Departamento de Engenharia Informática

Relatório Intercalar de
TRABALHO FINAL DE CURSO

do Curso de
LICENCIATURA EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES (LEIC)

Ano Lectivo 2004 / 2005

N.º da Proposta: 176a

Título: Comportamentos Relacionais para Robots Futebolistas

Professor Orientador:

Pedro Lima

_____(assinatura)_____

Co-Orientador:

Luís Custódio

_____(assinatura)_____

Professor Acompanhante:

Nome professor acompanhante

_____(assinatura)_____

Alunos:

48260, Gonçalo Pais de Mouro Vaz

_____(assinatura)_____

49681, João Rafael Risso Milhinhos

_____(assinatura)_____

Trabalho orientado por:

Pedro Manuel Urbano de Almeida Lima
Professor Orientador
Secção de Sistemas e Controlo
Departamento de Engenharia Electrotécnica e de Computadores
Instituto Superior Técnico

Luís Manuel Marques Custódio
Professor Co-Orientador
Secção de Sistemas e Controlo
Departamento de Engenharia Electrotécnica e de Computadores
Instituto Superior Técnico

AGRADECIMENTOS

GONÇALO:

Este é, sem dúvida alguma, o momento que mais alegria me dá….
São incontáveis todas as palavras, os sorrisos, os silêncios, as pessoas que me ajudaram ao longo deste meu peculiar percurso académico.
João foi, é e será um amigo que, na sua directa imensa maneira de ser, me fez descer à Terra sem nunca me deixar tocar no chão. Um obrigado do tamanho do mundo.
Nos meus olhos, vejo os dois grandes irmãos e os meus pais; é neles que eu vou buscar toda a minha boa disposição e paciência para levar todos os meus devaneios a um fim, por mais tresloucado que ele seja.
Um desaustinado corrido de palavras ao meu grupo de amigos desta vida, relembrando todo um calmo abanar que me dava forças para, no fim, poder ver o vosso sorriso. Obrigado a todos os Xaranguianos.
Um forte abraço a todos aqueles que me sorriram de volta.

"Para o momento em verdade viver,
é preciso esquecer, o nosso remoto passado e futuro que idealizamos.
O passado é necessário, mas não agora.
Para o agora só temos de levar o que somos no momento,
E sem pensarmos o que fomos e o que seremos,
Vivemos o já.
No segundo que agora passa, o passado está lá e o futuro nele se escreverá.
O presente é o tempo mais real.
Nele não pensamos nós. Aliás, pensamos no presente mas não sobre ele.
Parece que não se pode pensar o presente…só agir nele.
Quando pensamos no presente, imediatamente passou, imediatamente se trata de passado ou futuro, imediatamente não é acção,
não sendo presente então.
O presente é o tempo mais real.
Embora muitas coisas já tenham acontecido e muitas outras ainda estejam para ocorrer,
É no presente onde tu podes realmente ser e fazer acontecer."

JOÃO:

À minha mãe pela paciência e carinho inesgotáveis e pelos fantásticos cozinhados que eu, religiosamente ou não, aqueci no micro-ondas. À minha Super mana! Adoro-vos! Às longas e sempre morais histórias dos meus avós e toda a minha família: foram a minha bengala de apoio ao longo destes anos! Ao enorme Gonçalo, sem o qual nem a minha metade do trabalho seria realizada, por me tirar do trabalho quando estava prestes a arrancar os cabelos e pela grande lição de vida. A sempre presente Xaranga, obrigado, por toda a "pachorra" que tiveram para aturar o nosso mau humor nos bons dias e o péssimo humor nos maus. Aos finalistas 2005, pelos bons raios de Sol nas noites de Segunda e Quarta! Até já, encontramo-nos no mercado de trabalho!

Um agradecimento especial ao Eng. Mouro Vaz e à Eng. Helena Batista por toda a flexibildade e compreensão demonstradas nestes últimos 8 meses.

Aos AVIST: Vasco d'Orey, Duarte Sampaio, Rui Dias, Nuno Cordeiro, Ricardo Cristino, Carlos "Lulas frescas" Santos; aos FAB, aos meus afilhados e netinhos, ao Pedro "Bin" Santos e ao grande Tiago "Borralheira" Antunes: a minha vida académica acabou! Vivam a vossa!

À malta de Montemor-o-Novo, por não deixarem ficar em casa a descansar quem queria ficar em casa a descansar aos fins de semana.

Por último, à minha flor de estufa: foi um ano bem difícil, o grande futuro risonho está já aí!

AMBOS:

Em primeiro lugar um agradecimento especial ao professor Pedro Lima por todo o empenho demonstrado, inclusive a animar-nos na fase final deste trabalho, ao professor Luís Custódio e a toda a equipa SocRob.

RESUMO

O objectivo deste Trabalho Final de Curso passa pela criação e demonstração do funcionamento de comportamentos relacionais desenvolvidos no âmbito do projecto SocRob.
Fazendo uso e procedendo à reestruturação da metodologia implementada por Van der Vecht e Lima tendo como bases de fundamentação o trabalho apresentado por Cohen e Levesque[11], novos métodos e técnicas deram origem a um modo simples e objectivo de criação de comportamentos relacionais. Esses mesmos comportamentos foram desenvolvidos para duas situações distintas de jogo, estáticas e dinâmicas. São apresentados os devidos resultados experimentais ilustrando o funcionamento do trabalho desenvolvido.

PALAVRAS CHAVE

Comportamentos Relacionais, Comunicação, Sincronização, Futebol Robótico.

ABSTRACT

The goal of this Final Year Project aims is to design, develop and demonstrate relational behaviours developed in the scope of the SocRob project. Using and restructuring the framework implemented by Van der Vecht and Lima having the work presented by Cohen and Levesque [11] as our theory foundations, new methods and techniques lead to a simple and objective way of implementing relational behaviours. Those relational behaviours were developed for two clear and distinct situations, static and dynamic situations. Experimental results are presented illustrating the described concepts.

KEYWORDS

Relational Behaviours, Communication, Synchronization, Robotic Soccer.

ÍNDICE

1 Introdução 6

1.1 Motivação 6

1.2 Especificação do problema 8

1.3 Projecto SocRob 9

1.4 Enquadramento Científico 10

2 Fundamentos 12

2.1 Teoria Dos Compromissos Conjuntos 14

2.1.1 ABORDAGEM 15

2.1.2 COMPROMISSO INDIVIDUAL VS COMPROMISSO RELACIONAL 15

2.1.3 SINCRONIZAÇÃO 17

2.2 Redes de Petri 18

3 Modelação de Comportamentos Relacionais em Futebol Robótico 21

3.1 Comportamentos Relacionais 21

3.1.1 CONCEITO DE FASE 21

3.1.2 CONCEITO DE ESTADO 22

3.1.3 ALGORITMO DE IMPLEMENTAÇÃO DE NOVO COMPORTAMENTO 24

3.2 Modelos das Redes de Petri 25

3.2.1 PORQUÊ REDES DE PETRI 25

3.2.2 MODELAÇÃO DE UM COMPORTAMENTO RELACIONAL 25

4 Métodos e Algoritmos 33

4.1 Projecto SocRob 33

4.2 – Módulos Fundamentais 34

4.2.1 – Kernel 34

4.2.2 – Módulo HalSimWebots 35

4.3 – BlackBoard 35

4.4 - Implementação Metodologia 35

4.4.1 - Estrutura Interna 35

4.4.2 - Variáveis de configuração e sincronização 40

4.4.3 – Algoritmo de Implementação de um novo comportamento 41

5 Apresentação dos Comportamentos Relacionais 43

Comportamentos Relacionais *43*

5.1 Decisões Relacionais Estáticas *43*

5.1.1 Throwin 43

5.1.2 Goal - Kick 44

5.1.3 Kick – Off 46

5.2 Decisões Relacionais Dinâmicas *46*

5.2.1 OneTwoPass 47

6 Resultados e Discussão 48

6.1 Análise das Redes de Petri *48*

6.1.1 REDE CONSERVATIVA 48

6.2 Análise da máquina lógica *51*

6.2.1 PLAYER.LOG 52

6.2.2 BASICUNIT.LOG 52

6.2.3 CONTROL.LOG & FICHEIROS .LOG CORRESPONDENTES ÀS ACÇÕES PRIMITIVAS 53

7 Conclusões 55

8 Bibliografia 58

9 Anexos 61

9.1 A : Simulador Webots® *61*

9.1.1 O Mundo Webots 61

9.1.2 Princípios de Funcionamento 62

9.1.3 Implementação 63

9.2 B : TABELAS *66*

9.2.1 Capítulo 5, Secção 1, Ponto 1, Tabela Nº1 - Throwin 66

9.2.2 Capítulo 5, Secção 1, Ponto 1, Tabela Nº2 - Throwin 67

9.2.3 Capítulo 5, Secção 1, Ponto 2, Tabela Nº1 – Goal Kick 69

9.2.4 Capítulo 5, Secção 1, Ponto 2, Tabela Nº2 – Goal Kick 70

9.2.5 Capítulo 5, Secção 1, Ponto 3, Tabela Nº1 - KickOff 72

9.2.6 Capítulo 5, Secção 1, Ponto 3, Tabela Nº2 - KickOff 73

9.2.7 Capítulo 5, Secção 2, Ponto 1, Tabela Nº1 - OneTwoPass 75

9.2.8 Capítulo 5, Secção 2, Ponto 1, Tabela Nº2 - OneTwoPass 76

9.2.9 Capítulo 6, Secção 1, Ponto 1 – Análise das Redes de Petri 79

9.2.10 Capítulo 6, Secção 1, Ponto 2– Gestores de Compromisso 79

9.2.11 Capítulo 6, Secção 1, Ponto 3 – Kick Off 80

9.2.12 Capítulo 6, Secção 1, Ponto 4– Goal Kick 82

9.2.13 Capítulo 6, Secção 1, Ponto 5– Throw In 84
9.2.14 Capítulo 6, Secção 1, Ponto 6– One Two Pass 86
9.2.15 Capítulo 6, Secção 2, Ponto 1, Tabela Nº1 - PLAYER.LOG 88
9.2.16 Capítulo 6, Secção 2, Ponto 2, Tabela Nº1 - BASICUNIT.LOG 90
9.2.17 Capítulo 6, Secção 2, Ponto 3, Tabela Nº1 - CONTROL.LOG 93
9.2.18 Capítulo 6, Secção 2, Ponto 3, Tabela Nº2 - FINDPASSTRAJECTORY.LOG 97

9.3 C : FIGURAS 98

9.3.1 COMPORTAMENTO RELACIONAL THROWIN – CASO TOTAL 98
9.3.2 COMPORTAMENTO RELACIONAL THROWIN – CASO SUCESSO 99
9.3.3 COMPORTAMENTO RELACIONAL THROWIN – CASO INSUCESSO 100
9.3.4 COMPORTAMENTO RELACIONAL GOAL KICK – CASO TOTAL 101
9.3.5 COMPORTAMENTO RELACIONAL GOAL KICK – CASO SUCESSO 102
9.3.6 COMPORTAMENTO RELACIONAL GOAL KICK – CASO INSUCESSO 103
9.3.7 COMPORTAMENTO RELACIONAL KICK OFF – CASO TOTAL 104
9.3.8 COMPORTAMENTO RELACIONAL KICK OFF – CASO SUCESSO 105
9.3.9 COMPORTAMENTO RELACIONAL KICK OFF – CASO INSUCESSO 106
9.3.10 COMPORTAMENTO RELACIONAL ONETWOPASS – CASO TOTAL 107
9.3.11 COMPORTAMENTO RELACIONAL ONETWOPASS – CASO SUCESSO 108
9.3.12 COMPORTAMENTO RELACIONAL ONETWOPASS – CASO INSUCESSO 109

"O Webots é o man !!!"

1 Introdução

1.1 MOTIVAÇÃO

Todos os seres vivos apresentam uma série de características determinantes para a sua sobrevivência até aos dias de hoje. Uma dessas características, plenamente visível nas espécies de maior sucesso, reside na cooperação entre indivíduos para atingir um objectivo comum. Esta cooperação pode ser observada desde os seus casos mais simples, execução de pequenas tarefas individuais cujo resultado final pode ser observado como cooperação, até à sua forma mais complexa apenas ao alcance dos seres humanos, caracterizada pela sua flexibilidade, complexidade e planeamento prévio

A robótica, como ciência desenvolvida para ajudar o ser humano na execução de tarefas, centrou o seu desenvolvimento em torno da dinâmica dos indivíduos. Passar da imitação da mecânica humana para a realização de actividades mais complexas é um passo natural. A cooperação entre indivíduos, como actividade fulcral no comportamento humano, é um dos grandes objectivos da robótica e baseia-se naturalmente no exemplo humano.

No campo robótico, a implementação de comportamentos relacionais trará grandes vantagens em termos de facilidade e tempo de execução das tarefas a executar. Rapidamente se constata que ao serem utilizados mais do que um robot para efectuar determinada tarefa e no caso de o entendimento entre estes ser aceitável, a tarefa em causa será executada com rapidez e com maior precisão. Actualmente, existem ainda numerosas acções que não podem ser realizadas individualmente, logo não são efectuadas.

A robótica móvel não é isenta dos problemas intrínsecos a qualquer área de investigação recente. Problemas como a sincronização e comunicação entre os robots, tolerância a falhas dentro de comportamentos relacionais e optimização dos recursos existentes no grupo de robots revestem-se de grande complexidade. Tal é perfeitamente visível quando considerada a aplicação em áreas tão diferentes como a vigilância de fogos e a indústria metalúrgica.

O desenvolvimento de comportamentos relacionais num meio como o futebol robótico vem criar desequilíbrios em termos de jogadas corridas, levando a que, no caso corrente, a equipa ISocRob possa tirar partido desses benefícios e adiantar-se no marcador.

Este projecto situa-se no seguimento do trabalho desenvolvido por Van der Vecht e Lima no âmbito do projecto SocRob. Nesse trabalho foi desenvolvido uma metodologia que permite a formalização de comportamentos cooperativos entre robots.

1.2 ESPECIFICAÇÃO DO PROBLEMA

Um comportamento relacional define-se como um conjunto de acções primitivas que irão ser executadas de uma maneira coordenada por diversos agentes enquanto um conjunto de condições se verificarem, agentes esses que se conhecem de antemão ou pertencem a uma equipa comum. A coordenação entre os diversos agentes exige por parte destes um grande esforço de entendimento e comunicação. O comportamento do tipo individual apresenta-se hoje em dia com uma sólida base de conhecimento, com inúmeros exemplos da sua aplicação e com uma vasta bibliografia de apoio. Embora um comportamento relacional seja constituído por comportamentos individuais, o desenvolvimento de comportamentos do tipo relacional apresenta-se ainda numa fase relativamente embrionária no que respeita ao projecto SocRob.

A criação e desenvolvimento de comportamentos relacionais tem como intervenientes principais, dois ou mais robots da mesma equipa. Como tal, os intervenientes em causa terão antecipadamente de concordar em executar determinado comportamento. Cada comportamento exigirá a satisfação de um conjunto de condições que necessitarão de ser cumpridas e respeitadas para dar início, manter e finalizar o comportamento como um comportamento relacional.

Nesse conjunto de condições, englobam-se as intenções de cada robot em efectuar determinado comportamento enquanto este respeitar um outro conjunto de condições em que cada qual acredita como sendo válidas. Cada um dos mesmos é “livre” de abandonar o comportamento, caso as condições em que acredita já não sejam verificadas.

Neste caso específico, o problema consistia em introduzir uma maior consistência ao trabalho desenvolvido até então por colegas anteriores e dar suporte a alterações que pudessem surgir por parte da entidade reguladora, a RoboCup Federation.

O facto de o grau de desenvolvimento deste tipo de problemas ser reduzido permite uma maior margem de progressão e liberdade para explorar várias ideias que rapidamente tendem a surgir. Tendo como base e forte referência o trabalho descrito em [1] e sendo esta a referência bibliográfica que apresentava algumas ideias interessantes em termos práticos, desenharam-se novos comportamentos que pudessem de alguma forma reaproveitar a abordagem introduzida pelo anterior colega. Esta introduz preferencialmente o comportamento de passe entre dois robots futebolistas embora não se encontre apenas limitada a este tipo de comportamento.

Pense-se agora numa qualquer jogada de futebol humano. Para que esta resulte em golo, é quase impossível não haver um passe entre dois jogadores. É extremamente improvável que um jogador consiga marcar um golo sem qualquer colaboração de um colega seu. Como tal, um dos comportamentos individuais com elevado grau de importância será o comportamento de passe. Porém, a implementação de um passe representa só por si um comportamento com um nível de dificuldade elevado necessitando de um grande esforço de comunicação e sincronização entre dois robots.

Com a introdução de novas regras na liga MiddleSize do RoboCup disputada pelos robots futebolistas, surgiram situações de jogo que merecem agora ser repensadas. Situações como a reposição da bola após esta ter ultrapassado as linhas delimitadoras do campo e o começo de um jogo são, por si só, situações que exigem aos robots um determinado comportamento. Estas situações de jogo e outras imaginadas e desenhadas pelos autores do trabalho resultaram em diversos comportamentos relacionais.

O resultado final deste trabalho irá ser integrado no projecto SocRob, podendo este ser visualizado, analisado e avaliado nas diversas competições a que os robots futebolistas irão estar sujeitos.

Facilmente surgem algumas questões sobre as quais este trabalho tenta responder. Mas a dúvida central será saber se é possível ou não desenvolver uma metodologia que permita desenvolver comportamentos relacionais de um modo standard e automático. Algumas questões, também elas relevantes, serão:

- Que condições gerais deverão ser tidas em conta no desenvolvimento das comunicações dos comportamentos relacionais?

Que comportamentos poderão ser criados de modo a adaptar a equipa ISocRob às novas regras impostas pela RoboCup Federation?

- Ganhará a equipa ISocRob vantagem competitiva em utilizar comportamentos relacionais nestas novas situações? E de que modo?

1.3 PROJECTO SOCROB

O projecto SocRob (Sociedade de Robots ou Soccer Robots) é um projecto desenvolvido pelo Laboratório de Sistemas Inteligentes do Instituto de Sistemas e Robótica, no pólo do Instituto Superior Técnico. Este projecto visa dinamizar a investigação em robótica cooperativa, o ensino da Engenharia a estudantes do ensino superior e a divulgação da Ciência e Tecnologia através do futebol robótico. A complexidade e o

interesse científico e tecnológico deste domínio dão-lhe especial importância no que diz respeito a problemas de maior impacto social onde a robótica tem, e poderá vir a ter, um papel de grande relevo.

O projecto é também responsável pela participação do Instituto Superior Técnico no RoboCup, nomeadamente na Middle Size League[3].

O RoboCup, organizado pela Federação Internacional RoboCup, é um evento internacional que visa promover a Inteligência Artificial, Robótica e todas as diversas áreas associadas, através da formulação de um problema comum, o futebol robótico, que engloba uma grande diversidade de tecnologias. Esta pesquisa é efectuada a vários níveis nas variadas competições, ligas de simulação, robots autónomos e robots controlados por supervisor, liga de salvamento e liga júnior, onde são investigadas várias tecnologias: agentes autónomos, colaboração multi-agente, captação de dados e processamento tempo real, entre outras. O grande objectivo deste evento consiste no desenvolvimento de uma equipa de robots humanóides totalmente autónomos que consiga vencer a equipa humana campeã do mundo.

Na Middle Size League, aquela em que a equipa ISocRob participa e onde se insere este projecto, os robots são totalmente autónomos, jogam num mundo dinâmico e não-determinístico, obtendo informação do mundo a partir da visão, sensores e comunicação com os companheiros de equipa.

Neste âmbito, são extremamente valorizadas as temáticas de investigação relacionadas com o processamento de imagem, tomada de decisões, planeamento e cooperação multi-agente.

1.4 ENQUADRAMENTO CIENTÍFICO

O projecto SocRob abrange uma grande quantidade de temas de investigação na resolução dos sub problemas que constituem o problema geral: mecânica, inteligência artificial, processamento de imagem, tomada de decisões, planeamento, comunicação e cooperação entre robots. Constitui pois uma plataforma muito interessante para implementação e teste do tema objectivo deste projecto: os comportamentos relacionais.

Da mesma forma que a cooperação assume especial importância na sobrevivência da grande maioria das espécies animais e no comportamento diário dos seres humanos, ela irá também ter um papel fundamental no desenvolvimento da robótica, essencialmente no que toca à sua aplicação na sociedade humana. É neste ponto que a aplicação das investigações no futebol robótico têm especial importância: o abandono da simulação em ambiente simulado para a adopção de um mundo dinâmico em constante mutação vem aproximar a robótica da realidade.

A cooperação entre robots é também de importância extrema. Uma sociedade de robots irá conseguir realizar um conjunto de tarefas certamente mais amplo que apenas um robot isolado. Situações variadas como operações de salvamento em equipa em cenários de risco, transporte de cargas pesadas ou colocação dispersa de explosivos serão caracterizados por uma menor taxa de insucesso devido a avarias, maior cobertura de área, gama alargada de funções e principalmente pela menor exposição so ser humano a situações de risco. A aplicação dos comportamentos relacionais a operações de salvamento é algo que tem vindo a ser estudado, mesmo dentro do Instituto de Sistemas e Robótica através do projecto Rescue[21]. Planemanento do transporte de materiais pesados na exploração de Marte[22] e disposição de explosivos[23] são outros desenvolvimentos nesta área e nas quais é acentuado o uso da Robótica como substituto do Ser Humano em situações que acarretam risco. É perfeitamente visível a existência de um esforço comum para o desenvolvimento dos robots cooperativos, tal como a diversidade de abordagens utilizadas.

A competição RoboCup[20] tem-se tornado cada vez mais exigente, quer pela constante evolução das regras, requerendo maior complexidade por parte dos participantes, quer pela evolução das equipas participantes. Dentro do ambiente RoboCup, este é claramente um caminho a seguir: se, se pretende jogar contra os melhores jogadores humanos do mundo, ter-se-á que jogar como humano.

2 Fundamentos

A base dos comportamentos relacionais tem como componente vital a comunicação entre os participantes. Através de um conjunto de acções sincronizadas entre os vários participantes, estes conseguirão atingir no final um objectivo comum, beneficiando assim todo o conjunto envolvente.

O sincronismo entre as acções realizadas pode ser obtido através da comunicação estabelecida entre os vários robots.

A comunicação poderá resultar do envio de mensagens explícito entre os participantes, ou de acções tomadas que resultam da observação do comportamento dos outros participantes. Este segundo tipo de comunicação é denominado como comunicação implícita, uma vez que não existe qualquer tipo de envio de mensagens, derivando esta única e exclusivamente de assunções formuladas através de um conjunto de observações inicial.

