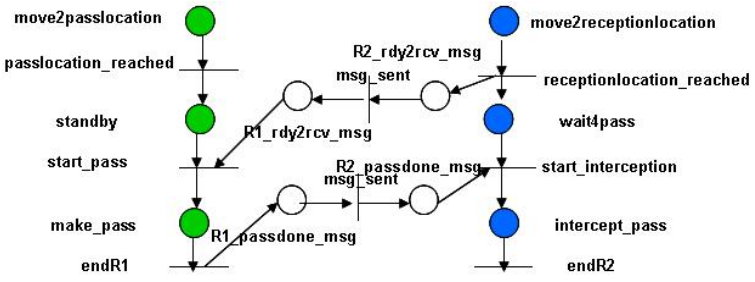


Figura 3.4.2: Rede de Petri para um comportamento relacional[15].

São necessários três lugares numa rede de gestão do compromisso:

- **Commitment <partner name>:** caso o token esteja neste lugar indica que o parceiro continua comprometido. Este lugar é complementar do <partner name>NotCommitted. Estabelecido o compromisso o token é colocado neste lugar.
- **End Commitment <own name>:** este lugar recebe um token quando o comportamento

não teve sucesso e portanto não existem condições para o robot continuar em compromisso.

- **<partner name>NotCommitted:** este lugar indica que o robot já não está comprometido o que leva à interrupção da acção primitiva. O token é colocado neste lugar caso o parceiro tenha tido insucesso no comportamento e quebrado o compromisso.

  A Secção 4.2.2 descreve a implementação da gestão do compromisso adaptada à arquitectura funcional de software existente nos robots e que teve como base o que foi descrito nesta secção.

3.5 Sincronização

Um requisito essencial a um comportamento relacional é a existência de sincronização. A sequência de acções que forma um comportamento pode ser correctamente executada pelos seus intervenientes, mas se não existir sincronismo entre os robots nessas acções o sucesso do comportamento poderá ser posto em causa. Relembrando o exemplo de duas pessoas a transportar uma tábua, se a pessoa que vai à frente andar demasiado, sem esperar pelo outro transportador, a tábua acabará por cair. Do mesmo modo, se num passe o robot passar a bola antes do receptor estar preparado a jogada perder-se-á.

A sincronização é conseguida através da transmissão de mensagens, ou seja, utilizando comunicação explicita, como referido na Secção 3.2. Na implementação do sincronismo do comportamento este facto deve ser tido em conta, dado que torna os pontos de sincronismo um alvo relativamente fácil de falhas. Sendo assim, um comportamento não deve ter muitos pontos de sincronismo e qualquer um deles, em caso de falha, não deve bloquear a acção do robot.

A Secção 4.2.3 descreve a implementação da sincronização para a arquitectura de software do projecto SocRob.

Capítulo 4

Implementação

Este capítulo apresenta a implementação dos conceitos introduzidos no Capítulo 3 na arquitectura de software desenvolvida no projecto SocRob. A Secção 4.1 apresenta-nos essa mesma arquitectura. A Secção 4.2 explica a implementação propriamente dita, referindo o algoritmo adoptado para criar comportamentos e a forma como foi implementado o estabelecimento do compromisso, a gestão do compromisso e a sincronização.

4.1 SocRob - Arquitectura MeRMaID

A implementação dos comportamentos relacionais teve como plataforma a arquitectura de software desenvolvida no projecto SocRob, designada por MeRMaID, Multiple-Robot Middleware for Intelligent Decision-making [16] e [17]. A plataforma é descrita nesta secção.

A plataforma MeRMaID apresenta-se como uma arquitectura de software para equipas de multi-robots desenvolvida para proporcionar um ambiente de programação sistemático para comportamentos de alto-nível, que simultaneamente torna o desenvolvimento de novos comportamentos simples e conduz a um código facilmente mantido.

4.1.1 Entidades

A principais entidades da plataforma MeRMaID[17] são: *roles* (papéis), *behaviors* (comportamentos), *primitive actions* (acções primitivas), *navigation primitives* (primitivas de navegação), *predicates* (predicados) e *events* (eventos). As suas definições são apresentadas de seguida:

- **Papel:** um papel é um subconjunto de comportamentos sobre o conjunto total de comportamentos disponíveis. Permitem definir a estrutura organizacional da equipa. (p.e. defesa, atacante);

- **Comportamento:** um comportamento é um conjunto de acções primitivas cuja sequência executa uma dada tarefa. Podem ser representados através de máquinas de estado, redes de petri ou de um algoritmo de decisão que utilize lógica fuzzy;

- **Acção Primitiva:** uma acção primitiva é o elemento atómico de um comportamento cuja decomposição é impossível. Usualmente consiste num ciclo STA (Sense-Think-Act) que realiza uma tarefa básica (p.e. mover-se para a bola, cobrir a baliza, rematar);
- **Primitiva de navegação:** é um algoritmo de navegação que, tendo como base a postura actual e a postura desejada para o robot, calcula a velocidade a impor às rodas do robot para mover o robot evitando obstáculos;
- **Predicado:** predicados são variáveis booleanas que indicam o estado do mundo ou do robot em relação ao mundo. (p.e. se o robot vê a bola, se o jogo está parado). A avaliação dos predicados permite tomar decisões sobre os comportamentos ou acções a executar;
- **Evento:** um evento indica a mudança de um estado de um predicado, de verdadeiro para falso ou de falso para verdadeiro. Ao contrário de um predicado, que armazena o valor de uma determinada variável, um evento é instantâneo.

4.1.2 Estrutura

O diagrama de blocos que representa a arquitectura encontra-se na Figura 4.1.1.

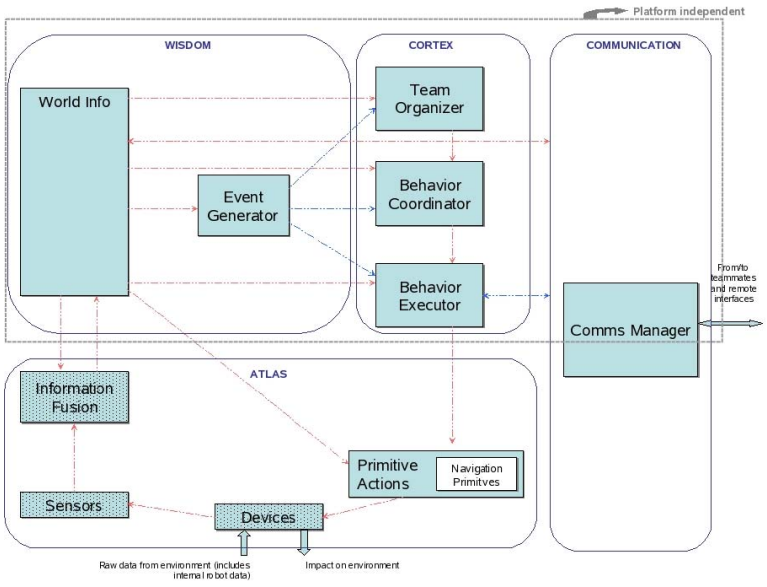


Figura 4.1.1: Diagrama de blocos da arquitectura de software MeRMaID[16].

A arquitectura está dividida em quatro componentes, cada uma constituída por sub-componentes:

- **ATLAS:** é responsável pelas tarefas que dizem respeito ao ambiente: sentir e agir. Engloba os seguintes subcomponentes: dispositivos, sensores, fusão de informação, acções primitivas e primitivas de navegação.
- **WISDOM:** armazena informação sobre o mundo. Essa informação é obtida através dos sensores, de mensagens vindas dos outros robots ou da referee box. A informação re-

cebida pelo robot inicialmente é declarada no ficheiro WorldInfoInit.xml e posteriormente é feito o seu *parsing*.

- **CORTEX:** neste bloco ocorre o processo de tomada de decisão sobre as acções e comportamentos a executar. Essa decisão é tomada com base na informação armazenada no módulo WISDOM. O Cortex está divido em três níveis:
  - **Team Organizer:** responsável pelas decisões tácticas da equipa. Atribui um papel (*role*) a cada robot.
  - **Behavior Coordinator:** responsável pela decisão de qual comportamento executar dentro de cada papel.
  - **Behavior Executor:** responsável pela decisão de qual acção primitiva deve correr dentro de cada comportamento.
- **COMMUNICATION:** responsável pela comunicação com os outros robots e com a referee box.

4.1.3 Redes de Petri

Os papéis, comportamentos e acções primitivas podem ser desenhados utilizando várias ferramentas de modelação. A plataforma MeRMaID suporta a execução de máquinas de estado, algoritmos de lógica Fuzzy ou Redes de Petri, utilizando este trabalho a última ferramenta.

A facilidade de desenvolver comportamentos apenas a alto-nível e permitida ao utilizador por esta arquitectura funcional reside na existência de uma ferramenta que executa as Redes de Petri dos três níveis do Cortex. Como tal, o desenho das redes obedece a certas regras:

- Caso os lugares tenham a designação **<action.>**nome representam papéis, comportamentos ou acções primitivas, consoante pertençam a uma rede do *Team Organizer*, do *Behavior Coordinator* ou do *Behavior Executor*, respectivamente. A presença no token nesse lugar indica que a entidade correspondente está a ser executada;
- Caso os lugares tenham a designação **<predicate.>**nome representam um predicado. Caso o predicado seja verdadeiro é colocado um token no seu lugar, caso contrário é colocado um token no lugar complementar, designado por **<predicate.>**NOT_nome . Tal como referido, um predicado expressa a relação do robot com o mundo, desta forma a *framework* que executa as redes não pode alterar directamente o seu valor ao colocar ou retirar um token num lugar deste género. Para que seja seguido este formalismo todas as transições que tenham um predicado como lugar de *input* têm-no como lugar de *output*;
- Caso o nome de um lugar seja precedido por **<macro>** esse lugar representa, tal como o nome indica, uma macro. Uma Rede de Petri para poder ser usada como macro tem

que ter os lugares especiais *GOAL_REACHED* e *GOAL_NOT_REACHED* para indicar o sucesso ou insucesso da macro. Na rede onde a macro é integrada deverão existir transições com os mesmo nomes e alimentadas pela macro, para fazer a rede evoluir no sentido do sucesso ou insucesso da macro executada;

- As transições podem ser imediatas ou associadas a eventos. Caso sejam associadas a eventos o seu nome deve ser precedido por **<event.>**;
- Um lugar sem nenhum dos prefixos indicados anteriormente denota uma lugar de memória não sendo responsável por qualquer execução.

É importante referir que todas as redes de Petri, com excepção das macros que correm sempre de início, estão a ser corridas em *background* pela *framework*. Sendo assim, é importante ter o cuidado de fechar as RdP desenhadas para que se mantenham num estado consistente e prontas a serem executadas.

O editor de Redes de Petri, neste caso o JARP[30], permite gravar as redes no formato PNML[29]. Posteriormente é feito o *parsing* desses ficheiros para que possam ser executados pela *framework* referida nesta secção.

4.2 Implementação

4.2.1 Estabelecimento do compromisso

Na Secção 3.3 introduziu-se o conceito de estabelecimento do compromisso e as etapas comuns a grande parte das implementações de algoritmos deste género. Nesta secção é apresentada a solução encontrada e como foi feita a sua integração na arquitectura funcional do projecto SocRob.

As Figuras 4.2.1 e 4.2.2 são um diagrama representativo da solução encontrada para o estabelecimento do compromisso.

Pela análise dos diagramas, e designando R1 o robot que lança o concurso e R2 a restante equipa, pode-se constatar que o estabelecimento do compromisso seguirá os seguintes passos:

1. **R1 - Pedido para estabelecimento do compromisso:** o robot R1 envia um convite aos companheiros de equipa para estabelecer um compromisso para um qualquer comportamento relacional.
2. **R2 - Análise e resposta ao pedido em 1:** o robot recebe o pedido e verifica se está em condições de entrar no compromisso, enviando uma mensagem de volta em caso afirmativo. Fica então à espera da resposta à sua candidatura.
3. **R1 - Análise e respostas às candidaturas em 2:** o robot R1 recebe as candidaturas de resposta ao seu pedido, durante um certo período. Findo esse período de tempo,

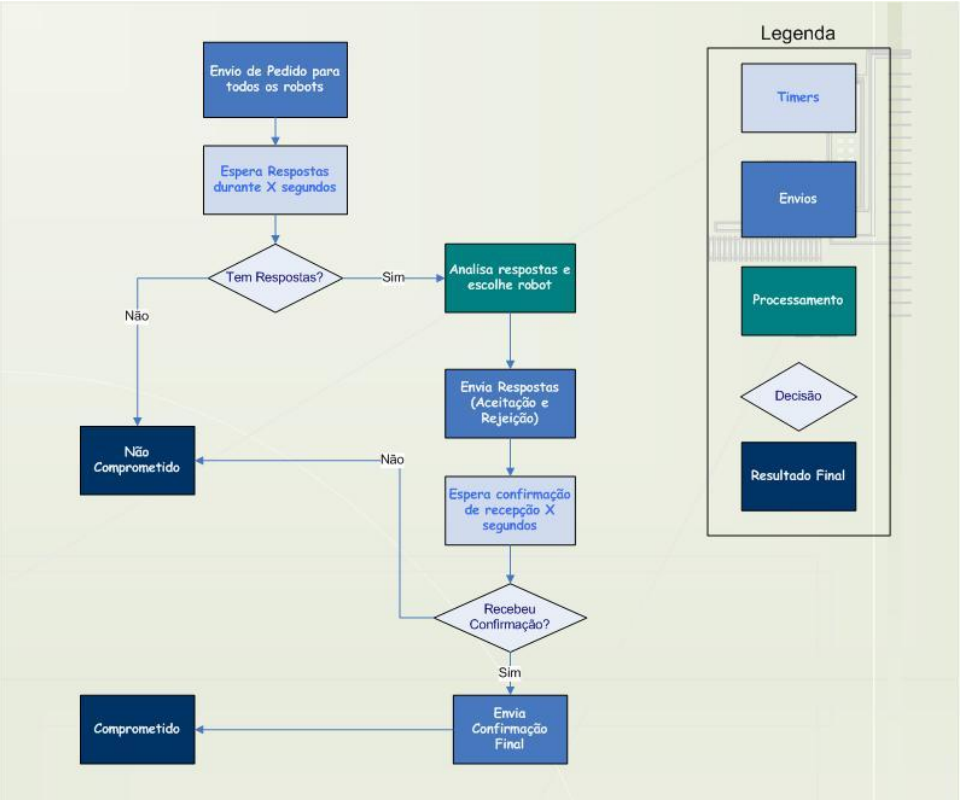


Figura 4.2.1: Diagrama de blocos para pedido de estabelecimento de compromisso.

analisa-as e responde com uma aceitação ou rejeição, conforme o caso. Fica então à espera da mensagem de compromisso de R2. Se não tiverem existidos candidaturas, considera-se não comprometido.

4. **R2 - Recepção da resposta à candidatura em 2:** o robot espera a sua resposta que, em caso positivo, significará que vai entrar no compromisso com R1 e em caso negativo significa que não obteve sucesso. No primeiro caso envia uma mensagem a R1 indicando que está comprometido, ficando à espera de uma mensagem do mesmo tipo.

5. **R1 - Recepção da mensagem de compromisso**: o robot recebe a mensagem enviada em 4, envia uma mensagem idêntica de volta, e considera-se comprometido.

6. **R2 - Recepção da mensagem de compromisso**: o robot recebe a mensagem de compromisso esperada e considera-se comprometido.

Em muitos pontos foi necessário a implementação de temporizadores para evitar que o robot fique bloqueado à espera de uma mensagem.

A Figura 4.2.3 apresenta a Rede de Petri para o estabelecimento de compromisso entre dois robots e que corresponde à descrição anterior.

A rede representada não é a rede executada pela ferramenta JARP referida na Secção 4.1.3, devendo sofrer alterações para que cada parte da rede (a parte do robot que lança o concurso e a parte do robot que aceita o compromisso) seja executada individualmente por

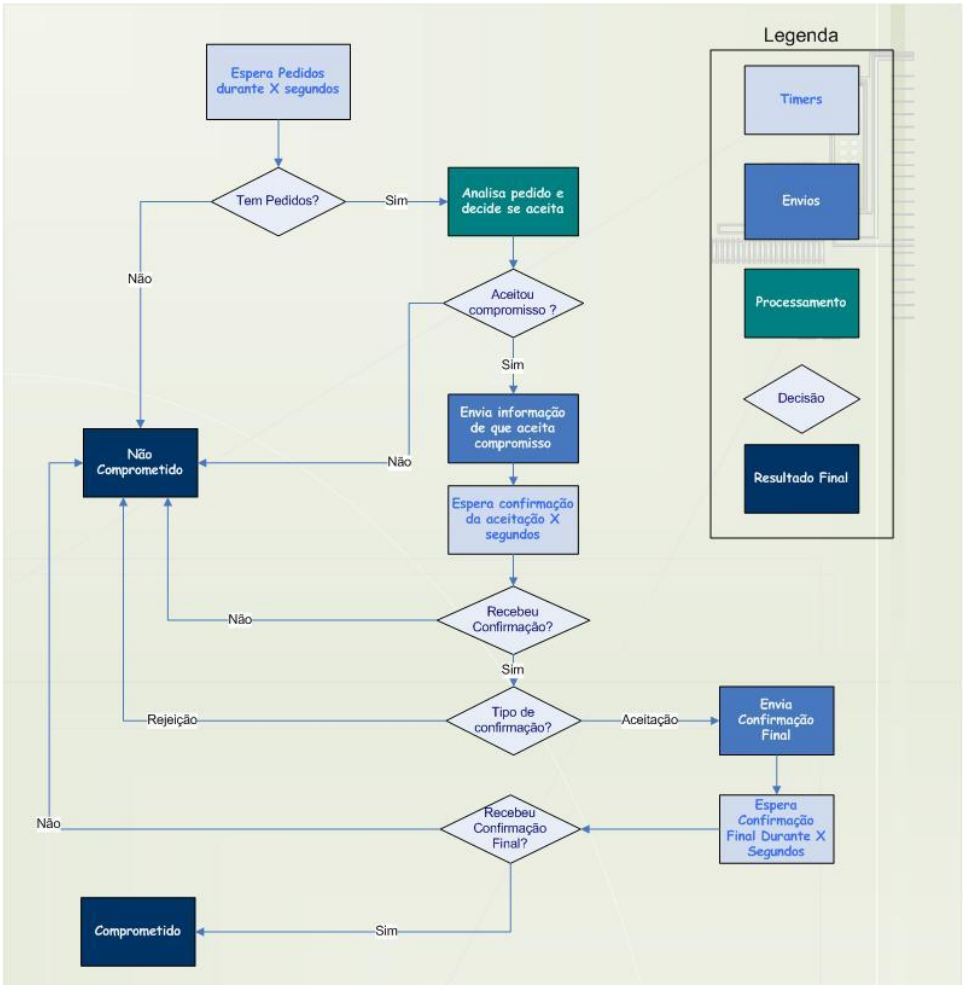


Figura 4.2.2: Diagrama de blocos para aceitação de estabelecimento de compromisso.

