Departamento de Engenharia Informática

Relatório de TRABALHO FINAL DE CURSO

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

Ano Lectivo 2004 / 2005

N.º da Proposta: 176b

Título: Comportamentos Relacionais para Robots Futebolistas

Professor Orientador:

Pedro Lima

_____(assinatura)_____

Aluno:

49766, Vasco Maria d'Athouguia de Albuquerque d'Orey

_____(assinatura)_____

Agradecimentos

Queria agradecer a algumas pessoas que julgo terem sido muito importantes durante a realização deste trabalho.

Um agradecimento muito especial é devido ao Prof. Pedro Lima pela sua invulgar disponibilidade, humanidade e pela capacidade de me motivar durante estes últimos 9 meses. A ele se deverá qualquer qualidade que este trabalho possa apresentar.

A dois ex-membros essenciais, o Gonçalo Mouro Vaz e o João Milhinhos, que sempre que puderam apoiaram, aconselharam e me 'puxaram' para o projecto SocRob. Obrigado aos dois, também pelo 'male-bonding' feito nos últimos 5 anos!

A toda a equipa do projecto SocRob, em especial o Hugo Pereira, João Esteves, Marco Barbosa e Nuno Lopes, que trabalharam mais perto dos AIBOs, por toda a disponibilidade que sempre mostraram e por trabalho, críticas e sugestões que fizeram. Vocês podem fazer desta equipa uma das melhores, basta terem mais tempo!

À Constança por ter suportado estudar e trabalhar comigo, e por ter pressionado a quantidade certa nas alturas certas durante os últimos 5 anos. Muito obrigado!

À minha família, Mãe, Pai, irmãos e cunhadas, Tio Jervis e Tia Sofia, sobrinhos e afilhadas, e Ana, por todo o apoio, sentido de humor e por aturarem pacientemente os meus acessos de mau humor.

À minha outra família, Tia Ana, Tio Luís, Rita, Francisca, Manuel e todos os outros núcleos por toda a amizade e felicidade que transmitem a quem está convosco.

A todos os amigos e outras pessoas que me tenha esquecido, muito obrigado por tudo.

Índice

Índice de figuras

Índice de Tabelas

Índice de Blocos de Código

Diccionário de Siglas

CR – Comportamento relacional

GT – *German Team*

SED – Sistema de Eventos Discretos

1 - Introdução

1.1 – Motivação

As constantes evoluções no mundo e na sociedade levaram ao desenvolvimento da robótica, que, em traços largos, consiste na realização de tarefas habitualmente executadas por homens por agentes, neste caso máquinas, denominados robots.

A utilização da robótica já provou ser um caso de sucesso em aplicações industriais, nomeadamente em linhas de produção e outras funções habitualmente repetitivas, no entanto, desde 1956 [10], tem havido uma evolução da robótica, consequência da evolução da inteligência artificial, que permitiu aos robots serem não apenas executores de tarefas repetitivas, mas poderem decidir por si próprios as acções que poderão tomar em cada circunstância.

Exemplos das aplicações da inteligência artificial que se perfilariam como importante para o homem seria a de dar capacidade aos agentes de ajudarem o em trabalhos que se apresentassem pesados, difíceis ou arriscados, como salvamento de pessoas em situações de acidente ou catástrofe, carregar volumes pesados ou, simplesmente, ajudar na decisão e resolução de problemas diversos em áreas, também elas, diversas e distintas como economia, gestão ou matemática entre muitas outras.

Na última década conseguiu-se provar que agentes autónomos conseguem igualar, ou mesmo superar o homem em alguns campos, como foi o caso da derrota do campeão mundial de Xadrez, Garry Kasparov, contra o *Deep Blue*, um sistema de inteligência artificial criado para jogar Xadrez. O projecto Robocup, no qual este trabalho se insere, tem o objectivo de criar uma equipa de robots humanóides plenamente autónomos que possa levar de vencida a equipa campeã mundial de futebol no ano de 2050.

1.2 – O projecto ISocRob

O projecto ISocRob (*ISR Soccer Robots*), no âmbito do qual este trabalho foi feito, contém duas equipas que participam nas competições organizadas no RoboCup, uma equipa de robots omnidireccionais e uma equipa de robots AIBO.

Uma das principais inovações que o projecto tenta introduzir é a existência de compromissos entre os diversos agentes permitindo uma coordenação entre eles, que será abaixo descrita. Esta característica permite aos agentes, neste caso robots, trabalhar em equipa o que pode apresentar vantagens quer no campo dos robots de salvamento como no campo dos robots futebolistas.

Para além de aproximar o funcionamento dos robots com as sociedades humanas, a existência de comportamentos relacionais poderá contribuir para uma melhoria substancial da competitividade das equipas ISocRob, dado que neste momento não há outras equipas participantes no RoboCup que tenham realizado projectos semelhantes.

1.3 Sumário executivo

O trabalho realizado dividiu-se em 3 fases essenciais:

- Modelação teórica dos comportamentos
- Implementação de comportamentos
- Implementação dos mecanismos de selecção e comunicação de comportamentos.

A primeira fase foi realizada na primeira fase deste projecto. No entanto houve diversas alterações que a tiveram de ser feitas à concepção inicial dos comportamentos por haver algumas impossibilidades na obtenção de informações, tais como as posições dos adversários.

No entanto tentou contornar-se este problema simplificando os comportamentos por forma a serem executáveis com as funcionalidades já existentes.

Quanto à implementação de comportamentos e de mecanismos de comunicação e estabelecimento destes até à data da entrega deste relatório, a sua realização poderá ser consultada na tabela 1.1.

Tarefa	Simulador	Robots reais
Comportamento individual *Sweep*	SIM	SIM
Comportamento individual Maradona	ILEGAL – NÃO PRIORITÁRIO	ILEGAL – NÃO PRIORITÁRIO
Comportamento individual *Clear The Ball*	NÃO	NÃO
Comportamento relacional *Attacking Kickoff*	SIM	SIM
Comportamento relacional *Defensive Kickoff*	SIM	SIM
Comportamento relacional *Defensive Throwin*	NÃO	NÃO
Comportamento relacional *Attacking Throwin*	SIM	SIM
Escolha do comportamento a fazer	SIM	SIM
Selecção de companheiro para CR	SIM	SIM
Exclusão mútua	SIM	NÃO
Aceitação / Recusa de comportamentos	SIM	NÃO
Quebra de compromissos	SIM	NÃO

Tabela 1.1 – Tarefas realizadas ou incompletas

As razões pelas quais alguns comportamentos não terão sido implementados serão descritas no anexo A. A descrição das funcionalidades será feita no capítulo 4 e as razões pelas quais não foram implementadas serão também aí explicadas, assim como os testes a elas feitos, descritos nos capítulos 5 e 6.

1.4 – Organização do trabalho

Neste documento irá ser encontrada a informação sobre os formalismos e métodos utilizados assim como a descrição do trabalho que foi feito.

No capítulo seguinte irão ser explicadas as ferramentas de modelação e o formalismo utilizado para a descrição dos comportamentos relacionais.

Em seguida as arquitecturas envolvidas no projecto serão descritas assim como uma possível correspondência entre a arquitectura da equipa alemã, campeã das últimas duas edições do RoboCup, e a arquitectura ISocRob.

No Capítulo 5 será exposto todo o trabalho e mecanismos implementados para conseguir a aproximação entre as duas arquitecturas, assim como os comportamentos relacionais definidos.

Por fim serão exibidos os testes realizados, assim como o seu resultado e as razões para as suas falhas. Os pontos que merecem especial atenção e que podem ser problemáticos no futuro irão ser, também, referidos nesta secção.

2 – Conceitos chave, notação e modelação

Como foi dito acima, um dos grandes objectivos do trabalho é a apresentação de um formalismo, já existente, e especificação de uma arquitectura que permita a cooperação entre elementos da equipa. A este formalismo dá-se o nome de comportamento relacional e será descrito neste capítulo.

2.1 – Motivação para comportamentos relacionais

Suponhamos que duas pessoas, a Maria e o Pedro, pretendem comer uns brigadeiros durante a tarde.

A receita para brigadeiros é simples, aquecem-se 2 latas de leite condensado durante 2 horas em água a ferver, retira-se a massa, deixa-se arrefecer, de seguida aquece-se no fogão uma panela com alguma manteiga e chocolate em pó. Deixa-se arrefecer novamente e enrola-se a massa com o chocolate granulado.

Considere-se a seguinte sequência de acções: os dois começam a preparar os brigadeiros, aquecendo ambos as latas de leite condensado, a Maria mais tarde tira as latas de leite condensado, deixando-as arrefecer, quando julga que já estão suficientemente cozidas e o Pedro, ao vê-las a esfriar decide misturar a manteiga e o chocolate, abandona pouco depois a cozinha e a Maria entra, vê a massa dos brigadeiros deixada ao acaso e enrola-a, queimando-se na massa quente e só consegue, até se aperceber da sua lesão, enrolar dois flácidos brigadeiros, resultado de um incumprimento grave da receita.

O resultado desta confusa sequência de acções é uma mão queimada, dois brigadeiros de pálida aparência e 2 latas de leite condensado desperdiçadas, tudo porque não houve qualquer tipo de coordenação entre o Pedro e a Maria.

Esta situação pode parecer improvável em humanos, no entanto se estivéssemos a falar de robots ou de agentes providos de inteligência artificial, esta situação é possível e provável se não forem definidas as sequências das operações e as operações que cada um dos agentes irá ter de realizar para atingir o objectivo final que, no caso acima especificado, é a confecção de alguns brigadeiros.

No entanto a especificação da ordem das operações e a atribuição das operações a cada um dos agentes pode não ser suficiente.

Suponhamos que o Pedro e a Maria, conscientes do fracasso que tinha sido a sua

primeira tentativa de satisfazer a sua gulodice, resolvem dividir entre si as tarefas. Desta vez estabelecem que o Pedro vai comprar o chocolate granulado ao supermercado, dado que não existe na dispensa, enquanto a Maria cozinha as latas de leite condensado e trata de tudo o que tem a ver com a massa.

Ainda que esta solução pareça óptima, não o é. A razão não é imediata, mas se tivermos em conta que os agentes que estamos a considerar são máquinas, torna-se visível.

Um bom caso para demonstrar a falibilidade do plano é o Pedro ter chegado ao supermercado e ter notado que não havia chocolate granulado. Desta forma a Maria irá ficar eternamente à espera que o chocolate granulado chegue para que possa enrolar os seus brigadeiros. O Pedro, ao ver que não era possível concretizar a sua missão e satisfazer a sua gula, comprou um gelado e foi para casa, deixando a Maria com a massa fria e inutilizável.

Esta triste história acontece pois houve uma falha de comunicação entre a Maria e o Pedro, não tendo este último comunicado o seu fracasso, deixando uma desolada Maria à espera do ingrediente essencial a qualquer brigadeiro que possa chamar por este nome.

2.2 – Cooperação entre agentes

Como vimos caso anterior, é essencial a definição de um formalismo quando se trata de cooperação entre agentes pois estes podem não discernir quando devem abandonar a execução de determinadas actividades.

O caso descrito no ponto anterior é paradigmático, pois a não comunicação da mudança de comportamento de um agente (no nosso caso o Pedro) pode fazer com que outro agente (Maria) fique envolvido em tarefas desnecessárias *ad aeternum* ou ficar um tempo limite (*timeout*) à espera de algo que não vai acontecer. Em qualquer um dos casos esse agente torna-se inútil para a obtenção do objectivo.

Define-se, então, um comportamento relacional [1]:

Definição (Comportamento Relacional): Um comportamento relacional contém um conjunto de acções primitivas que deverão ser executadas coordenadamente por um conjunto de agentes numa equipa cooperativa. Os participantes no comportamento deverão ter um objectivo comum (*joint persistent goal*) e comunicar entre eles para atingir esta coordenação. A este objectivo comum pode dar-se também o nome *commitment*.

Na definição de comportamento relacional é referido o conceito de 'objectivo comum'. Pode haver múltiplas formas de definir este conceito. A definição utilizada é a referida em [2]:

Definição (Objectivo comum):

Uma equipa de agentes tem um objectivo comum relativamente a *q*, para atingir *p* apenas quando todas as seguintes condições têm lugar:

- É conhecimento mútuo que *p* é actualmente falso.
- Os agentes desejam mutuamente que *p* venha a ser verdadeiro.
- É verdade (e conhecimento mútuo) que até os agentes venham a acreditar que *p* é verdade, que *p* nunca será verdade, ou que *q* é falso, eles continuarão a ter *p* como um objectivo *fraco* relativo a *q* e com respeito à equipa.

Teremos então de definir objectivo fraco, utilizando, de igual modo, o que vem referido em [2]:

Definição (Objectivo fraco):

Um agente tem um objectivo individual fraco relativamente a *q* e com respeito à equipa para atingir *p* quando qualquer uma destas condições tem lugar:

- O agente tem o objectivo persistente de atingir *p*.
- O agente acredita que *p* é verdadeiro, nunca será verdadeiro, ou é irrelevante (ie *q* é falso), mas tem o objectivo de fazer com que o estado de *p* seja **conhecimento mútuo** de todos os membros.

Adaptando estas definições à nossa culinária história, imaginemos que a Maria e o Pedro pretendem cozinhar uns brigadeiros para comer ao lanche.

O primeiro passo tem lugar quando o Pedro tem a ideia de pedir à Maria, que considera ser alguém que tem gosto e talento para a cozinha, para fazer uns brigadeiros, a Maria, vendo-se livre de outros trabalhos, decide aceitar e os dois resolvem dividir as tarefas como já havia sido estabelecido em outras ocasiões:

- O Pedro ficará encarregue de comprar o chocolate granulado.
- A Maria cozinhará a massa de manteiga e leite condensado enquanto o Pedro for comprar o chocolate granulado.
- O Pedro ficará encarregue de enrolar os brigadeiros no chocolate granulado, quando a massa que a Maria cozinha estiver pronta.

Se tudo correr pelo melhor, os brigadeiros estarão prontos a comer após o Pedro os enrolar.

Podemos começar a formular este problema como:

q - O Pedro e a Maria estão com vontade de comer guloseimas

p – O Pedro e a Maria resolvem fazer brigadeiros para satisfazer a sua vontade

Ou seja, enquanto estiverem com vontade de comer guloseimas *q*, farão os possíveis para satisfazer esta vontade ao tentar cozinhar uns brigadeiros, *p*, para os comer mais tarde.

Tendo em conta que o objectivo *p*, é um objectivo fraco de ambos, a Maria e o Pedro não terão qualquer problema em abandonar o projecto de cozinhar brigadeiros caso um deles note que é impossível de atingir este objectivo, ou que é irrelevante (i.e. já não há vontade de cozinhar brigadeiros) ou que o objectivo foi concluído. Em qualquer um destes casos a Maria ou o Pedro comunicarão ao outro que já não há vontade, que está concluído o projecto ou que é impossível a sua conclusão e o outro prontamente abandonará o projecto e dedicar-se-á a outras tarefas, o que, numa perspectiva optimista, poderá ser o engolir dos brigadeiros.

Nota-se também, que o Pedro terá de avisar a Maria que chegou a casa com o chocolate e perguntar-lhe se a massa já está pronta a ser enrolada. O Pedro, lembrando-se das terríveis consequências que meter a mão em massa quente pode ter, esperará que a Maria o avise que pode enrolar os brigadeiros, havendo, neste caso, sincronismo na execução das tarefas por parte de ambos.

Tendo definido a base teórica do projecto, passar-se-á à explicação da forma como se pode modelar um comportamento genérico utilizando uma rede de Petri.

2.3 - Redes de Petri

A principal ferramenta de modelação utilizada neste trabalho, assim como em toda a equipa ISocRob é a Rede de Petri.

É importante incluir-se neste trabalho uma pequena descrição do que é uma Rede de Petri, do formalismo desta rede, das análises que se podem fazer e propriedades que as redes representativas dos comportamentos deverão ter.

No entanto, antes de se passar à explicação de o que é uma Rede de Petri, é necessário descrever o que é um sistema de eventos discretos.

Os comportamentos deverão também ser definidos utilizando uma tabela de estados, que irá providenciar uma descrição destes em linguagem corrente, facilitando a compreensão das redes de Petri.

2.3.1 Sistema de Eventos Discretos

Um Sistema de Eventos Discretos [11] (SED) é um sistema que se baseia em estados. Estes sistemas evoluem dependendo da ocorrência assíncrona de eventos ao longo do tempo, ou seja, de eventos que ocorrem sem haver qualquer sincronização ou evolução temporal.

Estes eventos controlam a transição entre estados. Dependendo do tipo de evento, um sistema de eventos discretos (SED) 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 provoca a transição entre estados.

2.3.2 Definição Redes de Petri

As redes de Petri são representações matemáticas de sistemas de eventos discretos. São compostas por lugares, transições, arcos de transições para lugares e de lugares para transições que podem, ou não, ter um peso associado e pela marcação de um conjunto de lugares que define o número de *tokens* existente em cada lugar.

Normalmente às transições correspondem eventos, quando um evento ocorre, a transição ou transições a ele associada(s) irão consumir *tokens* dos lugares de entrada e colocá-las nos lugares de saída. Os lugares podem representar acções, comportamentos e uma série de outras coisas, dependendo do âmbito da utilização da rede, tais como quantidades em *stock*, fases de projecto entre outras.

Durante a execução de uma rede de Petri poderá haver várias transições que possam ser disparadas simultaneamente, o que faz com que a rede de Petri não seja determinística.

2.3.3 Porquê Modelar com Redes de Petri?

A utilização de redes de Petri na fase de modelação de comportamentos na equipa ISocRob prende-se com algumas características destas que serão aqui descritas e comparadas com outras possíveis formas de modelação, tais como autómatos.

A principal vantagem das redes de Petri sobre os autómatos é o maior poder expressivo. Com uma rede de Petri consegue-se representar tudo o que se consegue representar com autómatos e mais ainda.

Outra vantagem é o menor número de lugares, face ao número de estados. Ou seja, enquanto numa rede de Petri podemos ter, ou não, um *token* em dois lugares, representando o caso em que o robot está a efectuar duas operações em simultâneo, ou apenas uma delas ou nenhuma delas, isto num autómato representaria 4 estados diferentes, ou seja, o caso de uma rede de Petri com a limitação de apenas um *token* por lugar pode representar até 2^P estados num autómato, em que P é o número de lugares da rede de Petri.

Há também a vantagem de se poder correr funções de análise em redes de Petri [7], verificando se há possibilidade de entrar em *deadlocks*, ou seja, estados de onde não há saída, quais os estados alcançáveis e quantos *tokens* teremos em cada lugar, o que permite garantir que um determinado comportamento será sempre executado correctamente.

2.4 Como Modelar Comportamentos com Redes de Petri?

Como foi acima referido, as redes de Petri são uma ferramenta muito boa para a modelação de comportamentos, mas é necessário referir a forma como é feita a modelação de comportamentos utilizando-as.

A modelação de comportamentos é realizada fazendo corresponder a cada transição um evento e a cada lugar uma acção primitiva, que poderá ser de comunicação ou de movimento do robot.

Os lugares são também utilizados para representar o estado de compromisso no comportamento relacional, havendo lugares 'especiais' que representam o estado do robot no comportamento relacional, ou seja, se está, ou não, comprometido (*commited*) [figura 2.3].

Os arcos irão unir as acções primitivas a eventos e estes a outras acções primitivas, ou seja, o fluxo é controlado pelos eventos que irão surgir na rede como a entidade que faz trocar as acções primitivas a realizar por cada agente.

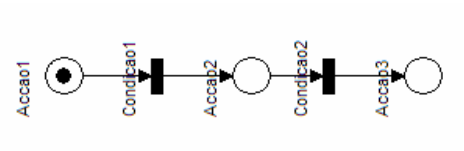


Figura 2.1 – Rede de Petri para um comportamento genérico

Quando existe um *token* num lugar, significa que o robot está a fazer a acção correspondente ao lugar ou, caso o lugar represente um *commitment*, significa que o robot

está envolvido num comportamento relacional.