Existem diversas abordagens que procuram dar resposta e criar uma base de entendimento que leve a uma standardização dos modelos e ideias desenvolvidas na criação de comportamentos relacionais entre vários participantes.

DEFINIÇÕES A MANTER

O estudo e desenvolvimento de um comportamento pode ser efectuado a diferentes níveis sociais. Veja-se a definição de sociedade:

Uma sociedade é um grupo de indivíduos que formam um sistema semi-aberto, no qual a maior parte das interacções é feita com outros indivíduos pertencentes ao mesmo grupo. Uma sociedade é uma rede de relacionamentos entre pessoas. Uma sociedade é uma comunidade interdependente. O significado geral de sociedade refere-se simplesmente a um grupo de pessoas vivendo juntas numa comunidade organizada.

A necessidade de distinguir os comportamentos criados e as relações existentes entre cada membro de uma sociedade levou a que Collinot et al [7] tenha desenvolvido uma arquitectura que defina e esclareça as decisões que englobem uma equipa de agentes. Mais tarde, esse trabalho foi estendido por Drogoul e Collinot[8] que o aplicou a uma equipa de futebol robótico.

Como tal, existem alguns conceitos cujo grau de incerteza requerem um esclarecimento e definição dos mesmos. O conceito de comportamento poderá ser dividido em três conceitos: organizacional, relacional e individual.

O nível organizacional envolve todas as relações e decisões a serem tomadas num núcleo de indivíduos, no nosso caso uma equipa de futebol robótico. Decisões que afectam a equipa como um todo e não um número particular de jogadores. Exemplos disso serão os casos da selecção dos papéis de cada um dos jogadores robots dentro da equipa (atacante ou defesa).

Dentro do nível relacional, poder-se-ão encontrar diferentes tipos de decisões que são tomadas por mais do que um jogador robot, de modo a desenvolver operações e entendimentos que levarão à criação e execução de compromissos entre esses mesmo jogadores.

O nível mais baixo da hierarquia define os comportamentos individuais. Cada um destes comportamentos é composto por um conjunto de acções primitivas que cada robot irá executar. Estas acções primitivas correspondem a comandos que serão enviados directamente ao hardware de modo a provocar uma resposta física, por parte dos robots, no mundo.

No trabalho a ser desenvolvido, o nível em foco será o nível relacional, onde cada comportamento do tipo relacional será única e exclusivamente constituído por acções primitivas. Esta abordagem, embora limitativa, permite simplificar e tornar explícito todo um conceito que até hoje permanecia envolto numa nuvem de indefinições.

A incerteza se um comportamento relacional deveria ser constituído por um conjunto de comportamentos individuais ou por um conjunto de acções primitivas não nos permitia definir claramente em que camada, ou camadas do projecto SocRob, este tipo de comportamentos se deveria incluir. A explicação do modelo adoptado surgirá posteriormente (Capítulo 3).

APRENDIZAGEM E DESENVOLVIMENTOS

As aproximações existentes encontram-se divididas em duas vertentes - aproximações baseadas em comportamentos ou baseadas em lógicas.

A abordagem baseada em comportamentos interage directamente com o trabalho em causa, tendo vindo a ser desenvolvida com particular atenção por dois investigadores. Matsubara et al [9] apresentou desenvolvimentos na área das redes neuronais, tentando demonstrar a capacidade autónoma de um robot optar entre efectuar um passe ou um remate em direcção a baliza. Pagello et al [5] explorou uma nova abordagem, a extensão de comportamentos individuais a comportamentos relacionais. Esta nova abordagem abre portas a novos desenvolvimentos no campo dos comportamentos. É possível a cada robot observar as acções dos seus colegas de equipa e perceber os seus motivos, iniciando variadas

interacções que desenvolvam novas jogadas. Este tipo de abordagem sustenta as suas bases no tipo de comunicação que é realizado implicitamente. Yokota et al.[6] procedeu a desenvolvimentos também na mesma área.

Por sua vez, a abordagem lógica tem apresentado vários desenvolvimentos ao nível de criação de novas metodologias, permitindo assim estender os comportamentos relacionais a equipas que necessitem de realizar acções cooperativas. Uma das abordagens que vale a pena realçar consiste no trabalho apresentado por Tambe em [10] e consistia numa metodologia com capacidade para tratar várias equipas com diversos tamanhos especializada em recuperação de falhas. Esta metodologia foi implementada com base na teoria de Compromissos Conjuntos apresentada por Cohen e Levesque [11].

As diversas abordagens são usualmente testadas em meios militares e com particular incidência, no futebol robótico.

A utilização da metodologia desenvolvida por Van der Vecht e Lima [1] teve como propósito desenvolvê-la, adaptando-a e expandindo-a a novos comportamentos e a novas acções que esta pudesse incluir.

2.1 TEORIA DOS COMPROMISSOS CONJUNTOS

Um trabalho a salientar foi desenvolvido por Cohen e Levesque [11] que apresentaram uma teoria tendo em vista dar suporte a uma área que permanecia sem desenvolvimentos concretos. As respostas apresentadas vêm fundamentar inúmeros conceitos utilizados e dar apoio às variadas ideias que surgiram ao longo deste trabalho.

A ideia de que um conjunto de acções realizadas em modo sincronizado entre vários indivíduos consiste em trabalho de equipa é completamente refutada por estes dois investigadores. Como exemplo, é sugerido o caso do trânsito existente numa cidade. Embora os condutores estejam todos a conduzir ao mesmo tempo, não produzem qualquer resultado em termos globais. Apenas o estariam a fazer se todos eles partilhassem de um objectivo comum, algo que os movesse a todos na mesma direcção de modo a atingirem esse mesmo objectivo. Neste caso, trata-se de coordenação ao invés de cooperação.

2.1.1 ABORDAGEM

A noção de compromisso é descrita como um objectivo que persiste ao longo do tempo, algo que, enquanto não for satisfeito, se mantém como um propósito a atingir. Veja-se o seguinte caso: imagine-se que existe um determinado conjunto de indivíduos que se encontra motivado a resolver determinada actividade. Porém, surge uma situação a dois desses indivíduos que os impossibilita momentaneamente de realizar a actividade anterior. Embora estes dois indivíduos se tenham ausentado da tarefa anterior, o objectivo inicial não será esquecido. Após terminarem a situação que inesperadamente retardou o objectivo inicial, comunicam com os outros membros e verificam se o objectivo inicial ainda se mantém. Nesse caso, voltam a reunir-se esforços no sentido de terminarem o objectivo inicial.

As divergências que tendem a aparecer em redor dos objectivos da equipa levam a que a comunicação estabelecida entre eles seja fundamental. No entanto, como deverá essa comunicação ser efectuada? Ou em que altura deverá ser realizada?

2.1.2 COMPROMISSO INDIVIDUAL VS COMPROMISSO RELACIONAL

Alguns conceitos a manter prendem-se com as definições que irão ser apresentadas posteriormente.

Eventos – ocorrem instantaneamente e separam os estados discretos existentes (Exemplo: separam os estados da máquina de estados)

Crença - proposições que são válidas para todos os comportamentos em que os variados agentes intervêm.

Objectivos - são constituídos por um conjunto de crenças que em reunião convergem para o mesmo fim.

Crença mútua - consiste na conjunção da crença de cada uma dos agentes.

Estes conceitos serão relevantes uma vez que irão clarificar e apresentar as noções de compromisso e intenção relacional e individual.

Num conjunto de agentes que se encontram num determinado local, é provável que existam, entre estes, alguns que pretendem executar a mesma acção, ou seja, têm um objectivo persistente que é comum a todos eles. Como tal, reunindo os esforços dos vários agentes, as acções para atingir esse objectivo tornar-se-ão facilitadas. Após ser estabelecida a comunicação entre cada um dos agentes, todos eles irão estar a trabalhar no mesmo sentido.

Formalizemos então a definição de objectivo persistente:

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 2ªcondição 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.

Após cada agente adoptar um objectivo persistente, não poderá descartar-se deste duma maneira livre e despreocupada; o agente terá de manter esse objectivo persistente como válido enquanto não surgirem novas condições que o permitam desistir do objectivo.

Pense-se agora no caso de se tratar de uma equipa. Após se ter estabelecido um objectivo persistente dentro da equipa, como se poderia tratar o caso em que um dos agentes acredita que esse objectivo é impossível ou inválido? Deve a equipa de agentes desistir imediatamente do objectivo apenas porque o agente X acredita ser impossível?

Seria imprudente toda a equipa desistir do objectivo. Neste caso, após ser informada pelo agente X que acredita que o objectivo é impossível, todos os agentes deverão chegar a um consenso levando à desistência ou não do objectivo. O aviso aos restantes membros da equipa por parte do agente torna-se, por si só, um segundo objectivo; porém, não deixa de ser um objectivo, mas um objectivo de baixa prioridade

Como tal, definir-se-á o conceito de objectivo de baixa prioridade a atingir.

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;

Veja-se, agora, o caso da equipa - esta acredita conjuntamente que o objectivo é verdadeiro; nesse caso, existe um objectivo persistente, mas agora relacional. Definindo o termo formalmente, obtém-se:

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;

Após o objectivo persistente relacional ter sido atingido com sucesso, todos os membros irão terminar o comportamento com sucesso. No caso de o sucesso do comportamento não se verificar, o objectivo de baixa prioridade é realizado e os restantes membros da equipa terminarão o comportamento com insucesso.

2.1.3 SINCRONIZAÇÃO

A existência de um objectivo persistente relacional não significa que o resultado do comportamento realizado pela equipa se traduza em sucesso. A comunicação estabelecida entre os vários membros da equipa torna-se num factor crucial.

Tal como no mundo real, se não existir um entendimento razoável na execução de uma operação que dependa de vários intervenientes, essa execução será certamente um fracasso. Esse entendimento traduz-se, em termos práticos, na comunicação estabelecida entre os vários agentes.

O modo como a comunicação é efectuada poderá ser realizado implícita ou explicitamente, tal como foi descrito anteriormente.

No caso de ser realizada implicitamente, parte do agente interessado verificar e perceber, através da visualização das acções do agente a comunicar, o que este pretende fazer. O sincronismo entre as acções a serem realizadas é mantido através das análises efectuadas posteriormente, bem como através dos valores captados pelos diversos sensores existentes. Note-se que, neste caso, o agente interessado nunca saberá se o seu parceiro ainda se mantém ligado ao comportamento, ou se já desistiu de perseguir esse objectivo. Não existe assim, nenhuma garantia que o comportamento seja efectuado com sucesso.

O sincronismo poderá também ser obtido através de uma comunicação efectuada explicitamente. Enveredando por este caminho, a comunicação é efectuada através do envio de mensagens que permitem aos agentes manter o mesmo ritmo ao longo da execução do objectivo e certificarem-se que esses

mesmos agentes ainda permanecem ligados à execução do objectivo. É assim possível a cada agente saber sempre o estado actual do comportamento, bem como o estado do seu parceiro.

Ao longo do trabalho, o tipo de comunicação utilizada foi do tipo explícito. As vantagens e desvantagens serão enunciadas posteriormente no ponto 1.2 do capítulo 3.

2.2 REDES DE PETRI

Sistemas de eventos discretos

Um sistema de eventos discretos[18] é um sistema baseado em estados dirigido por objectos. Estes sistemas dependem da ocorrência assíncrona de eventos ao longo do tempo, ou seja, de eventos que não dependem de qualquer sincronização temporal ou evolução temporal. Estes eventos controlam a transição entre estados. Dependendo do tipo de evento, um sistema de eventos discretos (DES) pode ser orientado pelo tempo ou por eventos. Num sistema orientado pelo tempo, todas as transições são sincronizadas pelo relógio do sistema, ao invés de um sistema orientado por eventos, cuja ocorrência de algo (um evento) provoca a transição entre estados.

Redes de Petri

As redes de Petri são uma possível representação matemática de sistemas de eventos discretos.

Uma rede de Petri é constituída por lugares, transições e arcos que ligam estados a transições e vice-versa. Um lugar está associado a uma acção, a uma tarefa ou ao processamento de algo. Em cada momento, cada lugar pode conter zero ou mais tokens. À marcação da rede, isto é, à distribuição de tokens pela rede, corresponde o estado actual do sistema.

Uma transição está associada a um evento, a algum acontecimento ou conjunto de condições que se verificam. Quando uma transição é disparada (executada), consome um ou mais tokens dos seus lugares de input e produz um ou mais tokens nos seus lugares de saída. Durante a execução de uma rede de Petri múltiplas transições podem ser activadas simultaneamente, tornando-a uma rede não determinística.

Definição Matemática

As redes de Petri são uma possível representação matemática de sistemas de eventos discretos.

Uma Rede de Petri (marcada) é um quíntuplo (P, T, A, w, x):

- P representa o conjunto finito de lugares;
- T representa o conjunto de finito de transicções;
- A representa o conjunto de arcos de lugares para transicções e arcos de transicções para lugares;
- w é a função de pesos associados a cada arco, em que cada w_i é um inteiro positivo;
- x representa a marcação do conjunto de lugares definida por um vector X = [x(p_1), x(p_2),....x(p_n)];

O modelo gráfico de uma rede de Petri completa é definido pelo seu grafo (conjunto de lugares, transições e arcos) , por um estado inicial e por uma marcação inicial. Dado que uma rede de Petri é um grafo bipartido, cada arco liga apenas nós diferentes da rede de Petri: lugares a transições ou transições a lugares.

Cada lugar pode ter 0 ou mais tokens. O número de tokens em cada lugar identifica o estado actual da rede, o que permite um bom seguimento da execução do sistema modelado.

Uma transição está associada a um evento, a algum acontecimento ou conjunto de condições que se verificam. Quando uma transição é disparada (executada), consome k_i tokens dos seus lugares de entrada e produz k_o tokens nos seus lugares de saída, sendo k_n o peso dos arcos ligados à transição.

Durante a execução de uma rede de Petri, podem haver momentos em que mais do que uma transição pode ser activada. Tal leva a que a execução de uma rede de Petri seja não-determinística.

Características do modelo

Os lugares de uma rede de Petri podem ser:

- ***Limitados***: um lugar de uma rede de Petri com estado inicial x_0 diz-se k-limitado se o número máximo de tokens em qualquer lugar alcançável da rede for k.
- ***Cobertos***: dada uma rede de Petri com estado inicial x_0 , um estado y diz-se coberto por x se $x(p_i) \geq y(p_i)$, para qualquer *i*.
- ***Rede de Petri Conservativa***: Uma rede de Petri diz-se estritamente conservativa se a soma dos tokens existentes em cada momento na rede for constante.
- ***Rede Viva***: Uma rede de Petri diz-se viva se qualquer transicção t_x possa ser disparada a partir de qualquer estado alcançável a partir do estado inicial x_0.

Existem ainda diversas extensões das redes de Petri, que aumentam o poder de modelação e reduzem o poder de decisão e restrições, as quais diminuem o poder de modelação mas aumentam o poder de decisão. Serão abordadas as extensões e restrições de importância relevante para o projecto mais adiante.

3 Modelação de Comportamentos Relacionais em Futebol Robótico

3.1 Comportamentos Relacionais

A modelação dos comportamentos relacionais apresenta alguns desafios, desafios esses que poderão ser solucionados com base em alguns dos modelos apresentados na Teoria dos Compromissos Conjuntos.

A execução de qualquer um dos comportamentos relacionais é previamente seleccionado por uma máquina lógica. Esta máquina lógica mantém-se encarregue de mediante as condições que o mundo lhe presenteia, avaliá-las e escolher o comportamento que melhor se adequar. Em suma, é estabelecido um objectivo persistente relacional ou individual. O controlo dessa máquina lógica não faz parte deste trabalho, de modo que não se irá proceder à explicação dos seus detalhes.

3.1.1 Conceito de Fase

O primeiro desafio na definição de um comportamento relacional consiste na definição das várias fases existentes ao longo do comportamento. Tal como apresentado por Cohen e Levesque [11], um comportamento poderá ser caracterizado por uma fase inicial onde são estabelecidas as condições necessárias ao compromisso dos dois agentes intervenientes no comportamento; uma fase intermédia onde é implementado o respectivo comportamento; e uma fase final que corresponde à indicação do sucesso ou insucesso do comportamento.

Este método apresenta como vantagens a garantia aos intervenientes que cada um deles verifica as condições necessárias à execução do comportamento, bem como cada um deles está sincronizado com o seu parceiro de modo a executar o comportamento com sucesso.

Na fase de Setup não existirão quaisquer tipos de compromissos seleccionados, mas apenas se verificarão se os respectivos agentes verificam as condições necessárias de modo a poderem executar o comportamento em causa. A título de exemplo, no caso de o robot querer executar um passe, algumas das condições que terá de verificar na fase de `Setup` serão:

- O seu papel será o de defesa

- O agente terá de ter a bola

- O agente terá de estar na zona do campo pertencente à sua equipa.

A fase de `Loop` será caracterizada pela escolha do comportamento a efectuar, comportamento esse seleccionado anteriormente pela máquina lógica.

A fase de `End` será extremamente semelhante à fase de `Setup`, ou seja, não serão seleccionados quaisquer tipos de comportamentos na fase de `End`.

3.1.2 CONCEITO DE ESTADO

Pense-se agora no seguinte caso: Imagine-se que determinada pessoa está a realizar uma acção complexa conjuntamente com um colega seu. A sua execução ficará extremamente facilitada no caso de dividirem essa acção em várias etapas. Entre as várias etapas, cada um deles irá parar um pouco para descansar ou mesmo esperar pelo colega que ainda não terminou a sua acção.

A implementação de cada uma das fases do comportamento poderá ser vista um pouco como o caso acima descrito, de tal forma que cada uma dessas fases será dividida num conjunto de estados que permitirão ser possível a sua descrição, bem como a sua análise. Este conjunto de estados facilitará ainda a sincronização entre os vários agentes à medida que executam o comportamento.

A fase inicial de `Setup` terá de comportar os estados necessários ao teste das condições, a fase de `Loop` terá tantos estados como os definidos na implementação da Rede de Petri e a fase de `End` apresentará apenas os estados indicativos do sucesso ou insucesso do comportamento.

Na fase inicial de `Setup`, apresentam-se dois estados: `request` e `de accept`. Cada um destes estados corresponde à verificação das condições e ao estabelecimento da ligação entre os dois agentes.

A fase de `End` apresenta os dois estados possíveis no final de qualquer comportamento, `Done` e `Failed`.

3.1.2.1 COMUNICAÇÃO

Tal como já foi referido anteriormente, a existência de dois tipos de comunicação poderia suscitar algumas dúvidas relativamente à sua implementação. Porém, a utilização de comunicação implícita é ainda hoje em dia um campo que ainda apresenta dificuldades do ponto de vista de desenvolvimento.

A utilização de comunicação explícita vem trazer algumas vantagens que se revelam fulcrais na implementação dos comportamentos.

Foi criado um estado que surgirá entre cada um dos estados desenvolvidos na fase de `Loop`. Este estado, denominado `COM SYNC`, permitirá proceder à sincronização dos vários agentes ao longo do comportamento. Esta sincronização obtém-se através da introdução de uma condição que irá atrasar, ou não, o comportamento do agente em causa. O agente irá assim aguardar que a condição seja verificada, só depois poderá passar ao próximo estado. A passagem aos estados seguintes é, assim, controlada através do envio de mensagens entre os vários intervenientes.

A sincronização obtida ao longo deste trabalho apresenta alguns desenvolvimentos. Ao contrário do método utilizado por [1], a sincronização obtida não será estática. Quer-se com isto dizer que o estado de comunicação criado poderá ser usado, ou não, como um meio de sincronização. Em suma, não é obrigatório que em cada estado `COM SYNC` exista alguma condição que obrigue os vários intervenientes a terem de esperar pelo seu parceiro(s). Consoante o comportamento em causa, é possível deixar a cada agente uma certa margem de manobra, não limitando as suas acções ao comportamento do seu parceiro. É atingido um maior grau de flexibilidade, possibilitando a cada agente ir executando determinada acção mesmo se o seu parceiro não aparenta estar sincronizado.

3.1.3 ALGORITMO DE IMPLEMENTAÇÃO DE NOVO COMPORTAMENTO

Um dos objectivos deste trabalho foi tornar standard um processo cujos métodos de implementação levantavam ainda bastantes questões.

Após redefinição de alguns conceitos e tendo como linha de desenvolvimento a metodologia implementada por [1] e com bases teóricas desenvolvidas por Cohen e Levesque [11], obteve-se o seguinte algoritmo.

- Desenho da Rede de Petri.

- Identificação dos vários estados existentes na rede.

- Definição das condições da fase de `Setup`.

- Definição das condições de mudança entre os vários estados.

- Implementação da sincronização pretendida entre os vários agentes.

- Definição das condições dos estados `COM SYNC`.

- Definição das condições necessárias para cada um dos comportamentos individuais (acções primitivas) incluídos no comportamento relacional

A criação de novos comportamentos transforma-se, assim, num processo facilitado, podendo por vezes ligeiras alterações em condições de fases, mudança de estados ou de comunicação levar à criação de novos e proveitosos comportamentos.

O ponto 4 do Capítulo 4 apresenta detalhadamente os vários passos a seguir no desenvolvimento de cada um dos pontos apresentados no algoritmo acima descrito.

3.2 MODELOS DAS REDES DE PETRI

3.2.1 PORQUÊ REDES DE PETRI

As máquinas de estado são a ferramenta utilizada para a modelação dos comportamentos individuais e relacionais no projecto SocRob. Uma abordagem mais aprofundada à modelação de comportamentos relacionais veio revelar que estas tinham manifestamente um poder de modelação inferior ao necessário.

- Uma máquina de estados é uma Rede de Petri restringida pela obrigação de cada lugar ter obrigatoriamente um lugar de Entrada e um lugar de Saída. A abordagem adoptada veio revelar situações onde é fundamental a existência de mais que um lugar de saída e/ou de entrada para uma dada transição
- Uma modelação por redes de Petri permite a modelação das comunicações como um conjunto de lugares e transições da rede de Petri
- A sincronização entre os diferentes robots está bem explicitada, permitindo a modelação de pontos de sincronização
- Uma rede de Petri permite a modelação de mais que uma sequência de execução de um comportamento relacional
- A modelação por Redes de Petri é bem mais expressiva que a por máquinas de estados, permitindo identificar claramente todas as condições de transição em qualquer estado.

As abordagens já realizadas à utilização das redes de Petri na modelação de um robot inteligente [24] alertaram para a sua utilização, pois permitem efectuar uma modelação bastante aproximada da codificação dos comportamentos, permitindo obter à priori a sua estrutura, a informação e as funções necessárias.