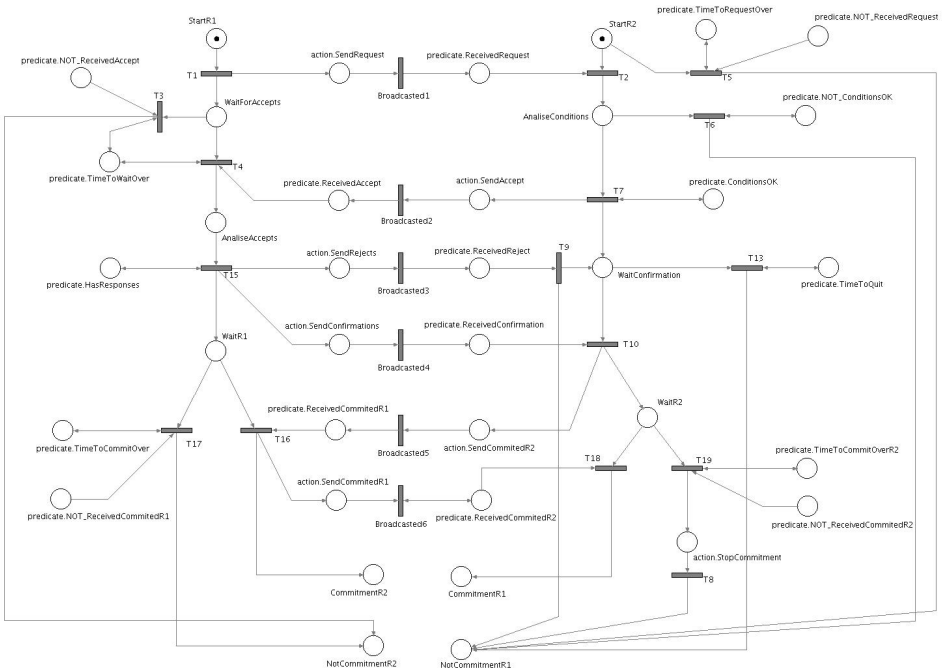
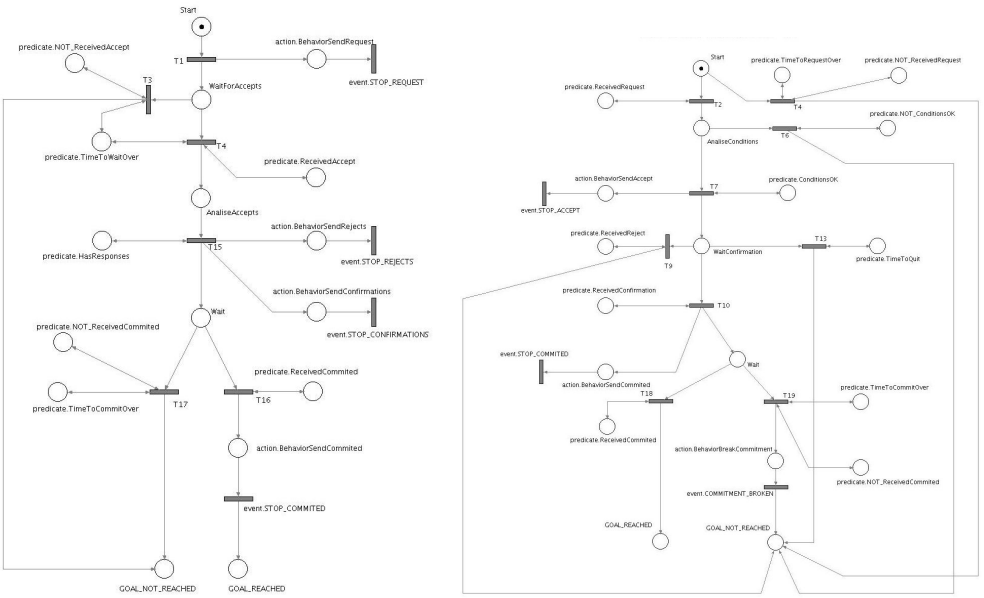


Figura 4.2.3: Rede de petri para estabelecimento de compromisso.

cada robot.

Para se obter as redes separadas basta retirar a transição *broadcasted*, tal como explicado na Secção 3.2. Estas redes estão na Figura 4.2.4.



(a) Robot que lança concurso. (b) Robot que recebe convite.

Figura 4.2.4: Redes de Petri para estabelecimento de compromisso.

As secções seguintes apresentam a rede dividida, um diagrama representativo do desenrolar do processo em questão e a respectiva implementação em código.

Pedido de Estabelecimento do Compromisso

A Figura 4.2.4(a) apresenta a rede executada pelo robot que dá inicio ao processo de estabelecimento do compromisso, devidamente preparada para ser executada como uma macro. Dividida a rede existe a necessidade de garantir que os lugares do tipo action.Send<...> não ficarão com o token para sempre, o que implicaria que a acção primitiva estivesse sempre a ser executada. Para tal, é utilizado um evento lançado pela própria acção para retirar esse mesmo token. Para uma melhor compreensão da Rede de Petri, a Figura 4.2.1 apresenta o diagrama de blocos referente a esta rede.

A rede implementada possibilita que respondam ao convite vários robots. Sendo assim, um momento importante de realçar na execução do estabelecimento de compromisso, por parte do robot que dá início ao processo, é o momento de escolha do parceiro. Esse processamento é feito no predicado *HasResponses*. Há medida que vão chegando candidaturas o robot vai actualizando o parceiro escolhido, guardando na variável *PartnerRobot* do WorldInfo. Neste caso, tratando-se de uma rede pensada para reposições de bola em jogo, é escolhido

o robot mais perto da bola. Contudo, esta implementação é bastante flexível bastando para isso alterar o predicado referido.

Aceitação de Estabelecimento do Compromisso

A Figura 4.2.4(b) apresenta a rede executada pelo robot que recebe o convite para o estabelecimento de um novo compromisso. Para uma melhor compreensão da Rede de Petri, a Figura 4.2.2 apresenta o diagrama de blocos referente a esta rede.

O robot que recebe o convite para estabelecimento do compromisso tem a oportunidade de decidir se quer ou não participar do comportamento relacional que terá início. Por exemplo, não terá muito sentido um robot sem linhas de passe candidatar-se a receptor da bola. Sendo assim o predicado *ConditionsOK* devolve se o robot pretende candidatar-se ou não. Este predicado foi implementado de forma a funcionar apenas como um *bypass* de um predicado desejado. Torna-se assim flexível para que, consoante o tipo de comportamento relacional, a decisão tenha um fundamento diferente. Caso o robot seja escolhido actualiza a variável *PartnerRobot* do WorldInfo com o nome do robot que lhe lançou o convite.

Comunicação e Implementação em código

Para que as redes referidas atrás pudessem ser executadas foi necessário definir a estrutura das mensagens trocadas e desenhar as acções primitivas e os predicados necessários.

O estabelecimento de compromisso foi implementado apoiando-se na troca de mensagens através do *Communication Manager*, referido na Secção 4.1.2. As mensagens contêm informação armazenada no *WorldInfo* sobre a forma de booleanos que indicam o estado do robot no que se refere à participação em compromissos. Cada um desses estados possui uma variável própria que toma valor *true* caso o robot se encontre nessa situação. Os estados são apresentados de seguida:

- **Standby:** o robot não está a participar em qualquer compromisso, todas as variáveis apresentam valor *false*;
- ***Request*:** neste estado o robot, R1 na Figura 4.2.3 está a enviar pedidos de estabelecimento compromisso para a equipa;
- ***Accept*:** neste estado o robot, R2 na Figura 4.2.3, recebeu um convite e aceitou candidatar-se;
- ***Confirm* e/ou *Reject*:** nestes casos o robot, R1 na Figura 4.2.3, responde às candidaturas dos outros robots com aceitação ou recusa. O caso de recusar um candidatura apenas se põe se responderem ao convite dois ou mais robots;
- ***Committed*:** este caso, comum a R1 e R2, indica que o robot está comprometido.

A Listagem 4.2.1 apresenta um exemplo da declaração da variável *Request*. Para os outros casos a declaração é feita do mesmo modo, apenas mudando o nome.

Listagem 4.2.1: WorldInfoInit.xml - Exemplo de declaração de variável para estado do robot num compromisso.

```
<Data>
  <Name>Request</Name>
  <Type>TimedData</Type>
  <Value>
    <Data>
      <Name>RequestBoolean</Name>
      <Type>Boolean</Type>
      <Value>false</Value>
    </Data>
  </Value>
</Data>
```

Cada robot transmite periodicamente informação relativa à sua envolvente para todos os outros elementos da equipa (posição da bola, comportamento que está a executar, entre outros). Do mesmo modo foi adicionada a transmissão de informação relativamente ao estado do robot no que se refere aos compromissos. É assim do conhecimento de cada robot o estado em que se encontram todos os seus parceiros quanto a um compromisso consigo. Naturalmente com o atraso associado ao período das transmissões e à transmissão em si. Cada tipo de variável (*request, accept, confirmation, reject e committed*) é transmitida utilizando uma estrutura de dados constituída por um *array* de booleanos, para endereçamento, e um outro booleano que transmite o valor da variável propriamente dita. A posição 0 do *array* a *true* indica que a mensagem é para todos e cada uma das outras posições endereça a mensagem a um dos robots. O código de preenchimento de uma mensagem deste tipo, neste caso um *request*, é apresentado na Listagem 4.2.2. Na Listagem 4.2.3 é apresentado a situação inversa, ou seja, neste caso o robot recebeu uma mensagem deste género.

Listagem 4.2.2: CommunicationManager.cpp - Envio de mensagem de *request*.

```
\\Vai buscar valor de request ao seu worldinfo
TimedData * tdRequest=
(TimedData*)WorldInfo::getInstance()->getData("Request");
    Boolean * request = (Boolean*) tdRequest->getData();
    bool robotRequest = false;

\\preenche valor a enviar
    if(request != NULL)
    {
        robotRequest = request->boolValue();
    }

    \\posição 0 a true, o request é sempre enviado para todos
    sws->request.to[0] = true;
    sws->request.value = robotRequest;
```

Foi ainda necessário criar acções primitivas para comunicação, facilmente identificáveis na Figura 4.2.4 pela designação action.BehaviorSend<...> .Estes comportamentos só englobam

Listagem 4.2.3: CommunicationManager.cpp - Recepção de mensagem de *request*.

```
 // actualiza Request apenas se a mensagem é para mim ou para todos
if(sws->request.to[myId] == true || sws->request.to[0] == true){
    \\retira valor recebido
    bool request = sws->request.value;
    \\adiciona ao mapeamento de informação sobre colegas de equipa
    TimedData * tdRequest = new TimedData((Data*)new
        Boolean(request),timestamp);
    robotInfo->addElement("Request", tdRequest)
    }
else{
    \\caso não tenha recebido mantém a false
    TimedData * tdDefaultFalse = new TimedData((Data*)
    new Boolean(false), timestamp);
    \\adiciona ao mapeamento de informação sobre colegas de equipa
    robotInfo->addElement("Request", tdDefaultFalse);
    }
```

a acção correspondente. Foram criados porque a implementação do estabelecimento do compromisso foi feita ao nível do *BehaviorCoordinator*, como será explicado no Capítulo 5. Estas acções alteram a variável de estado que lhe é respectiva, a qual vai posteriormente será transmitida da forma explicada acima. Para que o *token* não permaneça eternamente nestes lugares, o que levaria à continuação da execução da acção indeterminadamente, este tipo de acções produz um evento local o qual activa a transição que lhe é posterior na Rede de Petri. Na Listagem 4.2.4 dá-se um exemplo utilizando a acção primitiva *SendRequest*. Na Figura 4.2.4 pode-se ver uma aplicação de um evento desse tipo, p.e., o evento STOP_REQUEST que se segue à acção *SendRequest*.

Listagem 4.2.4: SendRequest.cpp - Exemplo de acção primitiva para comunicação durante estabelecimento do compromisso.

```
void SendRequest::serviceDefault2(){

    ActiveObject * eventRouter =  ActiveObjectFactory::
    getInstance()->getEventRouter();
    EventMap * em = EventMap::getInstance();

    \\coloca variável de request a true
   TimedData * requestTD = new TimedData(new Boolean(true) );
   this->putDataWI("Request", requestTD);

    int eventId;

    \\lança evento
    eventId = em->getEventID("STOP_REQUEST");

    eventRouter->putEvent(eventId);
};
```

A evolução da Rede de Petri é conseguida através da utilização de predicados que processam informação presente no WorldInfo, alguns já referidos anteriormente. Os predicados do tipo Receive<...> indicam que o robot recebeu uma mensagem de um dos tipos introduzidos

anteriormente: *request, accept, confirmation, reject e committed*. O valor deste tipo de predicados é retirado da informação recebida periodicamente acerca dos restantes robots. Por exemplo, caso alguns dos robots receba a variável *request* a *true*, vinda de um companheiro de equipa, o predicado ReceivedRequest tomará esse mesmo valor.

Outro tipo de predicados necessários são os temporizadores. A existência de *timeouts* é importantíssimo para que não haja estagnação na acção do robot caso alguma mensagem falhe. Sendo assim, associado a estas variáveis de estado do compromisso é adicionado um registo de tempo para uma implementação mais facilitada dos temporizadores presentes na rede da Figura 4.2.3. Essa característica está presente no excerto de código da Listagem 4.2.1 através da declaração da variável como *TimeData*. Basta conhecer o instante em que o robot entrou num determinado estado, o instante actual e o limite de tempo do temporizador em questão para indicar se esse mesmo limite foi ultrapassado.

4.2.2 Gestão do compromisso

Esta secção apresenta a implementação realizada para os conceitos introduzidos na Secção 3.4 acerca da gestão de compromisso.

A partir do momento em que o robot está comprometido é da responsabilidade do gestor de compromissos monitorizar a relação entre os dois robots envolvidos. Ao invés de uma monitorização constante, como a apresentada anteriormente, optou-se por uma monitorização apenas em pontos fundamentais. Esses pontos fundamentais são aqueles a partir dos quais a acção do robot seria significativamente mudada caso não estivesse comprometido. Por exemplo, o ponto em que o robot que faz o passe espera pelo robot que o vai receber assume-se como um ponto desse género. Nesses pontos é feita uma avaliação acerca do comprometimento do outro robot. Para isso foi criado um predicado chamado *PartnerCommited* cujo valor depende da informação transmitida pelo parceiro do compromisso. Caso num momento de avaliação se verifique que o valor do predicado é *false* o robot quebra o compromisso também do seu lado, dado que o compromisso já foi quebrado do lado do outro robot. Fazendo a ligação deste predicado com a Figura 3.4.1, nessa rede o predicado é designado por *R2Committed* para um dos robots e *R1NotCommited* para o outro robot.

No caso referido acima está-se perante o insucesso do comportamento devido à quebra de compromisso por parte do parceiro. Contudo, no caso de insucesso local ou de sucesso o compromisso também deve ser quebrado. A Listagem 4.2.5 apresenta o código para a acção responsável por quebrar o compromisso. A acção primitiva *BreakCommitment* consiste em escrever para a variável *Committed* do *WorldInfo* o valor *false*, indicando que o robot deixou de estar no compromisso. Note-se que na Figura 3.4.1 a quebra do compromisso é feita ao nível do BehaviorExecutor. No entanto, no sentido de adaptação aos três níveis da arquitec-

tura MeRMaID essa não foi a opção escolhida, sendo o compromisso terminado ao nível do BehaviorCoordinator. No Capítulo 5, aquando da descrição dos comportamentos implementados, é descrita a integração da acção para quebra de compromisso.

A opção de se implementar uma gestão do compromisso pontual, ao invés de constante, deve-se ao aumento do desempenho do comportamento se não se estiver constantemente a avaliar a situação do compromisso quando isso não trará uma maior valia para a acção do robot. Contudo, este tipo de abordagem obriga a uma maior atenção no desenvolvimento dos comportamentos para que uma melhor performance de um robot não tenha como consequência o insucesso dos companheiros no comportamento relacional.

Listagem 4.2.5: BreakCommitment.cpp - Acção primitiva para quebra de compromisso.

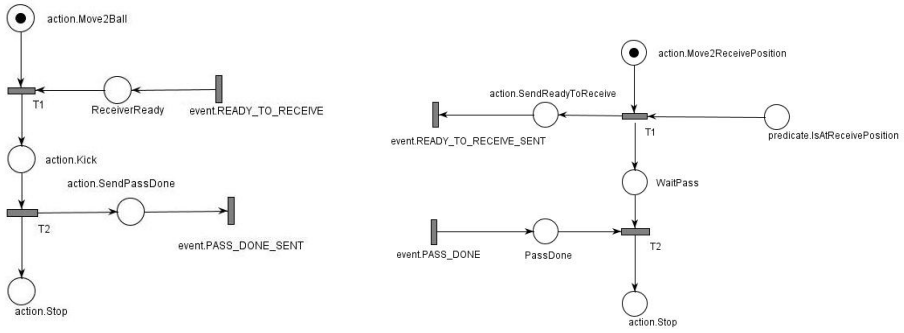
```
void BreakCommitment::serviceDefault2(){

    ActiveObject * eventRouter =  ActiveObjectFactory::
    getInstance()->getEventRouter();
    EventMap * em = EventMap::getInstance();

    \\coloca variável committed a false
    TimedData * commitedTD = new TimedData(new Boolean(false) );
    this->putDataWI("Commited", commitedTD );

    cout << "BreakCommitment => TRUE"<< endl;

    int eventId;

    \\envia evento local para parar acção primitiva
    eventId = em->getEventID("COMMITMENT_BROKEN");

    eventRouter->putEvent(eventId);

};
```

4.2.3 Sincronização

Na Secção 3.5 introduziu-se o conceito de sincronização nos comportamentos relacionais. Nesta secção é descrita a implementação para a plataforma de desenvolvimento do SocRob.