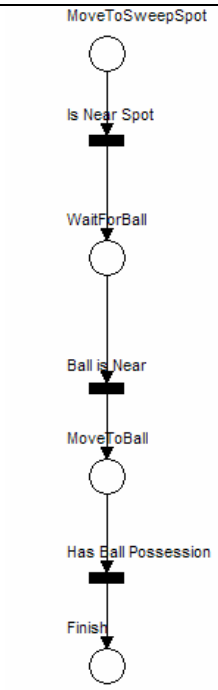


Figura 2.2 – Rede de Petri para um Comportamento individual

A modelação de um comportamento relacional pode ser vista na figura seguinte, retirada de [8]:

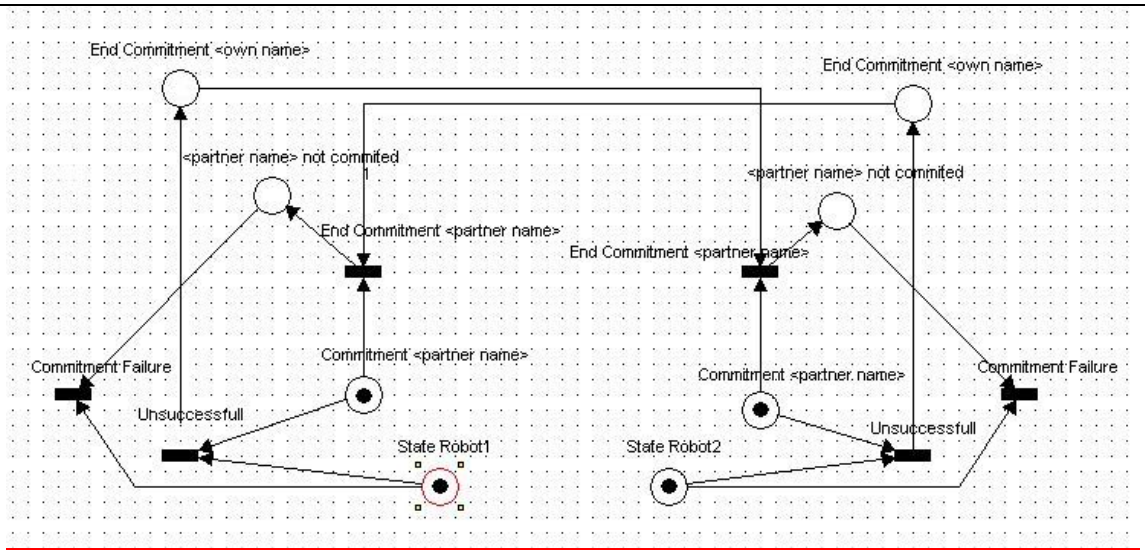


Figura 2.3 – Rede de Petri para um comportamento relacional genérico

Um exemplo de modelação de redes de Petri para comportamentos relacionais pode ser encontrado na figura abaixo. Note-se quando um agente verifica a conclusão de uma

tarefa ou que esta é impossível, irá informar o companheiro de comportamento para que este altere o seu estado para não comprometido.

2.4.1 Que propriedades deverão ter as redes de Petri de comportamentos?

As redes de Petri que modelam os comportamentos deverão ter algumas propriedades que assegurem o seu bom funcionamento.

As propriedades, bastante intuitivas, são, concretamente, a inexistência de *deadlocks*, para não haver estados em que um robot fique indeterminadamente à espera de algo que não possa acontecer, e a limitação em todos os lugares de haver, no máximo, um *token*. Todos os arcos deverão, também, ter peso constante e igual a 1.

Estas duas últimas limitações prendem-se com o significado que tem existir um *token* num lugar. Isto significará que o robot irá realizar a operação existente no lugar ou que está comprometido num comportamento relacional. Desta forma não faz sentido existir dois *tokens*, no mesmo lugar, dado que isso não teria significado prático e poderia causar problemas na execução dos comportamentos se não garantirmos que cada robot só faz uma acção primitiva de cada vez (ou duas no caso de uma delas ser de comunicação).

As propriedades que uma rede que modele um comportamento deverão, então, ser:

- Todos os lugares 1-limitados.
- A rede deverá ser viva.

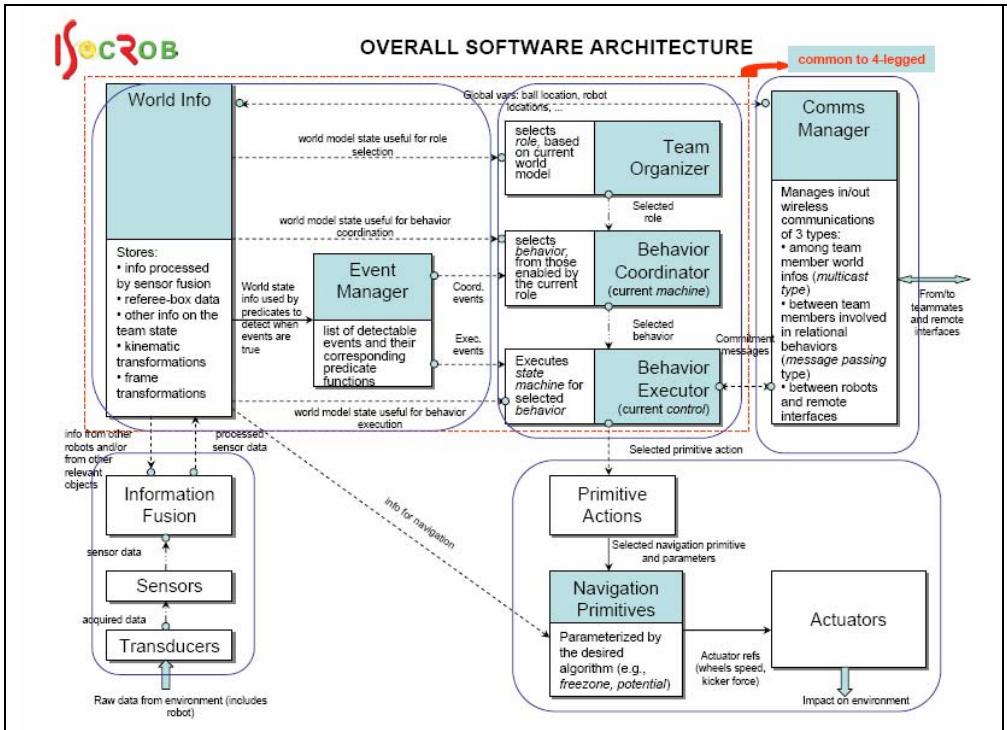
3 - Arquitecturas

Neste capítulo serão apresentadas as arquitecturas envolvidas na realização deste trabalho. As arquitecturas descritas serão a arquitectura ISocRob e a arquitectura da *German Team*.

3.1 - Arquitectura ISocRob

Os desafios existentes no projecto foram, numa fase inicial, especificar arquitecturas e construir sistemas suficientemente robustos para cada um dos grupos existentes no projecto ISocRob. No entanto nesta altura está a ser construída e especificada uma arquitectura que seja comum a todas os grupos do projecto ISocRob, tendo esta como objectivo a definição das estruturas que deverão constituir o esqueleto do projecto, sendo estas preenchidas com a informação específica a cada um dos grupos de trabalho.

A arquitectura que vem sendo desenvolvida no projecto ISocRob pretende atingir o objectivo ambicioso de uniformizar a arquitectura de tal forma a que seja possível partilhar toda a definição de comportamentos entre ambas as equipas de futebol robótico.



Legenda:

Processos.

Objectos

Classes de Objectos

Objecto ou classe que toma a iniciativa de pedir informação ou dados.

Figura 3.1 – Arquitectura ISocRob

(Nota: Na figura existem 3 tipos de setas, indicando comunicação entre processos ou entre *threads* ou dentro de *threads*. No sistema operativo dos AIBO esta questão não se põe, dado que toda a comunicação é feita entre objectos, não existem *threads.*)

Os pontos essenciais da arquitectura estão descritos na figura 3.1. Existe uma parte que será, tanto quanto possível, partilhada entre as equipas de futebol robótico que se encontra envolvida por um rectângulo vermelho. Isto faz com que os comportamentos sejam partilhados tanto quanto possível, ou seja, que as equipas tenham idealmente a mesma forma de jogar.

A arquitectura tem uma filosofia muito simples, a informação é capturada por sensores que irão actualizar o modelo do mundo (*World Info*) guardado na memória do robot. Este modelo do mundo é também actualizado por informação recebida em mensagens enviadas por companheiros de equipa (recebidas pelo Gestor de Comunicações).

Tendo em conta o modelo do mundo será seleccionado um papel para o robot (ex. avançado, defesa ou médio) que irá definir os possíveis comportamentos que o robot poderá tomar.

Durante a execução dos comportamentos os robots poderão enviar mensagens aos companheiros de equipa, também enviadas pelo Gestor de Comunicações. A execução dos comportamentos também irá requerer acções sobre o mundo, que são designadas como acções primitivas. Alguns exemplos destas acções são *goToPosture(x,y,theta)* ou *moveToBall().*

Este trabalho incidiu, sobretudo, nos comportamentos, ou seja, trabalhou-se com o *Behavior Coordinator* e *Behavior Executor*.

Do ponto de vista da classificação de comportamentos, que é a matéria em estudo neste trabalho, pode-se considerar 3 tipos distintos:

- Comportamentos organizacionais: relativos à organização da equipa, tais como a definição do papel de cada jogador.
- Comportamentos relacionais: comportamentos que envolvem mais que um jogador da equipa (por exemplo, jogadas, passes, tabelas, marcações).
- Comportamentos individuais: comportamentos relativos a apenas um jogador.

A arquitectura pressupõe que os comportamentos são definidos por acções primitivas, não havendo composição de comportamentos.

Um factor importante para a explicação da arquitectura é o papel da estratégia (ou organização) no funcionamento da equipa. Após a atribuição de um papel a um jogador, este terá disponível para si um conjunto de comportamentos, distinto do conjunto de comportamentos de outro jogador com um papel diferente. Desta forma podemos ter equipas com formas de jogar mais realistas e, possivelmente, que demonstrem uma maior competitividade devido a esta organização.

Uma boa explicação para a relação entre papeis e comportamentos é supor que temos 5 comportamentos diferentes:

A) Cobrir um jogador e tentar tirar-lhe a bola.

B) Driblar em direcção à baliza e chutar.

C) Desmarcar-se, ganhando posição para um remate.

D) Passar a bola a um jogador avançado.

E) Marcar um jogador avançado em posição perigosa.

E 3 papeis diferentes:

1. Defesa
2. Médio
3. Avançado

Desta forma podemos distribuir os comportamentos definidos pelos papeis existentes:

- Comportamentos(Defesa) = {A, D, E}
- Comportamentos(Médio) = {A, B, D}
- Comportamentos(Avançado) = {B, C, D}

A arquitectura ISocRob está, neste momento, em fase de implementação e é possível que venha a haver algumas alterações a este modelo.

3.2 Arquitectura da German Team 2004

Foi tornado claro que o objectivo da equipa ISocRob e deste trabalho em particular, prendia-se não tanto com a mecânica e sensores dos robots, mas com a implementação de comportamentos relacionais e a especificação da arquitectura onde estes possam ser executados, não havendo necessidade de re-implementar funções módulos que já existissem em outras equipas.

Sendo assim, fazia todo o sentido que fossem encontradas soluções já desenhadas,

implementadas e testadas para evitar que o trabalho realizado se prendesse demasiado com toda a parte de *hardware* dos robots, focando-se apenas, ou o máximo possível, toda a parte de escolha e controlo de comportamentos.

Optou-se por se escolher a arquitectura da equipa vencedora da edição de 2004 do RoboCup, a *German Team*, cujo código é altamente modular, podendo a arquitectura ISocRob aproveitar muita da funcionalidade já implementada e com sucesso, provado com os excelentes resultados que obteve em 2004 e 2005.

Serão descritos brevemente os módulos da *German Team*, e a sua arquitectura, sendo as alterações e adições realizadas descritas na próxima secção.

3.2.1 - Módulos, Representações e Soluções

Há 3 conceitos essenciais para perceber a arquitectura da German Team [5]:

- Módulos: Um módulo é uma solução específica para uma determinada tarefa (ver figura 2.2.3). Os módulos trocam informação com outros módulos, sensores e actuadores por intermédio de representações externas, nunca havendo referências explicitas nem invocações de módulos dentro de outros módulos. O processamento da informação recolhida pelos sensores ou das acções que os robots deverão efectuar é efectuado em módulos. Os módulos são representados por rectângulos na figura 2.2.2.
- Representações: Uma representação é uma estrutura de dados que é trocada entre módulos, representando informação proveniente de sensores (ex: imagens recolhidas pela *webcam*) ou de outros módulos (ex: pose do robot ou posição da bola). Estas estão definidas no directório */Src/Representations*. As representações deverão ter operadores de *streaming* para facilitar a sua troca entre módulos. As representações são apresentadas como elipses na figura 2.2.2.
- Soluções: Para cada módulo são definidas as representações nas quais podem escrever e ler. Graças a esta interface fixa é possível trocar entre diferentes soluções para cada módulo ou desligar o módulo totalmente (para efeitos de *debug*).

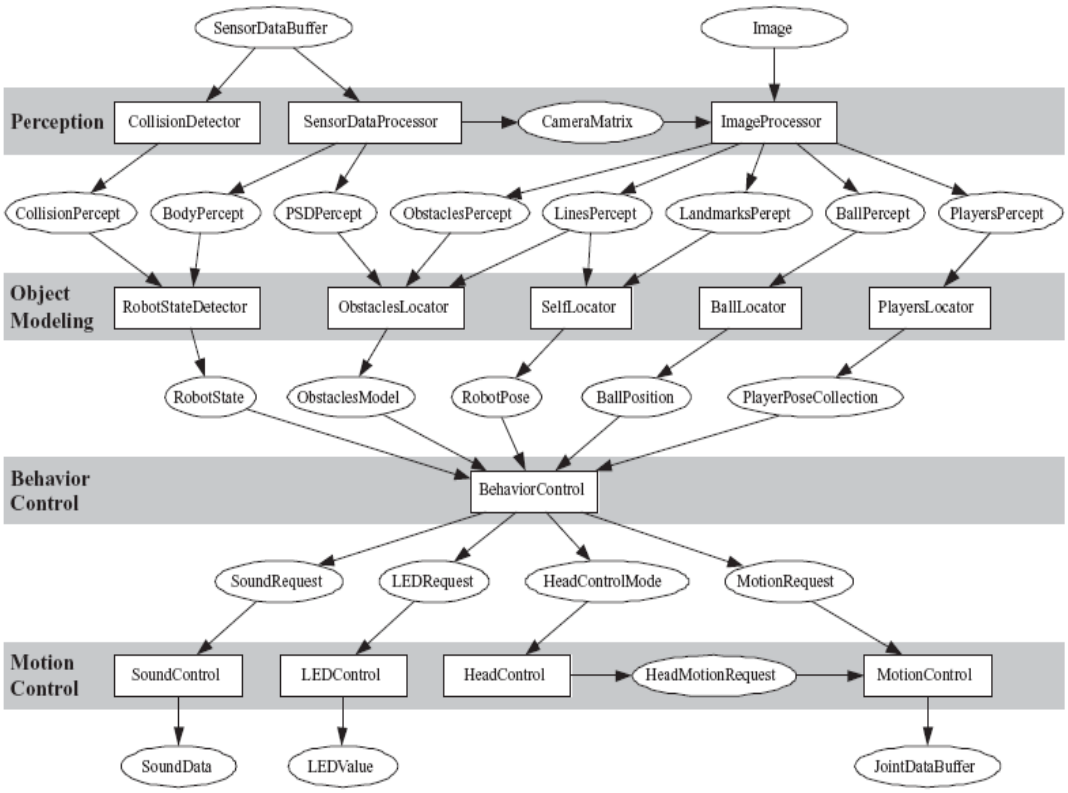


Figura 3.2 – Arquitectura *German Team* 2004

Devido à própria organização da German Team (composta por 4 equipa que desenvolvem em paralelo), foi criada uma arquitectura modular. Por módulo designa-se um conjunto de funções específico para uma certa tarefa. A arquitectura permite a troca dos módulos em *runtime*, o que traz vantagens, permitindo a alteração dinâmica do módulo a utilizar para uma determinada tarefa caso os resultados que estejam a ser produzidos não satisfaçam a exigência.

Para permitir a partilha de módulos, foram definidas interfaces para todas as tarefas necessárias para jogar futebol [5], interfaces que deverão ser respeitadas pela equipa de AIBOS do ISocRob, sendo que a equipa deverá trabalhar sempre que possível no campo das soluções, criando ou eliminando o mínimo possível os módulos existentes.

Um exemplo de um módulo com diferentes soluções é apresentado na figura 3.3, onde é apresentado o módulo *BallLocator*, que tem como finalidade localizar a bola, existindo 5

soluções diferentes, sendo estas *ATH2004-BallLocator*, *MSH2004BallLocator*, *PerceptBallLocator*, *PIDSmoothedBallLocator* e *VoteBasedBallLocator*.

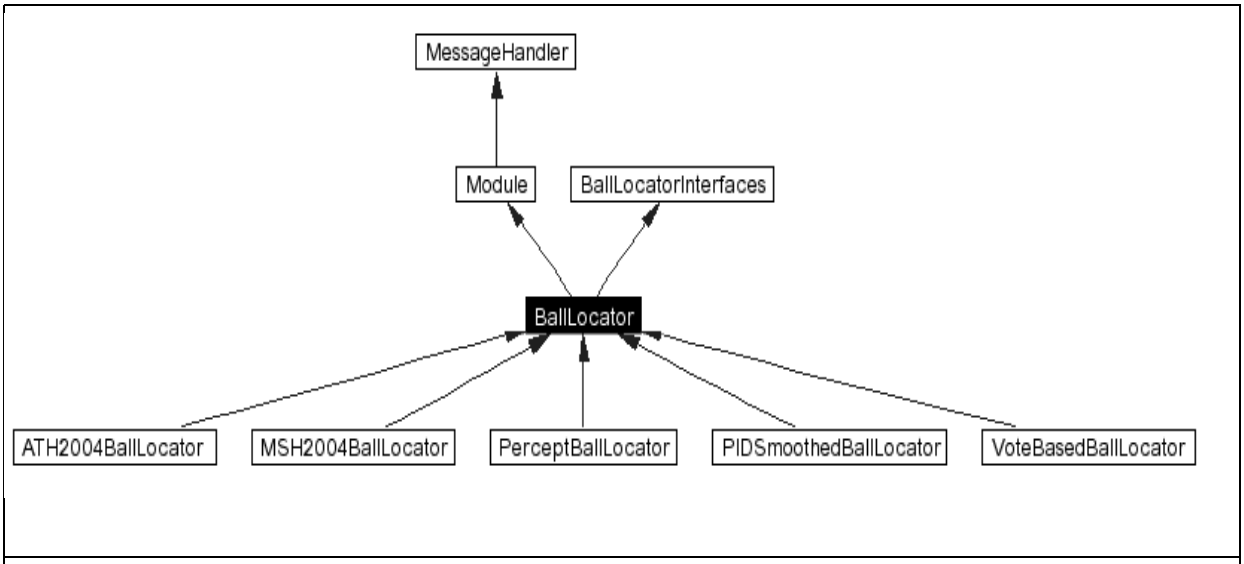


Figura 3.3 – Módulo *BallLocator*

Um *layout* de processos consiste na definição de quais os módulos que se agrupam em cada processo. Foi também definido um *layout* de processos que é muito simples, consistindo em apenas 3 partes, *Cognition*, *Motion* e *Debug*, que, segundo [5] provou ao longo dos anos ser o mais eficiente.

Os processos agruparão diversos módulos, por exemplo no *layout* abaixo definido os módulos constantes no processo *Motion* serão todos os módulos que terão como finalidade a locomoção dos robots e no processo *Cognition* serão os módulos que terão como objectivo a aquisição de informação do mundo, como por exemplo os módulos *BallModeling*, *CollisionDetector* ou *PlayerModeling*.