3.2.2 MODELAÇÃO DE UM COMPORTAMENTO RELACIONAL

No sentido de atingir um dos objectivos deste trabalho - conseguir uma modelação coerente e sistemática dos comportamentos relacionais, é essencial definir uma relação entre cada elemento da rede de Petri e a correspondente implementação.

3.2.2.1 A REDE DE PETRI

O comportamento relacional executa-se sempre entre dois ou mais robots. A execução desse comportamento é modelada por uma rede de Petri para cada robot, cuja sincronização é assíncrona e efectua-se através da comunicação. A rede de Petri modela:

- A execução normal do comportamento relacional, onde as condições ideais para a realização do comportamento se verificam. O posicionamento dos robots, o posicionamento dos adversários, a posição e movimento da bola e a comunicação entre os robots são algumas das condições que podem influenciar a execução normal do comportamento;
- A execução alternativa do comportamento para a qual não se verificam as condições ideias mas existem as condições suficientes para o comportamento relacional se executar com sucesso;
- Situações de falha em que o comportamento relacional não tem condições para continuar. Tal leva a uma condição de erro num robot, naquele que necessitaria desse conjunto de condições para continuar. Neste caso, a execução do comportamento é abortada em todos os seus intervenientes;

3.2.2.2 ACTORES

Um comportamento relacional é caracterizado pela intervenção de pelo menos dois jogadores que intervêm entre si: os *actores*.

O modelo geral do comportamento encontra-se dividido verticalmente nos actores intervenientes no comportamento. Os modelos dos vários intervenientes estão colocados paralelamente, partilhando os diferentes estados do comportamento (divisão horizontal).

3.2.2.3 ACÇÕES PRIMITIVAS

Uma acção primitiva é uma acção, de complexidade variável, indivisível e cuja execução não é interrompida nem avaliada em nenhum lugar excepto no final da mesma. A cada acção primitiva corresponde, no modelo, a um lugar com o mesmo nome. Cada lugar tem um estado e um actor correspondente, tal como se de uma matriz se tratasse.

É no lugar que se realiza:

- a acção pelo robot (avaliar o mundo, passar a bola, deslocar-se para uma posição);
- actualização da `Blackboard`;
- avaliação do resultado da execução do lugar;

Um lugar pode constituir um lugar de entrada para mais que uma transição, onde mais que uma transição disputam o seu token. Tais são os casos onde podem ocorrer erros ou falhas na execução normal do comportamento, direccionando-o para um desenrolar alternativo ou ao fim do comportamento relacional.

3.2.2.4 TRANSIÇÕES

A transição é o nó responsável pela progressão da execução do comportamento modelado. Cada transição tem condições lógicas associadas e um conjunto de lugares de entrada que decidem qual a transição que pode disparar a cada momento. Os eventos estão associados às transições.

A transição é o elemento responsável pela avaliação das condições de transição de estado. Estas condições são visíveis pela presença de tokens nos lugares de entrada e pelas condições lógicas que se encontram associadas a cada transição. As transições não estão associadas a qualquer estado, apenas os seus estados de saída e de entrada o estão. Como tal, são convencionalmente representadas no estado *`COM SYNC`*.

No modelo apresentado, uma transição pode ter mais que um lugar de saída. Tal acontece quando a conclusão de um estado implica uma comunicação para o robot cooperante. Neste caso, existe um lugar de saída na execução do comportamento e outro lugar de saída na comunicação.

3.2.2.5 COMUNICAÇÃO COM SYNC

A comunicação encontra-se modelada entre os vários actores e partilham, tal como eles, os diferentes estados do comportamento. Os lugares que respeitam à comunicação estão colocados no estado *COM SYNC*, estado característico da comunicação localizado entre os estados pertencentes ao comportamento relacional. Os lugares da comunicação não pertencem a qualquer estado do comportamento relacional, mas são essenciais na transição destes.

3.2.2.6 GESTORES DE COMPROMISSO

Os gestores de compromisso são os lugares responsáveis pela informação de compromisso entre os vários actores. Os gestores de compromisso não pertencem a qualquer estado, encontrando-se normalmente representados por cima da modelação de cada actor. A sua representação é efectuada devido à sua representatividade na execução dos comportamentos. No mínimo, são sempre necessários 3 gestores de compromisso por actor:

- *End Commitment <own name>*: Este lugar ganha um token quando, durante a execução do comportamento, não existem condições para o robot continuar em compromisso.
- *Commitment <partner name>*: A presença de token neste lugar indica que o companheiro está em compromisso. Este lugar é complementar do *<partner name> not commited*.
- *<partner name> not commited*: sempre que este lugar contém um token, o robot ao qual ser refere já não se encontra em compromisso. A execução do comportamento relacional é interrompida logo após a execução da acção primitiva, representada pelo lugar actual, seja qual for o estado actual. Este lugar ganha um token quando o colega falha e sai do compromisso.

Os gestores de compromisso são os responsáveis pela gestão do estado de compromisso entre os robots. É desta forma que são geridas nos comportamentos relacionais as situações de falha onde há cancelamento do comportamento relacional:

- A acção primitiva falha, o robot sai do comportamento relacional e avisa o seu colega;

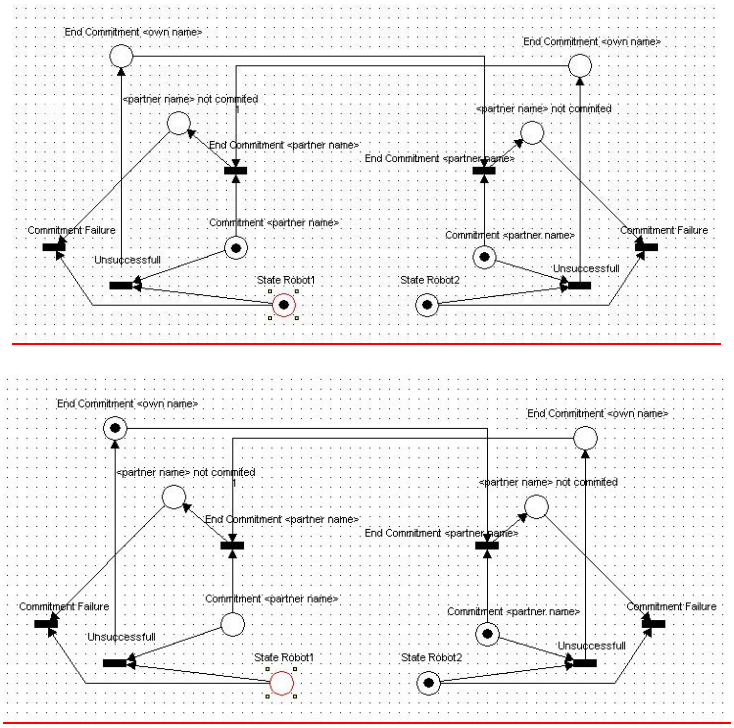

Figura 3.2.2.6.1 – Em cima: estado inicial dos gestores de compromisso no início do comportamento relacion

Em baixo: estado dos gestores de compromisso em caso de falha de execução da acção p

- O conhecimento de que o parceiro no comportamento relacional falhou e consequente saída do compromisso;

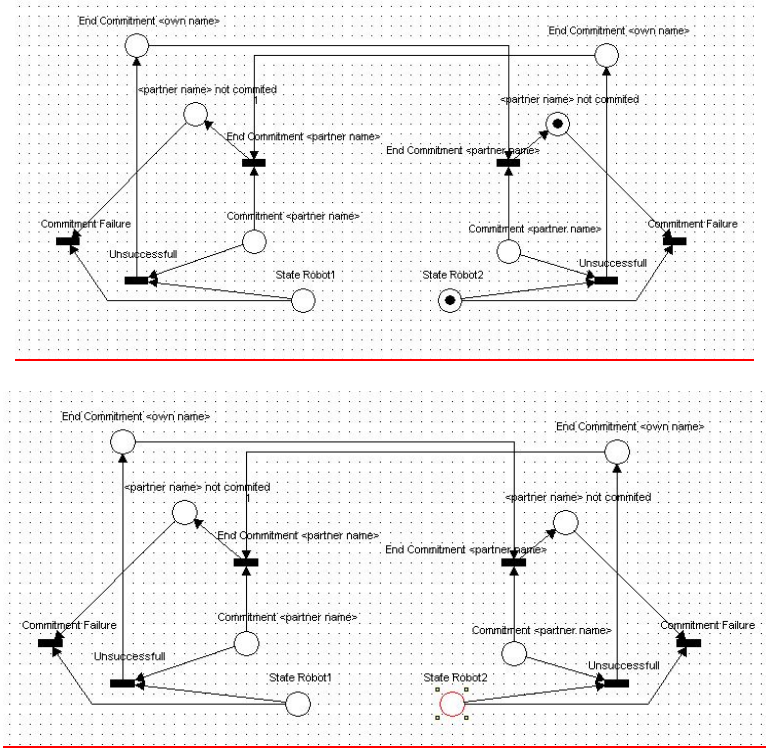

Figura 3.2.2.6.2 – Em cima: informação a um dos robots de que o parceiro quebrou o compromisso (evento: End commitment) . Em baixo: o robot sai do compromisso (commitment failure)

3.2.2.7 CARACTERÍSTICAS DA REDE DE PETRI

O modelo de um comportamento relacional tem de obedecer a um conjunto de regras de forma a poder ser considerado uma modelação válida. Existem dois tipos de regras que é imperativo qualquer modelação verificar:

- Propriedades da rede de Petri:
  - *Vivacidade*: o modelo deve ser *vivo*, não oferecendo quaisquer situações para a ocorrência de bloqueios (vulgo *deadlock*)
  - A rede deve ser *1-limitada*, uma vez que o número máximo de tokens por lugar é 1.

- Propriedades dos comportamentos relacionais

  - ***Estado final:***

    Em cada comportamento relacional existe um estado final, estado esse onde pertencem todos os lugares terminais do comportamento relacional, independentemente do resultado da sua execução.

  - ***Comunicação:***

    Usualmente os modelos dos comportamentos relacionais mais simples apresentam três ou mais estados de comunicação. Um menor número de lugares de comunicação indica que o comportamento modelado não é um comportamento relacional.

  - ***Marcação máxima:***

    O número máximo de tokens por lugar em qualquer instante da execução do comportamento é de 1. O token é apenas indicador do lugar que está actualmente em execução. Um número superior de tokens pode indicar uma situação de bloqueio da rede em que uma transição está a ser disparada consecutivamente.

    Cada lugar recebe um token apenas uma vez.

  - ***Marcação inicial nos commitments:***

    O modelo em rede de Petri de um comportamento relacional refere-se apenas à fase de *Loop*. Como tal, os actores já se encontram em compromisso no início da modelação. A existência de um *token* nos lugares *Commitment< actor >* indica isso mesmo.

- ***Terminação do comportamento por falha:***

Ao longo da execução de um comportamento relacional existem diversos motivos de falha, quer na execução de alguma acção, quer falha induzida por falha do companheiro de compromisso. As implicações deste facto reflectem-se na modelação das transições de erro e nas transições de falha:

- Transição de erro: transição existente para cada lugar, que é disparada quando um ou mais dos actores não se encontram em compromisso, levando à terminação da execução do comportamento no actor em questão.
- Transição de falha: estas transições são disparadas quando se verificam condições de insucesso na execução de uma acção primitiva.

4 Métodos e Algoritmos

4.1 PROJECTO SOCROB

A implementação dos novos comportamentos, tal como já foi referido anteriormente, será incluída no projecto desenvolvido pela equipa ISocRob constituída por diversos elementos do grupo de pesquisa SocRob do Instituto de Sistemas Robóticos, pólo de Lisboa.

Este projecto é constituído por inúmeros módulos de desenvolvimento; no entanto, apenas alguns se encontram relacionados com o tema abordado neste trabalho. Os módulos com maior importância são os módulos de `kernel` e `HalSimWebots`.

O módulo de kernel identifica todos os componentes intimamente relacionados com a navegação, visão, decisão, comunicação do robot, etc. No caso do trabalho em questão, as componentes fundamentais são constituídas pela componente de `control` e pela componente `LogicMachine`.

O módulo de `HalSimWebots` é responsável por efectuar toda a interface entre o código do projecto, o simulador `Webots` utilizado para o efeito e cada um dos robots que se encontra em funcionamento.

A arquitectura de software desenvolvida até ao momento tem como núcleo principal a utilização de diversos `micro-agentes`, onde cada um destes é executado através de um processo (`thread`) dedicados, de modo a ser possível funcionarem simultaneamente. Cada micro-agente é responsável por controlar uma determinada componente do robot [17] (visão, controlo, decisão, etc). Um exemplo será a escolha do comportamento a executar por parte de cada robot, onde existirá um micro-agente dedicado única e exclusivamente à selecção do melhor comportamento para determinada situação.

4.2 – MÓDULOS FUNDAMENTAIS

4.2.1 – Kernel

4.2.1.1 - COMPONENTE CONTROL

Esta componente é responsável pela criação do micro-agente que se irá encontrar dedicado ao processamento da informação desta componente. Este micro-agente, após inicialização, terá, como funções, analisar a situação actual da máquina lógica, verificar que pedidos são efectuados pela mesma, executar esses mesmo pedidos, ou mantê-los em execução. O desenvolvimento de cada um dos comportamentos será assegurado pelo micro-agente correspondente ao módulo da `LogicMachine`.

4.2.1.2 – COMPONENTE LOGICMACHINE

A coordenação, decisão e manutenção do progresso de cada um dos comportamentos será efectuado por um micro-agente dedicado, tal qual a situação apresentada no ponto anterior [17]. O trabalho desenvolvido tem como bases a metodologia apresentada por [1], incidindo na criação de novos comportamentos ao nível da camada relacional, bem como da camada individual.

A decisão e avaliação de cada uma das condições necessárias tem como factor crucial a quantidade de tempo dispendida na escolha de cada um dos comportamentos existentes. Como tal, a utilização de afirmações lógicas vem proporcionar tempos de resposta rápidos, sendo que ao nível da camada de comportamentos individuais, estes apenas serão escolhidos no caso de se verificarem todas as condições necessárias.

Tal como já foi referido anteriormente, cada comportamento relacional é composto por um conjunto de acções primitivas, sendo que se optou por tratar cada uma destas acções primitivas como possíveis comportamentos individuais. Assim sendo, será possível no futuro, se assim for desejado, adaptar essas acções primitivas a comportamentos individuais. Aumenta-se a flexibilidade de adaptação a situações futuras do código apresentado.

As decisões ao nível das camadas são efectuadas sequencialmente, de tal forma que é escolhido que papel irá o robot em causa desempenhar, seguindo-se a escolha do comportamento relacional que poderá ser inicializado e, só então a escolha do respectivo comportamento individual, ou acções primitivas a desempenhar.

Estas decisões são requisitadas através do micro-agente, que se encarrega de requisitar à máquina lógica por uma decisão que lhe permita executar algum tipo de comportamento.

4.2.2 – Módulo HalSimWebots

Este módulo foi criado de modo a tornar transparente e a identificar claramente todas as chamadas ao novo simulador que foi utilizado. O núcleo deste módulo é constituído pela interface entre o código desenvolvido e o simulador, bem como pelas soluções desenvolvidas para se proceder à comunicação entre os diversos robots.

4.3 – BLACKBOARD

Uma das soluções encontradas para transmitir os mais variados tipos de informação, consistiu na criação de um quadro de informação distribuída, onde se encontram definidos todos os tipos de informação que irão ser alterados ao longo de todo o processo.

A informação contida no `Blackboard` poderá respeitar apenas a um robot, podendo, assim, cada um deles guardar o estado de determinadas variáveis que em determinado momento de execução, irão ter um papel preponderante na sua acção (variáveis locais), ou respeitar a algum tipo de informação que um robot queira deixar acessível a algum colega de equipa (variáveis globais). Torna-se, assim, possível a comunicação entre vários robots, embora esta não seja directa, é de certo modo funcional, não ficando cada robot bloqueado à espera de alguma mensagem por parte do seu colega de equipa. A informação global contida no `Blackboard` poderá ainda respeitar a inúmeras configurações que cada robot deverá ter em conta ao iniciar todo o seu processo, nomeadamente, dimensões do campo, posições iniciais, etc.

A validade deste tipo de informação é extremamente relativa; como tal, é utilizado um valor temporal que permite verificar se o valor apresentado pela variável ainda é considerado válido, uma vez que o ambiente em que decorre todo o processo é um ambiente extremamente dinâmico.

4.4 - IMPLEMENTAÇÃO METODOLOGIA

4.4.1 - Estrutura Interna

A implementação de um novo compromisso relacional leva a que sejam efectuados alguns ajustamentos à estrutura definida por [1]. Tal como já foi mencionado anteriormente, um dos módulos preponderantes nesta estrutura é o módulo da `LogicMachine`. É neste que será avaliado constantemente a situação do mundo, de modo a verificar se será necessário proceder a alguma alteração no comportamento do robot. As decisões tomadas ao nível da camada lógica dependem da validade dos predicados lógicos e das condições estabelecidas anteriormente.

A estrutura que irá mover todo o processo relacional é composta pela interligação de quatro ficheiros fulcrais.

Foram ainda criadas diversas funções de modo a permitir tratar toda a comunicação necessária no desempenho dos vários comportamentos relacionais. Essas funções permitem alterar o estado de algumas variáveis que se encontram no `Blackboard`. De modo a ser possível interagir com a máquina lógica `Prolog`, as funções foram desenvolvidas na linguagem C, passando, assim, via apontadores, os valores das variáveis que eram relevantes no momento.

4.4.1.1 – BASICUNIT.PRO

Todo o processo de escolha de um comportamento relacional tem início através do requisitar de uma nova decisão do micro-agente responsável pela máquina lógica. Esta chamada é encaminhada para este ficheiro que avalia essa possibilidade e torna, assim, possível todo esse processo. Veja-se a figura 4.4.1.1.1, que apresenta parte do código responsável por efectuar essa verificação

```
makeBasicDecision( Decision, Name, Arg, X, Y, Theta) :-
  broadcastStatus,
  role( Role ),
  plAdaptRole( Role, NewRoleTemp),
  roleSet( NewRoleTemp ),
  dynamicRoleChange,
  role( NewRole ),
  chooseCommitment( Commitment ),
  basicBehaviour( Behaviour, Arg, X, Y, Theta, Commitment, NewRole).
```

Figura 4.4.1.1.1 – Acções responsáveis por verificar a possibilidade de efectuar uma nova decisão lógica.

Este ficheiro é ainda responsável por, após terem sido reunidas as condições necessárias para se estabelecer um determinado comportamento relacional, apresentar os diversos comportamentos individuais que irão constituir esse mesmo comportamento relacional. Ou seja, só após ter sido escolhido o comportamento relacional a ser executado, e todas as suas condições terem sido verificadas com sucesso nos ficheiros seguintes, só aí são executados os comportamentos individuais que incorporam o comportamento relacional seleccionado. Tal como foi referido anteriormente, esses comportamentos individuais poderão ser apenas constituídos por uma única acção primitiva.

A figura 4.4.1.1.2 apresenta alguns dos novos comportamentos individuais criados ao longo deste trabalho.

```
Findpasstrajectory,
Rodar,
Passing,
Finishsuccessful,
Findkicktrajectory,
Kick
```

Figura 4.4.1.1.2– Exemplos de novos comportamentos individuais

Na figura 4.4.1.1.3, é possível verificar que, para cada um desses comportamentos individuais ser executado, existe um conjunto de condições que necessitam de ser verificadas. Caso não se verifiquem, o comportamento relacional não será executado com sucesso.

```
basicBehaviour( findpasstrajectory, 0, 0.0, 0.0, 0.0, throwin, _ ) :-
      currentMode(play),
      plCheckCommitmentRole( throwin, kicker ),
      plCheckCommitmentState( throwin, local, prepare ),
      plGetID( throwin, kicker, MyID ),
      plCheckID( throwin, kicker, MyID ),
      \+ stateFinished( findpasstrajectory ).
```

Figura 4.4.1.1.3 – Código responsável por verificar a possibilidade de escolha de um comportamento individual, exemplo FindPassTrajectory

4.4.1.2 – COMMITMENTUNIT.PRO

Após ter sido invocado todo o processo de escolha de um comportamento relacional (`chooseCommitment`), a camada relacional da `LogicMachine` avalia quais dos comportamentos relacionais existentes melhor se adequam à situação actual.

A situação actual corresponderá a três situações do mundo, que poderão ser constatadas através da tabela 4.4.1.2.1

1ªSituação – Não é possível escolher qualquer tipo de comportamento

```
chooseCommitment( none ) :-
  \+ plCommitmentAllowed( any ).
```

2ªSituação – O agente já se encontra envolvido num comportamento.

```
chooseCommitment( DefCommitment ) :-
\+ plCheckCommitment( none ) – Não? Prossegue-se a execução
```

Sim? Que acções tomar?

```
getCommitmentInfo( Commitment, MyRole, MyState, PartnerRole,
PartnerState )
plGetCommitment( Commitment )
```

1. **– Deve o agente terminar o comportamento?**
   - Em caso afirmativo, termina-se o comportamento

   ```
   \+ endCommitment( Commitment, MyRole, MyState, PartnerRole )
   ```

2. **– O seu parceiro já se encontra num estado posterior ao seu?**
   - Em caso afirmativo, sincronizam-se os estados.

   ```
   confirmStateUpdate( Commitment, MyRole, MyState,
   PartnerRole, PartnerState )
   ```

3. **– O seu parceiro já se encontra num estado anterior ao seu?**
   - Em caso afirmativo, sincronizam-se os estados.

   ```
   repeatStateChange( Commitment, MyRole, MyState,
   PartnerRole, PartnerState )
   ```

4. **– Estão reunidas as condições para passar ao próximo estado?**
   - Em caso afirmativo, muda-se de estado e avisa-se o parceiro

   ```
   upgradeCommitmentState( Commitment, MyRole, MyState,
   NewState )
   ```

5. **– Por fim, verifica-se a possibilidade de selecção do comportamento.**

   ```
   selectCommitment( Commitment, NewState, DefCommitment )
   ```

3ªSituação – O agente não se encontra envolvido em qualquer comportamento.

```
chooseCommitment( none ):-
```

É possível estabelecer-se um comportamento? (request / accept)

- Em caso afirmativo, estabelece-se o comportamento.

```
setupConditions( _ )
```

Figura 4.4.1.2.1 – Acções iniciais tomadas após ter sido requisitada a selecção de um comportamento

Todas as condições que são avaliadas em cada um das situações partem da análise de variáveis que são actualizadas no `Blackboard`.

Este ficheiro contém todas as funções de análise, verificação e execução das condições necessárias à execução de qualquer uma das situações acima mencionadas.