A sincronização nos comportamentos desenvolvidos é necessária somente em dois pontos: quando o jogador que vai receber a bola quer indicar ao marcador da falta que está pronto para receber e quando o jogador que marca a falta quer indicar que a bola foi passada. Para realizar este tipo de comunicações optou-se pela utilização de eventos. A Figura 4.2.5 apresenta um exemplo para um passe. A opção de eventos justifica-se pela sua simplicidade de implementação. Um ponto de sincronismo representa uma situação pontual onde apenas se quer dar indicação ao outro robot do cumprimento de determinada tarefa, não havendo qualquer processamento ou armazenamento de informação. Contudo, é utilizado um local de memória, desta forma caso a execução da Rede de Petri ainda não tenha chegado ao ponto

de sincronismo, o evento não é perdido. Tal como nas redes de estabelecimento do compromisso, Figura 4.2.4, as acções do tipo *Send* devem produzir um evento local que retire o *token* para que a execução da acção não se prolongue por tempo indeterminado. Um exemplo prático pode ser visto na Figura 4.2.5 no caso da acção primitiva *SendPassDone* e do evento PASS_DONE_SENT.



(a) Robot que faz o passe. (b) Robot que recebe o passe.

Figura 4.2.5: Exemplo de utilização de eventos para sincronização.

A Listagem 4.2.6 apresenta um exemplo de código para a produção de um evento de sincronização.

Listagem 4.2.6: SendPassDone.cpp - Exemplo de utilização de eventos para sincronização.

```
void SendPassDone::serviceDefault2(){

    (...)
    //Envia evento local para retirar token

    eventId = em->getEventID("PASS_DONE_SENT");

    eventRouter->putEvent(eventId);

    (...)

    //Envia evento de sincronizacao
    ParameterList params;
    params = ParameterList();
    params.addParameter("msg", new String("sendMsg"));
    params.addParameter("msgToSend", new String("sendEventToRobot"));
    params.addParameter("recipientName",new String("communicationManager"));
    params.addParameter("eventName", new String("PASS_DONE"));
    //Envia evento para robot comprometido
    params.addParameter("robotId", new Integer(num)); //robot id

    ActiveObjectFactory::getInstance()->getDataRouter()->
    putMsg(params);

};
```

4.2.4 Algoritmo de implementação de um comportamento

Para uma implementação mais rápida e metódica dos comportamentos foi utilizado um algoritmo de implementação. O algoritmo teve como base o trabalho desenvolvido por Milhinhos e Moura Vaz[13], fazendo-se as alterações necessárias à implementação deste trabalho. Os seguintes passos compõe a metodologia referida:

1. Identificação do objectivo do comportamento
2. Desenho da rede de Petri
3. Identificação dos pontos de sincronismo, levantamento dos eventos e mecanismos de timeout necessários
4. Identificação dos pontos de gestão de compromisso necessários
5. Identificação das acções primitivas, predicados e eventos necessários ao comportamento e respectiva passagem para código dos elementos inexistentes
6. Teste do comportamento correndo cada robot individualmente no simulador
7. Teste do comportamento correndo dois ou mais robots no simulador (caso se trate de um comportamento não individual)
8. Teste do comportamento correndo cada robot real individualmente
9. Teste do comportamento correndo dois ou mais robots reais simultaneamente (caso se trate de um comportamento não individual)

Com este algoritmo automatizou-se o desenho de novos comportamentos promovendo uma fácil organização do trabalho no que diz respeito, por exemplo, à criação de novas acções primitivas, predicados e eventos.

No Capitulo 5 são apresentados os comportamentos desenvolvidos e como foram integrados os elementos apresentados neste capítulo.

Capítulo 5

Comportamentos Desenvolvidos

Este capítulo apresenta os comportamentos desenvolvidos, explicando para cada um a situação de jogo para o qual foi pensado e o seu funcionamento. É ainda referido onde foi introduzido o trabalho apresentado no Capítulo 4 na táctica da equipa.

Na Secção 5.1 são introduzidos os papéis e a táctica criados, na Secção 5.2 é a vez dos comportamentos relacionais e, por fim, na Secção 5.3 são descritos os comportamentos individuais. No que se refere à arquitectura do projecto, apresentada na Secção 4.1, a apresentação dos comportamentos é feita seguido uma abordagem *top-down*. Ou seja, é primeiramente apresentada a táctica desenvolvida para equipa, e corrida ao nível do *TeamOrganizer*. Seguem-se os papéis implementados ao nível do *BehaviorCoordinator*, e os comportamentos propriamente ditos, que são executados no nível designado por *BehaviorExecutor*.

5.1 Comportamentos Organizacionais

5.1.1 TacticBase

Este comportamento organizacional é corrido ao nível do *TeamOrganizer* da arquitectura MeRMaID (Secção 4.1.2) e define a táctica da equipa. Como tal, é corrido permanentemente por todos os robots de campo e atribui a cada um deles um papel no seio da equipa, papel esse que reúne um conjunto de comportamentos a executar pelo robot consoante o estado do jogo.

Esta táctica define cinco papéis:

- **Atacante** (*RoleAttacker*): o atacante é definido nas situações de bola corrida (jogo não parado nem falta a ser marcada). Os jogadores partilham informação da sua distância à bola entre todos e com base nessa informação é calculado localmente o predicado *ShouldIGo* que será *true* apenas para o jogador mais perto da bola que não seja guarda-redes e caso o jogo se encontre na situação explicada acima. Esse jogador é definido como atacante passando a correr a rede *RoleAttacker*. Existe apenas um atacante na equipa;

- **Defesa** (*RoleDefender*): o defesa é definido nas situações de bola corrida (jogo não parado nem falta a ser marcada) e é escolhido de entre os jogadores não escolhidos para atacante. Além da sua distância à bola, os jogadores partilham informação da sua distância à própria baliza. Com base nessa informação é calculado localmente o predicado *ShouldIDefend* que será *true* apenas para o jogador mais perto da própria baliza que não seja guarda-redes nem atacante. Esse jogador é definido como defesa passando a correr a rede *RoleDefender*. Existe apenas um defesa na equipa;

- **Apoio** (*RoleSupporter*): tal como o defesa e o atacante, os jogadores de apoio são também definidos nas situações de bola corrida. São escolhidos os restantes jogadores, ou seja, os robots que não foram escolhidos nem para defesa nem para atacante. Esses jogadores são definidos como apoio do atacante passando a correr a rede *RoleSupporter*. Sendo a equipa ISocRob constituída por cinco robots, existem dois que ocupam o papel de apoio ao atacante;

- **Marcador** (*RoleFoulTaker*): este papel, tal como o de receptor, é apenas definido para situações em que a equipa tem de repor a bola em jogo, independentemente da situação que lhe deu origem. É escolhido o jogador mais perto da bola (predicado *ShouldITakeFoul* a *true*) que posteriormente deverá estabelecer um compromisso com o jogador que irá receber a bola, o que será explicado na Secção 5.1.3. Este papel deixará de ser executado quando a camada abaixo, que executa a rede *RoleFoulTaker*, enviar um evento FOUL_FINISHED indicando o fim da marcação da falta, tanto em caso de sucesso como de insucesso;

- **Receptor** (*RoleFoulReceiver*): este papel, tal como o marcador, é apenas definido para situações em que a equipa tem de repor a bola em jogo, independentemente da situação que lhe deu origem. É escolhido o segundo jogador mais perto da bola (predicado *ShouldIReceiveFoul* a *true*) que posteriormente deverá estabelecer um compromisso com o jogador que irá passar a bola, o que será explicado na Secção 5.1.3. Este papel deixará de ser executado quando a camada abaixo, que executa a rede *RoleFoulReceiver*, enviar um evento FOUL_FINISHED indicando o fim da marcação da falta, tanto em caso de sucesso como de insucesso;

- **Parado** (*RoleStop*): caso os motores do robot sejam desligados. Neste caso o robot não entra nas contas para o cálculo dos predicados que definem os outros papéis.

Na Tabela 5.1.1 é resumida a informação sobre este comportamento descrito acima. No apêndice A.1.1 pode ser consultada a Rede de Petri desenhada.

5.1.2 RoleAttacker, RoleDefender, RoleSupporter

Estes papéis são modelados pelas redes RoleAttacker, RoleDefender e RoleSupporter, reunindo o conjunto de comportamentos executados pelo robot escolhido pelo *TeamOrganizer*

Tabela 5.1.1: Táctica *TacticBase*.

Nome	*TacticBase*.
Tipo	Comportamento Organizacional - Táctica.
Roles	*RoleAttacker, RoleDefender, RoleSupporter, RoleStop, RoleFoulTaker, RoleFoulReceiver*
Nível MeRMaID	*TeamOrganizer*.
Apêndice	A.1.1.

para atacante, defesa e apoio, respectivamente. São contempladas quatro situações distintas de jogo. As situações são explicadas de seguida:

- **Jogo corrido**: neste ponto os diferentes papéis diferem, sendo executados os comportamentos base *BehaviorBaseAttack, BehaviorBaseDefend e BehaviorBaseSupport* para o atacante, defesa e apoio, respectivamente. Estes comportamentos são explicados mais detalhadamente nas Secções 5.3.1, 5.3.2 e 5.3.3.

- **Reposição para a equipa**: o predicado *FoulOurs* a *true* indica que o jogo está neste estado. Nesta situação o robot executa os comportamentos *BehaviorFoulDefender*, *BehaviorFoulSupporter* ou *BehaviorFoulAttacker*, para o atacante, defesa e apoio, respectivamente. Estes comportamentos são explicados na Secção 5.3.4. O robot passará ao comportamento de jogo corrido se receber um evento FOUL_TAKEN proveniente do robot que repôs a bola em jogo ou se passarem os 10 segundos estipulados nas regras para uma reposição de bola.

- **Reposição para o adversário**: o predicado *FoulOpponent* a *true* indica que o jogo está neste estado. Tal como no ponto anterior, nesta situação o robot executa os comportamentos *BehaviorFoulDefender*, *BehaviorFoulSupporter* ou *BehaviorFoulAttacker*. O robot passará ao comportamento de jogo caso o predicado *FoulTaken* venha a *true*. Este predicado mede se a bola se moveu desde que foi fado o sinal para que o adversário jogue a bola, o que indicará que a bola foi reposta em jogo. De igual forma, caso os 10 segundos estipulados nas regras para uma reposição de bola passem o robot também passará a executar o comportamento de jogo corrido.

- **Golo**: caso a equipa marque ou sofra um golo, o robot desloca-se para a sua posição padrão no campo (comportamento *BehaviorMove2HomePosition*). Desta forma a equipa fica pronta para o reinicio da partida.

- **Jogo parado**: nesta situação o robot permanecerá parado, executando o comportamento *BehaviorStop*.

Na Tabela 5.1.2 é resumida a informação sobre os comportamentos descritos acima. Nos apêndices A.1.2, A.1.3 e A.1.4 podem ser consultadas as Redes de Petri desenhadas para o atacante, defesa e apoio, respectivamente.

Tabela 5.1.2: Papéis *RoleAttacker, RoleDefender e RoleSupporter*.

Nome	*RoleAttacker*, *RoleDefender*, *RoleSupporter*.
Tipo	Comportamento Organizacional - Papel.
Papel	Atacante.
Situação de jogo	Todo o jogo.
Comportamentos	*BehaviorBaseAttack, BehaviorFoulAttacker*, *BehaviorBaseDefend, BehaviorFoulDefender*, *BehaviorBaseSupport, BehaviorFoulSupporter*, *BehaviorStop, BehaviorMove2HomePosition*
Predicado (na táctica)	*ShouldIGo*
Nível MeRMaID	*BehaviorCoordinator*.
Apêndice	A.1.2, A.1.3, A.1.4.

5.1.3 RoleFoulTaker e RoleFoulReceiver

As RdP desenhadas para estes dois papéis reúnem o conjunto de comportamentos executados pelo marcador e receptor para uma reposição de bola para a sua equipa. É a este nível, *BehaviorCoordinator*, que é estabelecido e quebrado o compromisso entre os dois robots.

Inicialmente o robot executa o comportamento *BehaviorStop* que o mantém parado. Posteriormente, consoante a situação de jogo, o token é conduzido para uma das área da RdP. Essa especificação foi conseguida através da utilização de predicados que espelham o estado do jogo. Por exemplo, caso tenha sido marcada uma falta a favor o predicado *FreeKickOurs* tomará o valor *true*.

Tanto para o marcador como para o receptor, o compromisso foi apenas introduzido para os pontapé de baliza e livres, servindo como exemplo da implementação para qualquer outra situação.

Identificado o estado do jogo dá-se lugar ao estabelecimento do compromisso, representado na macro *macroCommitmentEstablishmentBC*. Caso seja atingido o objectivo são executados os comportamentos relacionais. Por exemplo, *BehaviorFoulTakerDirect* e *BehaviorFoulReceiverDirect*, para o marcador e receptor, respectivamente. Caso contrário, o marcador toma a iniciativa de repor a bola em jogo sozinho executando um comportamento individual, como por exemplo o comportamento *BehaviorFoulTakeAlone*. Neste último caso o papel do receptor terminaria assim que não tivesse sucesso no estabelecimento do compromisso.

Para cada situação de jogo o robot pode executar vários tipos de comportamento. Por exemplo, num lançamento de linha lateral pode passar a bola para a frente, pode desmarcar o receptor pela linha lateral ou pode tocar a bola com a parte lateral. A escolha do compor-

tamento a realizar é feita com base em predicados que implementam uma variável aleatória que toma valor 0 ou 1 com igual probabilidade, *RandomOne* e *RandomTwo*. No conjunto da informação transmitida periodicamente pelo robot está o comportamento que está a executar. Desta forma e com base em predicados o receptor saberá qual o comportamento que foi escolhido pelo marcador e poderá escolher o comportamento correspondente. Por exemplo, no caso de ter sido escolhido o comportamento *BehaviorFoulTakerDirect* pelo marcador, o predicado *KickerChooseDirect* tomará o valor *true*.

Os comportamentos executados por qualquer um destes papéis, ao terminarem, enviam um evento sinalizando o seu insucesso ou sucesso (BEHAVIOR_UNSUCCESSFUL e BEHAVIOR_SUCCESSFUL). O insucesso pode ser provocado pela falha do lado do próprio robot, do parceiro ou pela recepção de um sinal de jogo parado vindo da referee box.

Tanto no caso do marcador como do receptor, se estavam a executar o comportamento relacional e não tiveram sucesso, o evento BEHAVIOR_UNSUCCESSFUL conduzirá à macro *macroCommitmentBreakingBC*, responsável por quebrar o compromisso. Posteriormente, o marcador executa o comportamento individual de reposição da bola em jogo, enquanto o receptor produz um evento FOUL_FINISHED que indicará ao TO o fim da participação do robot na reposição da bola em jogo. Em ambos os papéis, no caso de sucesso é enviado o evento BEHAVIOR_SUCCESSFUL, é quebrado o compromisso utilizando a mesma macro e, finalmente, é produzido o evento FOUL_FINISHED, que informa o TO da finalização dos papéis de receptor e marcador, para que a táctica da equipa evolua.

Na Tabelas 5.1.3 e 5.1.4 é resumida a informação sobre o marcador e receptor, respectivamente. Nos apêndices A.1.5 e A.1.6 podem ser consultadas as RdP desenhadas.

Tabela 5.1.3: Papel *RoleFoulTaker*.

Nome	*RoleFoulTaker*.
Tipo	Comportamento Organizacional - Papel.
Papel	Marcador.
Situação de jogo	Pontapé de saída, grande penalidade, canto, pontapé de baliza, falta, fora.
Comportamentos	*BehaviorPenaltyTaker, BehaviorFoulTakerDirect, BehaviorFoulTakerDirect2, BehaviorFoulTakerTouch, BehaviorFoulTakerTouch2, BehaviorFoulTakerLine, BehaviorFoulTakerKickOff*
Tactic	*TacticBase*
Macros	*macroCommitmentEstablishment_BC, macroCommitmentBreaking_BC*
Predicado (na táctica)	*ShouldITakeFoul*
Nível MeRMaID	*BehaviorCoordinator*.
Apêndice	A.1.5.

Tabela 5.1.4: Papel *RoleFoulReceiver*.

Nome	*RoleFoulReceiver*.
Tipo	Comportamento Organizacional - Papel.
Papel	Receptor.
Situação de jogo	Pontapé de saída, grande penalidade, canto, pontapé de baliza, falta, fora.
Comportamentos	*BehaviorFoulReceiverDirect, BehaviorFoulReceiverDirect2, BehaviorFoulReceiverTouch, BehaviorFoulReceiverTouch2, BehaviorFoulReceiverLine, BehaviorFoulReceiverKickOff*
Tactic	*TacticBase*
Macros	*macroCommitmentEstablishment_BC, macroCommitmentBreaking_BC*
Predicado (na táctica)	*ShouldIReceiveFoul*
Nível MeRMaID	*BehaviorCoordinator*.
Apêndice	A.1.6.

5.2 Comportamentos Relacionais

Foram implementados diversos comportamentos relacionais para a reposição da bola em jogo, todos idênticos entre si, variando apenas a postura do marcador e do receptor. São apresentados nesta secção um desses pares de comportamentos desenvolvidos.

5.2.1 BehaviorFoulTakerDirect

Esta RdP implementa um comportamento relacional de reposição da bola em jogo que será executado pelo marcador. Tal como qualquer comportamento deste tipo, foi desenhado em paralelo com o comportamento correspondente executado pelo receptor e descrito na Secção 5.2.2.