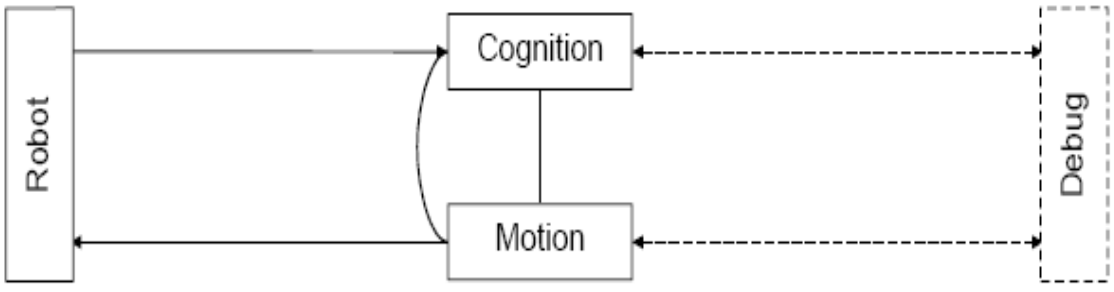


Figura 3.4 – Modelo de processos *German Team* 2004

É importante referir que os módulos não se referenciam mutuamente, de facto, nem se

conhecem mutuamente. Os módulos trocam informação com outros módulos através de uma representação externa. Isto significa que se um módulo é executado, ele lê a partir de uma estrutura de dados externa e guarda o resultado do processamento também em outras e representações (estruturas) externas.

Como já foi dito, estas representações têm operadores de conversão (*streaming*) que permitem que sejam transferidas facilmente entre processos, entre robot e robot e entre robot e PC. [5]

3.2.2 - Módulos existentes na *German Team* 2004.

Como foi descrito no ponto anterior, o código da *German Team* é extremamente modular. Desta forma pode-se criar módulos para representar tarefas que os robots deverão fazer e que não foram contempladas na *German Team* 2004, assim como definir-se soluções específicas para os módulos existentes.

Existem 9 módulos na *German Team* 2004, que serão brevemente descritos nas linhas que se seguem:

- ***Body Sensor Processing*** – O objectivo do *SensorDataProcessor* é o de obter dados de todos os sensores excepto da câmara e guardá-los em memória.
- ***Vision*** – O módulo *Vision* trabalha com as imagens oferecidas pela câmara do robot. O *output* do módulo preenche a estrutura *PerceptCollection.* A *PerceptCollection* contém informação sobre a posição relativa da bola, das linhas do campo, das balizas, das bandeiras, dos outros jogadores e de obstáculos. As posições e ângulos na *PerceptCollection* são relativas ao robot.
- ***Self-Localization*** – O sistema de auto-localização da German Team 2004 implementa um método de localização Markov, empregando a estratégia Monte-Carlo.
- ***Ball Modeling*** – No módulo *Ball Modeling* é criada um modelo da bola, incluindo a posição da bola e velocidade. Para reduzir o ruído dos valores obtidos é aplicado um filtro de Kalman à posição e velocidade da bola obtidas a partir da percepção da bola.
- ***Obstacle Model*** – No modelo de obstáculos é criada uma vista tipo radar das redondezas do robot.
- ***Collision Detector*** – Foi implementada uma solução para detecção de colisões. A informação de o que provém deste módulo, que indicará se o robot estará, ou não, a colidir contra algo pode ser utilizada para fazer com que o robot aja de uma maneira ou de outra e ajustar a informação de auto-localização.
- ***Player Modeling*** – Permite obter a posição de companheiros de equipa, a partir das

comunicações com recebidas.

- ***Behavior Control*** – Este módulo é o utilizado para tomada de decisões baseadas no estado do mundo. Este módulo foi o que mais alterações sofreu, pois é neste que incidia o tema deste trabalho. Os comportamentos na *German Team* 2004 são descritos utilizando uma linguagem baseada em XML, que é designada XABSL e é corrida neste módulo e descrita adiante.
- ***Motion*** – O módulo *MotionControl* gera as posições de juntas a serem enviadas para os motores e assim é responsável por controlar os movimentos do robot.

Apesar da alta modularidade o objectivo inicial de aproveitar os módulos da *German Team* e inseri-los na arquitectura ISocRob não foi conseguido, por restrições de tempo, dado que a integração de módulos da arquitectura ISocRob seria um processo moroso e neste momento a arquitectura ISocRob não está ainda totalmente implementada.

Desta forma optou-se por implementar na arquitectura da *German Team* os mecanismos para comportamentos relacionais, seguindo os modelos definidos, tendo-se incidido particularmente na comunicação e estabelecimento de comportamentos relacionais entre os robots, assim como em especificar uma descrição formal destes de como esta será feita.

3.3 - Diferenças e semelhanças entre as duas arquitecturas

Ainda que possam parecer muito diferentes, as arquitecturas da *German Team* e ISocRob são muito parecidas conceptualmente, sendo que a ISocRob prima mais pela abstracção, enquanto a da *German Team* é mais direccionada a uma equipa de robots futebolistas, como se conclui facilmente olhando para os módulos disponibilizados. Todos os problemas de possíveis incompatibilidades serão aqui abordados, assim como a forma que se encontrou para os contornar.

Nesta secção serão brevemente explicadas as semelhanças entre conceitos numa e noutra equipa, assim como a forma como as equipas solucionam alguns problemas em particular.

3.3.1 World Info vs. Representations

No essencial ambas as arquitecturas guardam o mesmo género de informação sobre o mundo, tal como a posição da bola, posição dos jogadores ou estado do jogo.

No entanto a forma como esta informação é guardada e acedida é substancialmente diferente num caso e no outro.

A *German Team*, guarda toda esta informação em objectos, chamadas *representations*, ou em português, representação, uma representação corresponde a um modelo de alguma pedaço de informação do mundo a que possamos querer aceder.

Desta forma, se quisermos aceder à posição da bola, vista pelo jogador, acederemos à representação *BallPosition*, ou melhor, a uma instanciação desta a que daremos o nome *ballModel*, e invocaremos sobre ela um método que nos desse a posição da bola vista pelo jogador, ou seja, genericamente: *ballModel.getSeenPosition()*.

Se, por alguma razão, o nosso jogador não estiver a ver a bola, podemos aceder à representação da bola e pedir a posição vista pelos jogadores de equipa.

ballModel.getPositionSeenByTeammate(teammateNumber).

Um ponto importante a reter sobre as representações é que a informação nela contida é preenchida pelos módulos ou sensores em todos os casos, ou seja, uma representação não é mais do que um repositório de dados que serve para trocar informação entre módulos.

O funcionamento da arquitectura ISocRob é um pouco diferente, havendo uma abstracção maior dos mecanismos de troca e armazenamento da informação, a que não será alheio o facto de haver duas equipas de futebol robótico, com *hardware* e sistemas muito diferentes.

Toda a informação sobre o mundo está localizada no objecto *World Info*, sendo a informação deste obtida através de sensores e informação comunicada por companheiros de equipa.

Desta forma se quisermos obter a posição da bola, faríamos algo como:

worldInfo.getBallPosition(), sendo que a informação sobre a posição da bola obtida através de mensagens de companheiros e / ou processamento da informação de sensores é colocada no *World Info*.

Resumindo, é possível traçar um paralelo entre ambas as arquitecturas caso haja abstracção na forma como a informação é obtida, no entanto organização e obtenção da informação realizada de maneira diferente. A informação guardada é, basicamente, a mesma e nisto se apoia este trabalho.

3.2.2 - Eventos

Na *German Team* não existe a noção de eventos, ao contrário do que se passa na equipa ISocRob, sendo esta uma das mais cruciais diferenças entre as duas arquitecturas.

Os eventos servem para 'disparar' transições entre estados, remetendo ao exemplo culinário, um exemplo de evento seria a Maria avisar o Pedro que a massa está pronta a enrolar, deixando Pedro a espera, ao ter a informação da ocorrência deste evento, passando ao estado seguinte, ou seja, enrolar a massa.

A forma de funcionamento da *German Team* é um pouco diferente, exigindo mais processamento, ou seja, o Pedro em vez de ser assinalado pela Maria, terá de ir verificando regularmente se a Maria lhe comunicou se a massa está fria.

Concretizando, em vez de eventos, na *German Team* as condições de passagem entre acções primitivas ou estados têm de ir sendo calculadas. Em vez de existir uma mensagem por parte de um companheiro de equipa que seja imediatamente integrada na execução do comportamento, tem de haver uma consulta regular às mensagens recebidas e verificar se houve alguma que lhe permita passar para o estado seguinte.

Apesar da aparente incompatibilidade, pode-se definir um evento como a passagem de uma condição de um estado verdadeiro para falso ou de falso para verdadeiro, ou seja, quando uma determinada condição, chamemos-lhe *aMariaComunicouQueAMassaJáEstáFria*, passa do estado falso, para verdadeiro, o Pedro poderá passar à fase de enrolar os brigadeiros, no entanto cabe ao Pedro verificar regularmente se a Maria, de facto, já comunicou.

3.2.3 - Comunicação

A comunicação feita entre companheiros de equipa é executada, também, de forma diferente.

A equipa ISocRob definiu que a comunicação poderá ser feita um robot e todos os outros (*multicast*), robots e interfaces remotas ou entre dois robots.

Na *German Team*, temos apenas as duas primeiras, não havendo explicitamente a comunicação entre dois companheiros específicos da equipa. As mensagens entre membros são recolhidas no módulo *BehaviorControl* e são enviadas em *multicast* para todos os outros companheiros de equipa em intervalos regulares (no caso 0.1 segundos).

Ainda que isto pudesse parecer um problema no que toque à execução de comportamentos relacionais, de facto não o é, pois pode enviar-se mensagens preenchendo

um campo de informação com o número de jogador do destinatário e quando um jogador recebe a mensagem apenas a interpretará se for lhe direccionada.

3.2.4 - Comportamentos

Também há algumas diferenças na classificação de comportamentos e na coordenação existente entre estes.

A *German Team*, assim como totalidade das equipas da *4 Legged League*, não tem qualquer mecanismo para comportamentos relacionais, tendo a implementação destes sido feita no âmbito deste trabalho e é descrita mais à frente.

Quanto à definição de comportamentos, a *German Team* utiliza apenas dois tipos de comportamento, *Basic Behaviors* e *Options*. É fácil traçar um paralelo entre estes e a definição de comportamentos da equipa ISocRob, correspondendo os *Basic Behaviors* a acções primitivas e as *Options* a comportamentos. Existe, também, variáveis de controlo de alto nível, que permitem que os jogadores se coordenem minimamente, por exemplo, fazendo que não haja simultaneamente dois jogadores a tentar controlar a bola.

No entanto o que é considerado uma *Option* na arquitectura da German Team é muitas vezes considerado uma acção primitiva na equipa ISocRob, exemplos disto são as *Options get-to-ball* ou *get-to-position*.

Isto poderia ser um problema, mas não o é pois as *Options* podem ser compostas por outras *Options*, ou seja, poderemos ter um comportamento que é composto pela *Option get-to-ball* e *do-kick*, representando um comportamento individual em que o jogador se aproximaria da bola e a chutaria.

O controlo de fluxo nas *Options* é em tudo semelhante ao dos comportamentos da equipa ISocRob, havendo condições para a passagem de estados, normalmente recorrendo a símbolos obtidos a partir do modelo do mundo, podendo aplicar-se operações de comparação, aritméticas e lógicas para melhor controlar o fluxo, desta forma é fácil adaptar e integrar os comportamentos no código da *German Team*.

4 - Implementação e adições ao código da *German Team*

Como qualquer projecto de programação com alguma complexidade, a programação deste projecto teve de seguir um plano cuidado para que o objectivo final fosse atingido com sucesso.

Nesta secção serão descritas as estruturas e algoritmos criados para atingir o objectivo de correr comportamentos relacionais na equipa de AIBOs, assim como o método seguido para a implementação da *framework* que dê base à execução de comportamentos relacionais no código original da *German Team*.

De igual modo será descrita a linguagem de modelação de comportamentos XABSL criada pela GT que irá também ser comparada com as redes de Petri.

4.1 - Método de programação e calendários de implementação

Devido à grande quantidade forças e condicionantes envolvidas durante o decurso deste projecto, como por exemplo o tempo que se demora a passagem de código para os robots reais, ou a estrutura de código da GT, o método de implementação foi definido à partida e sofreu poucas alterações.

O método de programação consistiu numa aproximação iterativa aos vários problemas em questão, resolvendo-se os problemas de menor dificuldade primeiro evoluindo-se para problemas com maior dificuldade à medida que o trabalho ia avançando, testando-se as soluções encontradas no final de cada resolução, para não haver propagação de erros para as fases seguintes.

Para cada problema definiu-se um *deadline*, de forma a que não se acumulasse trabalho para as fases finais, e só se avançava para a resolução e implementação de uma solução quando ficasse plenamente resolvido.

A resolução de cada problema envolveu, também, duas fases, sendo a resolução de cada problema um processo recursivo, envolvendo a implementação da solução, teste em simulador, correcção com base de resultados da simulação e teste de novo no simulador. Quando a solução estivesse funcional no simulador, passar-se-ia para o teste nos robots reais, dado que o simulador apresenta graves limitações, como a ausência de simulação da física do mundo ou deficiências no cálculo dos campos de visão dos robots, no entanto o

simulador consegue modelar perfeitamente as comunicações entre os robots e, tendo esse sido um dos principais temas deste trabalho, resolvia-se muitos problemas apenas no simulador, correndo perfeitamente no ambiente real.

Pode descrever-se o processo de implementação recorrendo ao seguinte diagrama:

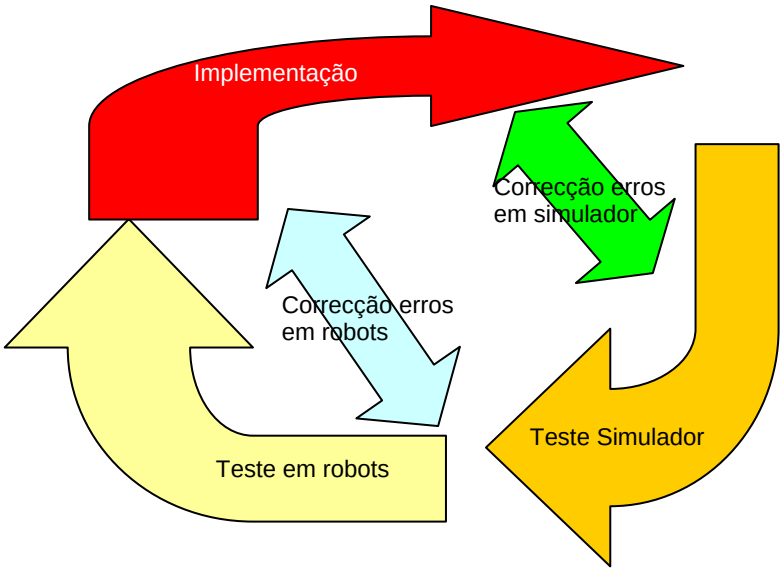


Figura 4.1 – Processo de implementação

Desta forma poucos erros passavam para os robots, sendo a correcção mais rápida no simulador, e quando existia erros nos robots resolviam-se corrigindo a implementação.

Os problemas que foram resolvidos utilizando este método foram, como foi atrás dito, de dificuldade crescente. Foram eles:

1. Implementação de comportamentos individuais
2. Implementação de comportamentos relacionais com mensagens enviadas manualmente
3. Implementação de comportamentos relacionais com mensagens enviadas pelos intervenientes no comportamento, mas com selecção dos comportamentos feita à mão
4. Implementação de sistema de selecção automática de companheiros
5. Implementação de sistema que permita a aceitação ou recusa de pedidos de

comportamento relacional

6. Implementação de sistema que permita a selecção do comportamento relacional a executar e faça, em seguida, um pedido ao companheiro calculado
7. Implementação de um despachante de comportamentos relacionais, para tornar o código mais compacto
8. Implementação de sistema que permita romper os compromissos já estabelecidos

Os problemas foram ordenados seguindo uma aproximação *bottom-up*, sendo primeiro resolvidos os problemas mais básicos, passando-se aos mais complexos ou que exijam maior automatismo por parte dos robots.

O orientador deste trabalho teve um papel essencial na definição da prioridade de cada problema a resolver e, dada a sua experiência nesta área de investigação e enorme disponibilidade, várias vezes apontou o caminho certo para as soluções que vieram a ser encontradas.

4.2 - A linguagem XABSL

A linguagem XABSL é uma linguagem baseada em XML criada com o objectivo de modelar comportamentos em agentes autónomos. Esta linguagem foi criada pela *German Team* [5], e foi utilizada neste projecto para modelação e implementação dos comportamentos relacionais.

É simples traçar um paralelo entre a linguagem XABSL e as redes de Petri. Há 4 componentes essenciais na linguagem XABSL:

- Estados
- Transições
- Símbolos
- Comportamentos e acções primitivas

As acções primitivas e comportamentos representação operações que o robot irá fazer em cada estado. As acções primitivas são mais básicas enquanto os comportamentos (designados por *options* na *German Team*).geralmente envolvem algum processamento e têm um fluxo.

Os estados são compostos por uma acção primitiva, ou comportamento, um ou mais

pedidos de *outputs* do robot (tais como LEDs, som ou movimentos de alguns motores) e por uma árvore de decisão que contenha regras de transição para outros estados.

Os símbolos representam informação processada pelos módulos e podem ser emparelhados com funções que nos dêem informações sobre o estado actual do Mundo. Há dois tipos de símbolos, símbolos de *input*, que representam informação que 'entra' no robot, ou seja, que o robot recolhe com os seus sensores ou processados a partir desta informação, e os símbolos de *output* que representam pedidos ao robot sobre certos actuadores (tais como LEDs ou som).

Apesar de a German Team ter criado esta linguagem com base em autómatos finitos deterministas, a flexibilidade das redes de Petri permite a modelação de comportamentos que se consigam representar muito facilmente nesta linguagem, como, aliás, foi feito.

4.3 - Framework *criada para comunicação de comportamentos relacionais*

Como já foi dito, a German Team, assim como todas as outras equipas da 4LL não tem um mecanismo de estabelecimento de comportamentos relacionais, tendo este de ter sido criado no âmbito deste trabalho.

Sendo assim, sobre o código da GT foi implementado o mecanismo de pedidos de comportamentos relacionais e trocas de informação na execução deste, uma estrutura que permite acrescentar comportamentos relacionais e regras para a sua aceitação ou recusa assim como funções que permitam a quebra de compromisso num comportamento relacional.

4.3.1 - A estrutura *ISocRobRelationalBehaviorRequest*

Uma das primeiras coisas que se verificou quando se começou a trabalhar especificamente na implementação de funções que permitam o estabelecimento de comportamentos relacionais, foi a necessidade de criação de uma estrutura própria para a comunicação de intenções de fazer um comportamento relacional.

No entanto a estrutura cresceu, sendo, neste momento, também utilizada para a comunicação de quebras de compromisso e aceitação ou recusa de pedidos de comportamento relacional.

A estrutura, denominada *ISocRobRelationalBehaviorRequest*, é utilizada para passar informação sobre comportamentos relacionais entre companheiros de equipa, sendo, portanto, uma *Representantion*. Os seus campos serão aqui explicados.

A estrutura é muito simples, contendo apenas quatro campos que podem assumir um número fixo de valores (tipo de dados enumerado), consoante a mensagem que se estiver a pretender passar.

Os campos que compõem a estrutura são os seguintes:

- *relationalBehavior* – O comportamento relacional que se pretende executar. Este campo é do tipo enumerado *RelationalBehavior*, tipo este que deverá conter todos os comportamentos relacionais implementados.
- *receiverNumber* – O número do jogador a quem se está a enviar a mensagem. Também se relaciona com um tipo enumerado de dados *ReceiverNumber*, que contem todos os números de jogadores, ou um de dois códigos especiais que definem que a mensagem se destina a todos os jogadores ou a ninguém. Todas as mensagens enviadas na *German Team* são enviadas a todos os jogadores, sendo assim todos os jogadores receberão a mensagem, mas apenas o jogador 'certo' a irá interpretar.
- *type* – Define o tipo de mensagem que está a ser trocada. Pode assumir um de três valores também eles enumerados, sendo estes *request*, *response* e *breakrb*, consoante seja um pedido, uma resposta a um pedido ou uma notificação de quebra de compromisso, respectivamente.
- *answer* – Define a resposta dada a um pedido. Este campo só é utilizado no caso de a mensagem se tratar de uma resposta a um pedido. Pode assumir um de três tipos enumerados *willDo*, para o caso em que se aceita o pedido de comportamento relacional, *wontDo*, para o caso em que se rejeita e *notUsed*, para o caso em que a mensagem não é do tipo *response*, tendo que se preencher o campo com alguma informação.