4.4.1.3 – THROWINUNIT.PRO

No caso de se verificar a terceira situação (`setupConditions( _ ))`, será seleccionado um comportamento relacional dos já existentes. Esta selecção poderá ser efectuada neste ficheiro através da verificação do pedido de `request` por parte do agente e da acção de `accept` ser realizada por parte do seu colega de equipa.

Este ficheiro contém ainda todas as condições respeitantes à transição entre os estados desenhados para o comportamento relacional que seja pretendido. É possível, com o auxílio da rede de Petri, identificar as várias alternativas possíveis, bem como as condições necessárias às transições entre cada um dos estados existentes nessas alternativas.

```
switchConditions( throwin, kicker, prepare, finish ) :-
        getState( findpasstrajectory ),
        stateError( findpasstrajectory ).
```

Figura 4.4.1.3.1 – Condições necessárias para efectuar a transição do estado prepare para o estado de finish no caso de erro, comportamento throwin.

4.4.1.4 – PROLOG.PRO

Este ficheiro contém todas as declarações das funções de C já existentes e que foram criadas para interagir com a máquina lógica. O ficheiro contém ainda todos os predicados que permitem tratar os comportamentos que se encontram em execução, concluindo se estes foram executados com sucesso, com insucesso ou com erro.

4.4.2 - Variáveis de configuração e sincronização

No que diz respeito às variáveis utilizadas para efectuar a comunicação e estabelecer algum grau de sincronização, foram criadas e utilizadas algumas das já existentes.

Todas as situações acima mencionadas são situações onde as condições que são analisadas partem da verificação de variáveis que se encontram actualizadas no `Blackboard`. Na figura 4.4.2.1, é possível observar algumas das variáveis utilizadas na selecção de um comportamento relacional.

```
local.commitment.any.allowed? BOOL  TRUE
local.commitment.current      string  none
local.commitment.role   string none
local.commitment.state  string none
```

Figura 4.4.2.1 – Variáveis Locais utilizadas na selecção de um comportamento relacional

Constata-se que o agente verifica se é possível estabelecer-se algum comportamento relacional, e só então verifica se já existe algum comportamento estabelecido.

A comunicação e consequente sincronização é efectuada através de um conjunto de variáveis que poderão ser acedidas pelos dois agentes, figura 4.4.2.2.

```
*.commitment.throwin.kicker   string  none
*.commitment.throwin.kicker.state   string  none
*.commitment.throwin.receiver string  none
*.commitment.throwin.receiver.state string      none

*.commitment.throwin.com.kicker.state     string  none
*.commitment.throwin.com.receiver.state   string  none
```

Figura 4.4.2.2 – Variáveis Globais utilizadas na comunicação e sincronização entre os dois agentes no comportamento relacional `Throwin`

Sendo o factor comunicação um ponto preocupante, uma vez que a estabilidade e fiabilidade desta componente não trazem uma certeza de funcionamento sem quaisquer tipos de erros ou falhas, as variáveis de comunicação apenas serão enviadas ao parceiro no caso de modificação dos valores destas. Ao invés do que foi apresentado por [1], a existência de comunicação na fase de `Loop` torna-se fulcral, de modo a introduzir um factor de sincronização que não dependa apenas dos estados. Ou seja, é possível agora aos dois agentes uma maior flexibilidade nas acções, não tendo cada um dos agentes de se encontrar no mesmo estado do comportamento à medida que vão avançando no mesmo.

4.4.3 – Algoritmo de Implementação de um novo comportamento

Na implementação de um novo comportamento, alguns dos passos definidos no algoritmo apresentado na secção 3.1.3 poderão apresentar alguma correspondência directa com as estruturas que foram acima documentadas. Nos passos 2, 3, 4 e 6 , poder-se-á introduzir alguma metodologia de apoio à criação de um comportamento relacional.

1 - Desenho da Rede de Petri

2 - Identificação dos vários estados existentes na rede.

Após ter sido desenhada a rede de Petri com todos os estados que a compõem, o utilizador poderá criar um ficheiro "`NomeComp.pro`" onde irá definir os estados que existem na rede, bem como as condições necessárias para se efectuar a transição entre cada um destes. Como exemplo, teria de definir os seguintes pontos:

2.1 – Definição dos papéis de cada um dos participantes

```
commitmentRoles( Commitment, MyRole, PartnerRole ).
```

2.2 – Definição do código a atribuir a cada um dos estados existentes no comportamento

```
commitmentState( Commitment, StateNr, State ).
```

2.3 – Definição dos estados e do comportamento em que cada um se encontra

```
selectCommitment( Commitment, State, DefCommitment ).
```

3 - Definição das condições da fase de Setup.

Neste ponto definem-se as condições que terão de ser verificadas durante a fase de `Setup`, de modo a que o comportamento relacional seja verificado. Estas condições serão também elas incluídas no ficheiro "`NomeComp.pro`".

3.1 – Definição das condições necessárias para entrar no comportamento relacional (request e accept)

```
setupConditions( Commitment ).
```

4 - Definição das condições de mudança entre os vários estados.

Neste ponto, definem-se as condições necessárias para que seja possível transitar entre cada um dos estados definidos anteriormente. Definem-se, assim, as várias alternativas (com sucesso, insucesso ou erro) que poderão existir num comportamento relacional. Estas condições serão também elas incluídas no ficheiro "`NomeComp.pro`".

4.1 – Definição das condições necessárias para transitar entre cada um dos estados existentes na rede de Petri.

```
switchConditions( Commitment, CommitmentRole, State, NewState ).
```

5 - Implementação da sincronização pretendida entre os vários agentes.

6 - Definição das condições dos estados COM SYNC.

Esta fase é caracterizada pela criação das variáveis que se irão alocar no Blackboard de modo a que seja possível aos dois agentes comunicarem entre si e introduzirem uma maior flexibilidade de sincronização no comportamento. Terão de ser criadas as funções que irão receber e analisar essas mesmas variáveis. As funções serão criadas no ficheiro "`CommitmentFunctions.c`" e as respectivas declarações no ficheiro "`Prolog.pro`".

5.1 – Exemplo da declaração de uma função criada para manipulação de variáveis do Blackboard

```
:- foreign(plSendCommSync(+string,+string,+string)).
```

7 – Definição das condições necessárias para cada um dos comportamentos individuais (acções primitivas) incluídos no comportamento relacional

Este ponto caracteriza-se pela definição das condições que terão de ser estabelecidas para que cada um dos comportamentos individuais definidos no comportamento relacional se verifique.
Estas condições encontram-se definidas no ficheiro "`BasicUnit.pro`".

7.1 – Exemplo da definição de um dos comportamentos individuais incluídos

```
basicBehaviour( Individual Behaviour, Arg, X, Y, Theta, Relational Commitment,
NewRole).
```

5 Apresentação dos Comportamentos Relacionais

COMPORTAMENTOS RELACIONAIS

Na realização deste trabalho, foram contempladas duas situações distintas.

Tem-se, como primeira situação, o caso em que o comportamento relacional é estabelecido numa situação de jogo estática, ou seja, numa situação em que a bola se encontra parada numa das linhas delimitadoras ou de marcação do campo, à espera que um dos robots se dirija para ela e tome alguma decisão.

Na segunda situação, o comportamento é estabelecido pelo robot que tem a bola em seu poder, decidindo o rumo da jogada através da selecção de um comportamento relacional adequado às condições do ambiente que o rodeia.

5.1 DECISÕES RELACIONAIS ESTÁTICAS

Existe um conjunto de decisões relacionais estáticas cuja implementação surge rapidamente como obrigatória. Pense-se no exemplo do futebol humano, em que existe um conjunto de situações em que a bola ultrapassa uma das linhas delimitadoras do campo e é necessário proceder à sua reposição em jogo. Neste caso, um dos jogadores dirige-se para a bola e efectua uma determinada acção na bola consoante as condições que se coloquem naquele instante.

Esse conjunto de jogadas consiste no pontapé de início de partida, num pontapé de baliza, num lançamento da bola na linha lateral ou mesmo num pontapé de canto. Essas jogadas são praticamente idênticas no que respeita ao futebol robótico. O conjunto que deu origem a comportamentos relacionais consiste no pontapé de início de partida, no pontapé de baliza e no lançamento da bola na linha lateral.

5.1.1 Throwin

A utilização deste comportamento relacional necessita da aprovação de um conjunto de condições obrigatórias; no entanto, vejamos em primeiro lugar o desenrolar da situação.

Esta situação ocorre quando a bola ultrapassa qualquer uma das linhas laterais delimitadoras do campo de futebol. Nesse caso, a bola é colocada na posição em que ultrapassou a linha e é assignada a

tarefa de repôr a bola em jogo a um dos robots da equipa. O robot R1 que reúna as melhores condições no ambiente que caracteriza a situação corrente torna-se no escolhido para efectuar a marcação. Neste momento, é também escolhido o melhor colega de equipa, o robot R2, que se encarregará de receber a bola.

Sendo assim, o robot R1 dirigir-se-á para a bola e tentará encontrar uma linha de passe, linha essa que irá definir a posição para onde o colega de equipa, o robot R2, se deverá dirigir. No caso de ter sido possível encontrar uma linha de passe, R1 deverá comunicar a R2 a posição<x,y> , para onde este se deverá dirigir. R1 inicia uma rotação de modo a poder passar a bola na direcção da linha de passe encontrada, enquanto R2 se dirige para a posição<x,y>. Seguidamente, R1 efectuará o passe, encontrando-se já R2 à espera de receber a bola. R2 interceptará a bola e terminará o comportamento com sucesso, podendo depois efectuar o que pretender consoante a situação corrente. O robot R1 já terminou o comportamento, podendo sair da posição em que se encontrava e optar por efectuar qualquer outra acção.

Existem várias alternativas no caso de a situação não se desenrolar como o previsto. Nesse caso, o comportamento poderá ser terminado com insucesso ou ainda proporcionar-se a alternativa em que embora o robot R2 termine o comportamento com insucesso, o robot R1 consegue terminar com sucesso.

Esta situação ocorre no caso em que não é possível obter uma linha de passe; nesse caso, R1 opta por rematar a bola contra os jogadores adversários, indo a bola sair novamente pela linha lateral mas, agora, numa posição mais avançada do campo. Embora o comportamento relacional não tenha sido um sucesso completo, a bola continua na posse da nossa equipa e encontra-se agora numa posição mais avançada.

As condições necessárias à selecção e execução com sucesso do comportamento encontram-se nas tabelas 5.1.1.1 e 5.1.1.2.

5.1.2 Goal - Kick

O comportamento relacional `Goal Kick` é utilizado nas reposições de bola em jogo única e exclusivamente em situação de pontapé de baliza. Esta é uma situação de jogo em tudo idêntica ao lançamento lateral, excepto na sua localização próxima e frontal à baliza. Como tal, esta é uma situação de jogo que resulta muitas vezes num golo sofrido quando não marcado atempadamente ou quando a bola é aí perdida para o adversário.

O comportamento relacional desenrola-se entre dois participantes: o *`kicker`* e o *`receiver`*. O primeiro passo consiste na tentativa de estabelecimento de uma trajectória de passe pelo *`kicker`*, de forma a que a bola possa ser recebida pelo *`receiver`*. Uma vez que é ao *`kicker`* que cabe a

responsabilidade deste cálculo, é transmitida ao *receiver* a posição para onde o passe será efectuado. Com o conhecimento desta informação o *receiver* desloca-se para a posição definida. Esta posição definida será sempre próxima das linhas laterais, para evitar perdas de bola junto da própria baliza e numa posição frontal. A execução normal do comportamento conduz à execução do passe pelo *kicker* e recepção da bola por parte do *receiver*.

Num cenário alternativo, é contemplada a impossibilidade de cálculo de uma trajectória de passe, pelo posicionamento dos robots adversários, pelo próprio posicionamento do *receiver*, entre outras. Este é um factor controlado pelo tempo utilizado para o cálculo, que não pode ultrapassar o tempo definido pelas regras para cobrança do pontapé de baliza. Neste caso, adoptou-se o comportamento adoptado no futebol humano: em casos extremos é sempre preferível colocar a bola fora de campo, o mais longe possível, pela linha lateral, a deixá-la jogável à mercê dos adversários. O *kicker* executa, então, um remate para a linha lateral, de forma a que a bola cruze a linha o mais longe possível da baliza.

As similaridades com o comportamento `Throwin` conduzem naturalmente a uma modelação idêntica à de este comportamento:

- Similaridade entre os estados de ambos os comportamentos;
- A comunicação ocorre, em ambos, no mesmo ponto da execução, com a mesma informação e com o mesmo impacto na execução dos comportamentos;
- As situações de erro e de execução alternativa consideradas existem em ambos os comportamentos.

O comportamento `Goal Kick` apresenta particularidades fundamentais para a execução correcta desta situação de jogo:

- Garantia de execução;
- Risco de perda de bola reduzido;
- Comportamento sem finalidades atacantes.;

Estas particularidades são intrínsecas à execução das acções primitivas, razão pela qual não têm qualquer impacto na modelação do comportamento (ver tabelas 5.1.2.1 e 5.1.2.2).

5.1.3 Kick – Off

O comportamento relacional `Kick Off` é utilizado em todas as situações de bola parada nas quais a bola está dentro do terreno de jogo, nomeadamente em casos de início / reinício da partida e cobrança de faltas. Nestas situações, o comportamento desenrola-se da seguinte forma:

- O robot *`kicker`* encontra-se perto da bola à espera que o robot *`receiver`* o informe de que assumiu uma boa posição para receber a bola. O cálculo da boa posição para receber a bola baseia-se na avaliação de uma boa trajectória, quer para dribble em direcção à baliza adversária, quer para possível remate;
- O robot *`kicker`* passa a bola ao companheiro de equipa;
- O compromisso é dissolvido logo após o passe efectuado, de forma a possibilitar o início de qualquer outro comportamento relacional ou individual.

As condições necessárias à selecção e execução com sucesso do comportamento encontram-se nas tabelas 5.1.3.1 e 5.1.3.2.

5.2 DECISÕES RELACIONAIS DINÂMICAS

As decisões relacionais dinâmicas definem-se pelas situações que são criadas durante o jogo de futebol através da interacção dos vários robots com o ambiente. Em diversas alturas do jogo, existem momentos que são mais favoráveis à criação de determinadas jogadas, sejam elas um simples passe ou um conjunto de passes e movimentações interligadas.

No entanto, a sua implementação apresenta um grau de dificuldade elevado, uma vez que a situação não sendo estática, existem inúmeros factores que são precisos avaliar e ter em conta.

A situação implementada tem o nome de `OneTwoPass` e é apresentada de seguida.

5.2.1 OneTwoPass

Este tipo de comportamento difere dos restantes, uma vez que a sua selecção parte de uma decisão tomada pela máquina lógica enquanto o jogo decorre.

A situação criada pela utilização deste comportamento visa colocar o robot que inicialmente detém a posse da bola, sem qualquer oposição adversária. O desenrolar do comportamento assemelha-se a uma situação já existente no futebol humano. A vulgar e conhecida "tabelinha" permite à pessoa que tem a bola em seu poder, de se desenvencilhar da oposição adversária fazendo uso do auxílio dos seus colegas de equipa.

Veja-se a seguinte situação, o robot R1 tem a bola em seu poder mas não pode avançar no campo, uma vez que existem robots adversários a obstruir a sua progressão. Como tal, R1 após comunicar com um colega de equipa, R2, e caso este esteja livre de oposição adversária, efectua um passe na sua direcção indicando-lhe que vai querer receber novamente a bola numa posição mais avançada. Após efectuar o passe para R2, a oposição adversária que se situa no robot R1 irá dirigir-se para R2, libertando R1 e dando-lhe a possibilidade de se deslocar para uma situação mais avançada no campo. R2 após receber a bola, irá devolver a bola a R1 que se encontra já livre de oposição e com possibilidades de seguir com a bola em direcção à baliza adversária.

Dado o carácter dinâmico da situação, o seu sucesso é reduzido; no entanto, caso este se verifique, permite colocar a equipa numa situação extremamente vantajosa (ver tabelas 5.2.1.1 e 5.2.1.2).

6 Resultados e Discussão

6.1 Análise das Redes de Petri

Após concluído o desenho das Redes de Petri segundo o algoritmo apresentado no início deste trabalho é fundamental o seu estudo de forma a garantir que estas se encontram dentro dos parâmetros estabelecidos: se a Rede de Petri é viva, conservativa e se os seus lugares são 1-limitados.

6.1.1 Rede Conservativa

De forma a analisar se o desenho dos vários comportamentos resultam em redes de Petri conservativas é necessário verificar o número de tokens existentes para cada estado atingível a partir de uma determinada marcação inicial – estado inicial.

Para tal dividiu-se a análise em três fases distintas:

- Análise dos Gestores de Compromisso
- Análise do núcleo do comportamento relacional e das situações de erro
- Análise da Comunicação

Adoptou-se uma sistematização da apresentação da informação e dos termos utilizados de forma a facilitar, quer a análise das redes, quer a apresentação dos dados dessa mesma análise. A nomenclatura utilizada ao longo de toda a análise está descrita na Tabela 6.1.1.1 – Anexo B.

Análise dos Gestores de Compromisso

Após análise da tabela 6.1.2.1 – Anexo B verificou-se que a variação do número de tokens existente no Gestor de Compromisso é sempre negativa. Os testes demonstram que esta sub-Rede, que apresenta 2 tokens no início do comportamento contém sempre 0 tokens quando esta termina. Cada vez que uma transicção de erro é executada ($t\text{-}_{e1}$, $t\text{-}_{e2}$, $t\text{-}_{e3}$, $t\text{-}_{e4}$) , 1 token é consumido tal como é demonstrado na tabela 6.1.2.2 – Anexo B. Como a cada transicção de erro segue-se imediatamente outra, obtemos o estado final esperado com 0 tokens.

Comportamento Kick Off

A variação total do número de tokens da rede é resultado exclusivo da variação do número tokens no Gestor de Compromisso, como pode ser verificado nas tabelas 6.1.3.1 e 6.1.3.2 do Anexo B. Verificou-se que o número de tokens do comportamento relacional é constante e igual a 0. A rede de Petri que modela o núcleo do comportamento `KickOff` é portanto conservativa.

Comportamento Goal Kick

A variação total do número de tokens da rede é resultado exclusivo da variação do número tokens no Gestor de Compromisso. Verificou-se que o número de tokens do comportamento relacional é constante e igual a 0. A rede de Petri que modela o núcleo do comportamento `GoalKick` é portanto conservativa, tal como é demonstrado nas tabelas 6.1.4.1 e 6.1.4.2 do Anexo B

Comportamento Throw In

A variação total do número de `tokens` da rede é resultado exclusivo da variação do número tokens no Gestor de Compromisso. Verificou-se que o número de tokens do comportamento relacional é constante e igual a 0. Este é um comportamento bastante similar ao comportamento `Goal Kick,` e como tal, era de esperar uma análise semelhante. As tabelas 6.1.5.1 e 6.1.5.2 do Anexo B confirmam essa observação. A rede de Petri que modela o núcleo do comportamento `ThrowIn` é portanto conservativa.

Comportamento OneTwoPass

A variação total do número de `tokens` da rede é resultado exclusivo da variação do número tokens no Gestor de Compromisso. Verificou-se que o número de tokens do comportamento relacional é constante e igual a 0, observação realizada a partir da análise das tabelas 6.1.6.1 e 6.1.6.2 do Anexo B. A rede de Petri que modela o núcleo do comportamento `OneTwoPass` é portanto conservativa.

Análise da Comunicação

A comunicação, cuja principal função consiste na sincronização entre os robots ao longo do comportamento, apresenta-se como caso particular:

- A contabilização do token existente num lugar de comunicação faz variar o número total de tokens na Rede de Petri,
- O lugar de comunicação, ao contrário de quaisquer outros lugares do modelo do comportamento relacional, depende dos Gestores de Compromisso, quer do `kicker`, quer do `receiver`.
- A sua existência é essencial para a execução do comportamento, mas totalmente indiferente em qualquer situação de erro (disparo de uma transicção de erro)
- O número máximo de tokens existentes em lugares de comunicação é 1.

Devido às particularidades referidas, concluiu-se que estes são lugares dispensáveis para a análise em questão, além de que a sua exclusão permite, em muito, simplificar e tornar mais objectiva a análise realizada.

Tabela Resumo

Propriedades	Comportamentos Relacionais			
	Throwin	Kick-Off	Goal Kick	OneTwoPass
Limitada	1k - Limitado	1k - Limitado	1k - Limitado	1k - Limitado
Conservativa	Sim	Sim	Sim	Sim
Vivacidade	Viva	Viva	Viva	Viva
Intervalo de Tokens	[2 – 5]	[2 – 5]	[2 – 5]	[2 – 5]
Estados Finais	FinishSuccessful FinishUnsuccessful	FinishSuccessful FinishUnsuccessful	FinishSuccessful FinishUnsuccessful	FinishSuccessful FinishUnsuccessful

A informação contida na Tabela Resumos indica claramente que todos os comportamentos relacionais desenhados obedecem às regras impostas inicialmente., validando-os sob a abordagem proposta.

Esta validação abrange também o algoritmo utilizado para modelação do modelo em Rede de Petri. O seguimento dos passos propostos é condição fundamental, se bem que insuficiente dado a necessária liberdade de modelação, para a construção de um comportamento que é Relacional e que é válido. Esta liberdade de modelação é essencial devido à larga diversidade de comportamentos relacionais que se pode ambicionar modelar, relacionados ou não, com o futebol robótico, e que podem ter ou exigir uma modelação mais complexa. Refira-se ainda que todo o estudo foi fundamentado em Comportamentos Relacionais com dois actores, nos quais foi focalizada toda a atenção neste trabalho. Tal não impossibilita a adopção da abordagem utilizada a comportamentos relacionados com três ou mais actores, possibilidade oferecida pela escabilidade dos mecanismos de sincronização.

6.2 ANÁLISE DA MÁQUINA LÓGICA

O funcionamento da máquina lógica implementada poderá ser analisado fazendo uso da escrita de "`check strings`" em ficheiros de `LOG`, permitindo verificar a passagem da respectiva máquina pelos estados definidos para cada um dos comportamentos relacionais.

No caso do fluxo de execução da máquina lógica ser o correcto para cada comportamento, respeitará a ordem de transicção entre cada um dos estados indicados na rede de Petri.

Analisando o comportamento relacional `Throwin` para o fluxo de execução principal, com o auxílio da tabela 5.1.1.1, a ordem de transição entre os estados para o robot `kicker` e `receiver` deverá respeitar a ordem dessa tabela. Apresenta-se na tabela 6.2.1 o conjunto de "`check strings`" impressas nos ficheiros de `LOG` relevantes, demonstrando a passagem na ordem correcta da máquina lógica.

Cada ficheiro `.LOG` é obtido através de funções de escrita colocadas no ficheiro `.c` com o nome idêntico.