O objectivo deste comportamento é colocar o robot junto à bola, alinhado e virado para a baliza adversária para finalmente efectuar um passe. Tratando-se de um comportamento relacional toda a sequência de acções tem por pressuposto um compromisso com outro elemento da equipa.

Começa-se por deslocar o robot e colocá-lo na postura desejada utilizando as acções primitivas *Move2Ball*, *ApproachBall* e *TurnAroundBallToOppGoal*. Os tempos de execução de cada uma destas primitivas são determinados pelos predicados *NearBall*, *CloseToBall* e *IsTurnedToOppGoal*, cujo nomes são auto-explicativos. De seguida o robot espera o sinal da *referee box* para que possa marcar a falta.

A *referee box* é uma aplicação que corre num computador controlado por um dos árbitros e que envia para o robot mensagens que indicam os eventos ocorridos no jogo. A marcação de uma falta segue os seguintes passos: é assinalado o tipo de falta e a equipa que tem a posse da bola, são dados alguns segundos para os robots se posicionarem e de seguida é enviado o sinal para o jogo ser reiniciado. A equipa que repõe a bola em jogo tem dez segundos para o fazer, passados esses dez segundos qualquer jogador pode seguir o jogo.

Logo que o jogo seja reiniciado, o robot passa a um ponto de sincronismo (*WaitForReceiver*). Neste ponto é avaliado o compromisso através do predicado *PartnerCommitted*. Caso o parceiro ainda esteja comprometido o robot espera, durante o tempo permitido para repor a bola em jogo, pelo sinal de que o receptor está em posição de receber a bola, ou seja, espera a chegada do evento READY_2_RECEIVE. Caso não seja recebido ao fim desse tempo nenhum evento o marcador envia o evento BEHAVIOR_UNSUCCESSFUL para a camada acima. Caso chegue o evento, é apanhada a bola, passada ao receptor (acções primitivas *CatchBall* e *KickPass*) e enviado o evento FOUL_TAKEN para que toda a equipa, inclusive o jogador que vai receber a bola, saiba que a bola foi passada. Finalmente, é enviado o evento BEHAVIOR_SUCCESSFUL comunicando ao *BehaviorCoordinator* que o comportamento teve sucesso.

Na Tabela 5.2.1 é resumida a informação sobre o comportamento descrito nesta secção. No apêndice A.2.1 pode ser consultada a RdP desenhada.

Tabela 5.2.1: Comportamento *BehaviorFoulTakerDirect*.

Nome	*BehaviorFoulTakerDirect*.
Tipo	Comportamento Relacional.
Situação de jogo	Reposição da bola em jogo tendo posse de bola: livres, pontapés de baliza e "lançamentos"de linha lateral.
Objectivo	Colocar-se junto à bola na direcção da baliza adversária. Quando recebida a indicação de que o jogador que vai receber a bola está pronto, passar.
Papel	*RoleFoulTaker*
Nível MeRMaID	*BehaviorCoordinator*.
Sincronismo (Eventos)	
Enviados Recebidos	FOUL_TAKEN READY_2_RECEIVE
Apêndice	A.2.1.

Foram implementadas variantes deste comportamento para outras situações de jogo, diferindo apenas na postura do robot tida como objectivo, nomeadamente:

- ***BehaviorFoulTakerDirect2***: coloca o robot de frente e alinhado com a sua baliza.É utilizado nos cantos.;

- ***BehaviorFoulTakerTouch***: coloca o robot de lado para a bola de forma a posteriormente tocá-la na direcção da baliza adversária. É utilizado nas mesmas situações que o comportamento descrito nesta secção;
- ***BehaviorFoulTakerTouch2***: coloca o robot de lado para a bola de forma a posteriormente tocá-la na direcção da sua baliza.É utilizado nos cantos;
- ***BehaviorFoulTakerLine***: coloca o robot alinhado com a linha lateral orientado no sentido ofensivo. É utilizado nos lançamentos;
- ***BehaviorKickOffTaker***: É utilizado nos pontapés de saída. RdP no Apêndice A.2.3.

5.2.2 BehaviorFoulReceiverDirect

Esta RdP implementa um comportamento relacional de reposição da bola em jogo que será executado pelo receptor. Tal como qualquer comportamento deste tipo, foi desenhado em paralelo com o comportamento correspondente executado pelo marcador e descrito na Secção 5.2.1.

O objectivo deste comportamento é colocar o robot junto à bola entre esta e a baliza adversária. Como já referido, tratando-se de um comportamento relacional toda a sequência de acções tem por pressuposto um compromisso com outro elemento da equipa.

A acção primitiva *Move2ReceivePositionDirect* leva o robot até à postura desejada. Se durante esta execução se verificar que o parceiro não está comprometido, através da avaliação do predicado *PartnerCommitted*, é enviado o evento BEHAVIOR_UNSUCCESSFUL para a camada acima terminar o comportamento. Ao ser atingida a postura pretendida o predicado *IsAtPositionReceiverDirect* tomará valor *true* e, estando o parceiro comprometido, será enviado o evento READY_2_RECEIVE para o marcador. Posteriormente o robot aguardará, parado, durante o tempo permitido para uma reposição de bola em jogo, o evento FOUL_TAKEN indicando que o marcador passou a bola. No caso de não ser recebido o evento ou do parceiro ter quebrado o compromisso é enviado o evento BEHAVIOR_UNSUCCESSFUL. Caso contrário o receptor move-se para a bola e envia para o BC a indicação de que o comportamento teve sucesso, ou seja, produz o evento BEHAVIOR_SUCCESSFUL.

Na Tabela 5.2.2 é resumida a informação sobre o comportamento descrito nesta secção. No apêndice A.2.1 pode ser consultada a RdP desenhada.

Tal como para o marcador, foram implementadas variantes deste comportamento para outras situações de jogo, diferindo apenas na postura do robot tida como objectivo, nomeadamente:

- ***BehaviorFoulReceiverDirect2***: coloca o robot junto à bola ente esta e a sua baliza .É utilizado nos cantos;

Tabela 5.2.2: Comportamento *BehaviorFoulReceiverDirect*.

Nome	*BehaviorFoulReceiverDirect*.
Tipo	Comportamento Relacional.
Situação de jogo	Reposição da bola em jogo tendo posse de bola: livres, pontapés de baliza e "lançamentos"de linha lateral.
Objectivo	Colocar-se junto à bola de costas para a baliza adversária. Quando recebida a indicação de que o jogador que vai passar a bola chutou, mover-se para a bola.
Papel	*RoleReceiverReceiver*
Nível MeRMaID	*BehaviorCoordinator*.
Sincronismo (Eventos)	
Enviados	READY_2_RECEIVE
Recebidos	FOUL_TAKEN
Apêndice	A.2.2.

- ***BehaviorFoulReceiverTouch***: É idêntico a este comportamento mas o jogador fica mais perto da bola, dado que a bola é tocada e não passada. É utilizado nas mesmas situações que o comportamento descrito nesta secção;
- ***BehaviorFoulReceiverTouch2***:É idêntico a este comportamento mas o jogador fica mais perto da bola, dado que a bola é tocada e não passada. O jogador fica de costas para a sua baliza. É utilizado nos cantos;
- ***BehaviorFoulReceiverLine***: coloca o robot alinhado com a linha lateral. É utilizado nos lançamentos.
- ***BehaviorKickOffReceiver***: É utilizado nos pontapés de saída. RdP no Apêndice A.2.4.

5.3 Comportamentos Individuais

5.3.1 BehaviorBaseAttack

Este comportamento foi desenhado para ser executado pelo atacante em situações de jogo corrido e tem como objectivo final marcar golo. Como requisito essencial ao sucesso de qualquer comportamento, o robot deverá conhecer a localização da bola. Essa verificação é feita utilizando o predicado *SeeBall* e caso este seja negativo o robot move-se para a sua posição padrão em campo, a qual está definida no *WorldInfo* e é distinta para cada elemento da equipa. Caso contrário dá-se inicio à sequência de acções que têm como meta rematar à baliza da equipa adversária com sucesso. Em primeiro lugar o robot desloca-se até à bola ao correr a acção primitiva *Move2Ball*. Chegado às imediações da bola, o que é indicado pelo predicado *NearBall* é corrida a primitiva *CatchBall*. Ao contrário dos comportamentos para

marcação de faltas ou foras, referidos anteriormente, neste caso não é executada a acção primitiva *ApproachBall*.

Esta opção é justificada pelo estado em que se encontra o jogo e pelas características das acções primitivas. A primitiva *CatchBall* alinha-se com a bola e leva o robot a mover-se na sua direcção, mantendo uma velocidade constante. Por outro lado, a primitiva *ApproachBall* aproxima-se tendendo a velocidade do robot para zero, o que é útil no caso da bola estar parada. Contudo, no caso de jogo corrido pretende-se que o jogador tenha uma acção mais ágil e rápida, dado que se disputa a bola com o adversário.

Caso a acção *CatchBall* tenha sucesso o robot ficará na posse da bola, essa circunstância é reflectida pelo predicado *HasBall* a *true*. De seguida é executada a primitiva *Dribble2Goal* cujo objectivo é deslocar o robot até perto da baliza adversária desviando-se dos oponentes que encontrar pelo caminho. Ao chegar perto da área adversária o predicado *NearGoal* fará evoluir o comportamento para a próxima etapa que consiste na escolha da abordagem à baliza. O predicado *RandomOne* implementa uma variável aleatória que toma valor *true* ou *false* com igual probabilidade. Utilizando este predicado é escolhida uma das seguintes acções primitivas: *Drible2Score*, em que o robot se desvia dos obstáculos orientando-se com a baliza, e *Aim2Score* em que o robot descreve uma diagonal em frente à baliza. Tanto num caso como noutro a acção primitiva tenta alinhar-se com uma área da baliza a descoberto e, atingida a direcção desejada, o que é indicado pelo predicado *ShootingOpportunity* a *true*, o robot remata. De notar que a acção primitiva de abordagem à baliza não deixa de ser executada durante o remate, para que o robot continue alinhado.

Se em algum momento do comportamento se perder a bola o robot executa novamente acção *Move2Ball* e o ciclo repete-se.

Na Tabela 5.3.1 é resumida a informação sobre este comportamento descrito acima. No apêndice A.3.1 pode ser consultada a Rede de Petri desenhada.

Tabela 5.3.1: Comportamento *BehaviorBaseAttack*.

Nome	*BehaviorBaseAttack*.
Tipo	Comportamento Individual.
Situação de jogo	Jogo não parado, tendo o robot que executa o comportamento a posse de bola.
Objectivo	Driblar até à baliza adversária e marcar golo.
Papel	*Attacker*.
Nível MeRMaID	*BehaviorExecutor*.
Apêndice	A.3.1.

5.3.2 BehaviorBaseDefend

Este comportamento define o desempenho do defesa em situações de jogo corrido. No caso do robot ver a bola é executada a acção primitiva *CoverGoal* que pretende cobrir a baliza, minimizando o ângulo de remate do adversário. Caso o robot não consiga localizar a bola de jogo, é executada primeiramente a acção primitiva *Move2DefenderPosition* que desloca o robot até às imediações da sua baliza e atingida essa posição, o que é indicado pelo predicado *IsAtDefenderPosition* a *true*, é executada a acção primitiva de procura designada por *SearchBallDefend* que faz o robot descrever um losango junto à sua baliza.

Na Tabela 5.3.2 é resumida a informação sobre este comportamento descrito acima. No apêndice A.3.2 pode ser consultada a Rede de Petri desenhada.

Tabela 5.3.2: Comportamento *BehaviorBaseDefend*.

Nome	*BehaviorBaseDefend*.
Tipo	Comportamento Individual.
Situação de jogo	Jogo não parado, sendo o robot o defesa da equipa.
Objectivo	Caso veja a bola cobrir o ângulo para a sua baliza, caso contrário mover-se para posição de defesa.
Papel	*Defender*.
Nível MeRMaID	*BehaviorExecutor*.
Apêndice	A.3.2.

5.3.3 BehaviorBaseSupport

Este é comportamento básico para os jogadores que desempenham a função de apoiar o atacante, ou seja, define o conjunto de acções executadas durante o jogo corrido. São apenas executadas duas acções primitivas e utilizado um predicado (*SeeBall*) cujo nome é auto-explicativo. Caso o robot veja a bola segue-a mantendo uma distância de dois metros, colocando-se assim numa situação privilegiada para recuperar a sua posse caso o atacante a perca. Caso contrário é executada uma acção primitiva de procura de bola no meio-campo ofensivo, descrevendo o robot um losango nessa zona.

Na Tabela 5.3.3 é resumida a informação sobre este comportamento descrito acima. No apêndice A.3.3 pode ser consultada a Rede de Petri desenhada.

5.3.4 BehaviorFoulAttacker, BehaviorFoulDefender, BehaviorFoulSupporter

Estes comportamentos individuais são executados pelos robots não envolvidos na reposição de bola em campo, tanto estando a bola na posse da própria equipa como na posse do adver-

Tabela 5.3.3: Comportamento *BehaviorBaseSupport*.

Nome	*BehaviorBaseSupport*.
Tipo	Comportamento Individual.
Situação de jogo	Jogo não parado, sendo o robot um apoio na táctica da sua equipa.
Objectivo	Seguir a bola mantendo-se a dois metros para apoiar atacante.
Papel	*Supporter*.
Nível MeRMaID	*BehaviorExecutor*.
Apêndice	A.3.3.

sário. Os comportamentos têm como objectivo colocar o robot a uma determinada distância da bola, de costas para a sua baliza, excepto no caso da conversão de uma grande penalidade para o adversário, em que não se pode colocar entre a bola e a baliza. Nesse caso são mantidos dois metros de distância à bola entre esta e a baliza adversária. Os comportamentos designados *BehaviorFoulDefender*, *BehaviorFoulAttacker* e *BehaviorFoulSupporter* são executados pelo defesa, atacante e apoio, respectivamente. O predicado *PenaltyOpponent* indica a situação de jogo, o predicados *IsAtAttackPosition* e *IsAtDefendPosition* indicam se o robot se encontra nas posições explicadas atrás, as quais são objectivo das acções primitivas *Move2AttackPosition* e *Move2DefendPosition*, respectivamente. Caso o robot não consiga ver a bola, ou seja, quando o predicado *SeeBall* é *false*, move-se para a sua posição padrão em campo. As acções primitivas e predicados referidos dizem respeito ao caso do *BehaviorFoulSupporter.* As únicas alterações para os outros dois comportamentos dizem respeito à acção utilizada para colocar o robot na postura desejada e ao respectivo predicado. O atacante manter-se-á mais perto da bola, seguido do apoio e, mais afastado, do defesa.

Na Tabela 5.3.4 é resumida a informação sobre os comportamentos descritos acima. No apêndice A.3.4 pode ser consultada a Rede de Petri desenhada para o exemplo do *BehaviorFoulSupporter.*

Tabela 5.3.4: Comportamentos *BehaviorFoulSupporter*, *BehaviorFoulDefender* e *BehaviorFoulAttacker*.

Nome	*BehaviorFoulAttacker*, *BehaviorFoulDefender*, *BehaviorFoulSupporter*
Tipo	Comportamento Individual.
Situação de jogo	Reposição da bola em jogo quer esteja na posse da própria equipa ou do adversário.
Objectivo	Colocar-se a uma distância da bola entre esta e a própria baliza ou a baliza adversária, esta última no caso de uma grande penalidade para a equipa contrária.
Papel	*RoleAttacker*, *RoleDefender*, *RoleSupporter*
Nível MeRMaID	*BehaviorExecutor*.
Apêndice	A.3.4.

5.3.5 BehaviorFoulTakerAlone

Este comportamento é executado no caso do marcador não ter sucesso no seu compromisso com o receptor durante uma reposição de bola em jogo. O comportamento é muito simples tendo apenas como objectivo colocar o robot junto à bola, em posição de seguir com ela logo que receba o sinal para reinicio do jogo. Ao receber esse sinal o predicado *GameIsStarted* tomará o valor *true*, são enviados os eventos FOUL_TAKEN e BEHAVIOR_SUCCESSFULL. O primeiro para toda a equipa avisando que a falta foi marcada e o segundo para a camada acima que, tal como explicado anteriormente, levará o papel de marcador a produzir o evento FOUL_FINISHED o que fará evoluir a táctica da equipa ao nível do TO. Estando o marcador na posse da bola passará naturalmente a ser o atacante e prosseguirá o jogo.

Na Tabela 5.3.5 é resumida a informação sobre este comportamento descrito acima. No apêndice A.3.5 pode ser consultada a Rede de Petri desenhada.

Tabela 5.3.5: Comportamento *BehaviorFoulTakerAlone*.

Nome	*BehaviorFoulTakerAlone*.
Tipo	Comportamento Individual.
Situação de jogo	Reposição da bola em jogo para a própria equipa
Objectivo	Colocar-se junto à bola.
Papel	*RoleFoulTaker*
Nível MeRMaID	*BehaviorExecutor*.
Apêndice	A.3.5.

Capítulo 6

Resultados e Análise

6.1 Resultados

Os resultados obtidos foram bastante satisfatórios, conseguindo-se executar com sucesso todos os comportamentos nos robot reais e com muito sucesso no simulador. Muitos dos comportamentos desenvolvidos foram utilizados pelo projecto durante as participações que a equipa ISocRob integrou no período de desenvolvimento deste trabalho: Festival Nacional de Robótica 2007 e RoboCup 2007.

Os vídeos da execução dos comportamentos tanto no simulador como nos robots reais podem ser consultados nos sítio:
`http://socrob.isr.ist.utl.pt/videos/behaviors/behaviors.html`
Como referido atrás, tanto no simulador como nos robots reais, o código corrido foi o mesmo.

6.2 Análise

Nesta secção são apresentados os resultados da análise quantitativa realizada. Os ensaios realizados forem feitos no sentido de responder a algumas questões pertinentes enquanto no processo de criação de um novo comportamento, por exemplo:

- Qual o impacto do insucesso das acções primitivas no insucesso de um comportamento?
- Para um determinado desempenho desejado para o comportamento, qual deve ser o desempenho das acções primitivas?
- Alguma acção primitiva da cadeia que constitui o comportamento terá maior impacto no seu sucesso do que as restantes?
- Qual o impacto, no sucesso do comportamento, de aumentar o desempenho de uma acção primitiva ?
- Qual o impacto do número de acções primitivas no sucesso de um comportamento?