Uma interacção típica entre dois robots utilizando esta estrutura para comunicar intenções de fazer comportamentos relacionais pode ser:

1. Robot1 envia *ISocRobRelationalBehaviorRequest(attackingKickoff, robot3, request,*

notUsed)

2. Robot3 recebe o pedido, avalia-o, e chegando à conclusão que pretende fazer o comportamento, responde *ISocRobRelationalBehaviorRequest(attackingKickoff, robot1, response, willDo)*
3. O comportamento está em execução e quando chega ao fim ou resolver , o Robot3 pedirá ao Robot1 para quebrar o compromisso com ele estabelecido, enviando uma mensagem: *ISocRobRelationalBehaviorRequest(attackingKickoff, robot1, breakrb, notUsed)*
4. O Robot1 receberá esta mensagem e quebrará, também, do seu lado o compromisso.

Apesar da estrutura parecer e ser, de facto, muito simples, foi necessário incluir formas de a manipular no código de comportamentos da *German Team*.

Foi, então, criada uma acção primitiva que permita enviar estas mensagens, que será descrita no ponto seguinte.

4.3.2 - Símbolos de *output rb-request*

A necessidade de comunicar a intenção de estabelecer ou romper um comportamento relacional com um companheiro fez com que se tivesse de criar funções para enviar a estrutura *ISocRobRelationalBehaviorRequest* a partir do código XABSL.

Para haver uma coincidência com a arquitectura dos comportamentos relacionais definida, criou-se vários símbolos de *output* que permitam preencher a estrutura a enviar para o companheiro de equipa. Estes símbolos de *output* irão preencher os campos da estrutura *ISocRobRelationalBehaviorRequest*, consoante a mensagem que se pretende enviar, como se verifica no bloco de código 4.1.

```
<set-output-symbol ref="rb-request-relational-behavior" value="rb-request-
relational-behavior.attacking-kickoff"/>
<set-output-symbol ref="rb-request-receiver-number" value="rb-request-
receiver-number.two"/>
<set-output-symbol ref="rb-request-type" value="rb-request-type.request"/>
<set-output-symbol ref="rb-request-answer" value="rb-request-
answer.notused"/>
```

Bloco de Código 4.1 – Envio de mensagem de comportamento relacional

O que produziria como efeito, o envio da mensagem: I*SocRobRelationalBehaviorRequest*(*attackingKickoff*, companheiro2, *request*, *notUsed*), que tem como objectivo pedir ao companheiro número 2 que faça com o emissor da mensagem o comportamento relacional *attackingKickoff*.

A vantagem dos *output symbols* em relação à definição de uma acção primitiva é a possibilidade de executar o envio de mensagem em simultâneo com a execução de algum movimento por parte do robot, como é descrito nos modelos de comportamento relacional por Redes de Petri.

4.3.3 - Comportamento de alto-nível *isocrob-play*

Apesar de todo o interesse que a execução de comportamentos individuais ou relacionais por si só possa ter, na competição RoboCup é necessário que a escolha destes comportamentos seja feita de forma totalmente autónoma, não podendo haver qualquer tipo de intervenção humana na decisão dos comportamentos a executar.

Sendo assim, tem de ser criado um comportamento de alto nível que seja um equivalente ao *Behavior Coordinator* na arquitectura *ISocRob*, que decida, automaticamente, qual o comportamento a correr em cada circunstância.

O comportamento foi criado como uma *option* XABSL que num estado inicial irá verificar quais as condições para executar os comportamentos e se estas se verificam. Caso se verifiquem, esta *option* irá ordenar o robot a execução do comportamento, podendo interromper a execução dos comportamentos caso haja uma condição que permita esta interrupção.

Em pseudo-código este comportamento será algo como:

```
estadoInicial:
loop:
      Condições:
            Condição1 -> executaComportamento1
            Condição2 -> executaComportamento2
            Condição3 -> pedeComportamentoRelacional1
            CondiçãoRecebeuPedido -> decideSeFazComportamentoRelacional
            True ->estadoInicial

executaComportamento1:
```

```
      -> Comportamento1
      IF(condiçãoFimComportamento)
            -> estadoInicial
      ELSE
            -> executaComportamento1
      (...)

decideSeFazComportamentoRelacional:
      IF(aceitaComportamento)
            -> enviaMensagemDeAceitação
            -> comportamentoRelacional
      ELSE-IF(recusaComportamento)
            -> enviaMensagemDeRecusa
            -> estadoInicial
      ELSE
            -> decideSeFazComportamentoRelacional
```

Bloco de Código 4.2 – *Behavior Coordinator*

É de notar que neste último estado temos três vias diferentes, e não apenas duas como seria de assumir. No entanto é necessário haver condições diferentes para aceitação e recusa, e não apenas uma, pois como as condições fazem referência às funções C++ a que estão associadas e assumem, à partida, o valor lógico falso, a condição de aceitação e de recusa irão sempre ter o valor falso à entrada para o estado, o que levaria sempre à execução da cláusula ELSE, nunca chegando a executar um comportamento relacional.

4.3.4 - Despachante de aceitação ou recusa de comportamentos relacionais

Como foi mostrado no ponto anterior, quando há uma recepção de pedido de comportamento relacional, existe um estado no qual se decide se se vai enveredar, ou não, pela sua execução.

Não seria uma boa solução descriminar todos os comportamentos relacionais, pois isso geraria um código extremamente longo e possivelmente confuso no comportamento *isocrob-play* (implementação do *Behavior Coordinator*).

Isto foi a motivação para a criação de um ‘despachante’ de aceitação ou recusa de comportamentos relacionais.

O funcionamento do despachante é muito simples, no momento em que o pedido de comportamento relacional chega, as variáveis *aceitaComportamento* e *recusaComportamento*

são acedidas para se decidir qual o próximo comportamento a correr.

Estas variáveis estão ligadas ao despachante de aceitação de comportamentos e recusa de comportamentos, respectivamente. Este despachante tem informação de qual o comportamento relacional pretendido e chama uma função, específica ao comportamento, que diz se deverá aceitá-lo.

Em pseudo-código.

```
aceitoComportamentoRelacional()
      condição:
            comportamentoPedido     ==     attackingKickoff     :     return
aceitoAttackingKickoff();
            comportamentoPedido     ==     defensiveKickoff     :     return
aceitoDefensiveKickoff();
            (...)
            true : return false;
```

Bloco de Código 4.3 – Despachante de Aceitação e Recusa de comportamentos

As funções aceitoAttackingKickoff() e aceitoDefensiveKickoff() devolverão o valor lógico verdadeiro ou falso, consoante se deva aceitar, ou não estes comportamentos.

Com a utilização deste despachante, consegue-se uma maior abstracção no código e poupança de muitas linhas nos comportamentos de mais alto nível, pois não é preciso descrever detalhadamente todos os possíveis pedidos de comportamentos, aceitando-se ou rejeitando-se de forma absolutamente genérica.

4.3.4 Despachante de escolha de companheiros

Um comportamento relacional tem como característica essencial ser realizado por mais do que um elemento da equipa.

Um dos problemas que merece especial atenção é como fazer a escolha de um companheiro de equipa para um comportamento relacional e esta foi a principal força que levou à criação de um despachante de escolha de companheiros.

Quando existe a vontade de executar um comportamento relacional, por parte de um dos elementos da equipa, este deparar-se-á com o problema da escolha do companheiro de equipa ideal para fazer o comportamento. Seria de todo indesejável que não houvesse regras

de escolha, dado que o elemento da equipa poderia escolher o guarda-redes para receber um passe na marcação de uma fora em terreno atacante ou forçar um avançado a defender junto à área.

O despachante é uma função que recebe como argumento o comportamento relacional que se pretende executar e devolve o companheiro de equipa ideal. O seu funcionamento é ilustrado na figura 4.2.

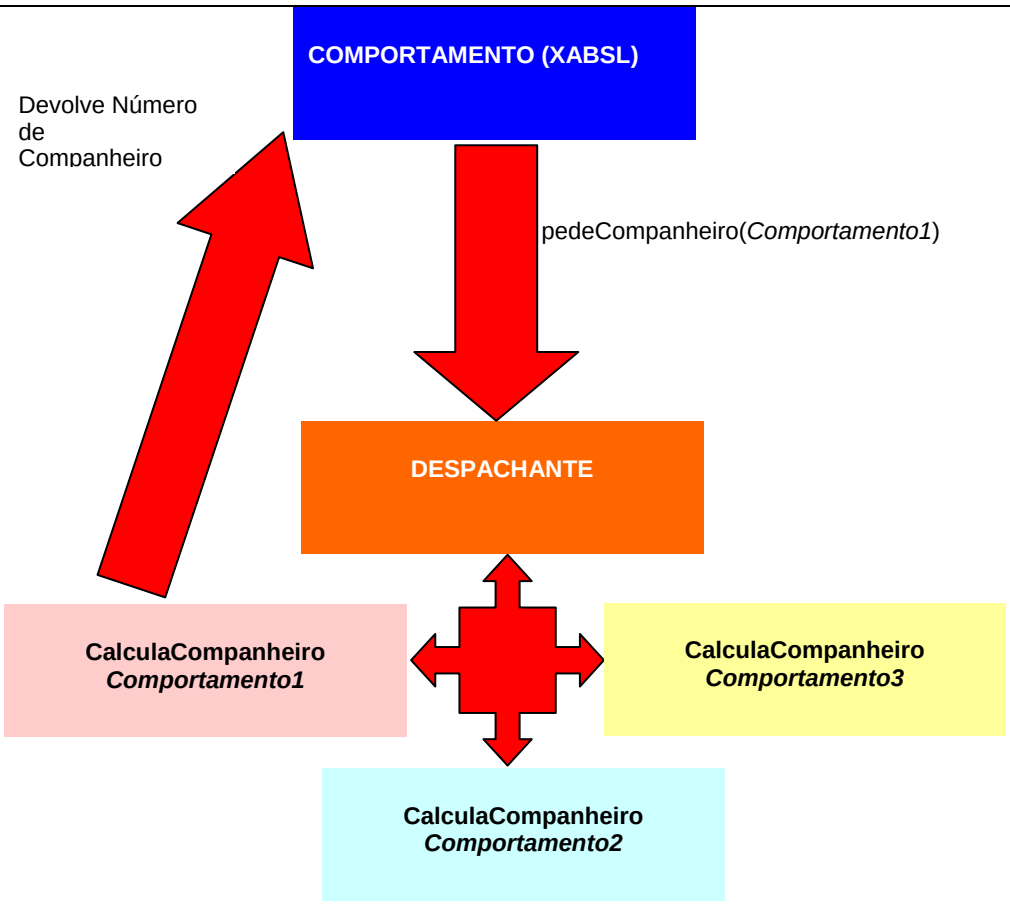


Figura 4.2 – Despachante de escolha de companheiros

Há que frisar que o despachante não faz qualquer cálculo de qual o companheiro a escolher, delegando esta tarefa para as funções que chama consoante o comportamento que lhe foi pedido.

4.3.5 Despachante de comportamentos relacionais

Para se obter uma abstracção maior e poupar-se código nos comportamentos de maior nível (*Team Organizer* e *Behavior Coordinator*), foi criado um despachante de

comportamentos relacionais.

Este despachante pressupõe que o jogador que pede o comportamento relacional e o jogador que recebe o pedido terão papeis pré-definidos na execução do comportamento, ou seja, que se um jogador, chamemos-lhe João Emissor, pede um comportamento relacional para um *attacking-kickoff* a outro jogador, chamemos-lhe António Receptor, tanto um como o outro saberão qual o papel que terão na execução do comportamento, partindo sempre da parte do jogador que irá enviar a bola a iniciativa de pedir o comportamento.

Neste caso, iria sempre partir do João Receptor o pedido de execução do comportamento caso pretendesse ter o papel de enviar a bola.

O despachante tem um papel essencial ao abstrair o comportamento que irá ser executado, não sendo necessário colocar todas as hipóteses de comportamentos relacionais nos comportamentos de alto nível, criando-se um código semelhante ao descrito no bloco de código 4.4

```
While(true)
{
(...)
  If(recebiPedido)
    If(aceitoComportamentoRelacional(comportamentoRecebido))
        despachanteCR(comportamentoRecebido,receptor);
(...)
  If(queroExecutarComportamento)
  {
    pedeComportamentoRelacional(comportamentoX,
despachanteEscolhaComp(comportamentoX));
    if(companheiroAceita)
      despachanteCR(comportamentoX, emissor);
  }
}
```

Bloco de Código 4.4 - Integração do despachante no comportamento isocrob-play (*Behavior Coordinator*)

Teria, claro, de haver um mapeamento que nos fizesse uma correspondência entre os papeis de emissor e receptor para cada comportamento, podendo ser, por exemplo:

Comportamento *AttackingKickoff*:

->Emissor: Chuta a bola

->Receptor: Recebe a bola

Comportamento *DefensiveKickoff*:

->Emissor: Persegue a bola

->Receptor: Cobre a zona da grande área.

4.3.6 Correspondência com a arquitectura ISocRob

As implementações realizadas tiveram como objectivos primordiais uma tentativa de tornar a arquitectura da GT o mais próxima possível da arquitectura definida no projecto ISocRob.

Podemos, então, sobrepor o trabalho realizado com a arquitectura original ISocRob, como é mostrado na figura 4.3:

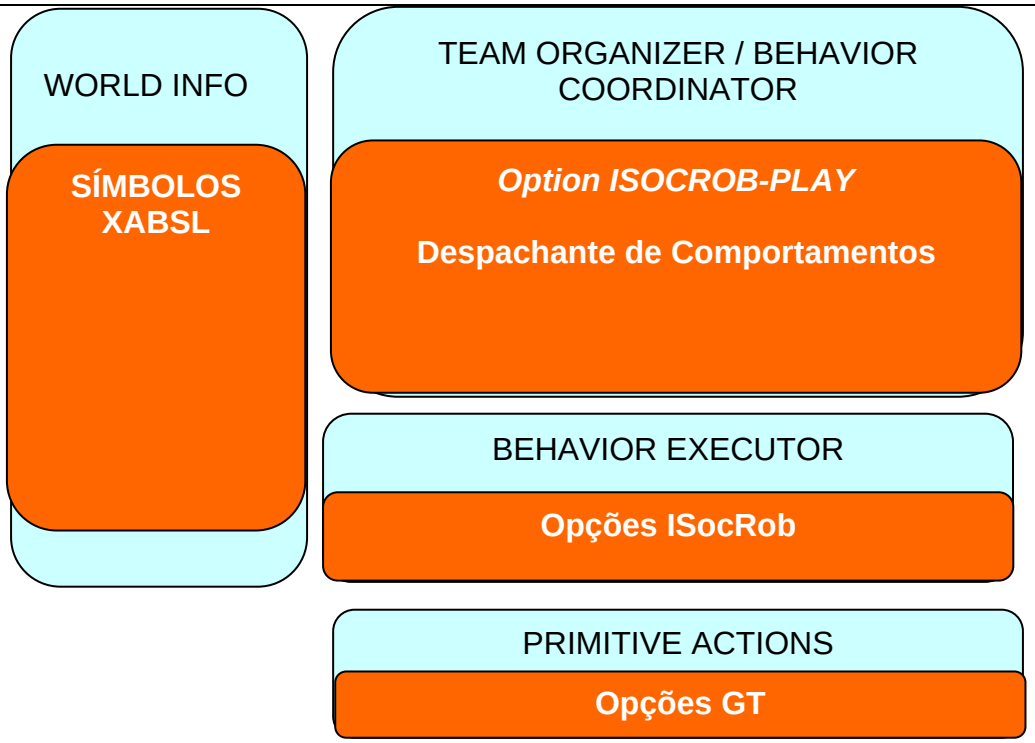


Figura 4.3- Correspondência da implementação com a arquitectura original ISocRob

4.4 – Comportamentos Individuais Definidos

Nesta secção serão mencionados os comportamentos individuais que iriam ser implementados. Alguns destes não chegaram, de facto, ao objectivo de serem implementados por algumas razões que serão descritas no Anexo A, onde irão ser formalmente descritos.

Um comportamento individual tem uma diferença fundamental quanto aos comportamentos relacionais que, logicamente, consiste no facto de estes comportamentos

serem executados apenas por um agente, não havendo necessidade de comunicação com os restantes elementos da equipa durante a sua execução.

No entanto existem semelhanças entre comportamentos relacionais e individuais, nomeadamente na estrutura de ambos, ou seja, tanto os comportamentos relacionais como os comportamentos individuais são descritos por conjuntos de acções primitivas executadas sequencialmente com regras de transição de uma acção para outra.

Os comportamentos individuais definidos foram:

- Maradona
- *Clear the ball*
- *Sweep*

4.5 - Comportamentos relacionais definidos.

A definição e implementação de comportamentos relacionais é o principal objectivo deste trabalho. Os comportamentos foram ou se prevê que sejam implementados serão enumerados nesta secção, sendo formalmente especificados em anexo, por meio de diagramas e tabelas.

Os comportamentos relacionais definidos no âmbito deste trabalho foram:

- *Attacking Throw In*
- *Attacking Kickoff*
- *Defensive Kickoff*
- *Defensive Throw In*

5 – Testes Realizados

Num trabalho desta envergadura é necessário testar os comportamentos implementados, tal como os sistemas que lhes servem de base.

Todos os sistemas e comportamentos foram testados, por forma perceber-se qual a probabilidade que há de virem a falhar.

Há, também, sistemas implementados que, na altura da entrega deste documento, apresentam ainda alguns erros graves, no entanto de resolução previsivelmente simples, mas no entanto funcionam no simulador. Daí haver duas sub-categorias para alguns testes, a primeira refere-se aos que foram feitos no simulador e a segunda nos robots reais, sendo previsível os melhores resultados no primeiro caso.

Os resultados dos testes serão apresentados no capítulo seguinte, em tabela construída para o efeito.

5.1 – Testes ao comportamento Individual Sweep

Este comportamento, dada a sua simplicidade, foi testado apenas nos robots reais. Os testes feitos incidiram sobretudo no funcionamento do comportamento já para fins competitivos, ou seja, se o comportamento já está suficientemente bom ou aproximadamente bom para ser utilizado em competições de futebol robótico.

O teste feito consistiu em ordenar ao robot a execução do comportamento e ir simulando as situações que poderiam surgir, ou seja, aproximar a bola do robot, para ver se este se aproximava dela e, quando a interceptasse, se finalizava correctamente a execução, e, caso a bola fosse afastada do robot, se este voltava para a sua posição de líbero.

5.2 – Testes ao comportamento relacional Attacking Kickoff

Foi pedido a dois robots que executassem este comportamento, cada um com o seu papel, e registou-se os resultados da sincronização e da execução com sucesso do

comportamento, que, previsivelmente, seriam mais baixos.

Os testes feitos ao comportamento *Attacking Kickoff* tiveram como principal objectivo a verificação da comunicação e sincronização entre dois companheiros de equipa, não tendo havido uma preocupação muito grande com falhas no passe entre os companheiros, apesar do robot que passa a bola o fazer em direcção do companheiro.

Todos os testes a este comportamento foram realizados em robots reais.

5.3 – Testes ao comportamento relacional Attacking Throw In

Tal como no comportamento anterior, a principal preocupação foi o registo da sincronização entre os companheiros de equipa neste comportamento, não devendo os resultados da execução correcta do comportamento ser interpretados, dado que este não se encontra optimizado.

Todos os testes foram, também, realizados nos robots reais.

5.4 – Testes ao comportamento relacional Defensive Kickoff

Para averiguar a qualidade deste comportamento foi pedido a dois companheiros de equipa que o executassem sincronamente.

Os testes feitos a este comportamento relacional prenderam-se, também, com a sincronização dos companheiros na execução do comportamento, mas também, por ser de menor grau de dificuldade do que os dois anteriores, na execução correcta e com sucesso.

5.5 – Testes ao despachante de escolha de companheiros

Os testes feitos ao despachante consistiram em corrê-lo e verificar se o companheiro escolhido seria o companheiro que o robot escolheria no caso teórico.

Estes testes foram executados tanto nos robots reais, como em simulador.

5.6 – Testes ao despachante de aceitação de comportamentos

Os testes realizados a esta funcionalidade consistiram em comparar os resultados da aceitação de comportamentos em robots reais e simulados com o modelo teórico.

Criaram-se condições fictícias de aceitação que eram verificadas (dado que as condições reais de aceitação ou não ainda não foram definidas) e comparava-se os resultados obtidos com os resultados teóricos.

Estes testes foram realizados em robots reais e simulados.

5.7 – Testes ao despachante de comportamentos

Este despachante apenas delega para outros comportamentos a execução de acções, mas dada a sua importância central na estrutura criada, foi necessário testar até à exaustão se esta delegação era bem efectuada.