6.2.1 PLAYER.LOG

No caso do ficheiro player.c, este contém uma chamada a uma função que requisita à máquina lógica uma decisão a tomar consoante as condições em que cada robot se encontra. Após a chamada à função "`decision = plGetBasicDecision( &stateName, &argument, &PosX, &PosY, &PosTheta )`", o seu valor de retorno, `decision`, é analisado e impresso demonstrando qual a decisão tomada pela máquina lógica. No caso de se encontrar correcto, o resultado impresso deverá apresentar uma ordem semelhante à apresentada na tabela 5.1.1.1. O comportamento desta função e a execução da máquina lógica já foi explicado anteriormente, como tal não voltará a ser referido.

Analisando a tabela 6.2.1.1 situada no Anexo B, constata-se que no caso da decisão a que a máquina chegou ser diferente da anterior, esta passa a ser a decisão a tomar, alterando assim o estado guardado anterior. Os números correspondentes a cada decisão correspondem ao `encoding` efectuado para cada uma das acções primitivas seleccionadas pela máquina lógica. É possível verificar que a sequência, quer para o robot kicker, quer para o robot receiver, corresponde à sequência apresentada na tabela 5.1.1.1.

6.2.2 BASICUNIT.LOG

Embora o resultado da chamada à função `plGetBasicDecision` seja testado no ficheiro PLAYER.LOG, poderá ser efectuado um rastreamento e consequente teste das chamadas desencadeadas dentro da máquina lógica. A função `plGetBasicDecision` invoca a função `makeBasicDecision` que se encontra no ficheiro BasicUnit.pro. Esta função irá efectuar todos os procedimentos necessários de modo a efectuar e seleccionar, caso seja possível, uma decisão lógica.

Estes procedimentos poderão ser analisados através da impressão no ficheiro BASICUNIT.LOG, dos resultados obtidos.

No caso de se encontrar correcto, o resultado impresso deverá apresentar uma ordem semelhante à apresentada na tabela 5.1.1.1. Os resultados encontram-se na tabela 6.2.2.1 situada no Anexo B.

Após ser seleccionado o comportamento relacional Throwin, é possível verificar para o `kicker` e para o `receiver`, que a impressão das "`check strings`" correspondentes à "`ExecutandoDecisaoBasica_behaviour`" segue a ordem apresentada na tabela 5.1.1.1. No final, após correcta execução das correspondentes acções primitivas seleccionadas, o comportamento relacional é terminado e um novo comportamento é seleccionado, neste caso "`none`".

É ainda possível verificar para cada um dos intervenientes, o sucesso da execução após a impressão da "`check string`" "`basicBehaviour_FINISHSUCCESS`".

6.2.3 CONTROL.LOG & FICHEIROS .LOG CORRESPONDENTES ÀS ACÇÕES PRIMITIVAS

A existência de um ficheiro `.LOG` para cada uma das acções primitivas executadas após selecção da máquina lógica permite verificar a correcta execução dessas acções, constatando-se que estas são percorridas e efectuadas com sucesso.

Tendo sido seleccionadas, cada uma destas acções primitivas é invocada através do micro-agente de `control` encarregue de gerir todo o processo de controlo de cada robot.

Analise-se em primeiro lugar a sequência de "`check strings`" existente no ficheiro `CONTROL.LOG` e verifique-se que a chamada das acções primitivas corresponde à sequência seleccionada pela máquina lógica.

Analisando a tabela 6.2.3.1 no Anexo B, é possível verificar algumas situações distintas.

1. No início do `loop` do micro-agente, "`controlMALoop INICIO`", a acção seleccionada por defeito é a acção primitiva standby. Esta acção é atribuída por defeito a qualquer robot quando este não se encontra comprometido com nenhum comportamento.
2. Após selecção de uma acção por parte da máquina lógica, dá-se início ao request da acção, sendo que depois de ter sido encontrado o `Plugin` correspondente ao código da acção seleccionada, essa mesma acção é executada.
3. A sequência das acções primitivas vem novamente demonstrar o sucesso de toda a operação.

4. A invocação dos métodos existentes em cada um dos ficheiros das acções primitivas é visualizada através das "check strings", "CONTROL-changeto" e "CONTROL-METHOD".

A demonstração da sequência gerada para cada acção primitiva será apenas demonstrada para o primeiro caso, uma vez que todos os outros serão idênticos.

A análise à tabela 6.2.3.2 situada no Anexo B, serve apenas para ilustrar a entrada em cada um dos métodos definidos para cada uma das acções primitivas, demonstrando a execução sem erros para qualquer das acções primitivas.

7 Conclusões

O desenvolvimento deste trabalho foi inserido no projecto SocRob, como já foi mencionado. Sendo este um projecto composto por diversos componentes, estando cada um deles ligados e dependentes do trabalho de variadas pessoas, existiam alguns factores a ter em conta no início do trabalho. Em que estado se encontrava o projecto SocRob? Que ferramentas de trabalho eram utilizadas pelo projecto? Qual o estado dos robots nos quais iam ser testados os comportamentos desenvolvidos? Estes factores foram verdadeiramente importantes e decisivos no desenrolar do trabalho.

No que diz respeito ao projecto SocRob, este foi um ano de transicção para todos os aspectos acima mencionados.

Relativamente ao código do projecto SocRob, deu-se início a um redesenho de toda a arquitectura de software existente, migrando todo o código da linguagem C para C++. Esta migração levou a profundas reestruturações e redefinições dos vários componentes do código, levando por vezes a atrasos na implementação de alguns desses componentes. O trabalho a ser desenvolvido por nós, encontrava-se dependente de alguns desses componentes, nomeadamente da estrutura de navegação e controlo dos robots.

No que diz respeito às ferramentas utilizadas pelo projecto, procedeu-se à mudança do simulador existente para testes de código desenvolvido, para o simulador desenvolvido pela empresa `CyberBotics`. Este simulador embora desenvolvido e apoiado pela empresa em causa, encontrava-se ainda numa fase embrionária, surgindo novas "`releases`" corrigindo inúmeros erros existentes nas versões anteriores. Todos estes erros e tempos de espera de correcção vieram atrasar consecutivamente o trabalho que até aí era desenvolvido. É, no entanto de realçar, que o simulador agora utilizado se apresenta numa fase bem mais estável e que permitirá com uma maior facilidade, o teste de qualquer código desenvolvido futuramente.

Existe ainda um terceiro ponto que se prende com os robots nos quais os comportamentos desenvolvidos iam ser testados. Dado que os robots nos quais o projecto SocRob testava o código desenvolvido não conseguiam dar resposta às exigências actuais, novos robots foram projectados procedendo-se à sua produção.

Em suma, todo o trabalho desenvolvido encontrou-se inserido numa fase de profundas mudanças e alterações de toda a estrutura do projecto SocRob, criando algumas perturbações e impossibilidades na concretização com sucesso do respectivo trabalho.

O objectivo final deste trabalho estando condicionado à partida, passava em grande parte pela criação de novos comportamentos relacionais, bem como pela demonstração do seu funcionamento. Existem no entanto um conjunto de sub-objectivos que se encontra intimamente ligado a este objectivo. A estrutura de objectivos a atingir encontra-se na tabela 7.1 abaixo apresentada.

CRIAÇÃO DE NOVOS COMPORTAMENTOS RELACIONAIS	✓
DEMONSTRAÇÃO DO FUNCIONAMENTO DE CADA UM DOS COMPORTAMENTOS CRIADOS	✗
Clarificação da metodologia existente	✓
- Definições e conceitos a definir	✓
- Terminologia utilizada	✓
Alteração do desenho de implementação de um comportamento relacional	✓
- Substituição das máquinas de estados por redes de Petri	✓
- Criar uma forte componente de proximidade na passagem das redes de Petri para o código de implementação	✓
Transformar a comunicação num valor acrescentado ao comportamento	✓
Criação de um algoritmo de construção de um comportamento relacional	✓
Análise das redes de Petri criadas para cada um dos comportamentos	✓
Implementação de cada comportamento	✗
- Análise da máquina lógica implementada para cada comportamento	✓
- Teste de cada comportamento no simulador Webots	✗
- Teste de cada comportamento nos robots omnidireccionais	✗

Tabela 7.1 – Tabela de objectivos existentes ao longo do trabalho

Analisando a respectiva tabela é possível verificar a existência de dois sub-objectivos que não foram concretizados acabando por influenciar o resultado final.

As razões do insucesso obtido para cada um destes pontos prende-se com o facto da não existência das funções de navegação e controlo, vindo deste modo impossibilitar qualquer tentativa da nossa parte em testar o trabalho até aí desenvolvido. Tal como mencionado anteriormente, estando este trabalho inserido num projecto que se encontra dependente de inúmeros módulos de implementação, módulos esses que não se encontram apenas sob a responsabilidade de uma única pessoa, existia a possibilidade de este trabalho estar dependente de trabalhos de outros. Este facto tornou-se uma realidade quando após se ter atingido o términus da implementação da máquina lógica, ser necessário toda a implementação de controlo e navegação que ainda não se encontrava finalizada. Confrontados com a escassez de tempo e com uma previsão indefinida para o completo términus dessas mesmas funções, foi obrigatório proceder à paragem da implementação terminando assim o respectivo trabalho.

Os objectivos concretizados com sucesso vêm apresentar a consolidação de toda uma área de desenvolvimento onde ainda existiam algumas incoerências e indefinições. A metodologia apresentada permite a criação de comportamentos relacionais estejam eles inseridos em situações estáticas ou dinâmicas de jogo, de uma forma simples e directa. Os casos apresentados apenas contemplam dois intervenientes, no entanto o número de agentes inseridos em cada comportamento não é restrição. É possível existirem mais do que 2 agentes, sendo dois o número mínimo permitido.

Com o trabalho desenvolvido aguarda-se com alguma ansiedade que a existência de jogadas cooperativas entre robots seja não uma possibilidade, mas sim uma certeza.

8 Bibliografia

- [1] Bob van der Vecht, Pedro Lima **"Formulation and Implementation of Relational Behaviours for Multi-Robot Cooperative Systems"**, RoboCup 2004 International Symposium, Lisboa, Portugal

- [2] Vasco Pires, Miguel Arroz, Luis Custódio, **"Logic Based Hybrid Decision System for a Multi-robot Team"**, 8th Conference on Intelligent Autonomous Systems, Amsterdam, The Netherlands, 2004.

- [3] MSL Technical Committee, "**Middle Size Robot League Rules and Regulations 2004"**

- [4] Bruno Damas, Pedro Lima **"Stochastic Discrete Event Model of a Multi-Robot Team Playing an Adversarial Game"**, 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles - IAV2004, Lisboa, Portugal

- [5] E. Pagello, A. d'Angelo, F. Montsello, F. Garelli, and C. Ferrari. **Cooperative behaviors in multi-robot systems through implicit communication. Robotics and Autonomous Systems**, 29(1):65–77, 1999.

- [6] K. Yokota, K. Ozaki, N. Watanabe, D. Matsumoto, A.and Koyama, T. Ishikawa, K. Kawabata, H. Kaetsu, and H. Asama. Uttori united: **Cooperative team play based on communication.** In RoboCup-98: Robot Soccer World Cup II, pages 479–484. Springer-Verlag, 1999.

- [7] A. Collinot, P. Carle, and K. Zeghal. Cassiopeia: **a method for designing computational organizations. In Proceedings of the First International Workshop on Decentralized Intelligent Multi-Agent Systems,** pages 124–131, 1995.

- [8] A. Drogoul and A. Collinot. Applying an agent-oriented methodology to the design of artificial organizations: **A case study in robotic soccer. Autonomous Agents and Multi-Agent Systems,** 1:113–129, 1998.

- [9] H. Matsubara, I. Noda, and K. Hiraki. Learning of cooperative actions in multiagent systems: **a case study of pass in soccer. In Adaptation, Coevolution and Learning in Multi-agent Systems:** Papers from the AAAI-96 Spring Symposium, pages 63–67, 1996**.**

- [10] M Tambe. Towards flexible teamwork. **Journal of Artificial Intelligence Research,** 7:83–124, 1997.

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

- [12] Pedro Lima, Luis Custódio **"Artificial Intelligence and Systems Theory Applied to Cooperative Robots"**, International Journal of Advanced Robotic Systems, No. 3, Sep 2004.

- [13] Frank Dylla, Alexander Ferrein, Thomas Rofer, et al "**Towards a League-Independent Qualitative Theory for RoboCup",** 2004

- [14] Vídeos dos jogos da Middle Size League dos Robocup 2002, 2003 e 2004

- [15] P. Lima, L. Custódio, and et al. Isocrob 2003: **Team description paper. In RoboCup 2003: Robot Soccer World Cup VII**. Springer-Verlag, 2003.

- [17] B. Machado, C. Cardoso, T.F.C: **Sistema de Projecto e Controlo de Missão de uma Equipa de Robots Cooperante**

- [18] Lafortune, S.; Cassandras, Christos G., **Introduction to Discrete Event Systems**, Kluwer Academic Publ., 1999

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

- [20] The Robocup Federation, **Robocup Brief Introduction,** http://www.robocup.org/Intro.htm

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

- [22] Paul S. Schenker et al, **Planetary Roger Development Supporting Mars Exploration, Sample return and Future Human-Robotic Colonization**, IEEE 10th International Conference on Advanced Robotics, Budapest, Hungary, 2001

- [23] James McLurkin, **Using Cooperative Robots for Explosive Ordnance Disposal**

- [24] Fei Yeu Wang, et al., **A Petri Net Coordination Model for an Intelligent Mobile Robot**

9 Anexos

9.1 A : SIMULADOR WEBOTS®

O Webots® é um simulador comercial da Cyberbotics que se encontra actualmente em utilização no polo de Lisboa do Instituto Superior de Robótica. Este simulador constitui um ambiente de prototipagem, modelação e simulação de robots, permitindo a simulação dos comportamentos num ambiente externo, fiável e perfeitamente controlado, uma vez que se admitem condições perfeitas e conhecimento total do mundo, algo que não acontece nos robots reais. Este simulador permite a execução do código original da equipa, requerendo para tal apenas pequenas alterações. O simulador apresenta, além de outras, as seguintes características:

- simulador gráfico baseado em OpenGL;
- apresenta editor do mundo e dos robots em VRML;
- possibilidade de criação/edição do robot e ambiente de teste;
- controlo total do ambiente de simulação;
- contém um leque de robots mais comuns já criados assim como diversos sensores, câmaras, tipos de locomoção e actuadores;
- utilizado sobretudo para investigação e simulação de agentes e algoritmos adaptativos;
- utilização de ODE (Open Dynamics Engine) para simulação da física.

9.1.1 O Mundo Webots

A correcta modelação do mundo onde se pretendem executar as simulações constitui o primeiro passo a realizar. É essencial que as principais características deste mundo sejam correctamente modeladas, assim como as características dos vários intervenientes. Foi assim desenvolvido pela equipa do projecto SocRob o modelo do campo de futebol, o modelo do robot omnidireccional e da bola.

No contexto deste trabalho, foi utilizado um ambiente de simulação composto por uma bola, dois robots omnidireccionais e o campo de futebol. (figs 1, 2 e 3, já existentes…)

9.1.2 Princípios de Funcionamento

No modelo utilizado, existem três identidades fundamentais: o robot, o supervisor e a física. Quando a aplicação `Webots` é executada, são lançados vários processos: um para cada robot presente em campo, um para o controlador e outro para a física. Deste modo, estas identidades são totalmente independentes entre si, não obstante a possibilidade da comunicação de dados entre eles. Estes processos são os responsáveis pela interacção do utilizador com os objectos no mundo simulado.

- Controlador do Robot - ste controlador é o responsável pela interacção com o robot modelado ao qual está associado pelo `Webots`. Esta interacção verifica-se sobretudo a nível da leitura e escrita da velocidade de rotação dos servos que, por sua vez, controlam a velocidade das rodas e dão movimento ao objecto na simulação.
- Controlador do Supervisor - este controlador tem conhecimento de todos os objectos presentes no mundo, permitindo ainda controlar todas as propriedades desses objectos. A sua utilização deve-se essencialmente à necessidade de obtenção da posição precisa da bola durante a simulação.
- Controlador da Física - o módulo de física é responsável pela configuração personalizada da física no modelo. Por omissão, o `Webots` recorre a um modelo de física similar ao mundo real e cujos parâmetros são definidos no mundo criado, o que permite uma simulação credível de situações tais como atrito, colisões, aplicação de forças. No âmbito deste projecto, os parâmetros por omissão são os utilizados. A utilização do módulo de física restringe-se à emulação do kicker, o qual é inexistente nos modelos utilizados.

9.1.3 Implementação

De forma a que a execução do código produzido neste projecto produza resultados no simulador, é necessário que os comandos produzidos lhe sejam comunicados. Para tal, a implementação do novo código contempla a criação de `Sockets TCP-IP` entre o código e o controlador do robot e a implementação de `sockets UDP` para a comunicação dos comandos entre o código e a física.

Sockets TCP-IP

Os `sockets TCP-IP` exigem que uma ligação bi-direccional seja efectuada entre o emissor e receptor, podendo esta durar toda a execução do programa. Este tipo de comunicação assegura que todos os pacotes são enviados/ recebidos e pela ordem correcta, sendo essencialmente indicada para situações em que a comunicação é frequente, bidireccional e cujos intervenientes estão bem definidos. Tal é o caso da comunicação entre o código SocRob e o controlador de cada um dos robots no `Webots`: há uma execução do código por cada robot existente no mundo simulado e existe comunicação frequente entre eles durante toda a execução.

Sockets UDP

Os `Sockets UDP` são `sockets` sem ligação, ou seja, enviam a informação para um endereço determinado, não dando quaisquer garantias da sua chegada ao destino e ordem de chegada. É especialmente indicado para situações onde não é necessário grande fiabilidade na comunicação. No caso deste trabalho, este tipo de ligação é utilizado na emulação do kicker, cuja comunicação ocorre entre o código SocRob e o módulo de física. Este foi o protocolo escolhido pelo facto de a comunicação ser unidireccional e não confirmada e pela simplicidade de implementação.

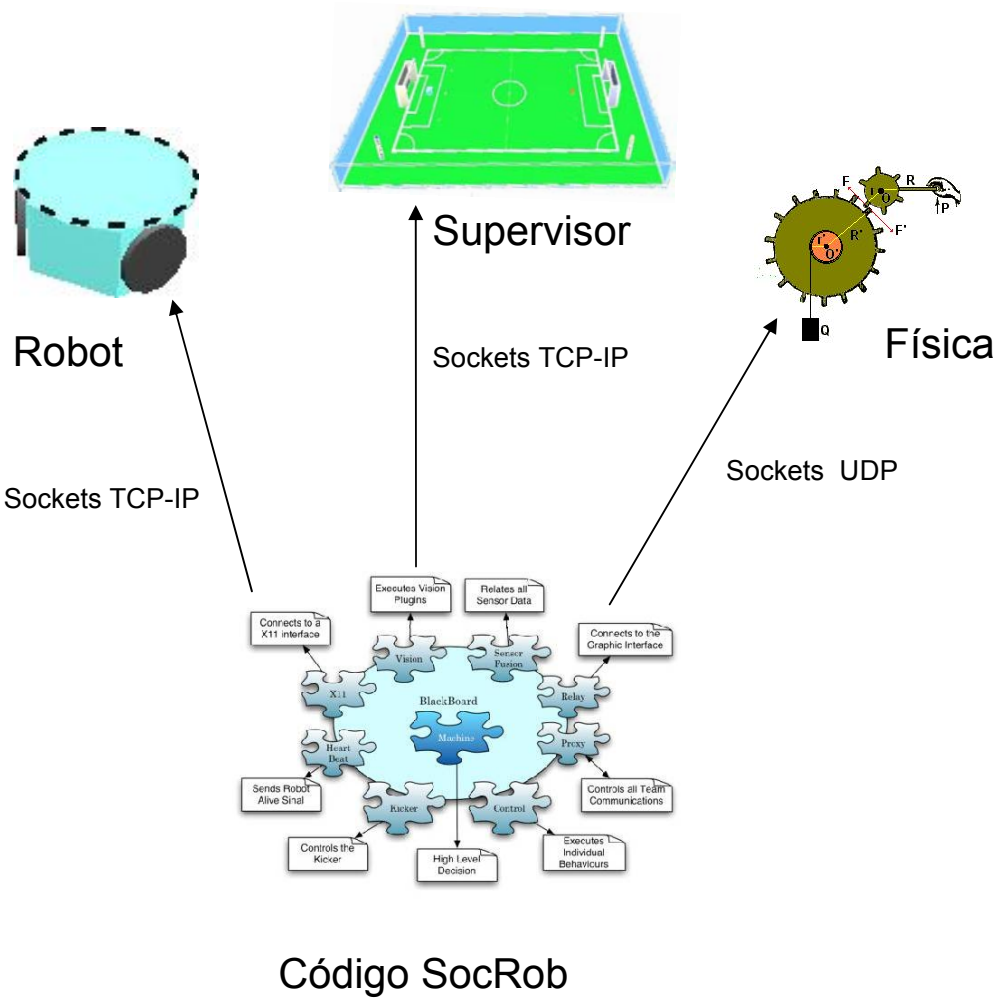

Figura 9.1.3.1: Esquemático das comunicações implementadas

Visão

A visão representa um aspecto fulcral no futebol robótico pelo seu papel na aquisição de informação do mundo. A utilização deste componente, tal como é usado nos robots reais, não se revestia de importância significativa o que, aliado à complexidade das transformações necessárias à sua adaptação, decretaram a sua não utilização. A informação, que deveria provir da visão, foi substituída pela utilização do Supervisor, para obter dados gerais do mundo tais como a posição da bola, e pela utilização de um nó específico dos robots modelados no `webots:` o GPS especialmente indicado para a gestão da odometria.

Implementação do Kicker

O `kicker` está implementado no modelo, não como um dispositivo físico, mas sim como uma força modelada pelo módulo de física do `webots`, a qual é aplicada apenas na bola. Tal é conseguido através da utilização do `ODE`, o qual permite a definição da direcção e intensidade de uma força e a sua aplicação a um objecto bem determinado no mundo `Webots`. Deste modo é possível a modelação de correntes em simulações subaquáticas, a acção dos ventos e, neste caso específico, de um equipamento de remate que não existe fisicamente no modelo. A execução do `kicker` depende apenas da verificação de pré-condições existentes no modelo implementado e não da acção física verificada no simulador.