Para esta análise foi utilizada a ferramenta TimeNet[31] que permite analisar RdP da classe GSPN. Seguindo a teoria apresentada na Secção 2.1.3 todas as análises consideraram para cada acção primitiva duas hipóteses de desfecho, sucesso e insucesso, as quais foram simuladas utilizando duas transições exponenciais. A Figura 6.2.1 apresenta um exemplo de uma rede desse género para um comportamento com quatro acções.

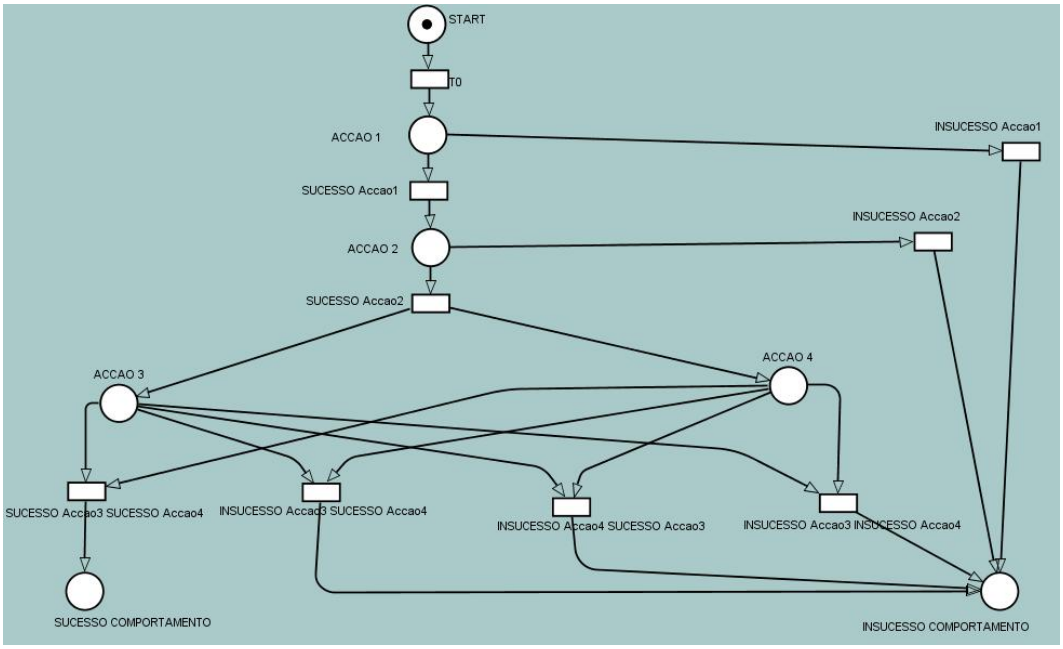


Figura 6.2.1: Exemplo de rede analisada.

Naturalmente, o insucesso de uma acção primitiva leva ao insucesso do comportamento, enquanto o consecutivo sucesso das mesmas acções conduzem ao seu sucesso. Fixando a taxa da transição que representa o sucesso e variando de forma crescente a taxa da transição que representa o insucesso foi conseguida a variação da probabilidade de sucesso de cada acção primitiva.

Cada uma das secções seguintes apresenta os resultados da análise realizada e as respostas retiradas para cada uma das perguntas.

6.2.1 Impacto do desempenho das acções primitivas

Esta análise tem como objectivo estudar o desempenho de um comportamento em função do sucesso das acções primitivas. O desempenho de qualquer acção primitiva está intimamente ligado ao sucesso do comportamento em que está inserido. Contudo, algumas acções primitivas tomam uma importância ainda mais fulcral, especialmente no caso dos comportamentos relacionais. Veja-se o caso das acções primitivas de comunicação, uma falha na transmissão de uma mensagem comprometerá o objectivo do comportamento, ou seja, para que o comportamento seja infalível as comunicações também o terão de ser. Do mesmo modo podemos inferir que caso as comunicações falhem em metade das execuções o comportamento nunca poderá ter uma probabilidade de sucesso superior a esses 50%.

Os resultados apresentados nesta secção foram retirados de simulações realizada para um comportamento individual, em que não são utilizadas comunicações e portanto pode-se perceber o verdadeiro impacto do desempenho das acções primitivas. Foi utilizado o comportamento *BehaviorBaseAttack*, descrito na Secção 5.3.1. Dado que este comportamento apresenta, na sua etapa final, comportamentos em paralelo, foi necessário contemplar os vários casos de sucesso/insucesso das duas acções, ou seja, apenas se considera o conjunto bem sucedido caso as duas acções cumpram o seu objectivo, considerando-se falha no comportamento no caso do insucesso de uma ou das duas acções. Esta situação é idêntica ao que acontece no exemplo da Figura 6.2.1

O gráfico da Figura 6.2.2 apresenta os resultados obtidos na simulação, utilizando a RdP do comportamento *BehaviorBaseAttack*. A probabilidade de sucesso das acções foi variada de 0 a 100%, considerando que todas as acções têm o mesmo desempenho, e foi calculado a percentagem de execuções do comportamento bem sucedidas.

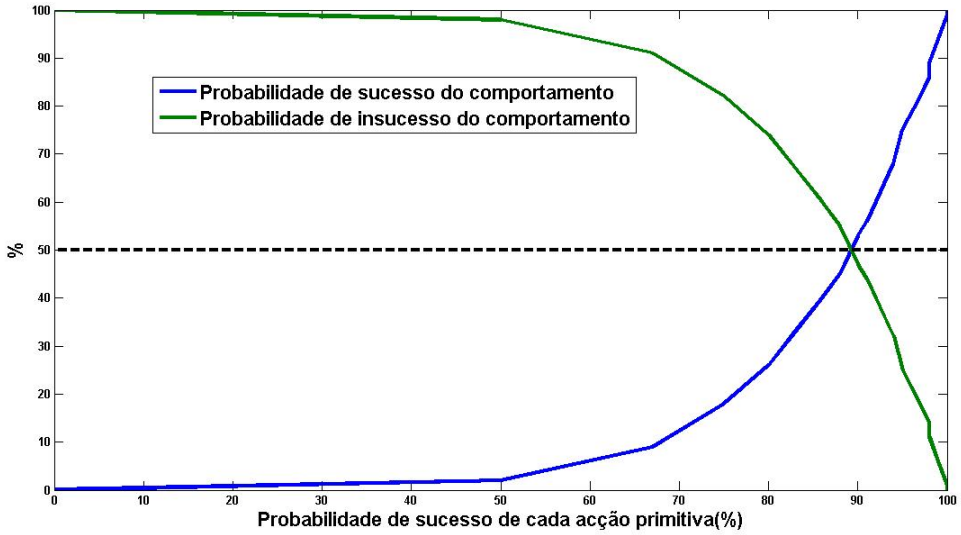


Figura 6.2.2: Impacto do desempenho das acções primitivas no sucesso do comportamento.

A análise do gráfico permite constatar que quanto melhor o desempenho de cada acção primitiva, maior o declive da curva. Facilmente se constata que uma evolução da probabilidade de sucesso das acções de 0 a 90% conduz a um crescimento de 0 a 50 % de comportamentos bem sucedidos, enquanto os restantes 10% têm como resultado o incremento presente na outra metade do gráfico. De facto, no caso de todas as acções terem o mesmo desempenho, é necessário uma grande qualidade na sua execução para que o comportamento apresente um bom índice de sucesso. Por exemplo, considerando este comportamento, para se obter sucesso 80 vezes em cada 100 execuções é necessário cada acção apresentar cerca de 96% de eficácia, o que é um valor bastante elevado. Uma das razões para este resultado prende-

se com o elevado número de acções primitivas que envolve este comportamento até que se cumpra o seu objectivo. Na Secção 6.2.3 é apresentada uma análise que teve como objectivo perceber o impacto do número de acções primitivas no desempenho de um comportamento.

6.2.2 Impacto do sucesso de cada acção primitiva

Esta análise teve como objectivo perceber o impacto do desempenho de uma única acção primitiva no sucesso do um comportamento. Foi usado o comportamento *BehaviorBaseAttack* tal como na Secção 6.2.1.

Foram feitos ensaios para cada acção primitiva fixando a probabilidade de sucesso das outras acções e variando a probabilidade de sucesso da acção primitiva em estudo.

O gráfico da Figura 6.2.3 apresenta os resultados obtidos na simulação. Naturalmente as acções primitivas que apenas figuram no comportamento uma vez apresentaram os mesmos resultados. Sendo assim distinguem-se dois casos: os resultados para a acção Dribble2Score / Aim2Score (Caso 2 no gráfico) e os casos para as restantes acções (Caso 1 no gráfico). Foi feita uma simulação para três situações: uma primeira em que a probabilidade fixa de cada acção primitiva seria 50% (A=50), uma segunda em que essa probabilidade tomaria o valor de 91%(A=91) e por fim o caso em que as acções se apresentariam quase infalíveis tendo índices de sucesso na ordem dos 98%(A=98).

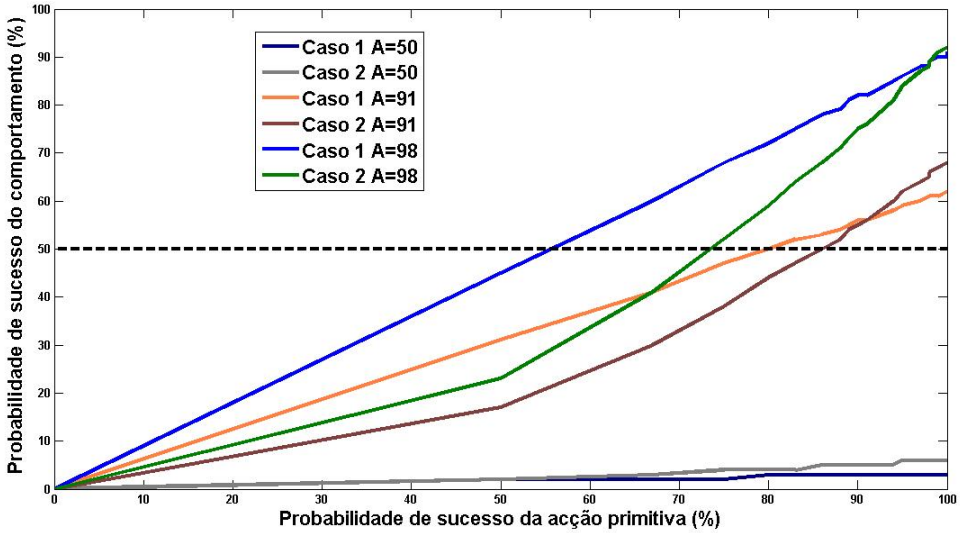


Figura 6.2.3: Impacto do sucesso de acções primitivas no sucesso do comportamento (1).

Rapidamente se constata que para todas as acções que apenas figuram uma vez no comportamento(linhas identificadas com sigla *Caso 1*) a variação do seu desempenho apresenta uma relação linear com a probabilidade de sucesso do comportamento. Pode-se ainda acrescentar que o declive da recta que descreve essa relação é tanto maior quanto maior a qual-

idade das acções primitivas que constituem o comportamento, ou seja, o melhoramento de uma acção primitiva terá maior impacto quanto melhores forem as outras acções primitivas. Veja-se a linha *Caso 1 A=50*, neste caso o melhoramento de uma acção, de uma probabilidade de sucesso de 50% para uma probabilidade de sucesso de 100%, apenas eleva o sucesso do comportamento dos 2% para os 3%. Facilmente se concluí que, caso as acções primitivas não tenham grandes desempenhos, será completamente inútil investir apenas numa, devendo-se investir no conjunto. A mesma situação acontece para o caso das acções que figuram uma segunda vez no comportamento. Contudo, neste caso verificam-se dois tipos de variação, sendo o impacto do desempenho da acção primitiva no desempenho do comportamento mais acentuado a partir de um certo valor, em torno dos 50%.

Foi ainda realizada um segundo tipo de análise considerando o seguinte cenário: uma das acções que apenas figura uma vez no comportamento e uma das acções que figura por duas vezes apresentam a mesma probabilidade inicial, no caso 70%. Para cada simulação fixou-se a probabilidade de sucesso das restantes acções (em 80, 91 e 98%) e variou-se em cada conjunto de ensaios a probabilidade de sucesso de uma das acções cuja probabilidade era inicialmente 70%. Esta simulação pretende recriar a situação em que o comportamento é constituído por acções com um desempenho relativamente bom e pretende-se decidir em qual das acções vale a pena investir. Apesar de ser intuitivo que se deve investir numa acção que figure por mais de uma vez, dado que contribui duas vezes para o execução do comportamento, o objectivo é perceber qual o verdadeiro impacto desse investimento ao invés de se investir numa acção que apenas tome lugar um vez no comportamento. Imaginemos que se pretende melhorar o comportamento mas a acção que figura duas vezes é muito mais complexa e melhorá-la revela um trabalho complicado, esta análise ajuda na decisão de iniciar esse trabalho ou optar por melhorar uma acção primitiva que tenha menos impacto mas seja mais simples.

O gráfico da Figura 6.2.4 apresenta os resultados obtidos na simulação.

Conforme referido anteriormente e ficando agora demonstrado, é preferível investir numa acção que figure por mais de uma vez no comportamento. Pode-se ainda retirar do gráfico que a diferença de impacto causado é tanto maior quanto melhor for o comportamento, ou seja, dado um estado de desenvolvimento será tão mais aconselhável investir numa acção que figure por mais de uma vez, p.e. *Dribble2Score* e do *Aim2Score*, quanto mais perto do 100% de sucesso estiver o comportamento em questão. Essa diferença chega quase aos 20% no caso de um comportamento cujas restantes acções apresentem uma probabilidade de sucesso de 98% e se tenha investido numa ou noutra acção primitiva por forma a igualar esse valor, o que é uma diferença bastante acentuada.

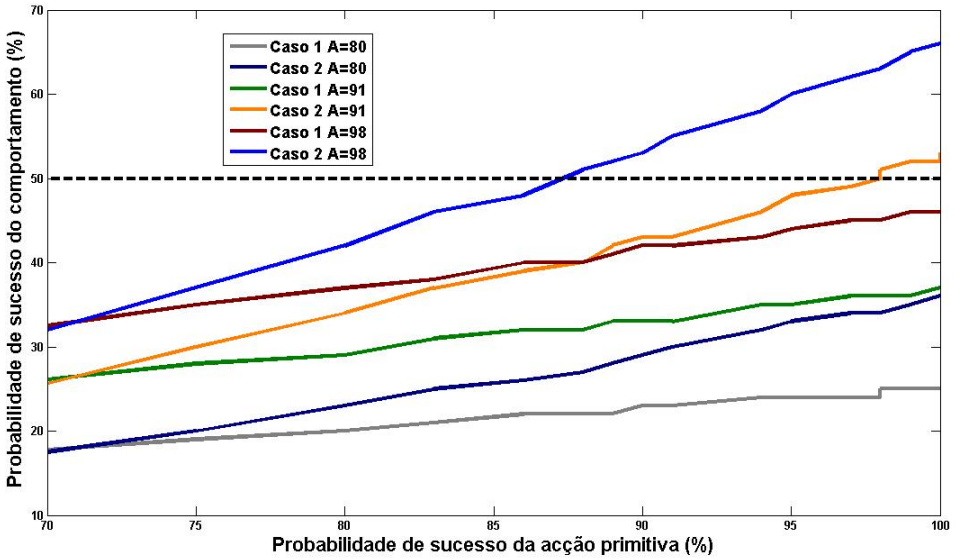


Figura 6.2.4: Impacto do sucesso de acções primitivas no sucesso do comportamento (2).

6.2.3 Impacto do número de acções primitivas

A maioria dos comportamentos apresenta-se como uma série de acções primitivas. Naturalmente, quanto maior o número de acções primitivas maior é a probabilidade de insucesso do comportamento dado que adicionar acções primitivas é sinónimo de adicionar ponto de possível falha a esses mesmos comportamentos. Nesta secção são apresentados os resultados da análise realizada com vista a perceber qual a dimensão dessa dependência.

O gráfico da Figura 6.2.5 apresenta os resultados obtidos na simulação. Foi variado o desempenho das acções primitivas para comportamentos de diversas dimensões.

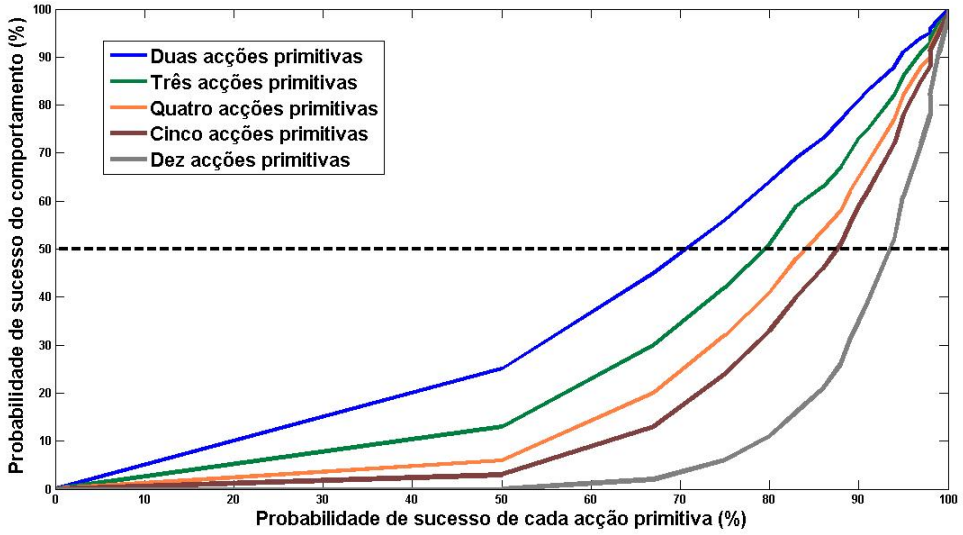


Figura 6.2.5: Impacto do número de acções primitivas no sucesso do comportamento.