Foi testado em robots reais e simulados.

5.8 – Testes ao comportamento isocrob-play *(*Behavior Coordinator*)*

Os testes realizados consistiram em correr este comportamento em todos os robots da equipa e anotar a coordenação entre eles.

Por haver ainda erros a corrigir na implementação, este comportamento foi testado em robots reais e simulados, apesar de ser previsível a ocorrência de várias falhas no primeiro caso.

6 – Resultados dos testes

6.1 – Resultados dos testes ao comportamento Individual Sweep

Em 20 testes realizados ao comportamento individual *sweep*, obteve-se os seguintes resultados:

	Realização com sucesso	Realização com Insucesso
Absoluto	17	3
Percentagem	85%	15%

Tabela 6.1 – Resultados dos testes ao comportamento individual *Sweep*

Os resultados consideram-se muito satisfatórios, havendo apenas uma razão para o insucesso, sendo esta o robot não ver a bola a passar a linha de 'perigo', por estar numa postura que não a pretendida.

6.2 – Resultados dos testes ao comportamento relacional Attacking Kickoff

Em 20 testes realizados foram registados os seguintes resultados:

	Boa sincronização	Sem sincronização
Absoluto	20	0
Percentagem	100%	0%

Tabela 6.2 – Resultados da sincronização no comportamento relacional *Attacking Kickoff*

Estes resultados são muito bons, no entanto quando se parte para o teste da execução correcta do comportamento, obtém-se os seguintes resultados:

	Passe bem executado	Passe executado para a frente	Passe não executado
Absoluto	2	17	1
Percentagem	10%	85%	5%

Tabela 6.3 – Execução do passe no comportamento *Attacking Kickoff*

6.3 – Resultados dos testes ao comportamento relacional **Attacking Throw In**

Em 20 testes realizados foram registados os seguintes resultados:

	Boa sincronização	Sem sincronização
Absoluto	20	0
Percentagem	100%	0%

Tabela 6.4 – Resultados da sincronização no comportamento relacional *Attacking Throw In*

Tal como nos resultados do teste anterior, a sincronização neste comportamento foi muito boa, no entanto os sucesso deste comportamento é mínimo, pois o companheiro que faz o passe ainda não o faz apontando para o companheiro com que faz o comportamento, desta forma obtemos os seguintes resultados:

	Passe bem executado	Passe mal apontado	Passe não executado
Absoluto	0	20	0
Percentagem	0%	100%	0%

Tabela 6.5 – Execução do passe no comportamento *Attacking Throw In*

Estes reflectem bem a necessidade de implementar mecanismos para apontar os passes, pois se se utilizasse este comportamento relacional em competição, a sua execução muito provavelmente iria ter como consequência a intercepção da bola por parte de um adversário.

6.4 – Resultados dos testes ao comportamento relacional **Defensive Kickoff**

Em 20 testes realizados a este comportamento, obteve-se os seguintes resultados na sincronização entre os companheiros:

	Boa sincronização	Sem sincronização
Absoluto	20	0
Percentagem	100%	0%

Tabela 6.6 – Resultados da sincronização no comportamento relacional *Defensive Kickoff*

Quanto aos resultados da execução do comportamento:

	Bola interceptada	Bola não interceptada
Absoluto	18	2
Percentagem	90%	10%

Tabela 6.7 – Resultados da execução do comportamento relacional *Defensive Kickoff*

Os casos em que a bola não foi interceptada, deveram-se ao mau posicionamento do marcador da zona que não viu a bola entrar no seu campo de acção, tendo esta sido interceptada, mas pelo companheiro de equipa.

6.5 – Resultados dos testes ao despachante de escolha de companheiros

Em 20 testes feitos ao despachante de escolha de companheiros, obteve-se o seguinte resultado, no simulador.

	Coincidente com modelo teórico	Não coincidente com modelo teórico
Absoluto	20	0
Percentagem	100%	0%

Tabela 6.8 – Resultados aos testes de escolha de companheiros em simulador

Nos robots reais obteve-se:

	Coincidente com modelo teórico	Não coincidente com modelo teórico
Absoluto	18	2
Percentagem	90%	10%

Tabela 6.9 – Resultados aos testes de escolha de companheiros em robots reais

A principal razão para estes resultados serem piores que no simulador, prende-se com a escolha dos companheiros ser, várias vezes, realizada consoante as posições comunicadas pelos companheiros de equipa havendo, por vezes, erros na auto-localização destes, erros estes inexistentes no simulador.

6.6 – Resultados dos testes ao despachante de aceitação de

comportamentos

Em 20 testes realizados ao despachante de aceitação de comportamentos, tanto no simulador, como em robots reais, obteve-se os seguintes resultados:

	Coincidente com modelo teórico	Não coincidente com modelo teórico
Absoluto	20	0
Percentagem	100%	0%

Tabela 6.10 – Resultados aos testes de aceitação de comportamentos em simulador e robots reais

Estes resultados são muito bons, mas, caso se utilize como condição de aceitação de comportamentos relacionais algumas variáveis que possam ser passíveis de erro, estes poderão vir a ter uma taxa de sucesso mais baixa, mas o funcionamento do mecanismo apresenta muito bons resultados.

6.7 – Resultados dos testes ao despachante de comportamentos

Tanto em simulação como nos robots reais, o despachante de comportamentos teve um funcionamento muito bom, a que não será alheio a sua simplicidade. A sua taxa de sucesso foi de 100%, em 20 repetições do teste, sendo os resultados apresentados na tabela seguinte:

	Executou o comportamento correcto	Não executou o comportamento
Absoluto	20	0
Percentagem	100%	0%

Tabela 6.11 – Resultados aos testes de despacho de comportamentos em simulador e robots reais

6.8 – Resultados dos testes ao comportamento **isocrob-play *(*Behavior Coordinator*)***

Os testes realizados consistiram em correr este comportamento em todos os robots da

equipa e anotar a coordenação entre eles. Em 20 execuções obteve-se os seguintes resultados:

	Executou coordenadamente	Não executou o coordenadamente
Absoluto	17	3
Percentagem	85%	15%

Tabela 6.12 – Resultados aos testes ao comportamento *isocrob-play* em simulador

Os erros na execução incorrecta foram erros grosseiros em que todos os robots executavam o mesmo comportamento com o mesmo papel, não tendo sido a sua origem identificada, apesar de a sua resolução ser aparentemente simples.

Nos robots reais o caso é mais problemático, não havendo qualquer espécie de coordenação, sendo esta controlada explicitamente no código.

Os resultados nos 20 testes em robots reais foram:

	Executou coordenadamente	Não executou o coordenadamente
Absoluto	0	20
Percentagem	0%	100%

Tabela 6.13 – Resultados aos testes ao comportamento *isocrob-play* em robots reais

7 - Conclusões

O trabalho até agora realizado deixa antever um futuro que parece ser promissor para as equipas de futebol robótico do grupo ISocRob.

Neste momento há cada vez mais a preocupação, e o trabalho até agora realizado tem reflectido isso mesmo, de tornar o processo de desenvolvimento de comportamentos cada vez mais metódico e formal, sendo por isso a sua implementação simplificada.

Neste momento a definição da *framework* para comportamentos relacionais encontra-se incompleta por incapacidade de resolver alguns problemas inesperados no desenvolvimento de *software*, até à altura da entrega deste relatório, problemas estes que, apesar de inviabilizarem a execução correcta dos comportamentos definidos, deverão ser de resolução simples.

Pelo que já foi feito há razões para acreditar na possibilidade de participar com bons resultados nas competições de futebol robótico, bastando apenas corrigir alguns erros que ainda têm lugar no código realizado, criar novos comportamentos relacionais e individuais e limar os já existentes de modo a que se possa ter uma equipa realmente competitiva e com grande interesse científico.

O trabalho a realizar de futuro deverá passar pela formalização e criação de mecanismos de estratégia da equipa, como a selecção de táctica consoante o resultado actual, e, talvez, tornar possível o alargamento de comportamentos relacionais a mais que dois elementos da equipa.

Anexo A – Descrição de comportamentos

Neste anexo serão descritos em maior detalhe todos os comportamentos anteriormente enumerados.

Em primeiro lugar serão definidos e formalizados os comportamentos individuais e em seguida os comportamentos relacionais.

Em ambos os casos os comportamentos serão definidos por uma rede de Petri consistindo nas tarefas, na sequência pela qual estas deverão ser executadas e coordenação das mesmas. No caso dos comportamentos relacionais será também apresentada uma tabela de 'tradução' da rede de Petri.

Tabelas e redes de comportamentos.

Cada estado dos comportamentos abaixo especificados baseiam-se em 3 conceitos essenciais:

- Pré-Condição
- Acção
- Transição

Pré-Condições são as condições ou informação sobre o mundo que o agente deverá ter para poder executar as acções constantes em cada estado.

Por acção designa-se as operações que o agente (no caso, um robot) deverá exercer sobre o mundo por forma a atingir o objectivo, que no nosso caso é a execução com sucesso do comportamento.

Transições são os **eventos** que ocorrem e que serão associados às alterações no mundo que as acções dos agentes deverão produzir.

Pré-Condições.

Por pré-condições designa-se a informação que o agente deverá ter sobre o mundo por forma a poder executar as acções listadas em cada estado. Estas têm uma relação muito próxima com os eventos ou transições, dado que a ocorrência de eventos irá alterar a informação existente sobre o mundo que irá permitir a execução, ou não, de determinados comportamentos.

Como pré-condições necessárias à execução das acções de um estado poderemos ter a posse de bola por parte do nosso robot, ou a posição de um outro robot ser defensiva,

ofensiva assim como o papel de cada um dos intervenientes nos comportamentos.

As pré-condições consideradas nas tabelas e redes foram as seguintes:

- *hasBallPossession* – Esta condição tem o seu valor verdadeiro quando um elemento da equipa tem a bola controlada.
- *isNearSpot* – Esta condição tem o seu valor verdadeiro quando o robot se encontra suficientemente perto de um ponto designado em *(x,y)*.
- *isNearBall* – Esta condição tem o seu valor verdadeiro quando o robot se encontra suficientemente perto da bola, ou seja, a diferença entre distância à bola e uma distância pré-definida é inferior a zero.
- *ballInRange* – Esta condição tem o seu valor verdadeiro quando o robot se encontra a uma distância inferior a um determinado valor da bola que é fornecido ao predicado.
- *ballHasMoved* – Esta condição tem o valor verdadeiro quando a bola se moveu uma distância mínima nos últimos *t* segundos.
- *ballWasPassed* – Esta condição tem o valor verdadeiro quando um jogador da equipa efectuou um passe com sucesso nos últimos *t* segundos.

Algumas destas condições são comunicadas (no caso relacional), sendo neste caso precedidas do nome Teammate (ex: TeammateIsNearBall).

Acções.

Como vem explicado acima, em cada estado o agente deverá efectuar uma ou mais acções sobre o ambiente com vista a atingir o objectivo que, no nosso caso, é a execução com sucesso de cada comportamento.

No caso do futebol robótico faz sentido que estas acções estejam relacionadas com futebol, havendo um paralelo com o futebol humano, no entanto, dada as características do jogo em si, faz sentido definir 2 grupos de acções:

- Acções de comunicação: acções que não afectam directamente a informação sobre o mundo, mas sim informações sobre o estado do robot no comportamento, providenciando informação necessária para manter robots comprometidos no comportamento relacional ou, caso seja caso disso, romper o compromisso, assim como a coordenação dos robots no comportamento.
- Acções primitivas sobre o ambiente: acções que alteram o estado actual do mundo, produzindo **movimento** quer dos robots quer da bola.

As acções primitivas consideradas nas tabelas e gráficos são as seguintes:

- *moveToBall* – Esta acção primitiva consiste numa procura da bola e deslocação em direcção a esta.
- *moveToSpot* – Esta acção primitiva consiste numa deslocação até a um ponto pré-definido em *(x,y)*.
- *passBallToSpot* – Esta acção primitiva consiste numa rotação e passe para um ponto pré-definido em *(x,y)*.
- *passBallToTeamMate* – Esta acção primitiva consiste numa rotação e passe para a posição *(x,y)* onde se encontra um companheiro de equipa.
- *controlBall* – Esta acção primitiva consiste num controlo de bola, quando esta se encontra suficientemente perto, por forma a ficar dominada pelo agente.
- *holdUpBall* – Esta acção pressupõe o domínio de bola (*controlBall*), mas é mais abrangente, sendo, não apenas o domínio e controlo de bola, mas também a retenção desta durante algum tempo, inferior a 3 segundos para não incorrer na infracção de *Ball Holding*, prevista nas regras da 4LL.
- *manMark* – Esta acção primitiva consiste em perseguir um jogador específico da equipa adversária.

Existem, também, acções que são específicas a cada um dos comportamentos que irão ser descritos, estas, por não serem partilháveis e por terem uma descrição mais complexa ou longa do que as acima descritas serão descritas junto aos comportamentos.

Transições:

As transições estão associadas a eventos, que alterarão a informação que o robot tem sobre o mundo. Para a definição dos comportamentos abaixo, foram considerados os seguintes eventos ou transições:

- *gotBall* – Este evento ocorre quando se dá a obtenção da posse de bola por parte do robot.
- *gotToPlace* – Este evento quando se dá a chegada do robot à posição *(x,y)* para onde se dirigia, ou a uma posição suficientemente perto desta.
- *passedBall* – Este evento ocorre quando um robot efectuou o passe da bola.
- *gotBallPossession* – Este evento ocorre quando um robot controlou com sucesso a bola.
- *ballMoved* – Este evento ocorre quando a bola, que estava parada há mais que

um certo tempo, ou foi reposta, foi chutada ou começou a deslocar-se.

- *gotNearBall* – Este evento ocorre quando o robot se encontra a uma distância suficientemente pequena da bola.
- *heldBall* – Este evento ocorre quando o robot começa a execução da acção primitiva *holdUpBall*.

A.1 Comportamentos Individuais

Nesta secção serão apresentadas as tabelas e gráficos dos comportamentos individuais especificados, assim como uma descrição mais detalhada do comportamento.

Alguns comportamentos não foram implementados por razões que serão explicadas em cada uma das secções. Outros foram alterados por forma a utilizarem informação que possa ser obtida através do código da GT. Em ambos os casos será descrita a implementação, caso tenha sido realizada, ou as razões pelas quais não foi.

Para muitos dos comportamentos será necessária uma descrição do campo, sendo por isso recomendável uma representação deste, que é mostrada na figura A.1.1, incluindo dimensões.

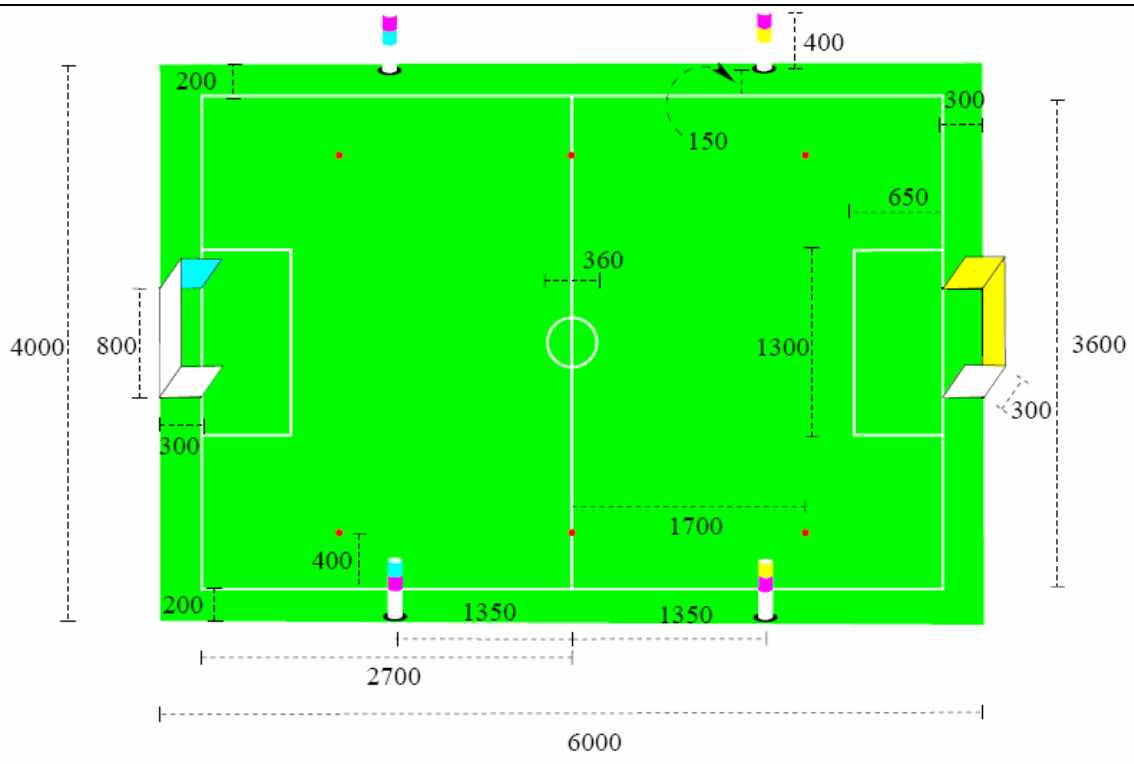


Figura A.1.1 – Dimensões do campo da liga 4LL

Caso seja necessário na descrição dos comportamentos abaixo descritos, poderá ser apresentada uma imagem com uma escala semelhante não contendo, por questões visuais de facilidade de compreensão da informação, as dimensões do campo.

A.1.1 Comportamento individual Maradona

Descrição

Este comportamento consiste em um robot que tenha o papel de avançado drible a bola e tente o remate quando estiver em posição de rematar à baliza.

O comportamento irá ser executado quando o jogador tem o papel de avançado, não tem oposição num determinado campo entre si e a baliza (definido abaixo) e se encontra com pelo menos um jogador adversário atrás de si.

A posição onde o remate irá ser efectuado será calculada e o jogador dirigir-se-á para esta, onde tentará o remate.

Pré Condições

- O jogador deverá ter o papel de avançado.
- O jogador deverá ter pelo menos um adversário atrás de si.
- O jogador deverá ter um ângulo aberto (sem oposição) para a baliza.
- O jogador deverá encontrar-se com a posse da bola.
- O jogador deverá estar orientado para a baliza.

Cálculo do Campo 'Maradona'

Como foi descrito, o jogador só irá tentar driblar a bola até à posição de remate (ou melhor, executar o comportamento Maradona) se tiver condições para progredir com a bola até à baliza. As condições para progredir com a bola terão de ser calculadas.

Tendo em conta as pré condições estabelecidas, há que medir alguns ângulos que para averiguar se é possível a progressão do jogador até à posição de remate.

O cálculo da orientação para a baliza é bastante simples, tendo em consideração o vector G que une o jogador à baliza e o vector O, que define a orientação do robot, basta calcular o ângulo entre estes dois vectores, utilizando a fórmula que relaciona o co-seno e o produto interno:

$$\cos(O \wedge G) = \frac{O.G}{\| O \| . \| G \|}$$

E, em seguida, calcular o seu inverso:

$$O \wedge G = \arccos(\cos(O \wedge G))$$

Se O^G for menor que uma determinada amplitude constante, imagine-se 20º, considera-se que o jogador está orientado para a baliza.

Um processamento análogo é feito em relação ao cálculo oposição até ao ponto de remate. Uma forma de o fazer é garantir que os adversários à frente do jogador não estarão numa distância mínima, podendo o jogador progredir. Tem então que se garantir que:

$$\| A_1 \|^2 < D^2$$
$$\| A_2 \|^2 < D^2$$

Em que D é a distância mínima, A1 e A2 são vectores que unem o jogador aos adversários. Um valor sugerido para D é 70cm. O facto das normas e as distâncias estarem ao quadrado vem do peso computacional do cálculo de raizes quadradas, podendo gerar alguns ganhos de performance.

No entanto garantir uma distância mínima não é suficiente pois os adversários podem encontrar-se no caminho entre o robot e o ponto de remate.

Desta forma dever-se-á calcular também o ângulo entre G e A1 e entre G e A2, tendo este que ser maior que um valor mínimo:

$$\cos(Ai \wedge G) = \frac{Ai.G}{\| Ai \| . \| G \|}$$

Logo,

$$Ai \wedge G = \arccos(\cos(Ai \wedge G))$$

Um valor mínimo sugerido para Ai^G é 30º.