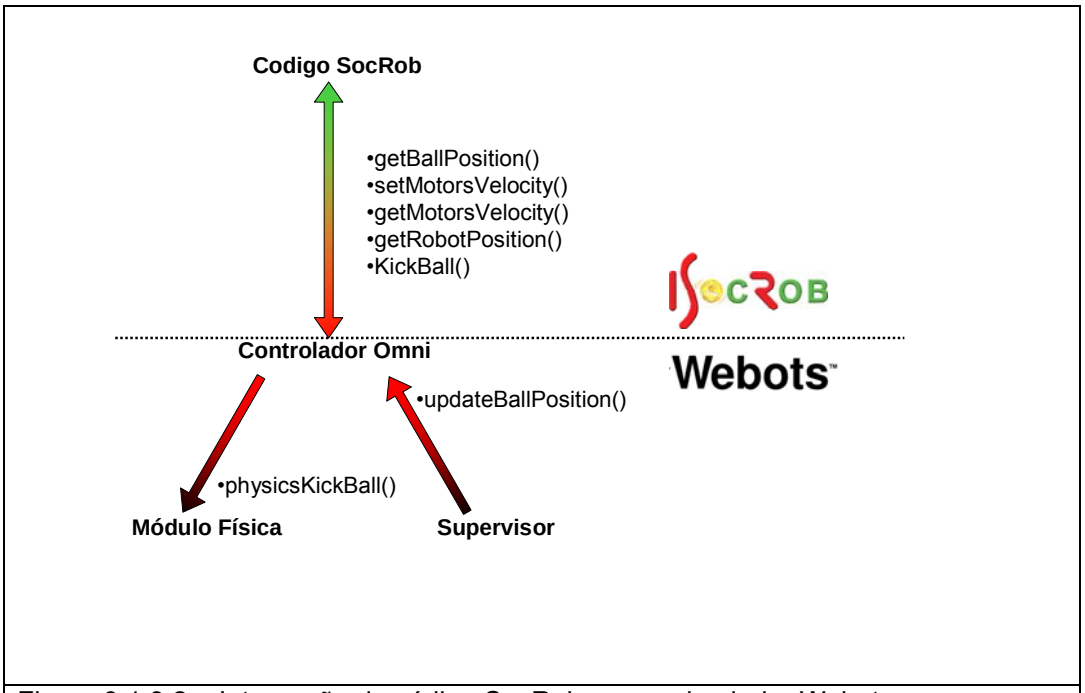

Figura 9.1.3.2 – Integração do código SocRob com o simulador Webots

9.2 B : TABELAS

9.2.1 Capítulo 5, Secção 1, Ponto 1, Tabela Nº1 - Throwin

Commitment Phases		Execução Principal		Execução Alternativa		Execução Falhada	
		Commitment States		Commitment States		Commitment States	
		Kicker	Receiver	Kicker	Receiver	Kicker	Receiver
Setup	*request*					-	-
	accept					-	-
Loop	*Prepare*	FindPass Trajectory	StandBy	FindPass Trajectory	StandBy	FindPass Trajectory	StandBy
		Rotate	Movetoxy	FindKick	–	–	–
	Move			Trajectory			
	Pass	Passing	StandBy	Rotate	–	–	–
	Intercept	–	Intercepting	Kick	–	–	–
		Finish Successful	Finish Successful	Finish Successful	Finish UnSuccessful	Finish UnSuccessful	Finish UnSuccessful
	Finish						
End	*Done*	-	-	-	-	-	-
	failed	-	-	-	-	-	-
Default	*none*	-	-	-	-	-	-

Tabela 5.1.1.1 – Tabela de estados do comportamento relacional Throwin

9.2.2 Capítulo 5, Secção 1, Ponto 1, Tabela Nº2 - Throwin

Estado do Comportamento	*Condições*		
	Execução Principal	*Execução Alternativa*	*Execução Falhada*
request	- o jogador pode ter qualquer papel	Idêntico à execução principal	Idêntico à execução principal
	- o jogador tem a bola	" "	" "
	- o jogador terá de efectuar a marcação do fora	" "	" "
accept	- **commitment.throwin.kicker.state** = *request*	" "	" "
	- papel do receptor no compromisso é *receiver*	" "	" "
	- o jogador pode ter qualquer papel	" "	" "
	- o jogador pode-se encontrar em qualquer ponto do campo	" "	" "
	- o receptor também se pode encontrar em qualquer ponto	" "	" "
	- o receptor representa o elemento mais bem posicionado para receber a bola	" "	" "
Prepare	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*
	commitment.throwin.receiver.state = *accept*	**commitment.throwin.receiver.state** = *accept*	**commitment.throwin.receiver.state** = *accept*
Move	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*
	- comportamento individual do kicker (acção primitiva) é FindPassTrajectory	- comportamento individual do kicker (acção primitiva) é FindPassTrajectory	- comportamento individual do kicker (acção primitiva) é FindPassTrajectory
	- comportamento individual do receiver (acção primitiva) é Standby	- comportamento individual do receiver (acção primitiva) é Standby	- comportamento individual do receiver (acção primitiva) é Standby
	- o robot kicker verifica o estado da oposição adversária	- o robot kicker verifica o estado da oposição adversária	- Em caso de insucesso UnS-Reason K1 ou K6 por parte do kicker, ocorre a transicção para o estado Finish.

	-o robot receiver passa a conhecer a posição <x,y> pretendida para o passe se não houver oposição adversária.	*-o robot receiver desiste do comportamento devido ao timeout expirar*	- Em caso de insucesso UnS-Reason R1 ou R5 por parte do receiver, ocorre a transicção para o estado Finish.
Pass	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*
	- comportamento individual do kicker (acção primitiva) é *Rotate*	- comportamento individual do kicker (acção primitiva) é Find*KickTrajectory*	- comportamento individual do kicker (acção primitiva) é *Rotate*
	- comportamento individual do receiver (acção primitiva) é *Movetoxy*	---	- comportamento individual do receiver (acção primitiva) é *Movetoxy*
	- o robot receiver prepara-se para receber a bola se a oposição adversária não constituir problema	---	- Em caso de insucesso UnS-Reason K2, K5 ou K7 por parte do kicker, ocorre a transicção para o estado Finish.
	- o robot receiver encontra-se na posição <x,y> definida pelo robot kicker e aguarda pelo passe do kicker ou de uma indicação de términus de comportamento	- o robot receiver termina o comportamento com insucesso	- Em caso de insucesso UnS-Reason R2 ou R6 por parte do receiver, ocorre a transicção para o estado Finish.
Intercept	- papel no compromisso é *receiver*	- papel no compromisso é *kicker*	- papel no compromisso é *receiver*
	- comportamento individual do kicker (acção primitiva) é *Passing*	- comportamento individual do kicker (acção primitiva) é *Rotate*	- comportamento individual do kicker (acção primitiva) é *Passing*
	- comportamento individual do receiver (acção primitiva) é *Standby*	---	- comportamento individual do receiver (acção primitiva) é *Standby*
	- a bola foi passada ou chutada	---	- Em caso de insucesso UnS-Reason K3, K8 ou K9 por parte do kicker, ocorre a transicção para o estado Finish.
	---	---	- Em caso de insucesso UnS-Reason R3 ou R7 por parte do receiver, ocorre a transicção para o estado Finish.
Finish	- papel no compromisso é *receiver ou kicker*	- papel no compromisso é *kicker*	- papel no compromisso é *receiver*

			ou kicker
	- comportamento individual do receiver (acção primitiva) é *Intercepting*	- comportamento individual do receiver (acção primitiva) é *kick*	- comportamento individual do receiver (acção primitiva) é *Intercepting*
	- a bola foi recebida pelo receiver	- a bola foi chutada pelo kicker	- Em caso de insucesso UnS-Reason K4 por parte do kicker, ocorre a transicção para o estado Finish.
	- o robot kicker observa apenas a continuação da jogada	---	- Em caso de insucesso UnS-Reason R4 ou R8 por parte do receiver, ocorre a transicção para o estado Finish.
Done	- comportamento executado com sucesso	- comportamento alternativo executado com sucesso	- comportamento executado com insucesso
Failed	- qualquer situação de falha	- qualquer situação de falha	- qualquer situação de falha
none	- o estado é *done* ou *failed*	- o estado é *done* ou *failed*	- o estado é *done* ou *failed*

Tabela 5.1.1.2 – Tabela de transições de estados do comportamento relacional Throwin

9.2.3 Capítulo 5, Secção 1, Ponto 2, Tabela Nº1 – Goal Kick

Commitment Phases		**Execução Principal**		**Execução Alternativa**		**Execução Falhada**	
		Commitment States		**Commitment States**		**Commitment States**	
		Kicker	**Receiver**	**Kicker**	**Receiver**	**Kicker**	**Receiver**
Setup	*request*					-	-
	accept					-	-
Loop	*Prepare*	FindPass Trajectory	StandBy	FindPass Trajectory	StandBy	FindPass Trajectory	StandBy
	Move	Rotating	Movetoxy	FindKick Trajectory	-	-	-
	Pass	Passing	StandBy	Rotate	-	-	-
	Intercept	-	Intercepting	Kick	-	-	-
	Finish	Finish Successful	Finish Successful	Finish Successful	Finish UnSuccessful	Finish UnSuccessful	Finish UnSuccessful
End	*Done*					-	-
	failed					-	-
Default	*none*					-	-

Tabela 5.1.2.1 – Tabela de estados do comportamento relacional Goal Kick

9.2.4 Capítulo 5, Secção 1, Ponto 2, Tabela Nº2 – Goal Kick

Estado do Comportamento	*Condições*		
	Execução Principal	*Execução Alternativa*	*Execução Falhada*
request	- o jogador tem o papel *defender* - o jogador tem a bola - o jogador terá que fazer a cobrança do pontapé de baliza	Idêntico à execução principal " " " "	Idêntico à execução principal " " " "
accept	- **commitment.goalkick.kicker.state** = *request* - papel do receptor no compromisso é *receiver* - o jogador pode ter qualquer papel - o receptor encontra-se no seu meio campo defensivo. - o receptor representa o elemento mais bem posicionado para receber a bola	" " " " " " " " " "	" " " " " " " " " "
Prepare	- papel no compromisso é *kicker* **commitment.goalkick.receiver.state** = *accept*	- papel no compromisso é *kicker* **commitment.goalkick.receiver.state** = *accept*	- papel no compromisso é *kicker* **commitment.goalkick.receiver.state** = *accept*
Move	- papel no compromisso é *kicker* - acção primitiva do kicker é FindPassTrajectory - acção primitiva do receptor é Standby - o robot kicker verifica o estado da oposição adversária	- papel no compromisso é *kicker* - acção primitiva do kicker é FindPassTrajectory - acção primitiva do receptor é Standby - o robot kicker verifica o estado da oposição adversária	- papel no compromisso é *kicker* - acção primitiva do kicker é FindPassTrajectory - acção primitiva do receiver é Standby - Em caso de insucesso UnS-Reason K1 ou K6 por parte do kicker, ocorre a transicção para o estado Finish.
	-o robot receiver passa a conhecer a posição <x,y> pretendida para o passe se não houver oposição adversária. Esta posição nunca é frontal à própria baliza.	-o robot receiver desiste do comportamento devido ao timeout expirar	- Em caso de insucesso UnS-Reason R1 ou R5 por parte do kicker, ocorre a transicção para o estado Finish.
Pass	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*	- papel no compromisso

			é *kicker*
	- acção primitiva do kicker é *Rotate*	- acção primitiva do kicker é Find*KickTrajectory*	- acção primitiva do kicker é *Rotate*
	- acção primitiva do receiver é *Movetoxy*	---	- acção primitiva do receiver é *Movetoxy*
	- o robot receiver prepara-se para receber a bola se a oposição adversária não constituir problema	---	- Em caso de insucesso UnS-Reason K2, K5 ou K7 por parte do kicker, ocorre a transicção para o estado Finish.
	- o robot receiver encontra-se na posição <x,y> definida pelo robot kicker e aguarda pelo passe do kicker ou de uma indicação de términus de comportamento	- o robot receiver termina o comportamento com insucesso	- Em caso de insucesso UnS-Reason R2 ou R6 por parte do kicker, ocorre a transicção para o estado Finish.
Intercept	- papel no compromisso é *receiver*	- papel no compromisso é *kicker*	- papel no compromisso é *receiver*
	- acção primitiva do kicker é *Passing*	- acção primitiva do kicker é *Rotating*	- acção primitiva é *Passing*
	- acção primitiva do receiver *Standby*	---	- acção primitiva é *Standby*
	- a bola foi passada.	---	- Em caso de insucesso UnS-Reason K3, K8 ou K9 por parte do kicker, ocorre a transicção para o estado Finish.
	---	---	- Em caso de insucesso UnS-Reason R3 ou R7 por parte do kicker, ocorre a transicção para o estado Finish.
Finish	- papel no compromisso é *receiver ou kicker*	- papel no compromisso é *kicker*	- papel no compromisso é *receiver ou kicker*
	- a acção primitiva do receiver é *Intercepting*	- acção primitiva do kicker é *kicking*	- acção primitiva do receiver é *Intercepting*
	- a bola foi recebida pelo receiver	- a bola foi chutada pelo kicker para a linha lateral	- Em caso de insucesso UnS-Reason K4 por parte do kicker, ocorre a transicção para o estado Finish.
	- o robot kicker observa apenas a continuação da jogada	---	- Em caso de insucesso UnS-Reason R4 ou R8 por parte do kicker, ocorre a transicção para o estado Finish.
Done	- comportamento executado com	- comportamento alternativo	- comportamento

Failed	sucesso – qualquer situação de falha	executado com sucesso – qualquer situação de falha	executado com insucesso – qualquer situação de falha
none	– o estado é *done* ou *failed*	– o estado é *done* ou *failed*	– o estado é *done* ou *failed*

Tabela 5.1.2.2 – Tabela de transições de estados do comportamento relacional Goal Kick

9.2.5 Capítulo 5, Secção 1, Ponto 3, Tabela Nº1 - KickOff

Commitment Phases		**Execução Principal**		**Execução Alternativa**		**Execução Falhada**	
		Commitment States		**Commitment States**		**Commitment States**	
		Kicker	**Receiver**	**Kicker**	**Receiver**	**Kicker**	**Receiver**
Setup	*request*					-	-
	accept					-	-
Loop	*Prepare*	StandBy	FindGoalPath	–	–	StandBy	FindGoalPath
	Positioning	Rotate	–	–	–	–	–
	Pass	Passing	StandBy	–	–	–	–
	Intercept	–	Intercepting	–	–	–	–
	Finish	Finish Successful	Finish Successful	–	–	Finish UnSuccessful	Finish UnSuccessful
End	*Done*					-	-
	failed					-	-
Default	*none*					-	-

Tabela 5.1.3.1 – Tabela de estados do comportamento relacional KickOff

9.2.6 Capítulo 5, Secção 1, Ponto 3, Tabela Nº2 - KickOff

Estado do Comportamento	*Condições*		
	Execução Principal	*Execução Alternativa*	*Execução Falhada*
request	- o cobrador pode ter qualquer papel	Não existe execução alternativa para o comportamento relacional kickoff	Idêntico à execução principal
	- o cobrador tem a bola	" "	" "
	- o cobrador é o jogador designado à cobrança da jogada	" "	" "
accept	- **commitment.kickoff.kicker.state** = *request*	" "	" "
	- papel do receptor no compromisso é *receiver*	" "	" "
	- o cobrador pode ter qualquer papel	" "	" "
	- o receptor pode-se encontrar em qualquer ponto do campo	" "	" "
	- o receptor representa o elemento mais bem posicionado para receber a bola	" "	" "
Prepare	- papel no compromisso é *receiver*	" "	" "
	commitment.kickoff.receiver.state = *accept*	" "	" "
Positioning	- papel no compromisso é *kicker*	" "	- papel no compromisso é *kicker*
	- acção primitiva do cobrador é StandBy	" "	- acção primitiva do cobrador é StandBy
	- acção primitiva do receptor é FindGoalPath	" "	- acção primitiva do receptor é FindGoalPath
	- o receptor avalia as possíveis trajectórias para a baliza adversária, calcula uma postura possível e assume-a.	" "	- Em caso de insucesso UnS-Reason K1 ou K2 por parte do cobrador, ocorre a transicção para o estado Finish.
	-o receptor informa o cobrador da posição para onde ele pretende ter a bola	" "	- Em caso de insucesso UnS-Reason R1 ou R2 por parte do receptor, ocorre a transicção para o estado

			Finish.
Pass	- papel no compromisso é *kicker*	" "	- papel no compromisso é *kicker*
	- acção primitiva do cobrador é *Rotate*	" "	- acção primitiva do cobrador é *Rotate*
	- o robot receptor encontra-se em standby na posição <X,Y> comunicada ao cobrador	" "	- Em caso de insucesso UnS-Reason K3 ou K4 por parte do cobrador, ocorre a transicção para o estado Finish.
Intercept	- papel no compromisso é *kicker*	" "	- papel no compromisso é *kicker*
	- acção primitiva é *Passing*	" "	- acção primitiva é *Passing*
	- comportamento individual do receptor (acção primitiva) é *Standby*	" "	- comportamento individual do receptor (acção primitiva) é *Standby*
	- a bola foi passada	" "	- Em caso de insucesso UnS-Reason K5, K6 por parte do cobrador, ocorre a transicção para o estado Finish.
	---	" "	- Em caso de insucesso UnS-Reason R3 por parte do receptor, ocorre a transicção para o estado Finish.
Finish	- papel no compromisso é *receiver*	" "	- papel no compromisso é *receiver*
	- acção primitiva do receptor é *Intercepting*	" "	- acção primitiva é *Intercepting*
	- a bola foi recebida pelo receptor	" "	- Em caso de insucesso UnS-Reason K4 por parte do receiver, ocorre a transicção para o estado Finish.
	- o robot cobrador observa apenas a continuação da jogada	" "	---
Done	- comportamento executado com sucesso	" "	- comportamento executado com insucesso

Failed	- qualquer situação de falha	" "	- qualquer situação de falha
none	- o estado é *done* ou *failed*	" "	- o estado é *done* ou *failed*

Tabela 5.1.3.2 – Tabela de transições de estados do comportamento relacional KickOff

9.2.7 Capítulo 5, Secção 2, Ponto 1, Tabela Nº1 - OneTwoPass

Commitment Phases		**Execução Principal**		**Execução Falhada**	
		Commitment States		**Commitment States**	
		Kicker	**Receiver**	**Kicker**	**Receiver**
Setup	*request*			-	-
	accept			-	-
Loop	*Prepare - Kicker*	FindPass Trajectory	StandBy	FindPass Trajectory	StandBy
	Move - Kicker	Rotate	Movetoxy	–	–
	Pass – Kicker	Passing	StandBy	–	–
	Intercept - Receiver	Movetoxy	Intercepting		
	Prepare – Receiver	StandBy	FindPass Trajectory		
	Rotate – Receiver	StandBy	Rotate		
	Pass - Receiver	–	Passing		
	Intercept - Kicker	Intercepting	–	–	–
End	*Finish*	Finish Successful	Finish Successful	Finish UnSuccessful	Finish UnSuccessful
	Done	-	-	-	-
	failed	-	-	-	-
Default	*none*	-	-	-	-

Tabela 5.2.1.1 – Tabela de estados do comportamento relacional OneTwoPass

9.2.8 Capítulo 5, Secção 2, Ponto 1, Tabela Nº2 - OneTwoPass

Estado do Comportamento	*Condições*	
	Execução Principal	*Execução Falhada*
request	- o jogador pode ter qualquer papel	Idêntico à execução principal
	- o jogador tem a bola	" "
accept	- **commitment.onetwopass.kicker.state** = *request*	" "
	- papel do receptor no compromisso é *receiver*	" "
	- o jogador pode ter qualquer papel	" "
	- o jogador pode-se encontrar em qualquer ponto do campo	" "
	- o receptor também se pode encontrar em qualquer ponto	" "
	- o receptor representa o elemento mais bem posicionado para receber a bola	" "
Prepare - Kicker	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*
	commitment. onetwopass.receiver.state = *accept*	**commitment .onetwopass.receiver.state** = *accept*
Move - Kicker	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*
	- comportamento individual do kicker (acção primitiva) é FindPassTrajectory	- comportamento individual do kicker (acção primitiva) é FindPassTrajectory
	- comportamento individual do receiver (acção primitiva) é Standby	- comportamento individual do receiver (acção primitiva) é Standby
	- o robot kicker verifica o estado da oposição adversária	- Em caso de insucesso UnS-Reason K1 ou K2 por parte do kicker, ocorre a transicção para o estado Finish.
	-o robot receiver passa a conhecer a posição <x,y> pretendida para o passe se não houver oposição adversária.	- Em caso de insucesso UnS-Reason R1 ou R2 por parte do receiver, ocorre a transicção para o estado Finish.
Pass - Kicker	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*
	- comportamento individual do kicker (acção primitiva) é *Rotate*	- comportamento individual do kicker (acção primitiva) é *Rotate*
	- comportamento individual do receiver (acção primitiva) é *Movetoxy*	- comportamento individual do receiver (acção primitiva) é *Movetoxy*