Como facilmente se constata, confirma-se o que foi inferido anteriormente, isto é, o aumento do número de acções na cadeia, sem melhoramento dessas mesmas acções, conduz à diminuição da probabilidade de sucesso do comportamento. Esse decréscimo é maior quanto pior o desempenho das acções primitivas. Contudo, para comportamentos com altos desempenhos o impacto é ainda grande. Veja-se o exemplo de um comportamento que envolva quatro acções em cadeia, o que olhando aos comportamentos desenvolvidos já apresenta alguma dimensão,e que apresente um índice de sucesso de 94%. Caso seja adicionada uma nova acção primitiva a probabilidade de sucesso do comportamento descerá dos 77% para os 72%, o que é uma queda significativa.

Conclui-se assim que o desenvolvimento de comportamentos deve promover primeiramente o melhoramento do desempenho das acções primitivas para que possam ser desenvolvidos comportamentos mais complexos. Caso contrário, ou seja, caso as acções primitivas não se apresentem extremamente eficazes, será mais lucrativo apostar em comportamentos mais simples.

Capítulo 7

Conclusões e Trabalho Futuro

Esta dissertação teve como objectivo a criação, demonstração e análise do funcionamento de comportamentos relacionais, desenvolvidos no âmbito do projecto SocRob, para situações de jogo estáticas. Para modelar os comportamentos utilizou-se Redes de Petri. Foi criado, a partir de trabalho anteriormente desenvolvido, um algoritmo para sistematizar o desenvolvimento de novos comportamentos. Como requisito essencial a uma tarefa que envolve vários agentes, foram implementados mecanismos de estabelecimento e gestão do compromisso bem como de sincronização. Os comportamentos desenvolvidos foram testados num simulador e nos robots reais, utilizando o mesmo código. Obtiveram-se resultados bastante satisfatórios, conseguindo-se executar com sucesso todos os comportamentos nos robot reais e com muito sucesso no simulador. Muitos dos comportamentos desenvolvidos foram utilizados pelo projecto durante as competições que a equipa ISocRob integrou no período de desenvolvimento deste trabalho: Festival Nacional de Robótica 2007 e RoboCup 2007. Finalmente, foi realizada uma análise quantitativa das Redes de Petri desenvolvidas.

Este trabalho permite concluir que os comportamentos relacionais são um assunto a investir na área do futebol robótico. Este facto é não só justificado por motivos competitivos, dado que uma equipa que jogue relacionando os seus jogadores obtém uma enorme vantagem sobre o adversário, como por motivos científicos, apresentado-se este tema como da maior importância para outras áreas da robótica.

Como trabalho futuro nesta área sera interessante para o projecto SocRob o desenho de comportamentos para a realização de passes dinâmicos, ou seja, passes realizados durante o jogo corrido. Ainda na área do trabalho em equipa, no próprio futebol robótico ou na busca e salvamento, poderá ser uma mais valia o estudo de diversos sistemas de decisão de que robots devem intervir num comportamento. Por exemplo, no caso de decidir que jogadores devem intervir num comportamento relacional seria interessante fazer um estudo experimentando os diversos sistemas políticos conhecidos e eleger qual o que terá mais resultado. Esta análise teria como extremos um regime absolutista, em que um robot tomará todas essas decisões, e um regime igualitário, em que cada robot tomará igual parte na decisão.

Bibliografia

[1] R. David, H. Alla, *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*, New York : Prentice Hall Editions, 1992

[2] Christos G. Cassandras *Discrete Event Systems - Modelling and Performance Analysis*, Aksen Associates, 1993

[3] James L. Peterson, *Petri Net Theory and the Modelling of Systems*, Prentice-Hall,1981

[4] Moody, J. and Antsaklis, *Supervisory Control of Discrete Event Systems*, Kluwer Academic Publishers, 1998

[5] N. Viswanadham and Y. Narahari., *Performance Modeling of Automated Manufacturing Systems*, Prentice Hall, New Jersey, 1992

[6] P. R. Cohen and H. J. Levesque. Teamwork. Nous, 35:487-512, 1991

[7] Vittorio Amos Ziparo and Luca Iocchi, *Petri Net Plans*, 2006

[8] Pedro U. Lima e Luis M. M. Custódio, *Artificial Intelligence and Systems Theory Applied to Cooperative Robots*, 2002

[9] T. Murata, *Petri Nets: Properties, Analysis and Applications*, 1989

[10] Francisco Melo, M. Isabel Ribeiro, Pedro Lima, *Navigation Controllability of a Mobile Robot Population*, Proc. RoboCup 2004 Symposium, Lisbon, Portugal, 2004

[11] D. Milutinovic and P. Lima, *Petri net models of robotic tasks*, In. Proc. of IEEE 2002 Int. Conf. on Robotics and Automation (ICRA. 2002), May 2002

[12] Bob van der Vecht, *Behaviour Coordination for Cooperative Multi-Robot Systems*, 2004

[13] João Rafael Risso Milhinhos e Gonçalo Pais de Mouro Vaz, *Comportamentos Relacionais para Robots Futebolistas*, Relatório de Trabalho Final de Curso da Licenciatura em Engª Informática e de Computadores (LEIC), 2005

[14] Vasco Maria d'Athouguia de Albuquerque d'Orey, *Comportamentos Relacionais para Robots Futebolistas*, Relatório de Trabalho Final de Curso da Licenciatura em Engª Informática e de Computadores (LEIC), 2005

[15] Pedro U. Lima, *Models of robotic tasks based on discrete event and hybrid systems*, 2006

[16] Hugo Costelha et al, *ISocRob 2007 Team Description Paper*, 2007

[17] Marco Barbosa, Nelson Ramos e Pedro Lima, *MeRMaID - Multiple-Robot Middleware for Intelligent Decision-making,*, 2007

[18] Projecto SocRob, *Projecto IsocRob - Descrição*, http://socrob.isr.ist.utl.pt/pt-desc.php

[19] The RoboCup Federation, *RoboCup Brief Introduction*, http://www.robocup.org/Intro.htm

[20] Projecto Rescue, Navegação Cooperativa para Robots de Salvamento, http://rescue.isr.ist.utl.pt/pt-desc.php

[21] Hans-Dieter Burkhard, Dominique Duhaut, Masahiro Fujita, Pedro Lima, Robin Murphy e Raul Rojas, *Past Progress Brings Us Towards a Research Road Map for Further Competitions and De velopments*, 2002

[22] MSL Technical Committee 1997-2007, *Middle Size Robot League Rules and Regulations for 2007*, 2007

[23] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli and G. *Franceschinis Modelling with Generalized Stochastic Petri Nets Wiley Series in Parallel Computing*, John Wiley and Sons, 1995

[24] Simona Bernardi, *Building Stochastic Petri Net models for the verification of complex software systems*, Universit'a degli Studi di Torino

[25] Montgomery, Douglas C. e Runger, George C. (2003) *Applied Statistics and Probability for Engineers*, 3rd Edition. John Wiley & Sons, New York.

[26] Pawel Zebrowski, Simon Fraser University, *Communication In Multi-Robot Systems* Spring 2004, http://www.sfu.ca/ pzebrows/cmpt816/

[27] D. Agrawal, *The Performance of Two-Phase Commit Protocols in the Presence of Site Failures*, 1994

[28] *Distributed Computing, Lesson 24: Three-Phase commit, recovery, stable storage*, RAI University

[29] Matthias Jüngel, Ekkart Kindler, Michael Weber *The Petri Net Markup Language*, Agosto 2000

[30] Ricardo Sangoi Padilha, *JARP*, http://jarp.sourceforge.net/

[31] Zimmermann, A.; Freiheit, J.; German, R. Hommel, G.: *Petri Net Modelling and Performability Evaluation with TimeNET 3.0.*, 11th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'2000), LNCS 1786, pp. 188-202, http://pdv.cs.tu-berlin.de/ timenet/, Springer-Verlag, Schaumburg, Illinois, USA, 2000

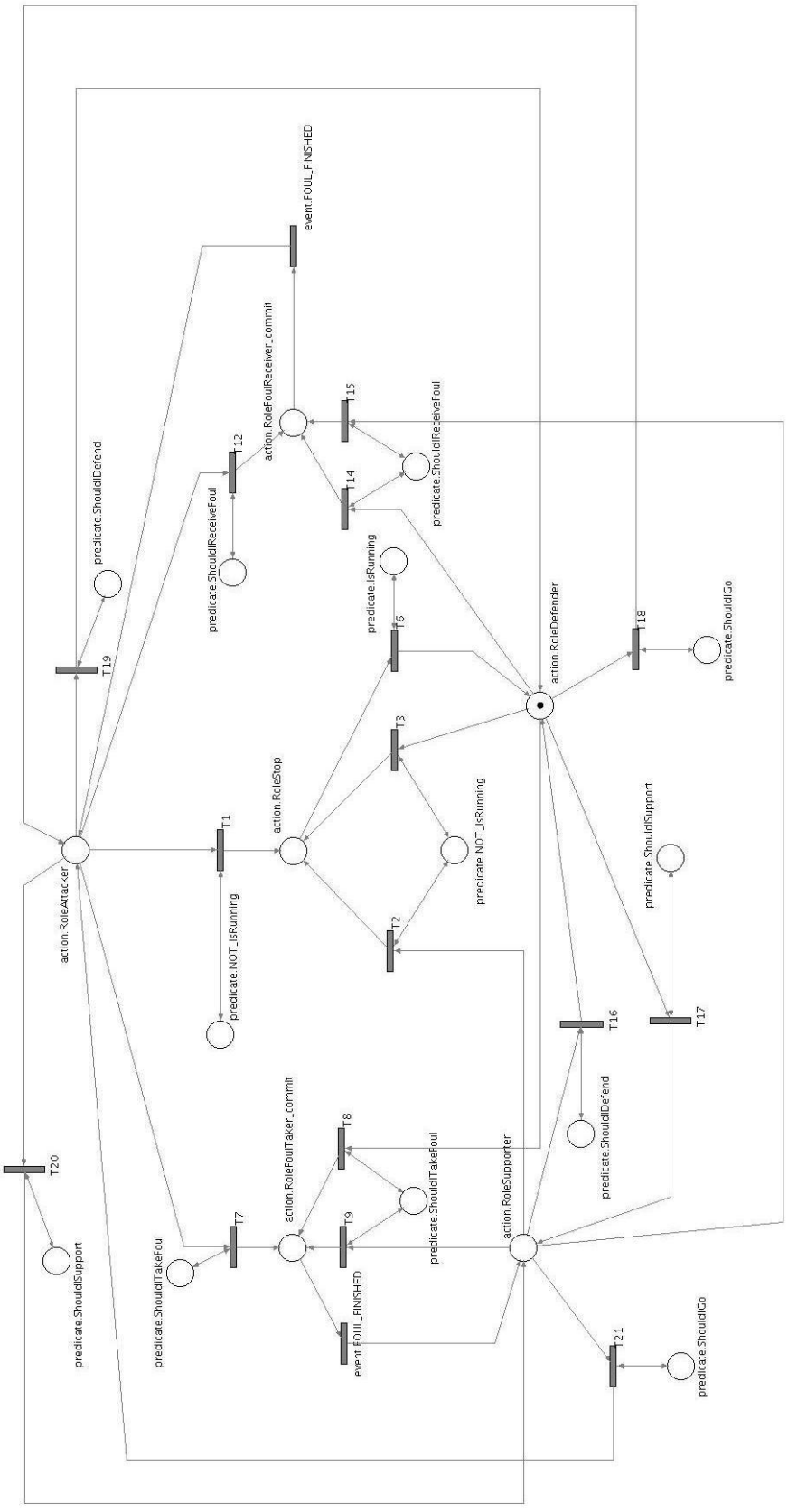
[32] WEBOTS, http://www.cyberbotics.com/

Apêndice A

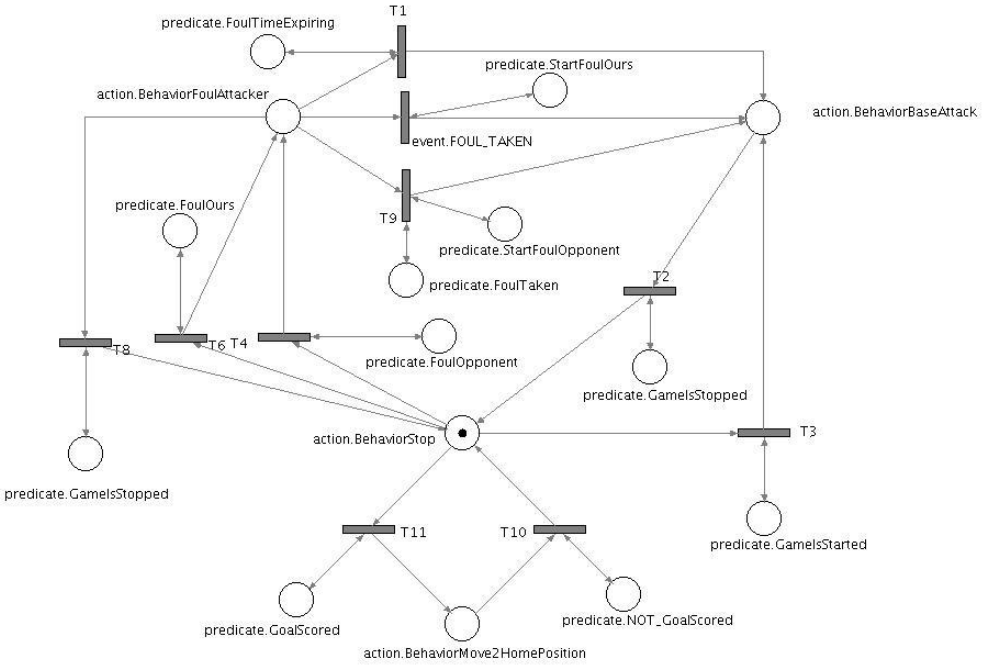
Redes de Petri dos Comportamentos Desenvolvidos

A.1 Comportamentos Organizacionais - Redes de Petri

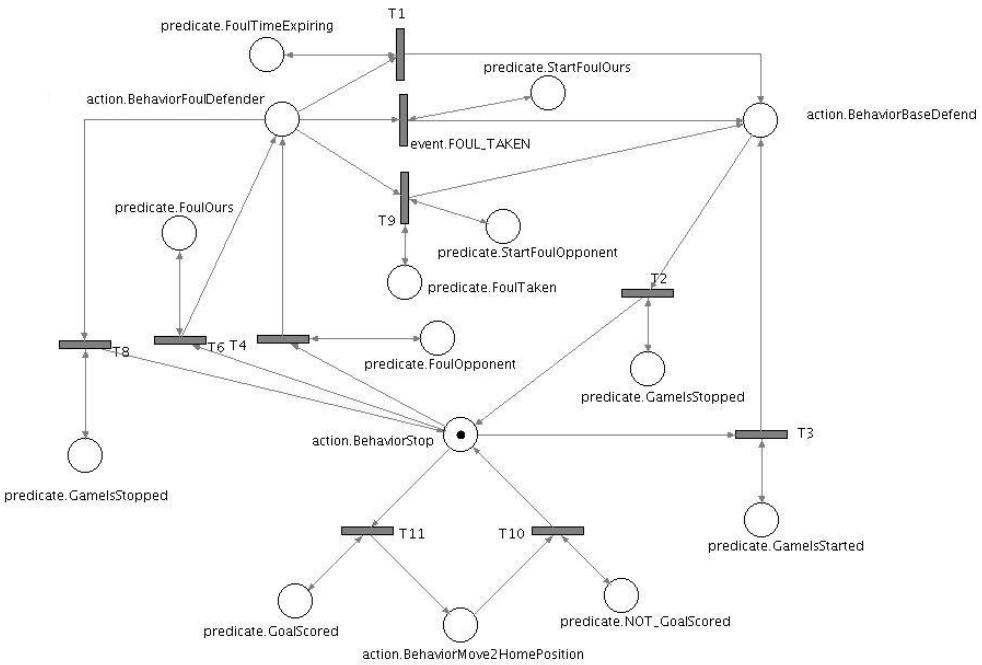
A.1.1 TacticBase - Rede de Petri


event.FOUL_FINISHED
action.RoleFoulReceiver_commit
T15
T12
T14
predicate.ShouldIReceiveFoul
predicate.ShouldIReceiveFoul
predicate.ShouldIDefend
T19
predicate.IsRunning
T6
action.RoleDefender
T18
predicate.ShouldIGo
T3
action.RoleStop
T1
predicate.NOT_IsRunning
action.RoleAttacker
predicate.NOT_IsRunning
T2
predicate.ShouldISupport
T16
T17
T20
T8
action.RoleFoulTaker_commit
predicate.ShouldIDefend
predicate.ShouldISupport
T7
T9
predicate.ShouldITakeFoul
action.RoleSupporter
predicate.ShouldITakeFoul
event.FOUL_FINISHED
T21
predicate.ShouldIGo