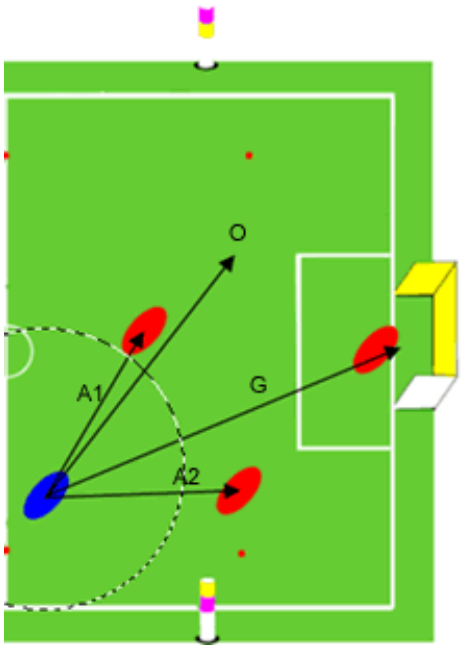


Figura A.1.2 – Cálculo do campo

Cálculo do ponto de remate

O ponto de remate será senpre a uma distância L da linha de fundo, tornando os cálculos menos complexos do que, por exemplo, calculá-lo num raio em torno da baliza e, desta forma, existe sempre a garantia que o jogador terá sempre algum ângulo de remate.

Figura A.1.3 – Cálculo do ponto de remate

O ponto de remate é de cálculo simples, utiliza-se a equação vectorial da recta que une o robot ao centro da baliza:

$(x, y) = G.K + R_0$, em que Ro é a posição do jogador e G o vector que une o jogador à baliza e que tem a forma: $G = (G_x, G_y)$.

Desdobrando-a obtém-se:

$y = G_y K + R_y$

$x = G_x.K + R_x$, ora o valor de X é conhecido e tem o valor (Xmax – L), temos que o valor de K para o ponto de remate é:

$$K = \frac{X_{\max} - L - R_x}{G_x}$$

Substituindo em Y:

$$y = \frac{G_y(X_{\max} - L - R_x)}{G_x} + R_y \,.$$

Temos então as coordenadas do ponto de remate, Pr:

$$P_r = (X_{\max} - L, \frac{G_y(X_{\max} - L - R_x)}{G_x} + R_y)$$

Rede de Petri.

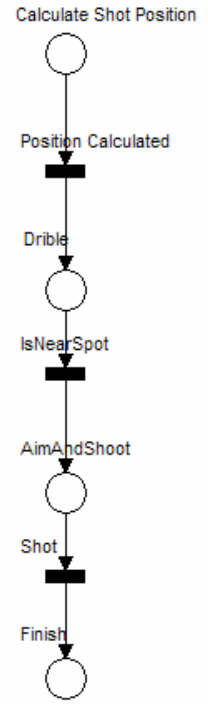


Figura A.1.4 – Rede de Petri para o comportamento individual Maradona

A.1.2 Implementação do comportamento individual Maradona

Apesar de parecer simples, a implementação do comportamento individual Maradona não foi realizada por duas questões essenciais:

- Incapacidade de localizar jogadores adversários
- Alteração nas regras dos campeonatos

A tendência nas competições Robocup é, cada vez mais, a utilização de cooperação entre os elementos das equipas participantes. Sendo assim este comportamento seria desactualizado pouco depois da sua especificação no primeiro relatório apresentado, pois não é permitido a um jogador segurar a bola durante mais de 50 cm, de acordo com as novas regras.

A.1.3 Comportamento individual *Clear the Ball*

Descrição

Este comportamento individual é um comportamento defensivo de emergência, sendo portanto executado por defesas. O seu objectivo é aliviar a bola para longe da área, não havendo qualquer preocupação em construir jogadas.

O comportamento só deverá ser executado quando existirem 2 jogadores adversários no meio campo defensivo. Quando tal acontece o jogador irá aproximar-se da bola, tentando interceptá-la e mal o faça deverá voltar-se para o meio campo e tentar chutar a bola segundo a forma abaixo descrita.

Pré Condições

- Existem 2 jogadores adversários no meio campo defensivo.
- O jogador deverá ter o papel de defesa.
- O jogador é o mais próximo da bola, de entre todos os companheiros de equipa.
- A bola encontra-se no último terço de terreno.

Cálculo do ponto de alívio

Como foi referido acima o comportamento é executado em situações de emergência, não havendo preocupação de construir jogadas. No entanto é um ponto fulcral definir-se cuidadosamente a forma como aliviar a bola pois um cálculo errado poderá fazer com que a bola seja colocada num adversário, criando mais perigo do que originalmente.

O valor obtido como posição para onde chutar a bola deverá ser função das posições dos robots adversários, sendo que não se tentará aliviar a bola para uma zona povoada por um, ou mais, adversários e também deverá ser função da própria postura do jogador da equipa ISocRob, pois o ponto para onde se vai aliviar a bola deverá exigir o mínimo de

rotação possível, pois o jogador pode perder a bola enquanto se posiciona para o passe, ou ser punido por uma infracção *Ball Holding*.

Quando o robot obtém a posse de bola irá calcular uma função de qualidade para várias soluções possíveis. Esta função irá atribuir um valor maior para os pontos onde a equipa ISocRob estará mais livre de perigo.

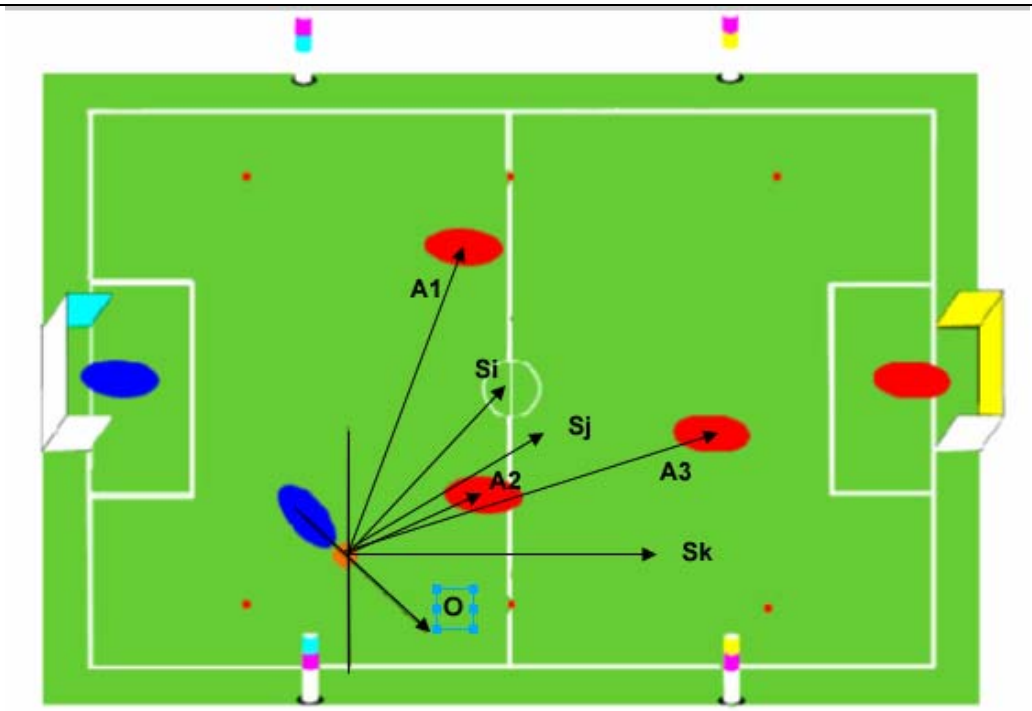


Figura A.1.5 – Soluções possíveis para *Clear The Ball*

Uma sugestão é utilizar uma função que considere todos os ângulos que apontem para o meio campo adversário e discretizá-los por forma a considerar apenas um número finito de soluções. Estas soluções serão apenas as que consistam em chutar a bola para a frente, não sendo considerados alívios para trás.

Na figura A.1.5 estão desenhadas algumas possíveis soluções, representadas por vectores S, os vectores S terão uma amplitude fixa entre eles, tendo esta amplitude fixa que oferecer um bom compromisso entre qualidade da solução e tempo de processamento.

Um valor que, ainda que empiricamente, parece aceitável é de 5º, oferecendo 180/5 = 36 soluções possíveis nas quais haverá uma solução que será muito próxima da solução óptima.

Para cada uma das soluções Sk será calculado o ângulo entre estas e o vector de orientação do jogador da equipa ISocRob, quanto menor este ângulo melhor a solução,

contribuindo, então, este valor de forma negativa para a qualidade da função final.

A função de qualidade da solução *k* será então definida, por enquanto, por:

$$f_k = -O \wedge S_k$$

Há, também, que considerar as posições dos robots adversários. Considera-se então, apenas os adversários que se encontrem a menos de 1.5 metros de distância não terão influência no cálculo da função, pois estarão demasiado longe para criar perigo.

Sejam então Ai, os vectores que unem as posições de robots adversários à posição do jogador da equipa ISocRob que irá executar o comportamento, construir-se-á uma tabela com os valores dos vectores Ai, tais que $\| Ai \|^2 > 2.25$ (condição de distância mínima).

O cálculo da função de qualidade para a solução *k* irá, então, assumir o seguinte valor:

$$f_k = (\sum_{i=1}^{3} (Ai \wedge Sk)) - O \wedge Sk .$$

Na figura A.1.5, a solução considerada seria enviar a bola para a linha lateral, dado que é a que maximiza a função F.

Rede de Petri

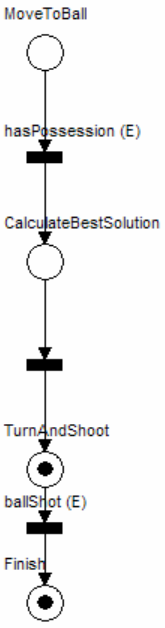


Figura A.1.6 – Rede de Petri para o comportamento individual *Clear the Ball*.

A.1.4 implementação do comportamento *Clear The Ball*

Por ser impossível a localização de jogadores adversários utilizando a arquitectura da GT, foi decidido não implementar este comportamento. No entanto seria possível adaptar o próximo comportamento a ser descrito (*sweep*), fundindo-o com este.

Desta forma teríamos um jogador a fazer o papel de *sweeper* mediante algumas condições (suponhamos, a equipa ter de defender uma vantagem), pontapeando a bola para longe sempre que se desse o caso de esta passar para uma posição perigosa no campo.

A.1.5 Comportamento individual *Sweep*

Descrição

O comportamento individual *sweep* é um comportamento puramente defensivo, sendo, desta forma, apenas executado por jogadores com o papel de defesa.

O jogador que executar este comportamento irá posicionar-se atrás dos restantes companheiros de equipa em situações de perigo para a própria baliza, tentando interceptar a bola caso esta passe uma linha imaginária no meio campo defensivo, sendo o jogador que executa o comportamento um líbero, como se designa em linguagem futebolística.

O comportamento irá chegar ao fim quando o jogador interceptar a bola, ou quando houver um *timeout* ou quando houver alguma mensagem por parte do *Behavior Coordinator* ordenando o jogador à interrupção do comportamento.

Pré Condições:

- O jogador tem o papel de defesa.
- A bola está na posse de um adversário.
- Não há outro jogador (excepto o guarda-redes) mais atrasado no terreno.
- Não existe outro jogador a executar este comportamento.

Posição de Sweep

A posição de *Sweep* deverá ser constante sempre que possível. O jogador irá sempre posicionar-se o mais possível recuado no terreno, tendo, no entanto, em conta que as regras [4] definem que o jogador nunca deverá estar totalmente dentro da área. Desta forma a posição que o jogador deverá ocupar será exactamente na entrada da área como está indicado na figura A.1.7. A linha a partir da qual o jogador avançará para a bola está desenhada a tracejado preto.

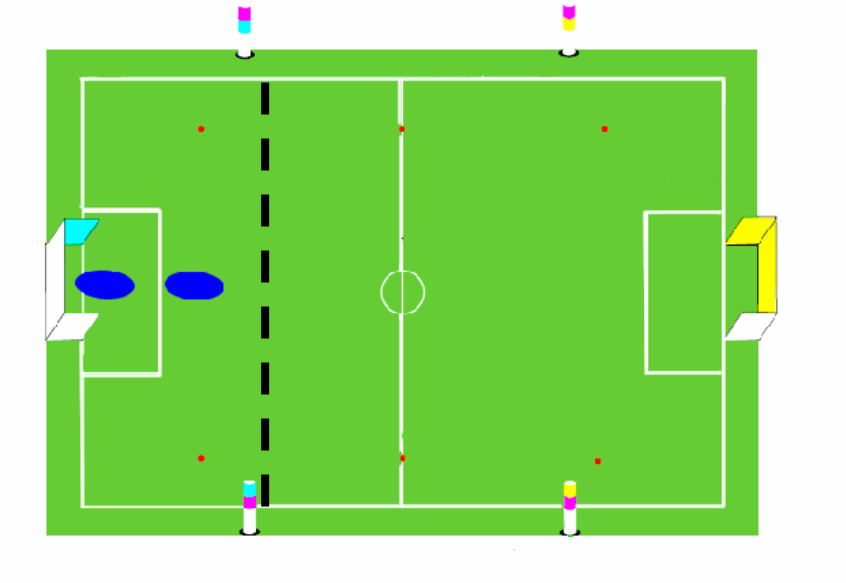


Figura A.1.7 – Posicionamento do *sweeper*.

Rede de Petri

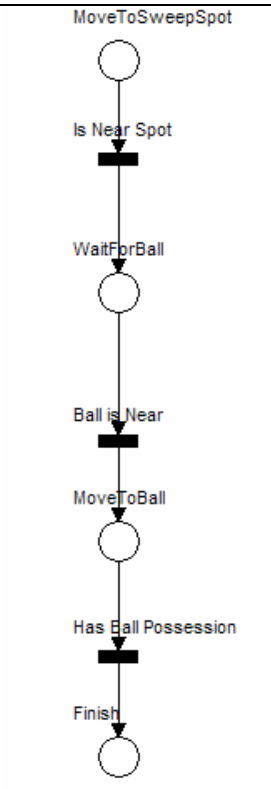


Figura A.1.8 – Rede de Petri para o comportamento individual *Sweep*.

A.1.6 Implementação do comportamento relacional *Sweep*

A implementação deste comportamento foi realizada e testada com bons resultados (ver capítulo 6).

A.2 Comportamentos relacionais

Foram definidos 4 comportamentos relacionais que serão descritos em pormenor abaixo. Os comportamentos podem ser ofensivos ou defensivos e têm como característica diferenciadora em relação aos comportamentos individuais o facto de serem executados por mais que um jogador e exigirem a comunicação entre os intervenientes.

Tenta-se que os comportamentos coincidam ao máximo com a definição em 2.2 de comportamentos relacionais. Ou seja, caso a clausula de escape seja accionada, o jogador que o detecta irá tentar fazer com que o seu companheiro no comportamento desista de o fazer, acabando, assim com o compromisso estabelecido.

Desta forma, se um jogador detecta que perdeu a bola, falhou um passe ou, simplesmente, esperou demasiado tempo por outra acção de outro, este irá desistir do comportamento e irá comunicar esta desistência ao seu companheiro no comportamento, fazendo que o companheiro desista de fazer o comportamento, dado que se torna irrelevante para a obtenção do objectivo.

Também é rompido o compromisso assumido pelos dois jogadores no caso em que um deles chegue à conclusão que o objectivo está cumprido com sucesso.

A.2.1 Comportamento relacional *Attacking throw in*

Descrição

Como o nome indica, o comportamento *attacking throw in*, define a forma de dois companheiros de equipa se posicionarem e interagirem quando a bola é reposta num dos pontos de reposição de foras do meio campo para a frente.

Para decorrer o comportamento em causa, o robot *kicker* deverá encontrar-se em posição ofensiva, assim como o robot *receiver*. O robot *receiver* posiciona-se de forma a ter ângulo aberto para a baliza e espera o passe. Quando o robot *receiver* recebe o passe o comportamento é concluído com êxito.

Determinação do jogador receiver

Um dos pontos importantes e que precede a execução do comportamento relacional é a determinação de qual o jogador que irá recepcionar o passe.

Este deverá encontrar-se em posição ofensiva (à frente do meio campo) e simultaneamente atrás da linha paralela à linha de fundo à distância *L*, como indicado na

figura A.2.1.

Para haver maior probabilidade de recepção do passe em condições, o jogador escolhido deverá ser o que se encontra a menor distância do ponto onde se encontra a bola, desde que se encontre na zona entre a linha *L* e o meio campo.

Cálculo do ponto de recepção do passe

Visto haver diversos pontos para os quais o jogador poderá deslocar-se para receber a bola, adoptar-se-á uma estratégia de posicionamento que ofereça um bom compromisso entre a possibilidade de rematar à baliza e a possibilidade de receber o passe em condições.

Como noutros comportamentos utiliza-se uma discretização dos pontos para onde o robot se poderá deslocar. Para cada um destes pontos irá ser calculada a abertura para um possível remate à baliza. Considerar-se-á então a abertura do ângulo de remate, tendo em conta os adversários mais próximos do ponto, e a distância ao jogador que irá passar a bola.

Todos os pontos solução Sj situar-se-ão a uma distância L da linha de fundo. Se considerarmos *n* soluções estas irão dividir o segmento de recta no qual se encontram todas as soluções em *n* partes.

Na figura seguinte considera-se *n* = 9.

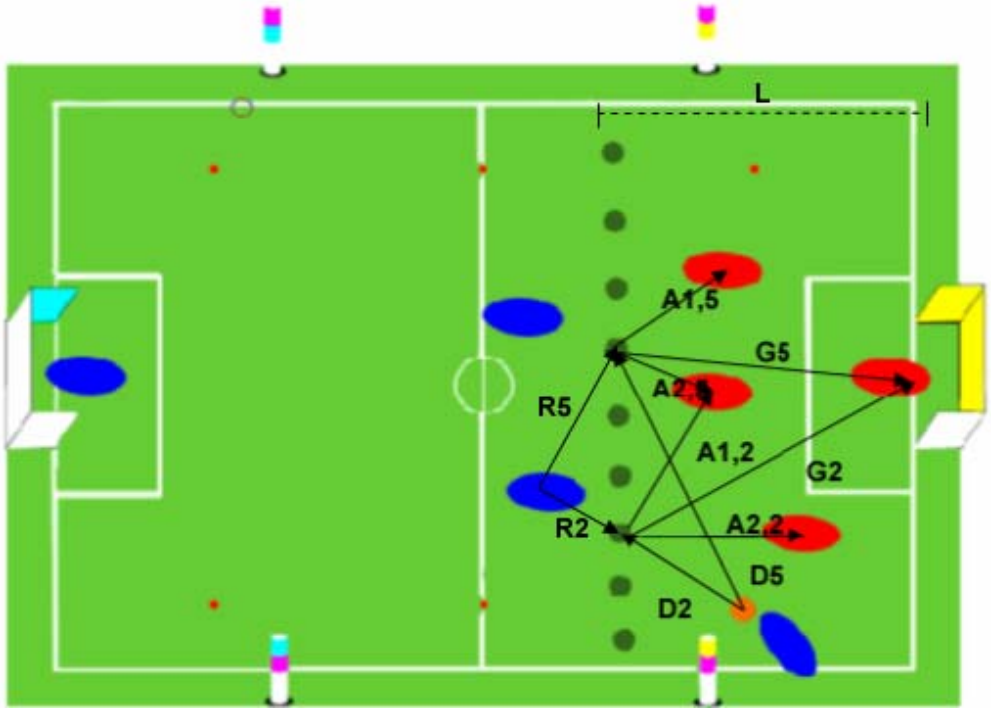


Figura A.2.1 – Soluções possíveis para o ponto de recepção do passe.

Escolhe-se então a solução com o maior valor de:

$$f_j = \frac{\sum_{i=1}^{2} G_j \wedge A_{i,j}}{\| D_j \| + \| R_j \|}$$, em que:

Dj – vector que une a bola à ao ponto solução Sj.

Rj – vector que une o ponto ao receptor do passe em cada solução Sj.

Gj – vector que une o ponto da solução Sj ao centro da baliza

Ai,j – vector que une o ponto solução Sj aos 2 adversários mais próximos.