	- o robot receiver prepara-se para receber a bola se a oposição adversária não constituir problema	- Em caso de insucesso UnS-Reason K3, ou K4 por parte do kicker, ocorre a transicção para o estado Finish.
	- o robot receiver encontra-se na posição <x,y> definida pelo robot kicker e aguarda pelo passe do kicker ou de uma indicação de términus de comportamento	- Em caso de insucesso UnS-Reason R3 ou R4 por parte do receiver, ocorre a transicção para o estado Finish.
Intercept - Receiver	- papel no compromisso é *receiver*	- papel no compromisso é *receiver*
	- comportamento individual do kicker (acção primitiva) é *Passing*	- comportamento individual do kicker (acção primitiva) é *Passing*
	- comportamento individual do receiver (acção primitiva) é *Standby*	- comportamento individual do receiver (acção primitiva) é *Standby*
	- a bola foi passada ou chutada	- Em caso de insucesso UnS-Reason K5, ou K6 por parte do kicker, ocorre a transicção para o estado Finish.
	---	- Em caso de insucesso UnS-Reason R5 ou R6 por parte do receiver, ocorre a transicção para o estado Finish.
Prepare - Receiver	- papel no compromisso é *receiver*	- papel no compromisso é *receiver*
	- comportamento individual do kicker (acção primitiva) é *Movetoxy*	- comportamento individual do kicker (acção primitiva) é *Movetoxy*
	- comportamento individual do receiver (acção primitiva) é *Intercepting*	- comportamento individual do receiver (acção primitiva) é *Intercepting*
	- a bola foi recebida pelo receiver	- Em caso de insucesso UnS-Reason K7, ou K8 por parte do kicker, ocorre a transicção para o estado Finish.
	- o robot kicker encontra-se na posição <x,y> indicada previamente ao robot receiver e aguarda pelo passe do receiver ou de uma indicação de términus de comportamento	- Em caso de insucesso UnS-Reason R7 ou R8 por parte do receiver, ocorre a transicção para o estado Finish.
Rotate - Receiver	- papel no compromisso é *receiver*	- papel no compromisso é *receiver*
	- comportamento individual do kicker (acção primitiva) é StandBy	- comportamento individual do kicker (acção primitiva) é StandBy
	- comportamento individual do receiver (acção primitiva) é *FindPassTrajectory compatible with kicker's new position*	- comportamento individual do receiver (acção primitiva) é *FindPassTrajectory compatible with kicker's new position*
	- o robot receiver verifica o estado da oposição adversária	- Em caso de insucesso UnS-Reason K9, ou K10 por parte do kicker, ocorre a transicção para o estado Finish.
	-o robot kicker recebe a confirmação de	- Em caso de insucesso UnS-Reason R9 ou

	que é possível efectuar um passe para a posição definida por ele previamente.	R10 por parte do receiver, ocorre a transicção para o estado Finish.
Pass – Receiver	- papel no compromisso é *receiver*	- papel no compromisso é *receiver*
	- comportamento individual do kicker (acção primitiva) é *StandBy*	- comportamento individual do kicker (acção primitiva) é *StandBy*
	- comportamento individual do receiver (acção primitiva) é *Rotate*	- comportamento individual do receiver (acção primitiva) é *Rotate*
	- o robot kicker continua a aguardar pelo passe do receiver ou de uma indicação de términus de comportamento	- Em caso de insucesso UnS-Reason K11, ou K12 por parte do kicker, ocorre a transicção para o estado Finish.
	---	- Em caso de insucesso UnS-Reason R11 ou R12 por parte do receiver, ocorre a transicção para o estado Finish.
Intercept – Kicker	- papel no compromisso é *receiver*	- papel no compromisso é *receiver*
	- comportamento individual do receiver (acção primitiva) é *Passing*	- comportamento individual do receiver (acção primitiva) é *Passing*
	- a bola foi passada ou chutada	---
		- Em caso de insucesso UnS-Reason R13 ou R14 por parte do receiver, ocorre a transicção para o estado Finish.
Finish	- papel no compromisso é *kicker*	- papel no compromisso é *kicker*
	- comportamento individual do kicker (acção primitiva) é *Intercepting*	- comportamento individual do kicker (acção primitiva) é *Intercepting*
	- a bola foi recebida pelo kicker	- Em caso de insucesso UnS-Reason K13 ou K14 por parte do kicker, ocorre a transicção para o estado Finish.
	- o robot receiver observa apenas a continuação da jogada	---
Done	- comportamento executado com sucesso	- comportamento executado com insucesso
Failed	- qualquer situação de falha	- qualquer situação de falha
none	- o estado é *done* ou *failed*	- o estado é *done* ou *failed*

Tabela 5.2.1.2 – Tabela de transições de estados do comportamento relacional OneTwoPass

9.2.9 Capítulo 6, Secção 1, Ponto 1 – Análise das Redes de Petri

p_n: lugar **n**

t_n:transicção **n**

$t\text{-}_{en}$: Grupo de transições de erro. Este agrupamento verifica-se devido ao igual peso que estas transições têm na execução do comportamento

S_n: Estado número **n**

S_n: [p_1 p_2 p_3 p_4 p_5 p_6]: Descrição do estado **S_n**

T: Transição Disparada

$K_g(S_n)$: Número de tokens do gestor de compromisso no estado Sn

$K_c(S_n)$: Número de tokens do comportamento no estado Sn

Tabela 6.1.1.1 – Tabela explicativa da nomenclatura utilizada em todas as tabelas ao longo da análise

9.2.10 Capítulo 6, Secção 1, Ponto 2– Gestores de Compromisso

Nomenclatura utilizada:

p_1	Commitment Receiver	**p_4**	Commitment Kicker
p_2	Receiver Not Committed	**p_5**	Kicker Not Committed
p_3	Commitment Kicker End	**p_6**	Commitment Receiver End

t_1	Transicção do kicker	**$t\text{-}_{e1}$**	Transicção para **e3** (externa ao mecanismo)
t_2	Transicção do receiver	**$t\text{-}_{e2}$**	Transição que consome **e2**
		$t\text{-}_{e3}$	Transicção para **e6** (externa ao mecanismo)
		$t\text{-}_{e4}$	Transição que consome **e5**

S_n:	**[p_1 p_2 p_3 p_4 p_5 p_6]**	**S_{n-1}**	**T**	**$K_g(S_n)$**	**$K_g(S_n) – K_g(S_{n-1})$**
S_0:	[1 0 0 1 0 0]	-	-	2	-
S_1:	[1 0 0 0 0 1]	S_0	$t\text{-}_{e3}$	2	0
S_2:	[0 1 0 0 0 0]	S_1	t_1	1	- 1

S_3:	[0 0 1 1 0 0]	S_0	$t\text{-}_{e1}$	2	0
S_4:	[0 0 0 0 1 0]	S_1	t_2	1	- 1
S_5:	[0 0 0 0 0 0]	S_2 e S_4	$t\text{-}_{e2}$ ou $t\text{-}_{e4}$	0	- 1

Tabela 6.1.2.1 – Tabela de variação do número de tokens no Gestor de Compromisso

9.2.11 Capítulo 6, Secção 1, Ponto 3 – Kick Off

Nomenclatura Utilizada:

$\mathbf{p_1}$	Stand By	$\mathbf{p_5}$	FindGoalPath
$\mathbf{p_2}$	Rotate	$\mathbf{p_6}$	StandBy
$\mathbf{p_3}$	Passing	$\mathbf{p_7}$	Intercepting
$\mathbf{p_4}$	Finish (kicker)	$\mathbf{p_8}$	Finish (Receiver)

$\mathbf{t_1}$	Receiver Avaliable	$\mathbf{t_6}$	Ball intercepted Successfully
$\mathbf{t_2}$	Pass executed Successfully	$\mathbf{t\text{-}_{e1}}$	Transicções insucesso k2,k4 e k6
$\mathbf{t_3}$	Rotate Done	$\mathbf{t\text{-}_{e2}}$	Transicções insucesso k1,k3,k5
$\mathbf{t_4}$	Found Goal Trajectory	$\mathbf{t\text{-}_{e3}}$	Transicções insucesso R2,R4
$\mathbf{t_5}$	Got Confirmation	$\mathbf{t\text{-}_{e4}}$	Transicções insucesso R1,R3

$\mathbf{S_n}$:	$\mathbf{[p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ p_8]}$	$\mathbf{S_{n-1}}$	**T**	$\mathbf{K_c(S_n)}$	$\mathbf{K_c(S_n) - K_c(S_{n-1})}$
S_0:	[1 0 0 0 1 0 0 0]	-	-	2	-
S_1:	[1 0 0 0 0 1 0 0]	S_0	t_4	2	0
S_2:	[0 1 0 0 0 1 0 0]	S_1	t_1	2	0
S_3:	[0 0 1 0 0 1 0 0]	S_2	t_3	2	0
S_4:	[0 0 0 1 0 1 0 0]	S_3	t_2	2	0
S_5:	[0 0 0 1 0 0 1 0]	S_4	t_5	2	0
S_6:	[0 0 0 1 0 0 0 1]	S_5	t_6	2	0

$S\text{-}_{e1}$	[0 0 0 0 - - - -]	$[S_0,S_5]$	$t\text{-}_{e1}$	2	0
$S\text{-}_{e2}$	[0 0 0 0 - - - -]	$[S_0,S_5]$	$t\text{-}_{e2}$	2	0
$S\text{-}_{e3}$	[- - - - 0 0 0 0]	$[S_0,S_5]$	$t\text{-}_{e3}$	2	0
$S\text{-}_{e4}$	[- - - - 0 0 0 0]	$[S_0,S_5]$	$t\text{-}_{e4}$	2	0

Tabela 6.1.3.1 – Tabela de variação do número de tokens no comportamento KickOff

T	$\Delta_{kg} = K_g(S_n) - K_g(S_{n-1})$	$\Delta_{kc} = K_c(S_n) - K_c(S_{n-1})$	$\Delta_g + \Delta_c$
$t\text{-}_{e1}$	- 1	0	-1
$t\text{-}_{e2}$	- 1	0	-1
$t\text{-}_{e3}$	- 1	0	-1
$t\text{-}_{e4}$	- 1	0	-1

Tabela 6.1.3.2 – Tabela de variação do número de tokens no comportamento KickOff em casos de erro

9.2.12 Capítulo 6, Secção 1, Ponto 4– Goal Kick

Nomenclatura Utilizada:

$\mathbf{p_1}$	FindPassTrajectory	$\mathbf{p_9}$	StandBy
$\mathbf{p_2}$	Rotate	$\mathbf{p_{10}}$	MovetoXY
$\mathbf{p_3}$	Passing	$\mathbf{p_{11}}$	StandBy
$\mathbf{p_4}$	FindKickTrajectory	$\mathbf{p_{12}}$	Intercept
$\mathbf{p_5}$	Rotate	$\mathbf{p_{13}}$	Finish (receiver)
$\mathbf{p_6}$	kick	$\mathbf{p_{14}}$	Finish Unsuccessful (receiver)
$\mathbf{p_7}$	Finish Successfull(kicker)		
$\mathbf{p_8}$	Finish Unsuccessfull (kicker)		

$\mathbf{t_1}$	Found Pass Trajectory	$\mathbf{t_8}$	Position Received
$\mathbf{t_2}$	Rotate Done	$\mathbf{t_9}$	Arrived at XY
$\mathbf{t_3}$	Pass executed successfully	$\mathbf{t_{10}}$	Got confirmation
$\mathbf{t_4}$	No free Space found	$\mathbf{t_{11}}$	Ball Intercepted Successfully
$\mathbf{t_5}$	Found Kick Trajectory	$\mathbf{t\text{-}_{e1}}$	Transicções insucesso k2,k2,k3,k4,k5,k10
$\mathbf{t_6}$	Rotate Done	$\mathbf{t\text{-}_{e2}}$	Transicções insucesso k6,k7,k8
$\mathbf{t_7}$	Ball kicked Successfully	$\mathbf{t\text{-}_{e3}}$	Transicções insucesso R1,R2,R3,R4
		$\mathbf{t\text{-}_{e4}}$	Transicções insucesso R5,R6,R7,R8

S_n:	$[p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ p_8\ p_9\ p_{10}\ p_{11}\ p_{12}\ p_{13}\ p_{14}]$	S_{n-1}	T	$K_c(S_n)$	$K_c(S_n) - K_c(S_{n-1})$
S_0	[1 0 0 0 0 0 0 0 1 0 0 0 0 0]	-	-	2	-
S_1	[0 1 0 0 0 0 0 0 1 0 0 0 0 0]	S_0	t_1	2	0
S_2	[0 1 0 0 0 0 0 0 0 1 0 0 0 0]	S_1	t_8	2	0
S_3	[0 1 0 0 0 0 0 0 0 0 1 0 0 0]	S_2	t_9	2	0
S_4	[0 0 1 0 0 0 0 0 0 0 1 0 0 0]	S_3	t_2	2	0
S_5	[0 0 0 0 0 0 1 0 0 0 1 0 0 0]	S_4	t_3	2	0
S_6	[0 0 0 0 0 0 1 0 0 0 0 1 0 0]	S_5	t_{10}	2	0
S_7	[0 0 0 0 0 0 1 0 0 0 0 0 1 0]	S_6	t_{11}	2	0
S_8	[0 0 0 1 0 0 0 0 1 0 0 0 0 0]	S_0	t_4	2	0
S_9	[0 0 0 0 1 0 0 0 1 0 0 0 0 0]	S_8	t_5	2	0
S_{10}	[0 0 0 0 0 1 0 0 1 0 0 0 0 0]	S_9	t_6	2	0
S_{11}	[0 0 0 0 0 0 1 0 0 0 0 0 0 1]	S_{10}	t_7	2	0
$S\text{-}_{e1}$	[0 0 0 0 0 0 0 1- - - - - -]	$[S_0,S_{10}] / S_7$	$t\text{-}_{e1}$	2	0
$S\text{-}_{e2}$	[0 0 0 0 0 0 0 1- - - - - -]	$[S_0,S_{10}] / S_7$	$t\text{-}_{e2}$	2	0
$S\text{-}_{e3}$	[- - - - - - - -0 0 0 0 0 1]	$[S_0,S_{10}] / S_7$	$t\text{-}_{e3}$	2	0
$S\text{-}_{e4}$	[- - - - - - - -0 0 0 0 0 1]	$[S_0,S_{10}] / S_7$	$t\text{-}_{e4}$	2	0

Tabela 6.1.4.1 – Tabela de variação do número de tokens no comportamento GoalKick

T	$\Delta_g = K_g(S_n) - K_g(S_{n-1})$	$\Delta_c = K_c(S_n) - K_c(S_{n-1})$	$\Delta_g + \Delta_c$
$t\text{-}_{e1}$	- 1	0	- 1
$t\text{-}_{e2}$	- 1	0	- 1
$t\text{-}_{e3}$	- 1	0	- 1
$t\text{-}_{e4}$	- 1	0	- 1

Tabela 6.1.4.2 – Tabela de variação do número de tokens no comportamento GoalKick em casos de erro.

9.2.13 Capítulo 6, Secção 1, Ponto 5– Throw In

Nomenclatura Utilizada:

$\mathbf{p_1}$	FindPassTrajectory	$\mathbf{p_9}$	StandBy
$\mathbf{p_2}$	Rotate	$\mathbf{p_{10}}$	MovetoXY
$\mathbf{p_3}$	Passing	$\mathbf{p_{11}}$	StandBy
$\mathbf{p_4}$	FindKickTrajectory	$\mathbf{p_{12}}$	Intercept
$\mathbf{p_5}$	Rotate	$\mathbf{p_{13}}$	Finish (receiver)
$\mathbf{p_6}$	kick	$\mathbf{p_{14}}$	Finish Unsuccessful (receiver)
$\mathbf{p_7}$	Finish Successfull(kicker)		
$\mathbf{p_8}$	Finish Unsuccessfull (kicker)		

$\mathbf{t_1}$	Found Pass Trajectory	$\mathbf{t_8}$	Position Received
$\mathbf{t_2}$	Finished Rotate	$\mathbf{t_9}$	Arrived at XY
$\mathbf{t_3}$	Pass executed successfully	$\mathbf{t_{10}}$	Got confirmation
$\mathbf{t_4}$	No free Space found	$\mathbf{t_{11}}$	Ball Intercepted Successfully
$\mathbf{t_5}$	Found Kick Trajectory	$\mathbf{t\text{-}_{e1}}$	Transicções insucesso k2,k2,k3,k4,k5,k10
$\mathbf{t_6}$	Finished Rotate	$\mathbf{t\text{-}_{e2}}$	Transicções insucesso k6,k7,k8
$\mathbf{t_7}$	Ball kicked Successfully	$\mathbf{t\text{-}_{e3}}$	Transicções insucesso R1,R2,R3,R4
		$\mathbf{t\text{-}_{e4}}$	Transicções insucesso R5,R6,R7,R8

S_n:	$[p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ p_8\ p_9\ p_{10}\ p_{11}\ p_{12}\ p_{13}\ p_{14}]$	S_{n-1}	T	$K_c(S_n)$	$K_c(S_n) - K_c(S_{n-1})$
S_0	[1 0 0 0 0 0 0 0 1 0 0 0 0 0]	-	-	2	-
S_1	[0 1 0 0 0 0 0 0 1 0 0 0 0 0]	S_0	t_1	2	0
S_2	[0 1 0 0 0 0 0 0 0 1 0 0 0 0]	S_1	t_8	2	0
S_3	[0 1 0 0 0 0 0 0 0 0 1 0 0 0]	S_2	t_9	2	0
S_4	[0 0 1 0 0 0 0 0 0 0 1 0 0 0]	S_3	t_2	2	0
S_5	[0 0 0 0 0 0 1 0 0 0 1 0 0 0]	S_4	t_3	2	0
S_6	[0 0 0 0 0 0 1 0 0 0 0 1 0 0]	S_5	t_{10}	2	0
S_7	[0 0 0 0 0 0 1 0 0 0 0 0 1 0]	S_6	t_{11}	2	0
S_8	[0 0 0 1 0 0 0 0 1 0 0 0 0 0]	S_0	t_4	2	0
S_9	[0 0 0 0 1 0 0 0 1 0 0 0 0 0]	S_8	t_5	2	0
S_{10}	[0 0 0 0 0 1 0 0 1 0 0 0 0 0]	S_9	t_6	2	0
S_{11}	[0 0 0 0 0 0 1 0 0 0 0 0 0 1]	S_{10}	t_7	2	0
$S\text{-}_{e1}$	[0 0 0 0 0 0 0 1- - - - - -]	$[S_0,S_{10}] / S_7$	$t\text{-}_{e1}$	2	0
$S\text{-}_{e2}$	[0 0 0 0 0 0 0 1- - - - - -]	$[S_0,S_{10}] / S_7$	$t\text{-}_{e2}$	2	0
$S\text{-}_{e3}$	[- - - - - - - -0 0 0 0 0 1]	$[S_0,S_{10}] / S_7$	$t\text{-}_{e3}$	2	0
$S\text{-}_{e4}$	[- - - - - - - -0 0 0 0 0 1]	$[S_0,S_{10}] / S_7$	$t\text{-}_{e4}$	2	0

Tabela 6.1.5.1 – Tabela de variação do número de tokens no comportamento Throw In

T	$\Delta_g = K_g(S_n) - K_g(S_{n-1})$	$\Delta_c = K_c(S_n) - K_c(S_{n-1})$	$\Delta_g + \Delta_c$
$t\text{-}_{e1}$	- 1	0	- 1
$t\text{-}_{e2}$	- 1	0	- 1
$t\text{-}_{e3}$	- 1	0	- 1
$t\text{-}_{e4}$	- 1	0	- 1

Tabela 6.1.5.2 – Tabela de variação do número de tokens no comportamento Throw In em casos de erro

9.2.14 Capítulo 6, Secção 1, Ponto 6– One Two Pass

Nomenclatura Utilizada:

p_1	FindPassTrajectory	p_{10}	StandBy
p_2	Rotating	p_{11}	MovetoXY
p_3	Passing	p_{12}	StandBy
p_4	MoveToXY	p_{13}	Intercepting
p_5	StandBy	p_{14}	FindPassTrajectory
p_6	StandBy	p_{15}	Rotating
p_7	Intercepting	p_{16}	Passing
p_8	FinishSuccessfull(kicker)	p_{17}	FinishSuccessfull (receiver)
p_9	FinishUnsuccessfull (kicker)	p_{18}	FinishUnSuccessfull (receiver)

t_1	Found Pass Trajectory	t_8	Position Received
t_2	Finished Rotating	t_9	Arrived at position <XY>
t_3	Pass executed successfully	t_{10}	Got confirmation
t_4	Arrived at position <XY>	t_{11}	Ball Intercepted Successfully
t_5	Continue in StandBy	t_{12}	Found Pass Trajectory
t_6	Got Confirmation	t_{13}	Finished Rotating
t_7	Ball intercepted Successfully	t_{14}	The passa was executed successfully
		$t\text{-}_{e1}$	Transicções insucesso k2,k2,k3,k4,k5,k10
		$t\text{-}_{e2}$	Transicções insucesso k6,k7,k8
		$t\text{-}_{e3}$	Transicções insucesso R1,R2,R3,R4
		$t\text{-}_{e4}$	Transicções insucesso R5,R6,R7,R8

S_n:	$[p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ p_8\ p_9\ p_{10}\ p_{11}\ p_{12}\ p_{13}\ p_{14}\ p_{15}\ p_{16}\ p_{17}\ p_{18}]$	S_{n-1}	T	$K_c(S_n)$	$K_c(S_n) - K_c(S_{n-1})$
S_0	[1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]	-	-	2	-
S_1	[0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]	S_0	t_1	2	0
S_2	[0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]	S_1	t_8	2	0
S_3	[0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]	S_2	t_9	2	0
S_4	[0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]	S_3	t_2	2	0
S_5	[0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0]	S_4	t_3	2	0
S_6	[0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0]	S_5	t_{10}	2	0
S_7	[0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0]	S_6	t_4	2	0
S_8	[0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0]	S_7	t_{11}	2	0
S_9	[0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0]	S_8	t_{12}	2	0
S_{10}	[0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0]	S_9	t_5	2	0
S_{11}	[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0]	S_{10}	t_{13}	2	0
S_{12}	[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0]	S_{11}	t_{14}	2	0
S_{13}	[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0]	S_{12}	t_6	2	0
S_{14}	[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0]	S_{13}	t_7	2	0
$S\text{-}_{e1}$	[0 0 0 0 0 0 0 0 1- - - - - -]	$[S_0,S_{13}]$	$t\text{-}_{e1}$	2	0
$S\text{-}_{e2}$	[0 0 0 0 0 0 0 0 1- - - - - -]	$[S_0,S_{13}]$	$t\text{-}_{e2}$	2	0
$S\text{-}_{e3}$	[- - - - - - - -0 0 0 0 0 0 0 0 1]	$[S_0,S_{13}]$	$t\text{-}_{e3}$	2	0
$S\text{-}_{e4}$	[- - - - - - - -0 0 0 0 0 0 0 0 1]	$[S_0,S_{13}]$	$t\text{-}_{e4}$	2	0

Tabela 6.1.6.1 – Tabela de variação do número de tokens no comportamento OneTwoPass

T	$\Delta_g = K_g(S_n) - K_g(S_{n-1})$	$\Delta_c = K_c(S_n) - K_c(S_{n-1})$	$\Delta_g + \Delta_c$
$t\text{-}_{e1}$	- 1	0	- 1
$t\text{-}_{e2}$	- 1	0	- 1
$t\text{-}_{e3}$	- 1	0	- 1
$t\text{-}_{e4}$	- 1	0	- 1

Tabela 6.1.6.1 – Tabela de variação do número de tokens no comportamento OneTwoPass em casos de erro