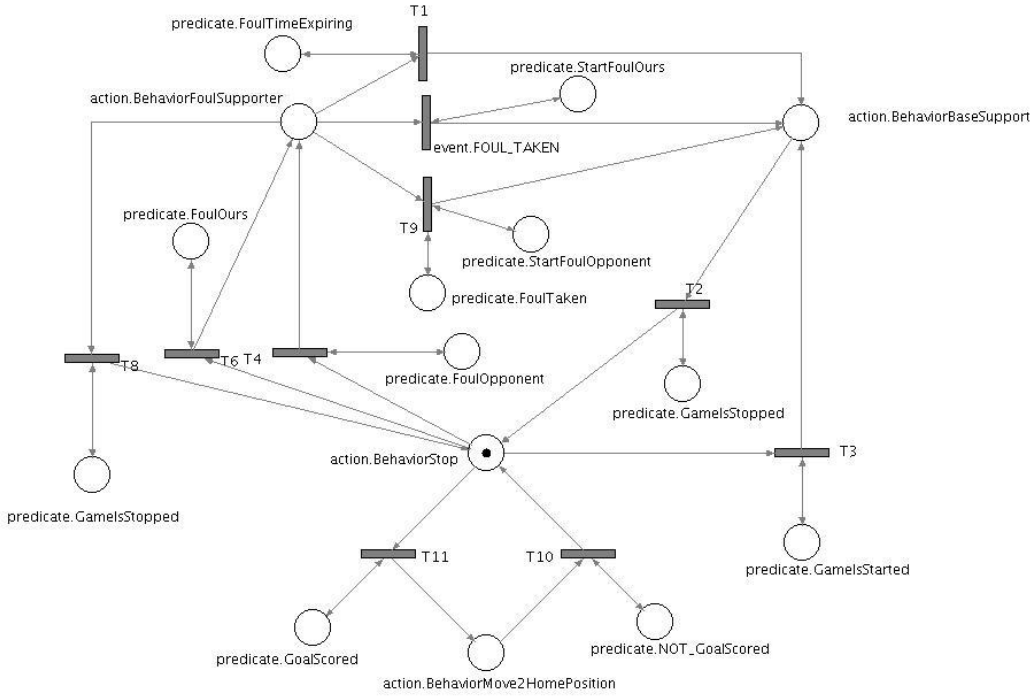
A.1.2 RoleAttacker - Rede de Petri



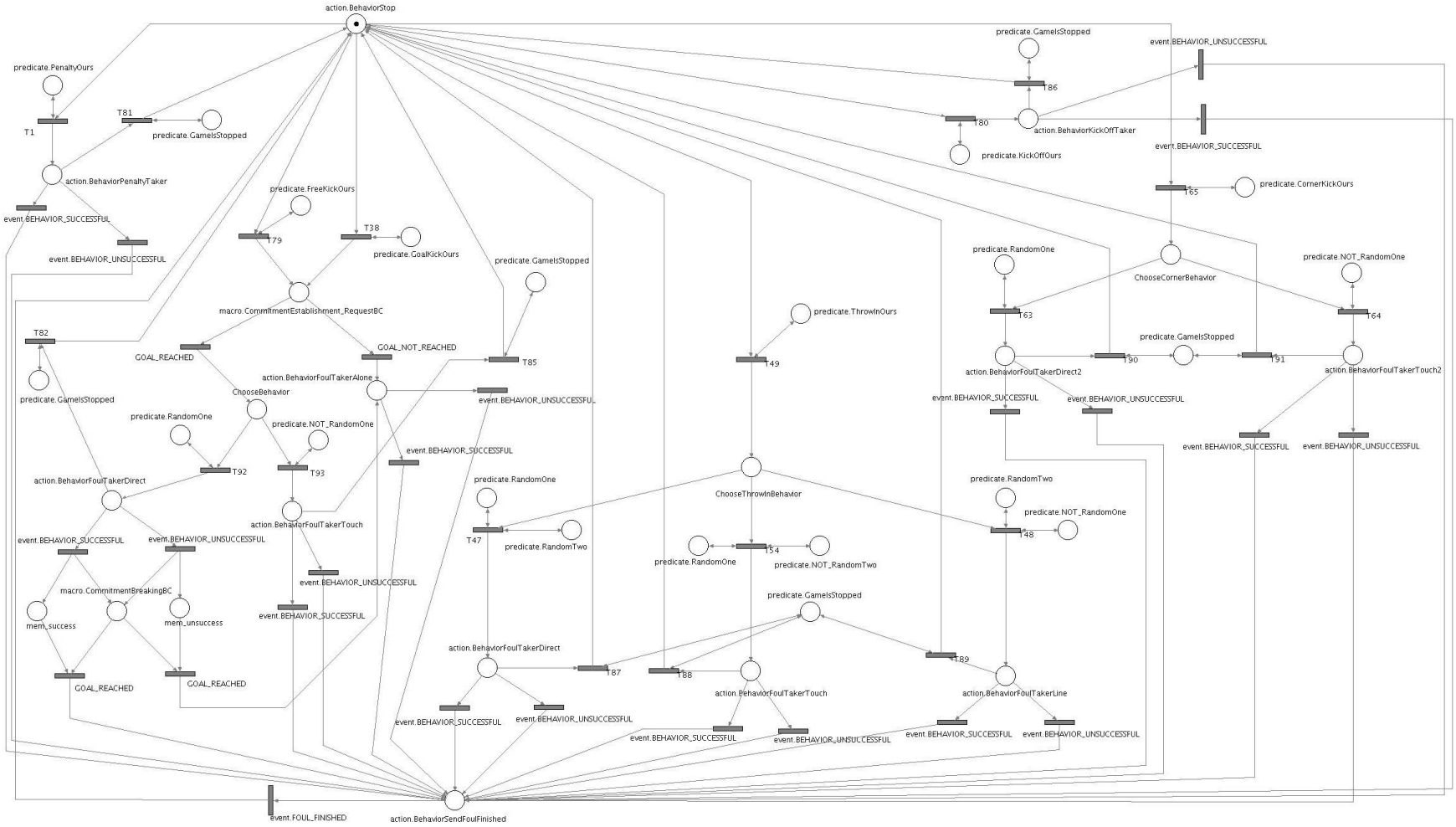
A.1.3 RoleDefender - Rede de Petri



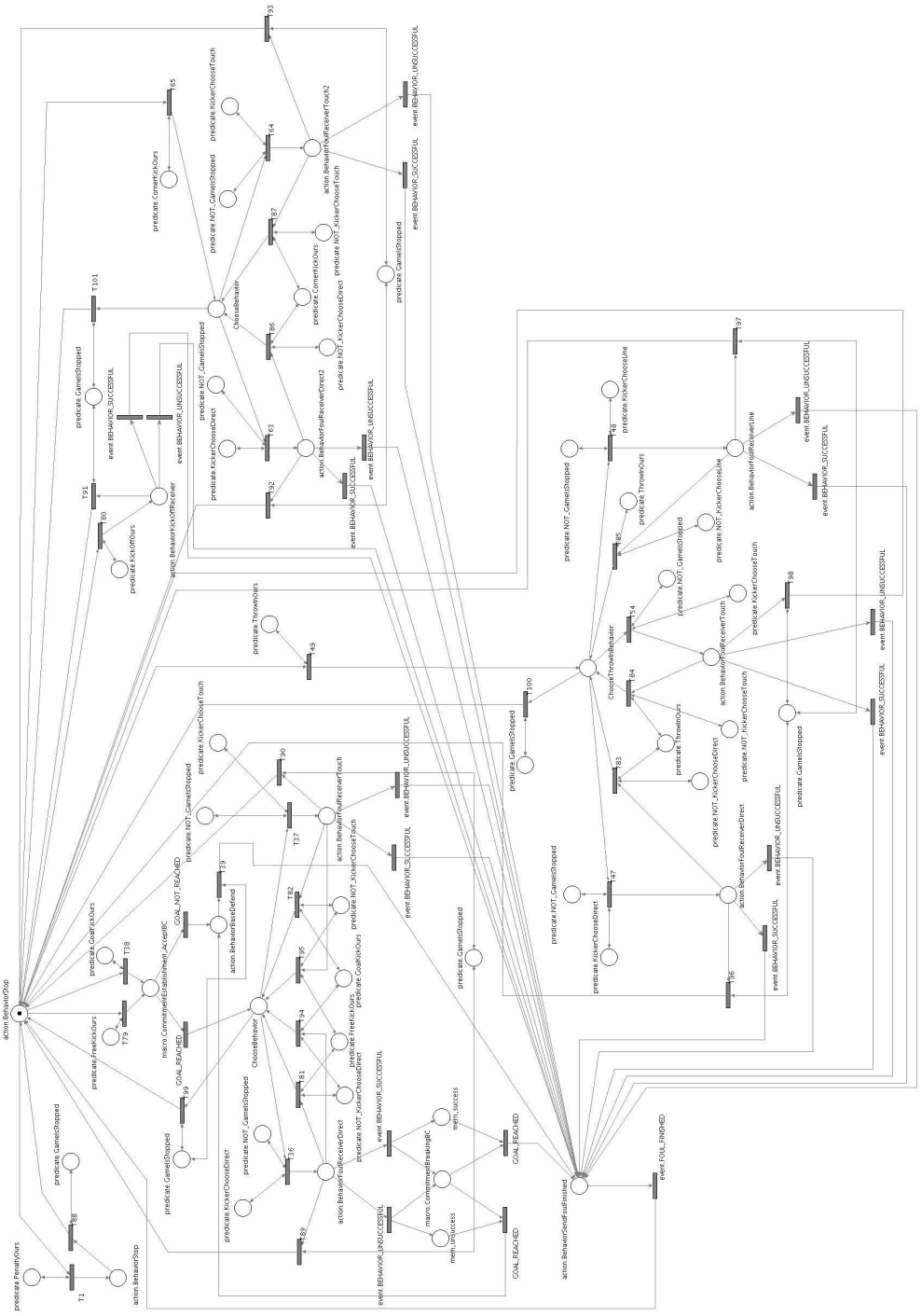
A.1.4 RoleSupporter - Rede de Petri



A.1.5 RoleFoulTaker - Rede de Petri

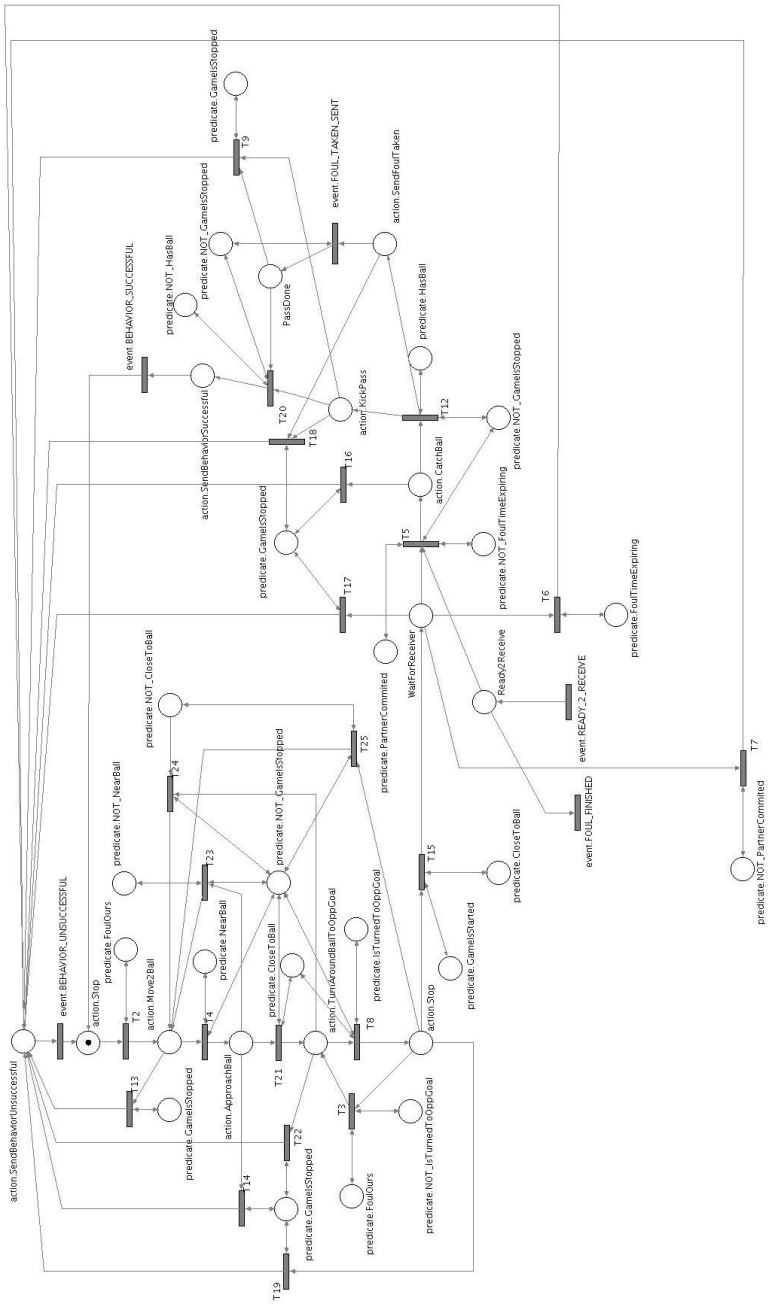

action.BehaviorStop
predicate.PenaltyOurs
T1
T81
predicate.GameIsStopped
action.BehaviorPenaltyTaker
event.BEHAVIOR_SUCCESSFUL
event.BEHAVIOR_UNSUCCESSFUL
predicate.FreeKickOurs
T79
T38
predicate.GoalKickOurs
macro.CommitmentEstablishment_RequestBC
T82
predicate.GameIsStopped
GOAL_REACHED
GOAL_NOT_REACHED
T85
action.BehaviorFoulTakerAlone
ChooseBehavior
event.BEHAVIOR_UNSUCCESSFUL
predicate.RandomOne
predicate.NOT_RandomOne
T92
T93
event.BEHAVIOR_SUCCESSFUL
action.BehaviorFoulTakerDirect
action.BehaviorFoulTakerTouch
event.BEHAVIOR_SUCCESSFUL
event.BEHAVIOR_UNSUCCESSFUL
event.BEHAVIOR_UNSUCCESSFUL
event.BEHAVIOR_SUCCESSFUL
macro.CommitmentBreakingBC
mem_success
mem_unsuccess
GOAL_REACHED
GOAL_REACHED
predicate.GameIsStopped
predicate.GameIsStopped
T86
T80
action.BehaviorKickOffTaker
predicate.KickOffOurs
event.BEHAVIOR_UNSUCCESSFUL
event.BEHAVIOR_SUCCESSFUL
T65
predicate.CornerKickOurs
ChooseCornerBehavior
predicate.RandomOne
predicate.NOT_RandomOne
T63
T64
predicate.GameIsStopped
action.BehaviorFoulTakerDirect2
T90
T91
action.BehaviorFoulTakerTouch2
event.BEHAVIOR_SUCCESSFUL
event.BEHAVIOR_UNSUCCESSFUL
event.BEHAVIOR_SUCCESSFUL
event.BEHAVIOR_UNSUCCESSFUL
predicate.ThrowInOurs
T49
ChooseThrowInBehavior
predicate.RandomOne
T47
predicate.RandomTwo
predicate.RandomTwo
predicate.NOT_RandomOne
T48
predicate.RandomOne
T54
predicate.NOT_RandomTwo
predicate.GameIsStopped
action.BehaviorFoulTakerDirect
T87
T88
T89
action.BehaviorFoulTakerTouch
action.BehaviorFoulTakerLine
event.BEHAVIOR_SUCCESSFUL
event.BEHAVIOR_UNSUCCESSFUL
event.BEHAVIOR_SUCCESSFUL
event.BEHAVIOR_UNSUCCESSFUL
event.BEHAVIOR_SUCCESSFUL
event.BEHAVIOR_UNSUCCESSFUL
event.FOUL_FINISHED
action.BehaviorSendFoulFinished