Pré condições

- Robot *kicker* encontra-se em posição ofensiva.
- Robot *receiver* encontra-se em posição ofensiva.
- A bola foi reposta num dos pontos de reposição de *throw in* em posição ofensiva.
- Robot *kicker* está mais próximo da bola que qualquer adversário.

Rede de Petri

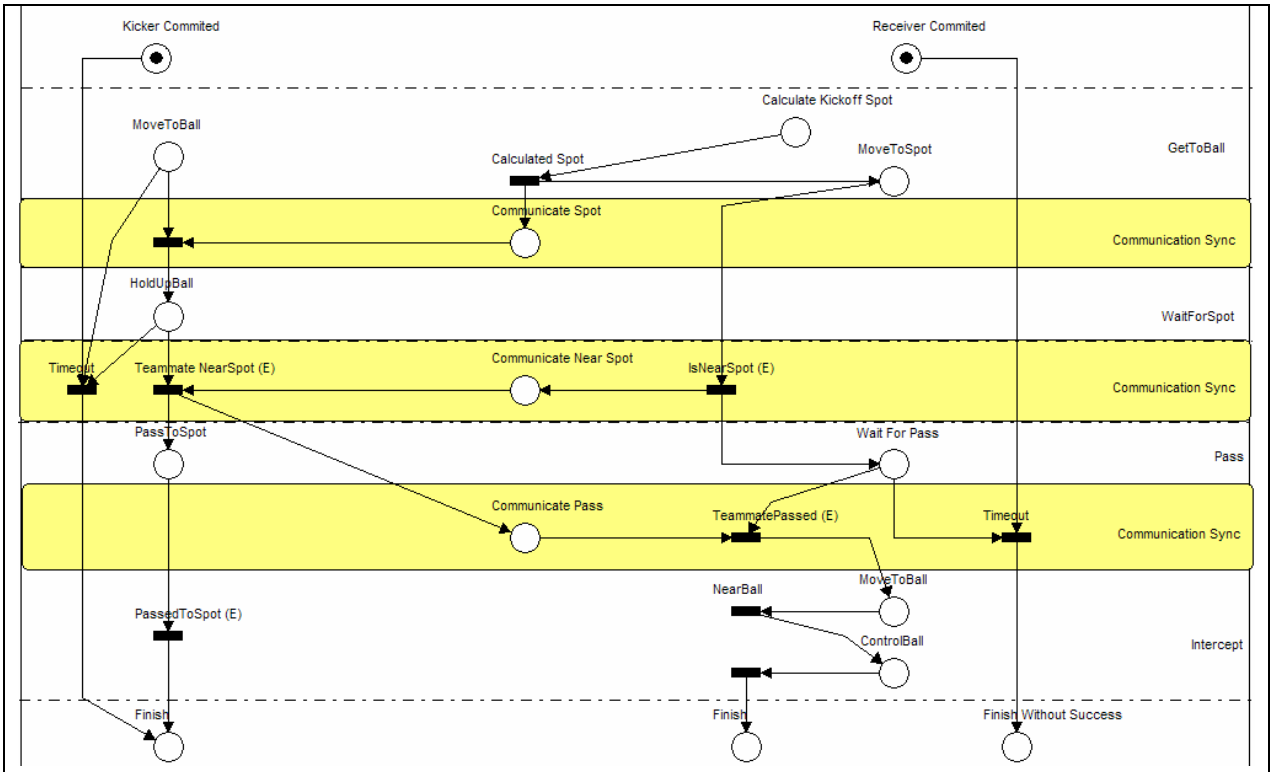


Figura A.2.2 – Rede de Petri para o comportamento relacional *Attacking Throw In*

Relational Behavior Phase	*Setup*		*Loop*			*End*	
Relational Behavior State	*Request*	*Accept*	*Preparing*	*Pass*	*Intercept*	*Success*	*Insuccess*
Robot Kicker	Procura o colega melhor posicionado para receber o passe e pede-lhe a iniciação do comportamento relacional (o mais próximo da baliza e com menos adversários a uma determinada distância).	Aceita o compromisso.	**Pré condições**: • Aceitação do compromisso **Acções:** • Movimenta-se para a bola. (*moveToBall*) **Transição:** • Obteve a posse de bola. (*gotBall*)	**Pré condições:** • Robot *kicker* encontra-se com a posse de bola. • Recebeu a informação sobre posição para onde passar. O robot receiver deverá estar perto desta. **Acções:** • Passa a bola para a posição de remate. (*passBallToSpot*). • Comunica que o passe foi feito. **Transição:** • Passou a bola. (*passedBall*) • Recebeu a confirmação que o *receiver* se encontra perto do ponto onde deverá receber a bola (*nearSpot*)			**Pré Condições:** Não obteve a posse de bola em tempo útil. Não obteve confirmação da colocação do *receiver* em tempo útil. **Acção:** • Comunicar insucesso ao companheiro de equipa.
Robot Receiver	É o colega melhor posicionado para um remate à baliza.	Aceita o compromisso.	**Pré condições:** • Aceitação do compromisso. **Acções**: • Calcula a posição de recepção do passe. • Movimenta-se para a posição calculada. (*moveToSpot*) • Comunica esta posição ao robot *kicker*. **Transição:** • Chegada ao ponto de remate. (*gotToPlace*)	**Pré-condições:** Encontra-se próximo do ponto de recepção da bola. **Acção:** Comunica ao kicker quando se encontra próximo da posição onde deverá receber a bola. **Transição:** A bola é passada pelo receiver (*ballPassed*) Recebe a confirmação do passe.	**Pré condições:** • A bola foi controlada pelo *kicker*. • O robot encontra-se suficientemente perto da posição onde deverá receber a bola. **Acções**: • O robot *receiver* aproxima-se da bola. (*moveToBall*) • O robot *receiver* controla a bola. (*controlBall*) **Transição:** • A bola está controlada pelo robot *receiver*. (*GotBallPossession*)	**Pré condições:** • A bola foi interceptada com sucesso após o passe ter sido feito. (*hasBallPossession*) **Acções:** • Assinalar como sucesso a execução do comportamento à BC.	**Pré Condições:** • Ter recebido informação de insucesso do robot *kicker*. • Não ter recebido confirmação do passe em tempo útil **Acção:** Comunicar insucesso ao companheiro de equipa. Assinalar como insucesso a execução do comportamento à BC.

Tabela A.2.1 – Descrição do comportamento relacional *Attacking throw in*.

A.2.2 Implementação do comportamento relacional *Attacking Throw In*

Este comportamento relacional foi implementado, mas, por restrições de tempo e incapacidade da arquitectura da GT em localizar jogadores adversários, o seu funcionamento denota algumas diferenças face à concepção original.

Devido à incapacidade em obter a informação sobre a posição dos adversários, o jogador que irá receber o passe deslocar-se-á para um de dois pontos pré-definidos na linha onde se localizam as soluções na concepção original, pontos localizados junto às linhas laterais, escolhendo um dos dois consoante estejam mais ou menos próximos. Não se tomam, então, em conta as posições dos adversários.

Por restrições de tempo não foi possível depurar o passe, na actual implementação, o jogador que irá realizar o passe irá passar 'ao acaso', tendo sido a principal área de interesse a coordenação entre os dois jogadores (ver cap. 6).

A.2.3 Comportamento Relacional *Defensive Throw In*

Descrição

Este comportamento relacional tem a intenção de melhorar as acções defensivas no caso de haver um *throw in* numa posição defensiva. Em traços largos o comportamento estabelece que os robots nele envolvidos deverão ter objectivos defensivos distintos de forma a não haver uma descoordenação defensiva que permita remates à baliza por parte dos adversários.

Um dos robots (*chaser*) irá tentar interceptar a bola, enquanto o outro (*marker*) irá cobrir o jogador adversário melhor posicionado, que assumiremos ser o jogador no meio campo defensivo com o ângulo mais aberto para um remate à baliza, posicionando-se entre este e o local onde está a bola.

Determinação do jogador melhor colocado

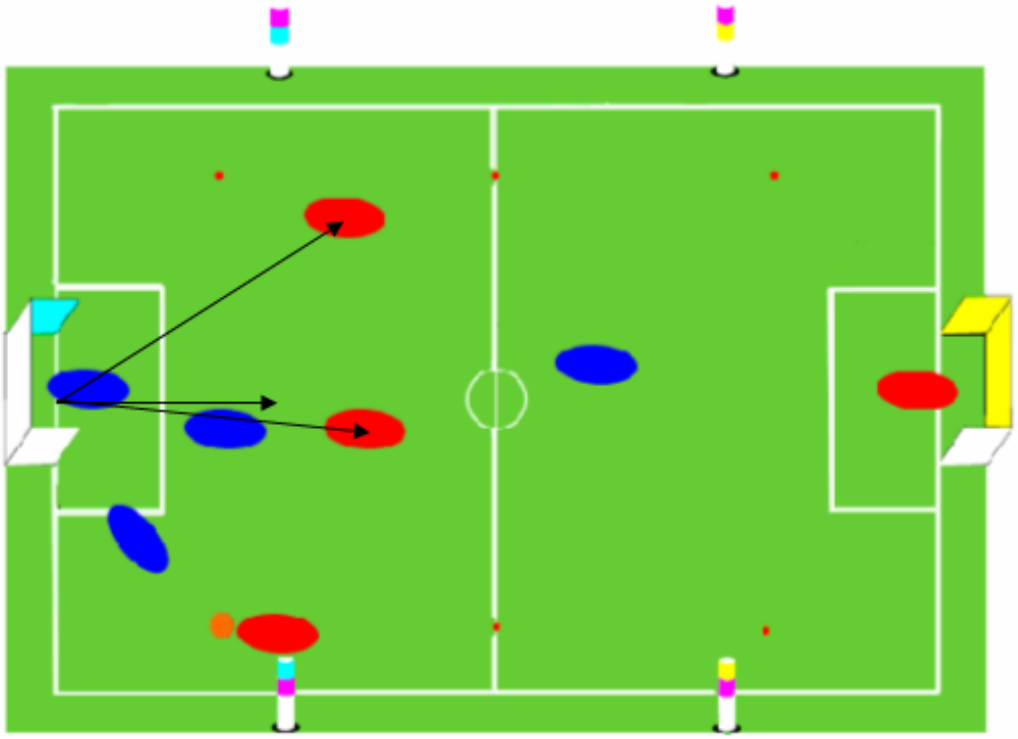


Figura A.2.3 – Determinação do jogador melhor colocado

A forma de determinação do jogador melhor colocado para criar perigo é medindo o ângulo entre os vectores que unem o centro da baliza aos jogadores adversários que se encontrem à frente do meio campo que seja normal às linhas de fundo (ver figura A.2.3)

Quanto menor for este ângulo mais perigosa será a posição do jogador adversário.

Há, no entanto, o problema de o jogador com uma posição calculada como sendo a mais perigosa de entre os adversários ser o jogador que tem a posse de bola. Para salvaguardar este caso retirar-se-ão da lista de possíveis jogadores em posições ofensivas não apenas os jogadores que se encontrem para lá da linha do meio campo, mas também o jogador que se encontra mais próximo da bola que é o identificado como sendo o marcador da fora.

Cálculo do ponto de marcação

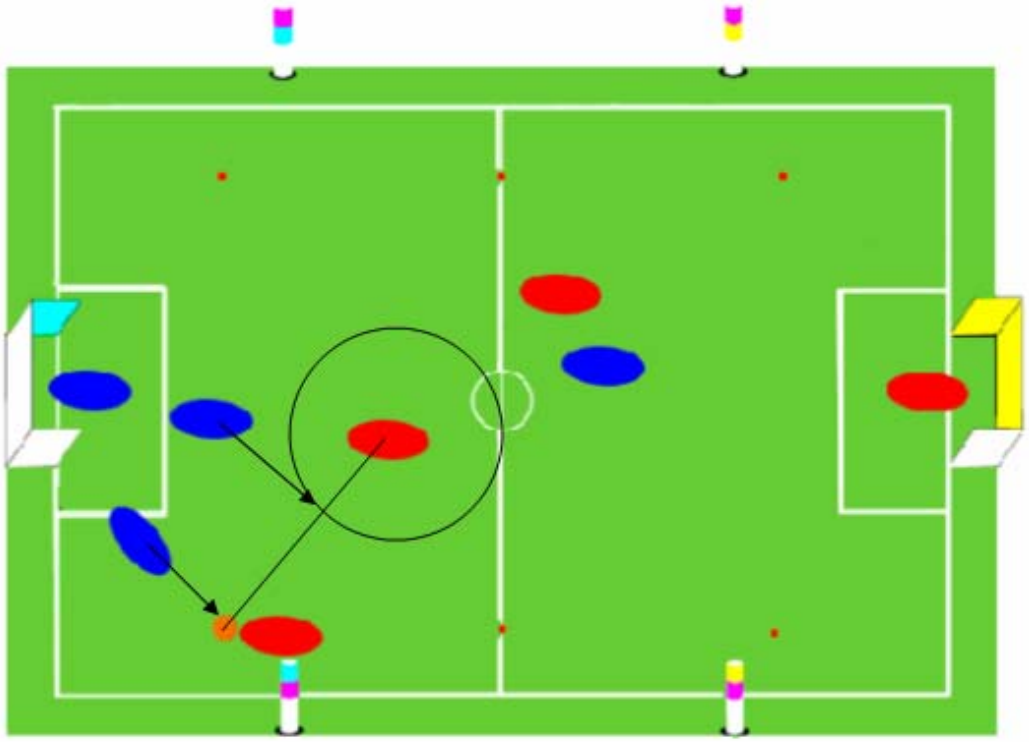


Figura A.2.4 – Cálculo do ponto de marcação *Defensive Throw In*

Considere-se uma distância pré-definida D, representando o raio de um círculo imaginário em torno do jogador melhor colocado para criar perigo para a equipa ISocRob. A forma de calcular o ponto de marcação é bastante simples e consiste em calcular a intersecção entre o círculo imaginário em torno do jogador melhor colocado e o segmento de recta que o une à bola.

Pré condições

- A bola foi reposta num ponto de reposição de foras no meio campo defensivo.
- A bola está mais próxima de um adversário do que de qualquer jogador da equipa.
- O robot *marker* deverá ter o papel de defesa.

Rede de Petri

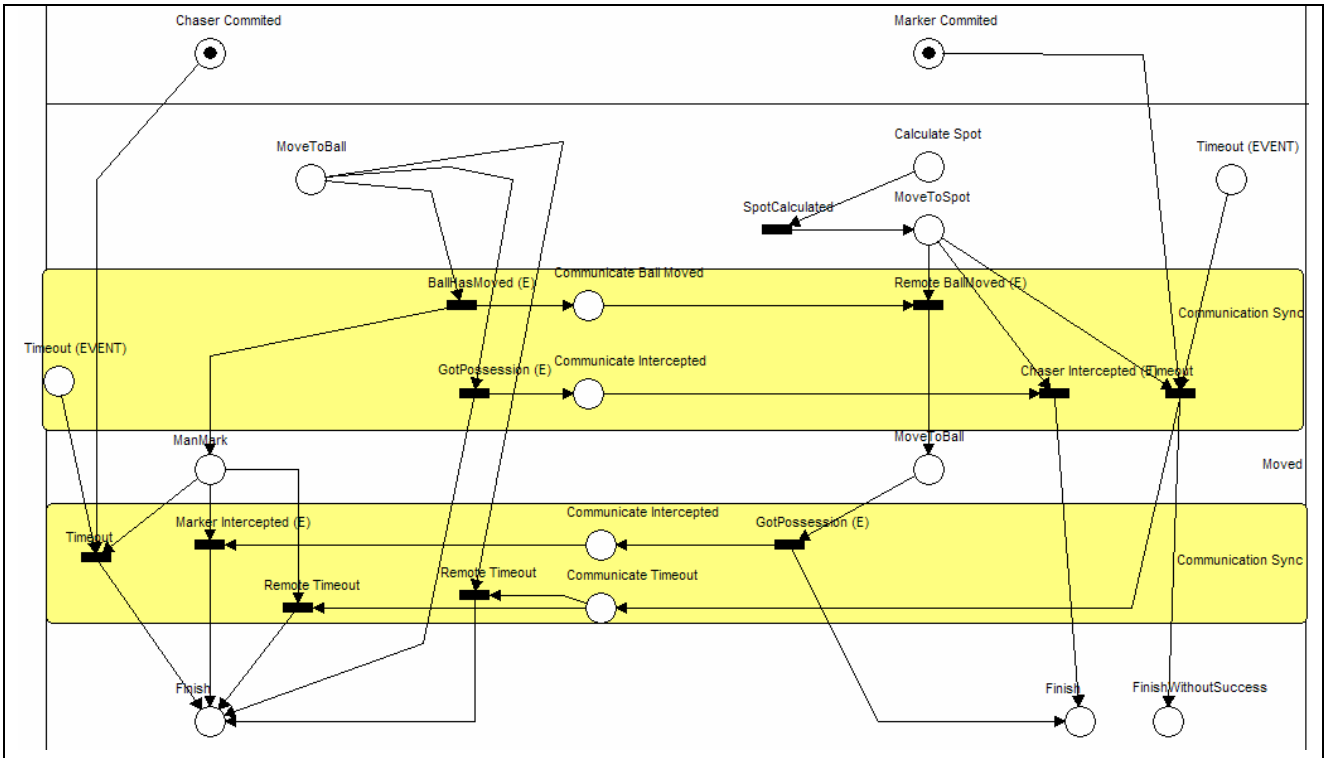


Figura A.2.5 – Rede de Petri para o comportamento relacional *Defensive throw in*

Relational Behavior Phase	*Setup*		*Loop*			*End*	
Relational Behavior State	*Request*	*Accept*	*In Throw-in Spot*	*Moved*	*Intercepted*	*Success*	*Insuccess*
Robot Chaser	Procura o colega mais próximo do jogador adversário com maior ângulo para remate à baliza que esteja no meio campo defensivo.	Aceita o compromisso.	**Pré-Condições:** • Aceitação do compromisso. **Acções:** • Persegue a bola. (*moveToBall*). **Transição:** • A bola movimentou-se (*ballMoved*). • Conquistou a posse de bola. (*gotBallPossession*)	**Pré-Condições:** • A bola movimentou-se (*ballHasMoved*). **Acções:** • Comunica ao robot *marker* que a bola se movimentou. • Acompanha o jogador que marcou a fora. (*manMark*) **Transição:** • A bola foi interceptada pelo *marker*. (*gotBallPossession*)		**Pré-Condições:** • A bola foi interceptada com sucesso. (*hasBallPossession*) **Acção:** • Comunicar ao robot *marker* que obteve a posse de bola.	**Pré Condições:** • Fica demasiado tempo em marcação individual. **Acção:** • Comunica o insucesso ao robot *marker*.
Robot Marker	É o robot mais próximo do robot adversário mais perigoso.	Aceita o compromisso.	**Pré-Condições:** • Aceitação do compromisso. • Papel é de defesa. **Acções:** • Calcula o ponto para onde se deverá dirigir. • Movimenta-se para o ponto calculado. (*moveToSpot)* **Transição:** • A bola movimentou-se (*ballMoved*)	**Pré-Condições:** • A bola moveu-se. (*ballHasMoved*) **Acções:** • Tenta interceptar a bola. (*moveToBall*) **Transição:** • A bola foi interceptada. (*gotBallPossession*)	**Pré-Condições:** • A bola foi interceptada. (*hasBallPossession*) **Acções:** • Comunica ao *chaser* que a bola foi interceptada.	**Pré-Condições:** • A bola foi interceptada com sucesso. (*hasBallPossession*) • Recebeu a confirmação que o robot marker obteve a posse de bola. **Acção:** • Assinalar como sucesso a execução do comportamento à BC.	**Pré Condições:** • Recebeu informação de insucesso do companheiro de equipa. • A bola não se movimentou em tempo útil. **Acção:** • Comunica o insucesso ao robot *chaser*. • Assinala como insucesso a execução do comportamento à BC.

Tabela A.2.2 – Descrição do comportamento relacional *Defensive throw in*

A.2.4 Implementação do comportamento relacional *Defensive Throw In*

Este comportamento não foi implementado pois depende muito obtenção da posição do jogador adversário. No entanto é possível vir-se a adoptar uma táctica semelhante à que é realizada no *Defensive Kickoff*, mas com resultados que se prevêem piores.