9.2.15 Capítulo 6, Secção 2, Ponto 1, Tabela Nº1 - PLAYER.LOG

Comportamento Throwin – Execução Principal - `plGetBasicDecision`	
Robot Kicker	**Robot Receiver**
player_role -- INICIO	player_role -- INICIO
player_role -- CURRENTMODE? PLAY	player_role -- CURRENTMODE? PLAY
player_role -- DECISAO? 21	player_role -- DECISAO? 21
player_role -- LAST STATE? 21	player_role -- LAST STATE? 21
player_role -- FIM	player_role -- FIM
plGetState - INICIO	plGetState - INICIO
plGetState - GET_STATE: findpasstrajectory	plGetState - GET_STATE: movetoxy
plGetState - GET_STATE: 14	plGetState - GET_STATE: 22
plGetState - FIM	plGetState - FIM
player_role -- INICIO	player_role -- INICIO
player_role -- CURRENTMODE? play	player_role -- CURRENTMODE? play
player_role -- DECISAO? 14 findpasstrajectory	player_role -- DECISAO? 22 movetoxy
player_role -- LAST STATE? 21	player_role -- LAST STATE? 21
player_role -- FIM	player_role -- FIM
plGetState - INICIO	plGetState - INICIO
plGetState - GET_STATE: rodar	plGetState - GET_STATE: standby
plGetState - GET_STATE: 15	plGetState - GET_STATE: 21
plGetState - FIM	plGetState - FIM
player_role -- INICIO	player_role -- INICIO
player_role -- CURRENTMODE? play	player_role -- CURRENTMODE? play
player_role -- DECISAO? 15 Rodar	player_role -- DECISAO? 21 standby
player_role -- LAST STATE? 14	player_role -- LAST STATE? 22
player_role -- FIM	player_role -- FIM
plGetState - INICIO	plGetState - INICIO
plGetState - GET_STATE: passing	plGetState - GET_STATE: intercepting

plGetState - GET_STATE: 16	plGetState - GET_STATE: 23
plGetState - FIM	plGetState - FIM
player_role -- INICIO	player_role -- INICIO
player_role -- CURRENTMODE? play	player_role -- CURRENTMODE? play
player_role -- DECISAO? 16 passing	player_role -- DECISAO? 23 intercepting
player_role -- LAST STATE? 15	player_role -- LAST STATE? 21
player_role -- FIM	player_role -- FIM
plGetState - INICIO	plGetState - INICIO
plGetState - GET_STATE: finishsuccessful	plGetState - GET_STATE: finishsuccessful
plGetState - GET_STATE: 17	plGetState - GET_STATE: 17
plGetState - FIM	plGetState - FIM
player_role -- INICIO	player_role -- INICIO
player_role -- CURRENTMODE? play	player_role -- CURRENTMODE? play
player_role -- DECISAO? 17	player_role -- DECISAO? 17
player_role -- LAST STATE? 16	player_role -- LAST STATE? 23
player_role -- DECISION calculada 17 finishsuccessful	player_role -- DECISION calculada 17 finishsuccessful
player_role -- LASTSTATE 16	player_role -- LASTSTATE 23
player_role -- FIM	player_role -- FIM
plGetState - INICIO	plGetState - INICIO
plGetState - FINISHSUCCESS	plGetState - FINISHSUCCESS
plGetState - GET_STATE: finishsuccessful	plGetState - GET_STATE: finishsuccessful
plGetState - GET_STATE: 17	plGetState - GET_STATE: 17
plGetState - FIM	plGetState - FIM

Tabela 6.2.1.1 – Tabela de "`check strings`" relativa ao ficheiro PLAYER.LOG

9.2.16 Capítulo 6, Secção 2, Ponto 2, Tabela Nº1 - BASICUNIT.LOG

Comportamento Throwin – Execução Principal - `makeBasicDecision`	
Robot Kicker	**Robot Receiver**
ExecutandoDecisaoBasica -> inicio_decisao	fazendoDecisaoBasica -> inicio_decisao
ExecutandoDecisaoBasica -> defender	ExecutandoDecisaoBasica -> attacker
ExecutandoDecisaoBasica_Commitment -> none	ExecutandoDecisaoBasica_Commitment -> none
ExecutandoDecisaoBasica_NewRole -> defender	ExecutandoDecisaoBasica_NewRole -> attacker
basicBehaviour_standby -> standby	basicBehaviour_standby -> standby
ExecutandoDecisaoBasica_behaviour -> standby	ExecutandoDecisaoBasica_behaviour -> standby
ExecutandoDecisaoBasica -> inicio_encodeState	ExecutandoDecisaoBasica -> inicio_encodeState
ExecutandoDecisaoBasica -> standby	ExecutandoDecisaoBasica -> standby
ExecutandoDecisaoBasica -> fim_decisao	ExecutandoDecisaoBasica -> fim_decisao
ExecutandoDecisaoBasica -> inicio_decisao	ExecutandoDecisaoBasica -> inicio_decisao
ExecutandoDecisaoBasica -> defender	ExecutandoDecisaoBasica -> attacker
ExecutandoDecisaoBasica_Commitment -> throwin	ExecutandoDecisaoBasica_Commitment -> throwin
ExecutandoDecisaoBasica_NewRole -> defender	ExecutandoDecisaoBasica_NewRole -> attacker
basicBehaviour_FindPass -> inicio_FindPass	basicBehaviour_FindPass -> inicio_FindPass
basicBehaviour_FINDPASS -> ph	basicBehaviour_Rodar -> inicio_rodar
basicBehaviour_FINDPASS -> passei_plSendCommSync	basicBehaviour_RODAR -> bp
basicBehaviour_FindPass -> fim_FindPass	basicBehaviour_passing -> inicio_passing
ExecutandoDecisaoBasica_behaviour -> findpasstrajectory	basicBehaviour_passing -> passei_plCheckCommSync
ExecutandoDecisaoBasica -> inicio_encodeState	basicBehaviour_finishsuccessful -> inicio_finishsuccessful
ExecutandoDecisaoBasica -> findpasstrajectory	basicBehaviour_standby -> inicio_standby_PREP
ExecutandoDecisaoBasica -> fim_decisao	basicBehaviour_standby -> inicio_standby_PASS
ExecutandoDecisaoBasica -> inicio_decisao	basicBehaviour_movetoxy -> inicio_movetoXY
ExecutandoDecisaoBasica -> defender	basicBehaviour_movetoxy -> passei_plCheckCommSync
ExecutandoDecisaoBasica_Commitment -> throwin	basicBehaviour_MOVETOXY -> bp
ExecutandoDecisaoBasica_NewRole -> defender	basicBehaviour_movetoxy -> passei_plSendCommSync
basicBehaviour_FindPass -> inicio_FindPass	basicBehaviour_movetoxy -> fim_movetoxy
basicBehaviour_Rodar -> inicio_rodar	ExecutandoDecisaoBasica_behaviour -> movetoxy
basicBehaviour_RODAR -> ph	ExecutandoDecisaoBasica -> inicio_encodeState
basicBehaviour_rodar -> fim_rodar	ExecutandoDecisaoBasica -> movetoxy
ExecutandoDecisaoBasica_behaviour -> rodar	ExecutandoDecisaoBasica -> fim_decisao

ExecutandoDecisaoBasica -> inicio_encodeState	ExecutandoDecisaoBasica -> inicio_decisao
ExecutandoDecisaoBasica -> rodar	ExecutandoDecisaoBasica -> attacker
ExecutandoDecisaoBasica -> fim_decisao	ExecutandoDecisaoBasica_Commitment -> throwin
ExecutandoDecisaoBasica -> inicio_decisao	ExecutandoDecisaoBasica_NewRole -> attacker
ExecutandoDecisaoBasica -> defender	basicBehaviour_FindPass -> inicio_FindPass
ExecutandoDecisaoBasica_Commitment -> throwin	basicBehaviour_Rodar -> inicio_rodar
ExecutandoDecisaoBasica_NewRole -> defender	basicBehaviour_passing -> inicio_passing
basicBehaviour_FindPass -> inicio_FindPass	basicBehaviour_passing -> passei_plCheckCommSync
basicBehaviour_Rodar -> inicio_rodar	basicBehaviour_PASS -> bp
basicBehaviour_passing -> inicio_passing	basicBehaviour_finishsuccessful -> inicio_finishsuccessful
basicBehaviour_passing -> passei_plCheckCommSync	basicBehaviour_standby -> inicio_standby_PREP
basicBehaviour_PASS -> ph	basicBehaviour_standby -> inicio_standby_PASS
basicBehaviour_passing -> passei_plSendCommSync	basicBehaviour_STANDBY_A -> bp
basicBehaviour_passing -> fim_passing	basicBehaviour_standby -> fim_standby_PASS
ExecutandoDecisaoBasica_behaviour -> passing	ExecutandoDecisaoBasica_behaviour -> standby
ExecutandoDecisaoBasica -> inicio_encodeState	ExecutandoDecisaoBasica -> inicio_encodeState
ExecutandoDecisaoBasica -> passing	ExecutandoDecisaoBasica -> standby
ExecutandoDecisaoBasica -> fim_decisao	ExecutandoDecisaoBasica -> fim_decisao
ExecutandoDecisaoBasica -> inicio_decisao	ExecutandoDecisaoBasica -> inicio_decisao
ExecutandoDecisaoBasica -> defender	ExecutandoDecisaoBasica -> attacker
ExecutandoDecisaoBasica_Commitment -> throwin	ExecutandoDecisaoBasica_Commitment -> throwin
ExecutandoDecisaoBasica_NewRole -> defender	ExecutandoDecisaoBasica_NewRole -> attacker
basicBehaviour_FindPass -> inicio_FindPass	basicBehaviour_FindPass -> inicio_FindPass
basicBehaviour_Rodar -> inicio_rodar	basicBehaviour_Rodar -> inicio_rodar
basicBehaviour_passing -> inicio_passing	basicBehaviour_passing -> inicio_passing
basicBehaviour_passing -> passei_plCheckCommSync	basicBehaviour_passing -> passei_plCheckCommSync
basicBehaviour_finishsuccessful -> inicio_finishsuccessful	basicBehaviour_finishsuccessful -> inicio_finishsuccessful
basicBehaviour_FINISHSUCCESS -> ph	basicBehaviour_standby -> inicio_standby_PREP
basicBehaviour_finishsuccessful -> fim_finishsuccessful	basicBehaviour_standby -> inicio_standby_PASS
ExecutandoDecisaoBasica_behaviour -> finishsuccessful	basicBehaviour_movetoxy -> inicio_movetoXY
ExecutandoDecisaoBasica -> inicio_encodeState	basicBehaviour_movetoxy -> passei_plCheckCommSync
ExecutandoDecisaoBasica -> finishsuccessful	basicBehaviour_intercepting -> inicio_intercepting
ExecutandoDecisaoBasica -> fim_decisao	basicBehaviour_movetoxy -> passei_plCheckCommSync
ExecutandoDecisaoBasica -> inicio_decisao	basicBehaviour_INTERCEPT -> bp
ExecutandoDecisaoBasica -> defender	basicBehaviour_intercepting -> fim_intercepting

ExecutandoDecisaoBasica_Commitment -> none	ExecutandoDecisaoBasica_behaviour -> intercepting ExecutandoDecisaoBasica -> inicio_encodeState ExecutandoDecisaoBasica -> intercepting ExecutandoDecisaoBasica -> fim_decisao ExecutandoDecisaoBasica -> inicio_decisao ExecutandoDecisaoBasica -> attacker ExecutandoDecisaoBasica_Commitment -> throwin ExecutandoDecisaoBasica_NewRole -> attacker basicBehaviour_FindPass -> inicio_FindPass basicBehaviour_Rodar -> inicio_rodar basicBehaviour_passing -> inicio_passing basicBehaviour_passing -> passei_plCheckCommSync basicBehaviour_finishsuccessful -> inicio_finishsuccessful basicBehaviour_FINISHSUCCESS -> bp basicBehaviour_standby -> inicio_standby_PREP basicBehaviour_standby -> inicio_standby_PASS basicBehaviour_movetoxy -> inicio_movetoXY basicBehaviour_movetoxy -> passei_plCheckCommSync basicBehaviour_intercepting -> inicio_intercepting basicBehaviour_movetoxy -> passei_plCheckCommSync basicBehaviour_finishsuccessful -> inicio_finishsuccessful basicBehaviour_FINISHSUCCESS -> bp basicBehaviour_finishsuccessful -> fim_finishsuccessful ExecutandoDecisaoBasica_behaviour -> finishsuccessful ExecutandoDecisaoBasica -> inicio_encodeState ExecutandoDecisaoBasica -> finishsuccessful ExecutandoDecisaoBasica -> fim_decisao ExecutandoDecisaoBasica -> inicio_decisao ExecutandoDecisaoBasica -> attacker ExecutandoDecisaoBasica_Commitment -> none

Tabela 6.2.2.1 – Tabela de "`check strings`" relativa ao ficheiro BASICUNIT.LOG

9.2.17 Capítulo 6, Secção 2, Ponto 3, Tabela Nº1 - CONTROL.LOG

Comportamento Throwin – Execução Principal – CONTROL.LOG	
Robot Kicker	**Robot Receiver**
controlMALoop INICIO	controlMALoop INICIO
controlPlugins[i]->id takeball2goal	controlPlugins[i]->id takeball2goal
numero de plugins 17	numero de plugins 17
request standby	request standby
controlPlugins[i]->id standby	controlPlugins[i]->id standby
numero de plugins 17	numero de plugins 17
request standby	request standby
CONTROL-changeto controlPlugin[i]: 1	CONTROL-changeto controlPlugin[i]: 1
CONTROL-changeto controlPlugin[i]-id: standby	CONTROL-changeto controlPlugin[i]-id: standby
CONTROL-METHOD controlPlugin[i]: 1	CONTROL-METHOD controlPlugin[i]: 1
CONTROL-METHOD controlPlugin[i]-id: standby	CONTROL-METHOD controlPlugin[i]-id: standby
controlPlugins[i]->id standby	controlPlugins[i]->id standby
numero de plugins 17	numero de plugins 17
request findpasstrajectory	request movetoxy
controlPlugins[i]->id selflocalize	controlPlugins[i]->id selflocalize
numero de plugins 17	numero de plugins 17
request findpasstrajectory	request movetoxy
controlPlugins[i]->id rodar	controlPlugins[i]->id rodar
numero de plugins 17	numero de plugins 17
request findpasstrajectory	request movetoxy
controlPlugins[i]->id passing	controlPlugins[i]->id passing
numero de plugins 17	numero de plugins 17
request findpasstrajectory	request movetoxy
controlPlugins[i]->id nothing	controlPlugins[i]->id nothing
numero de plugins 17	numero de plugins 17
request findpasstrajectory	request movetoxy
controlPlugins[i]->id movetoxy	controlPlugins[i]->id movetoxy
numero de plugins 17	numero de plugins 17
request findpasstrajectory	request movetoxy

controlPlugins[i]->id kick	CONTROL-changeto controlPlugin[i]: 7
numero de plugins 17	CONTROL-changeto controlPlugin[i]-id: movetoxy
request findpasstrajectory	CONTROL-METHOD controlPlugin[i]: 7
controlPlugins[i]->id intercepting	CONTROL-METHOD controlPlugin[i]-id: movetoxy
numero de plugins 17	controlPlugins[i]->id kick
request findpasstrajectory	numero de plugins 17
controlPlugins[i]->id halt	request standby
numero de plugins 17	controlPlugins[i]->id intercepting
request findpasstrajectory	numero de plugins 17
controlPlugins[i]->id getclose2ball	request standby
numero de plugins 17	controlPlugins[i]->id halt
request findpasstrajectory	numero de plugins 17
controlPlugins[i]->id finishunsuccessful	request standby
numero de plugins 17	controlPlugins[i]->id getclose2ball
request findpasstrajectory	numero de plugins 17
controlPlugins[i]->id finishsuccessful	request standby
numero de plugins 17	controlPlugins[i]->id finishunsuccessful
request findpasstrajectory	numero de plugins 17
controlPlugins[i]->id findpasstrajectory	request standby
numero de plugins 17	controlPlugins[i]->id finishsuccessful
request findpasstrajectory	numero de plugins 17
CONTROL-changeto controlPlugin[i]: 14	request standby
CONTROL-changeto controlPlugin[i]-id: findpasstrajectory	controlPlugins[i]->id findpasstrajectory
CONTROL-METHOD controlPlugin[i]: 14	numero de plugins 17
CONTROL-METHOD controlPlugin[i]-id: findpasstrajectory	request standby
controlPlugins[i]->id findkicktrajectory	controlPlugins[i]->id findkicktrajectory
numero de plugins 17	numero de plugins 17
request rodar	request standby
controlPlugins[i]->id clearball	controlPlugins[i]->id clearball
numero de plugins 17	numero de plugins 17
request rodar	request standby
controlPlugins[i]->id takeball2goal	controlPlugins[i]->id takeball2goal
numero de plugins 17	numero de plugins 17
request rodar	request standby
controlPlugins[i]->id standby	controlPlugins[i]->id standby

numero de plugins 17	numero de plugins 17
request rodar	request standby
controlPlugins[i]->id standby	CONTROL-changeto controlPlugin[i]: 1
numero de plugins 17	CONTROL-changeto controlPlugin[i]-id: standby
request rodar	CONTROL-METHOD controlPlugin[i]: 1
controlPlugins[i]->id selflocalize	CONTROL-METHOD controlPlugin[i]-id: standby
numero de plugins 17	controlPlugins[i]->id standby
request rodar	numero de plugins 17
controlPlugins[i]->id rodar	request intercepting
numero de plugins 17	controlPlugins[i]->id selflocalize
request rodar	numero de plugins 17
CONTROL-changeto controlPlugin[i]: 4	request intercepting
CONTROL-changeto controlPlugin[i]-id: rodar	controlPlugins[i]->id rodar
CONTROL-METHOD controlPlugin[i]: 4	numero de plugins 17
CONTROL-METHOD controlPlugin[i]-id: rodar	request intercepting
controlPlugins[i]->id passing	controlPlugins[i]->id passing
numero de plugins 17	numero de plugins 17
request passing	request intercepting
CONTROL-changeto controlPlugin[i]: 5	controlPlugins[i]->id nothing
CONTROL-changeto controlPlugin[i]-id: passing	numero de plugins 17
CONTROL-METHOD controlPlugin[i]: 5	request intercepting
CONTROL-METHOD controlPlugin[i]-id: passing	controlPlugins[i]->id movetoxy
controlPlugins[i]->id nothing	numero de plugins 17
numero de plugins 17	request intercepting
request finishsuccessful	controlPlugins[i]->id kick
controlPlugins[i]->id movetoxy	numero de plugins 17
numero de plugins 17	request intercepting
request finishsuccessful	controlPlugins[i]->id intercepting
controlPlugins[i]->id kick	numero de plugins 17
numero de plugins 17	request intercepting
request finishsuccessful	CONTROL-changeto controlPlugin[i]: 9
controlPlugins[i]->id intercepting	CONTROL-changeto controlPlugin[i]-id: intercepting
numero de plugins 17	CONTROL-METHOD controlPlugin[i]: 9
request finishsuccessful	CONTROL-METHOD controlPlugin[i]-id: intercepting
controlPlugins[i]->id halt	controlPlugins[i]->id halt

numero de plugins 17	numero de plugins 17
request finishsuccessful	request finishsuccessful
controlPlugins[i]->id getclose2ball	controlPlugins[i]->id getclose2ball
numero de plugins 17	numero de plugins 17
request finishsuccessful	request finishsuccessful
controlPlugins[i]->id finishunsuccessful	controlPlugins[i]->id finishunsuccessful
numero de plugins 17	numero de plugins 17
request finishsuccessful	request finishsuccessful
controlPlugins[i]->id finishsuccessful	controlPlugins[i]->id finishsuccessful
numero de plugins 17	numero de plugins 17
request finishsuccessful	request finishsuccessful
CONTROL-changeto controlPlugin[i]: 13	CONTROL-changeto controlPlugin[i]: 13
CONTROL-changeto controlPlugin[i]-id: finishsuccessful	CONTROL-changeto controlPlugin[i]-id: finishsuccessful
CONTROL-METHOD controlPlugin[i]: 13	CONTROL-METHOD controlPlugin[i]: 13
CONTROL-METHOD controlPlugin[i]-id: finishsuccessful	CONTROL-METHOD controlPlugin[i]-id: finishsuccessful
controlPlugins[i]->id findpasstrajectory	controlPlugins[i]->id findpasstrajectory
numero de plugins 17	numero de plugins 17
request standby	request standby
controlPlugins[i]->id findkicktrajectory	controlPlugins[i]->id findkicktrajectory
numero de plugins 17	numero de plugins 17
request standby	request standby
controlPlugins[i]->id clearball	controlPlugins[i]->id clearball
numero de plugins 17	numero de plugins 17
request standby	request standby
controlPlugins[i]->id takeball2goal	controlPlugins[i]->id takeball2goal
numero de plugins 17	numero de plugins 17
request standby	request standby
controlPlugins[i]->id standby	controlPlugins[i]->id standby
numero de plugins 17	numero de plugins 17
request standby	request standby
	CONTROL-changeto controlPlugin[i]: 1
	CONTROL-changeto controlPlugin[i]-id: standby
	CONTROL-METHOD controlPlugin[i]: 1
	CONTROL-METHOD controlPlugin[i]-id: standby

Tabela 6.2.3.1 – Tabela de "`check strings`" relativa ao ficheiro CONTROL.LOG

9.2.18 Capítulo 6, Secção 2, Ponto 3, Tabela Nº2 - FINDPASSTRAJECTORY.LOG

Comportamento Throwin – Execução Principal – FINDPASSTRAJECTORY.LOG
Robot Kicker
INIT FINDPASSTRAJECTORY-KICKER
INIT-JOGADOR EM CAUSA: ph
CHANGETO FINDPASSTRAJECTORY
CHANGETO - JOGADOR EM CAUSA: ph
CHANGETO - Initialization-KICKER
ENTERED METHOD
METHOD-JOGADOR EM CAUSA: ph
SWITCH-KICKER-FINAL FINDPASSTRAJECTORY

Tabela 6.2.3.2 – Tabela de "check strings" relativa ao ficheiro FINDPASSTRAJECTORY.LOG

9.3 C : FIGURAS

9.3.1 COMPORTAMENTO RELACIONAL THROWIN – CASO TOTAL

9.3.2 COMPORTAMENTO RELACIONAL THROWIN – CASO SUCESSO

9.3.3 COMPORTAMENTO RELACIONAL THROWIN – CASO INSUCESSO

9.3.4 COMPORTAMENTO RELACIONAL GOAL KICK – CASO TOTAL

9.3.5 COMPORTAMENTO RELACIONAL GOAL KICK – CASO SUCESSO

9.3.6 COMPORTAMENTO RELACIONAL GOAL KICK – CASO INSUCESSO

9.3.7 COMPORTAMENTO RELACIONAL KICK OFF – CASO TOTAL

9.3.8 COMPORTAMENTO RELACIONAL KICK OFF – CASO SUCESSO

9.3.9 COMPORTAMENTO RELACIONAL KICK OFF – CASO INSUCESSO

9.3.10 COMPORTAMENTO RELACIONAL ONETWOPASS – CASO TOTAL

9.3.11 COMPORTAMENTO RELACIONAL ONETWOPASS – CASO SUCESSO

9.3.12 COMPORTAMENTO RELACIONAL ONETWOPASS – CASO INSUCESSO