A.1.6 RoleFoulReceiver - Rede de Petri

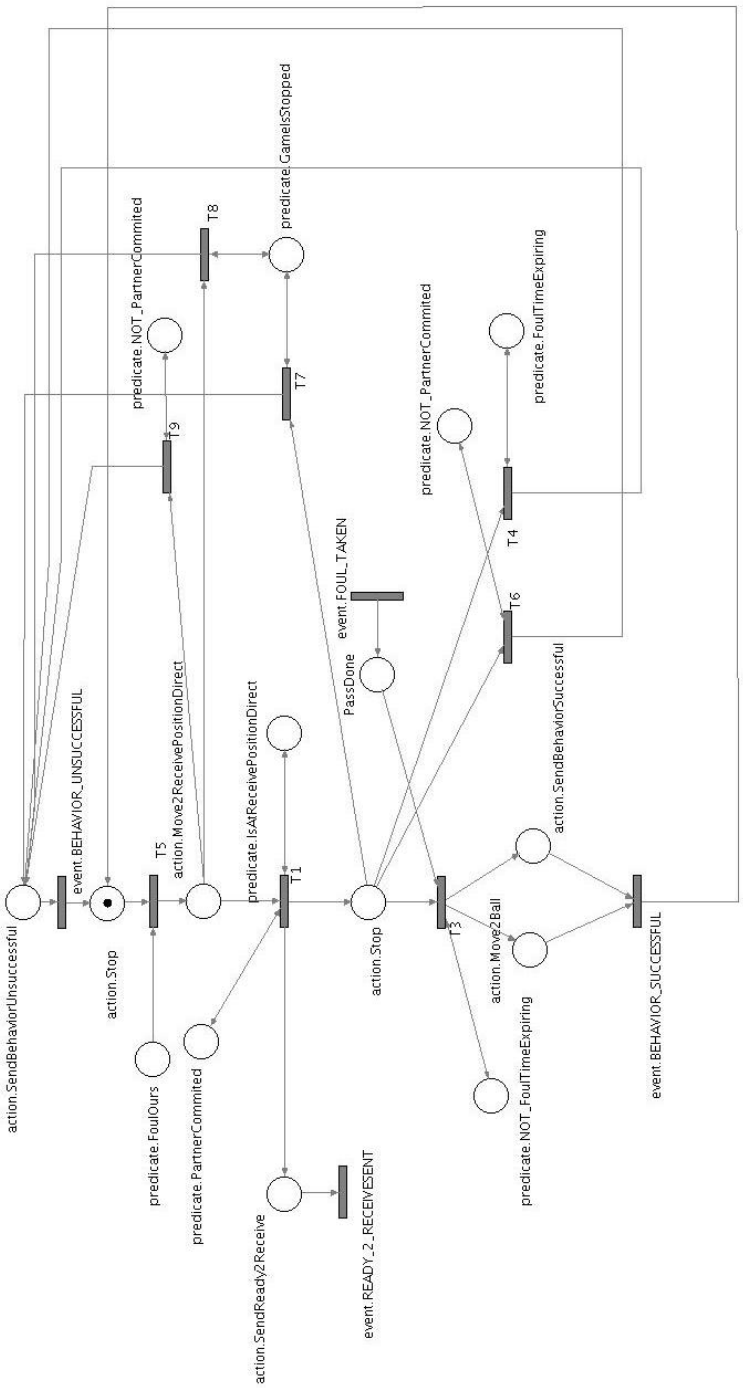


A.2 Comportamentos Relacionais - Redes de Petri

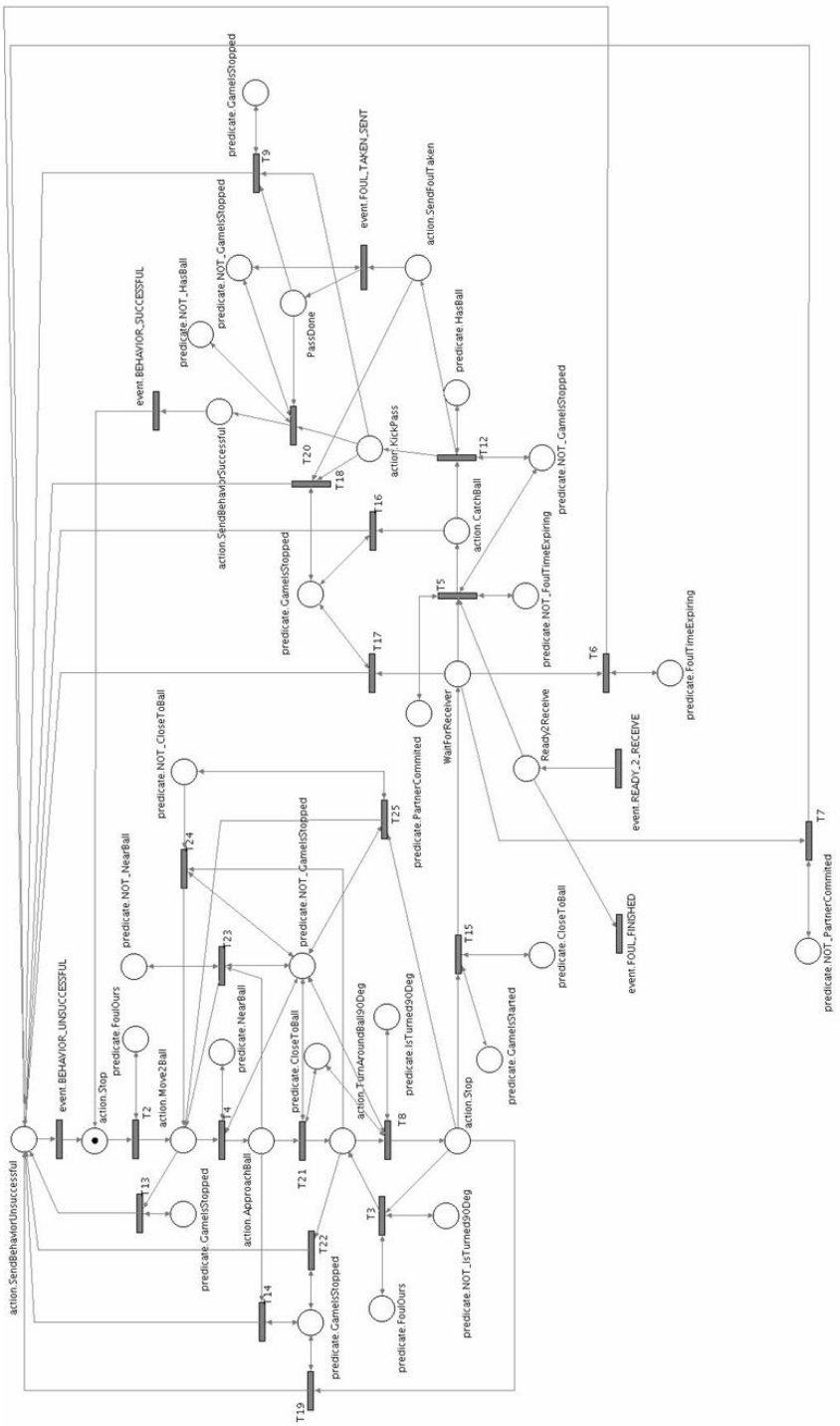
A.2.1 BehaviorFoulTakerDirect - Rede de Petri



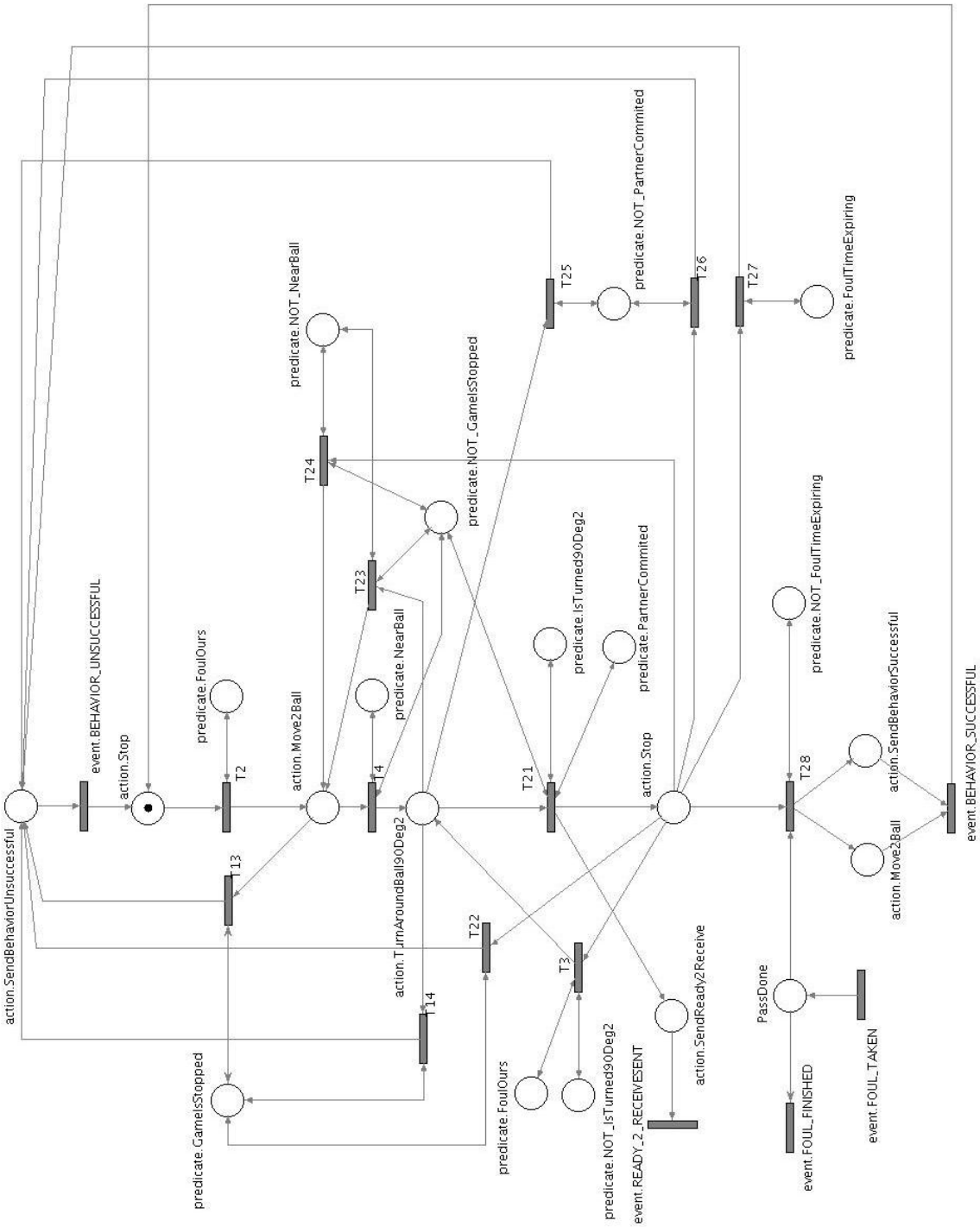
A.2.2 BehaviorFoulReceiverDirect - Rede de Petri



A.2.3 BehaviorKickOffTaker - Rede de Petri

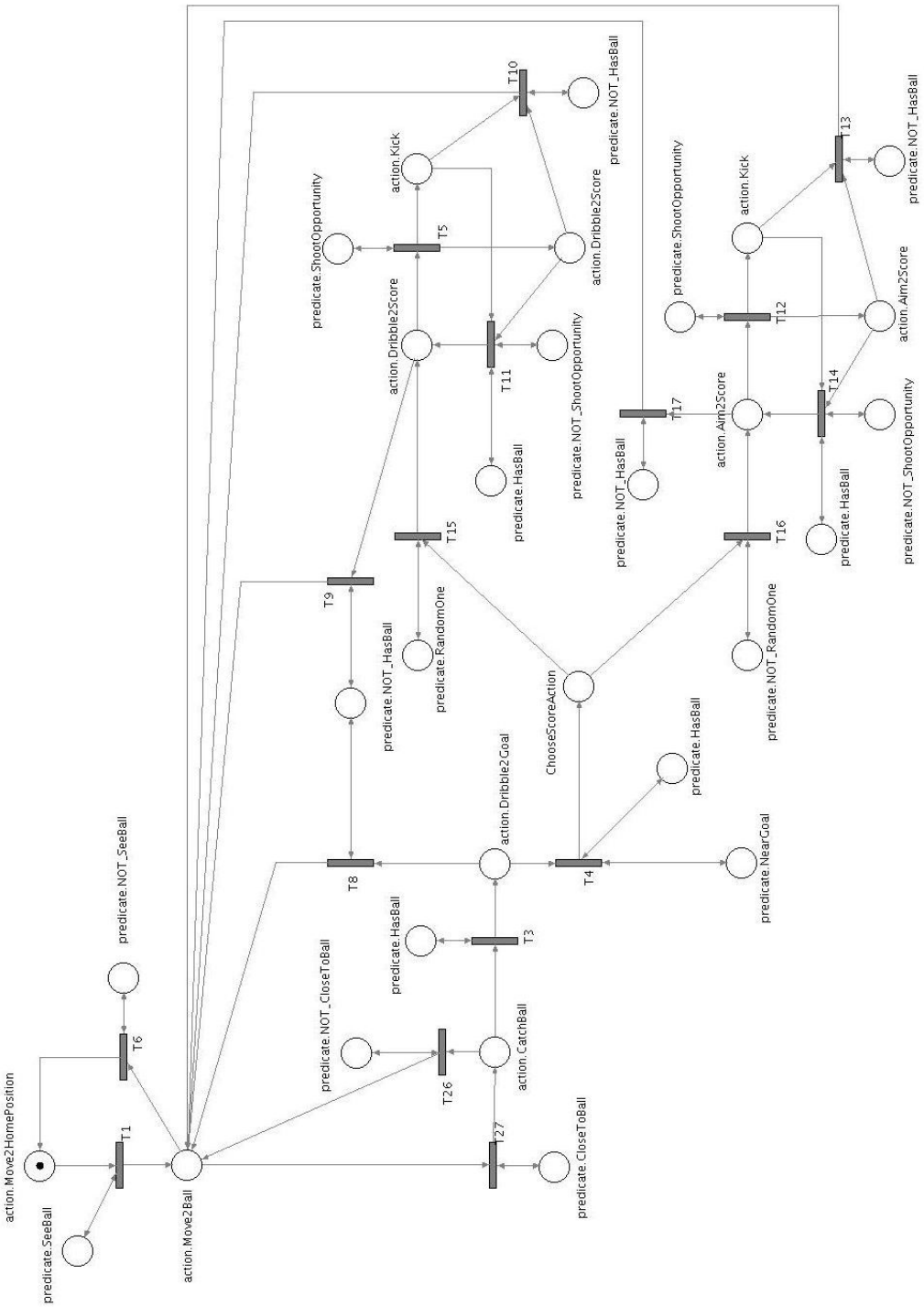


A.2.4 BehaviorKickOffReceiver - Rede de Petri

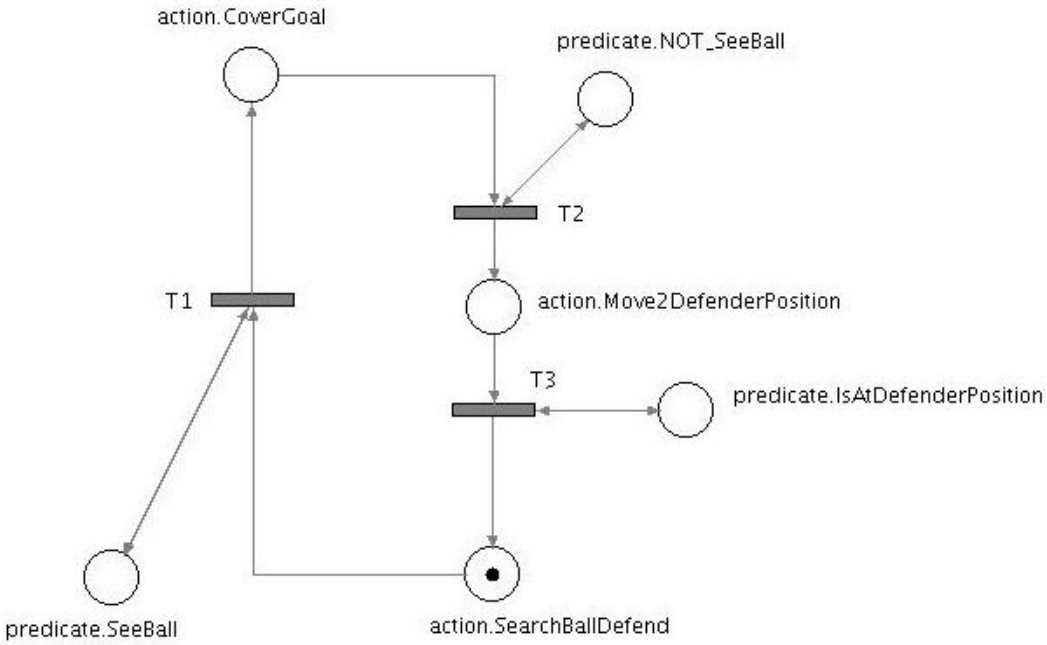


A.3 Comportamentos Individuais - Redes de Petri

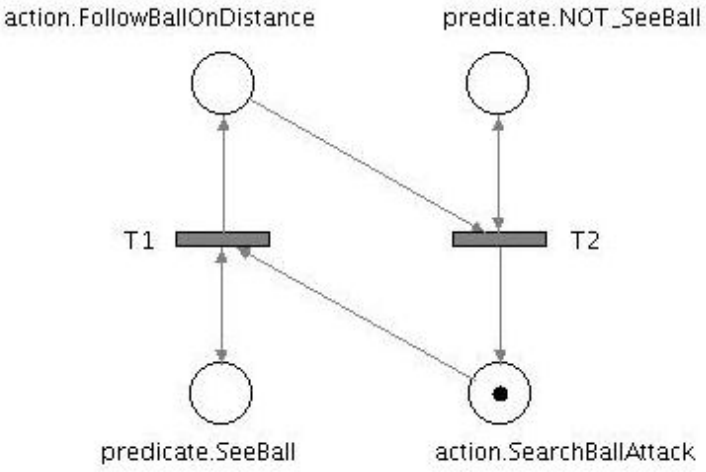
A.3.1 BehaviorBaseAttack - Rede de Petri



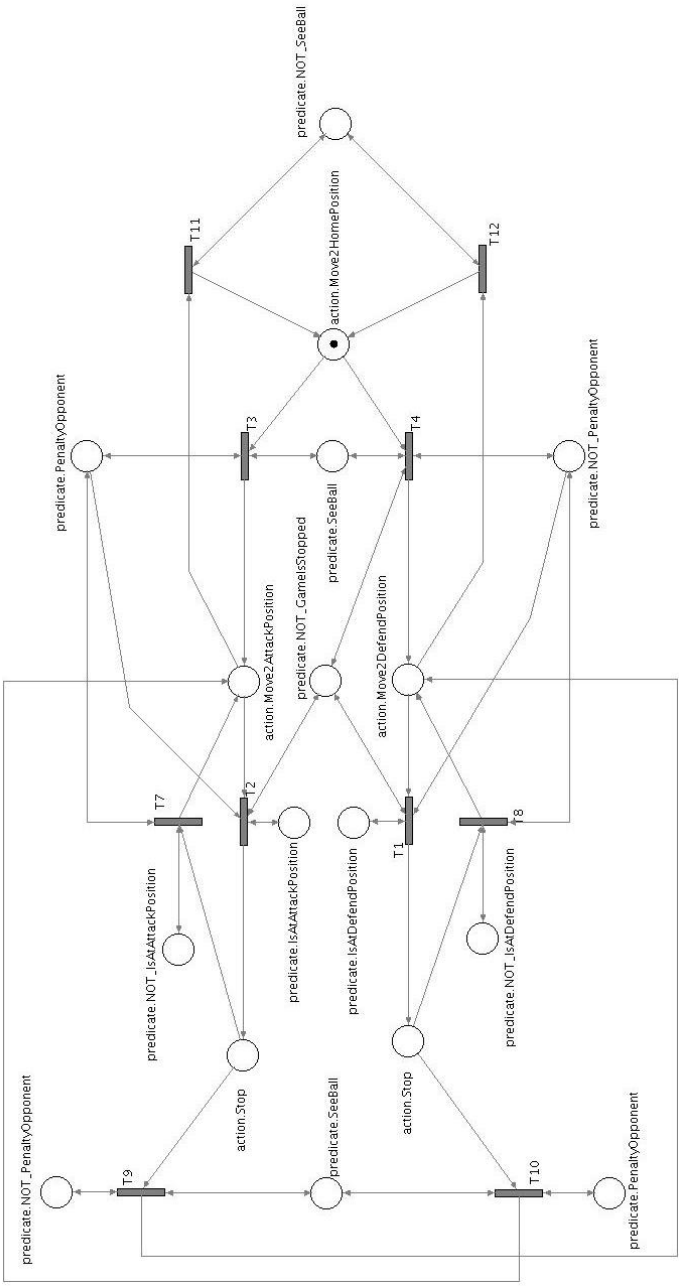
A.3.2 BehaviorBaseDefend - Rede de Petri



A.3.3 BehaviorBaseSupport - Rede de Petri



A.3.4 BehaviorFoulSupporter - Rede de Petri



A.3.5 BehaviorFoulTakerAlone - Rede de Petri