A.2.5 Comportamento Relacional *Attacking Kick Off*

Descrição

Este comportamento relacional serve para definir a forma como os jogadores deverão interagir quando ocorrer um pontapé de baliza a favor da equipa ISocRob. O funcionamento é muito simples, um dos robots deverá aproximar-se da bola, dominá-la e passá-la para um outro que se deverá encontrar-se numa posição com ângulo aberto para um eventual remate.

O funcionamento do comportamento é muito semelhante ao do *Attacking throw-in*, com a diferença de poder potenciar situações de maior perigo, dado que no pontapé de baliza não existe oposição adversária até à metade do meio campo adversário [4].

A diferença ao nível de estrutura ou definição do comportamento reside apenas na forma de calcular o ponto de recepção do passe.

Cálculo da posição de recepção da bola

Para tornar mais simples o cálculo da posição de recepção do passe considera-se um número discreto de soluções possíveis na linha do meio campo.

O jogador irá calcular uma função de qualidade que receberá como argumentos as posições dos adversários e calculará *n* valores da função, em que *n* é o número de pontos de recepção do passe especificados.

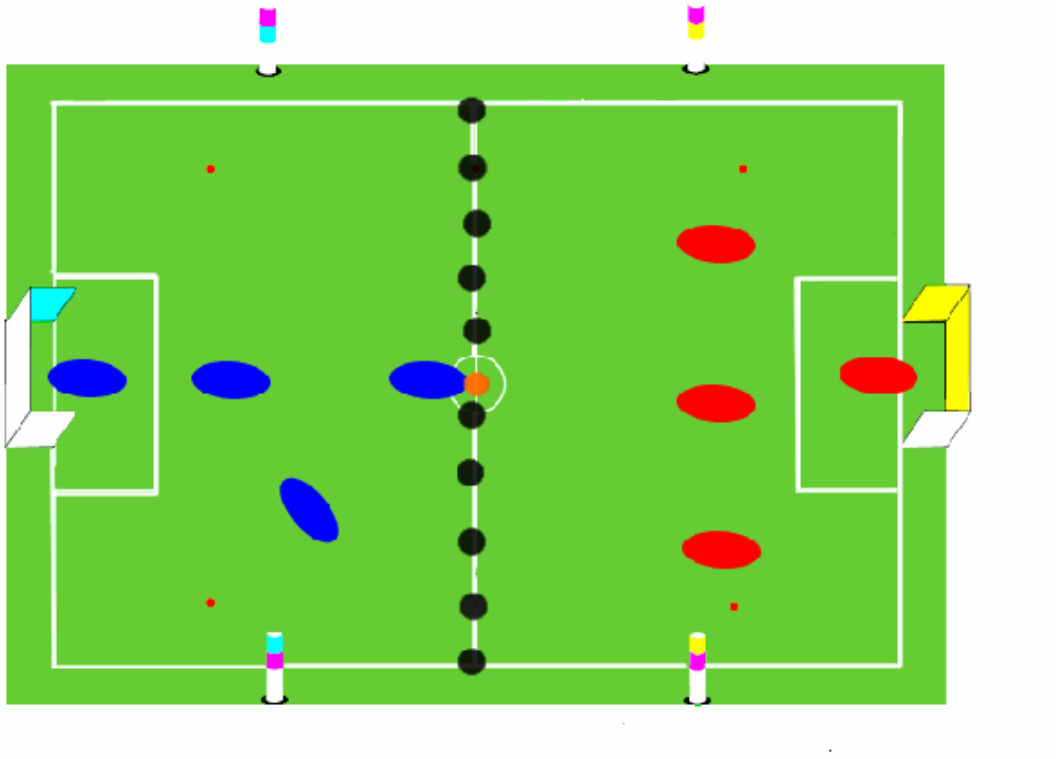


Figura A.2.6 – Divisão da linha do meio campo

Em primeiro lugar a linha de meio campo é dividida em *n* partes iguais, como é demonstrado com pontos pretos na figura A.2.6.

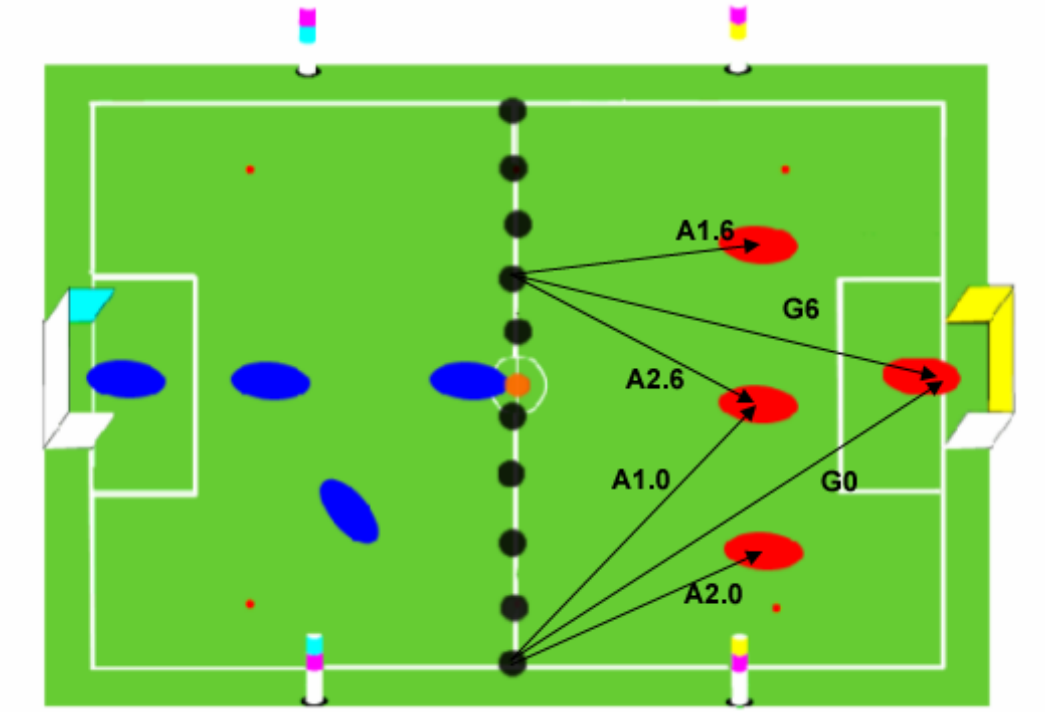


Figura A.2.7– Possíveis soluções para o comportamento relacional

Para cada um destes pontos, ou possíveis soluções, é calculado o vector que a une ao centro da baliza (vector G) e, igualmente, os vectores que unem os dois jogadores adversários (vectores A) mais próximos ao ponto, ver figura A.2.7

A solução escolhida será a que minimiza o valor:

$$f_j = \sum_{i=1}^{2} G_j \wedge A_{i,j}$$

Pré condições

- A bola encontra-se em posição de *kickoff*.
- O *kickoff* é a favor.
- O robot *receiver* tem o papel de avançado.

Rede de Petri

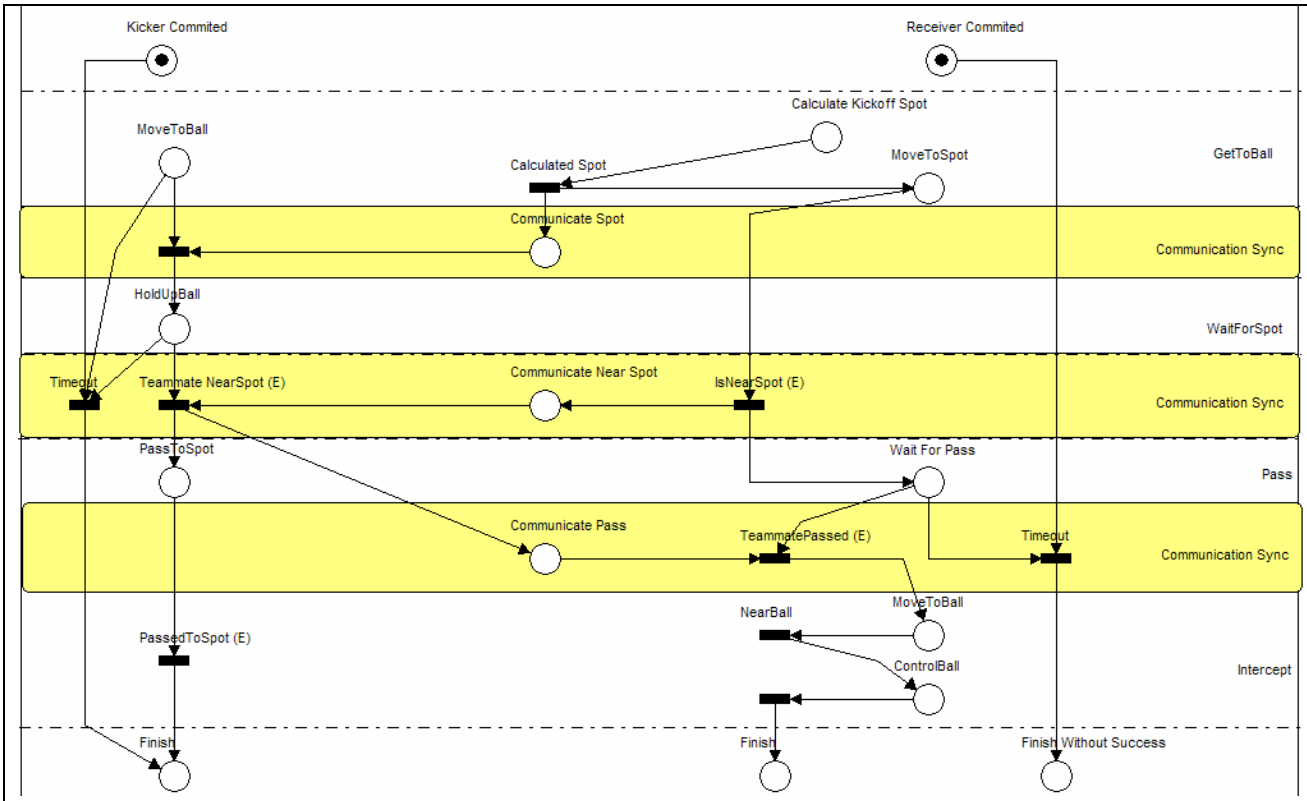


Figura A.2.8– Rede de Petri para o comportamento *Attacking kickoff*

Relational Behavior Phase	*Setup*		*Loop*				*End*	
Relational Behavior State	*Request*	*Accept*	*GetToBall*	*GotToSpot / PreparePass*	*Pass*	*Intercept*	*Success*	*Failure*
Robot kicker	O robot *kicker* inicia o comportamento relacional com um companheiro de equipa que tenha o papel de avançado.	Aceita o compromisso.	**Pré-Condições:** • Aceitação do compromisso. **Acção:** • Aproximação até à bola. (*moveToBall*) **Transição:** • O robot encontra-se suficientemente perto da bola. (*gotNearBall*)	**Pré-Condições:** • O robot encontra-se suficientemente perto da bola. (*isNearBall*) **Acção:** • O robot segura a bola. (*holdUpBall*) • Comunica ao robot *receiver* que está preparado a passar a bola. **Transição:** • O robot tem a posse de bola. (*heldBall)*	**Pré-Condições:** • O robot tem a posse da bola. (*hasBallPossession*) • O robot *receiver* encontra-se suficientemente perto da posição de recepção. (*closeToSpot*) **Acções:** • Passa a bola para a posição o robot receiver. (*passBallToSpot*) • Comunica ao robot receiver que passou a bola. **Transição:**			**Pré Condições:** • Não obteve a posse de bola em tempo útil. • Não recebeu posição do *receiver* em tempo útil. **Acção:** • Comunica o insucesso ao robot *receiver*.
Robot receiver		Aceita o compromisso.	**Pré-Condições:** • Aceitação do compromisso. **Acções:** • Calcula o ponto de recepção da bola • Deslocação até ao ponto de recepção da bola. (*moveToSpot*) • Comunica ao robot *kicker* a posição onde deseja receber a bola. **Transição:** • A bola foi passada pelo robot kicker. (*ballPassed*)	**Pré-Condições:** • O robot encontra-se suficientemente perto da posição onde deverá receber a bola. (*isNearSpot*) **Acções:** • Comunica ao robot *kicker* que se encontra suficientemente perto da posição onde deverá receber a bola. **Transição:** • A bola foi passada. (*ballPassed*)		**Pré-Condições:** • A bola foi passada pelo robot *kicker*. (*ballWasPassed*) **Acções:** • O robot deverá tentar interceptar a bola. (*moveToBall*) **Transição:** • A bola foi interceptada com sucesso. (*gotBalPossessionl*)	**Pré-Condições:** • A bola foi interceptada com sucesso após o passe do robot *kicker* (*hasBallPossession*) **Acção:** • Assinalar como sucesso a execução do comportamento à BC..	**Pré Condições:** • Recebeu informação de insucesso do companheiro de equipa. • Não recebeu confirmação do passe em tempo útil. **Acção:** • Comunica o insucesso ao robot *kicker*. • Assinala como insucesso a execução do comportamento à BC.

Tabela A.2.3 – Descrição do comportamento relacional *Attacking Kickoff*

A.2.6 Implementação do comportamento relacional *Attacking Kickoff*

Tal como o comportamento relacional *Attacking Throw In*, este comportamento foi implementado, mas, também, em versão simplificada.

O jogador desloca-se sempre para um ponto pré-definido. Quando chega a este ponto pedirá ao companheiro de equipa que lhe enderece um passe, cabendo a este último a execução do pontapé que terá como destino o companheiro. No momento em que este relatório foi escrito, o passador ainda não se preocupa em fazer o passe na direcção do receptor da bola, no entanto este deverá ser um dos passos a concretizar brevemente pela equipa, não devendo ser excessivamente complicado.

A.2.7 Comportamento relacional *Defensive Kickoff*

Descrição

Este comportamento relacional serve para definir um conjunto de acções que os agentes nele participantes deverão seguir quando existe um pontapé de saída a favor do adversário.

O seu funcionamento é extremamente simples, quando ocorre um *kickoff* a favor dos adversários, os robots que decidirem iniciar este comportamento irão posicionar-se junto da sua área. Quando a bola é reposta em jogo, um dos robots perseguirá a bola enquanto o outro cobrirá a zona da baliza, impedindo possíveis jogadas de perigo dos adversários, perseguindo a bola apenas quando esta estiver a uma determinada distância (representada por um círculo na figura A.2.9.

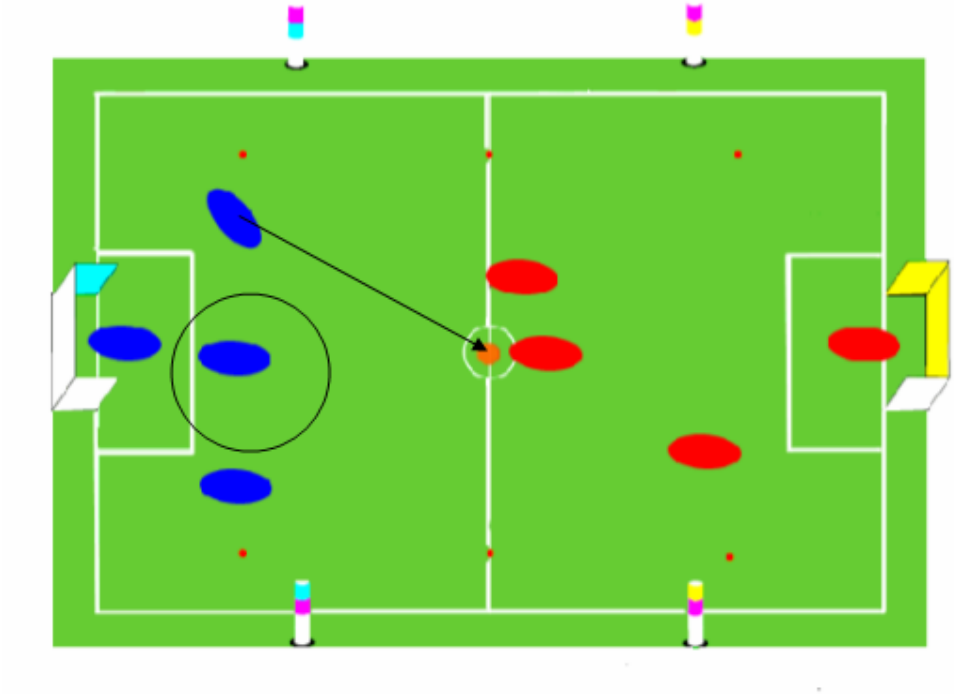


Figura A.2.9– Comportamento relacional *Defensive kickoff*

Pré Condições

- A bola foi reposta num dos pontos de *kickoff* a favor do adversário.

Rede de Petri

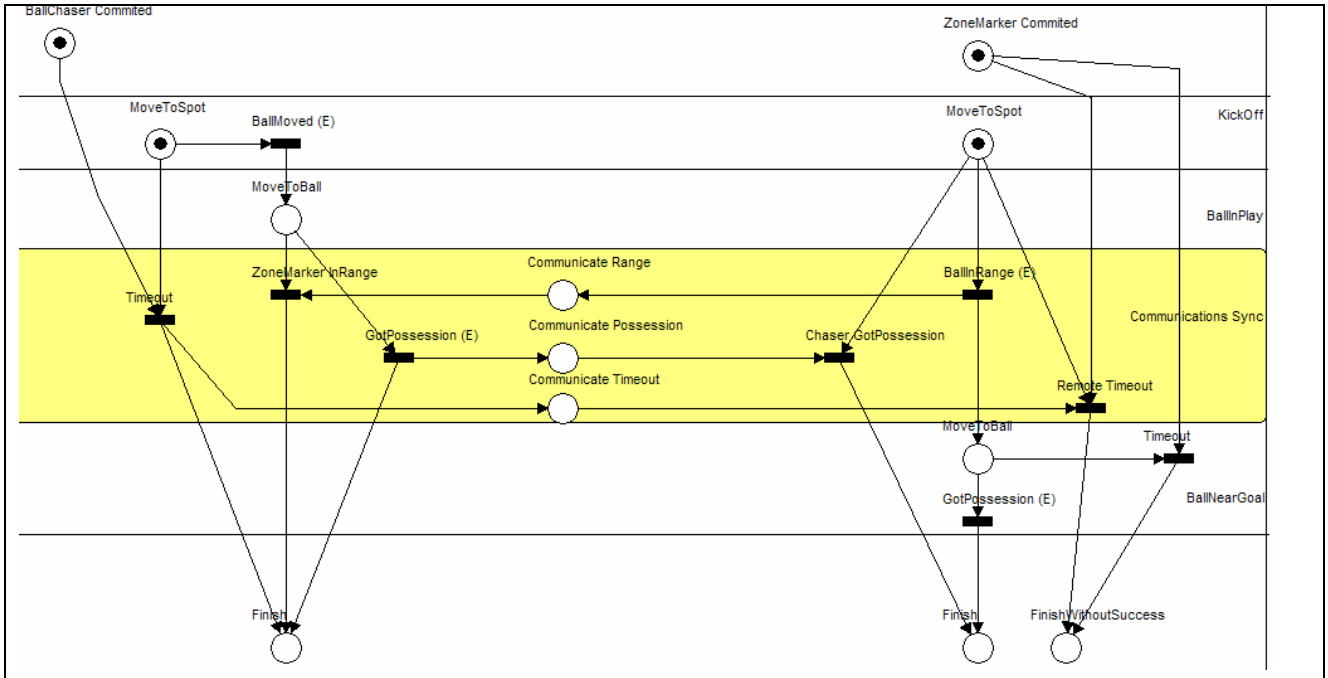


Figura A.2.10 Rede de Petri para o comportamento relacional *Defensive Kick Off*

Relational Behavior Phase	*Setup*		*Loop*			*End*	
Relational Behavior State	*Request*	*Accept*	*KickOff*	*BallInPlay*	*BallNearGoal*	*Success*	*Failure*
Robot BallChaser	O robot *kicker* inicia o comportamento relacional com um companheiro de equipa que tenha o papel de avançado.	Aceita o compromisso.	**Pré-Condições:** • Aceitação do compromisso. **Acção:** • Movimenta-se para a posição de partida. (*moveToSpot*) **Transição:** • A bola foi reposta em jogo (*ballMoved*)	**Pré-Condições:** • A bola foi reposta em jogo. **Acção:** • Tenta interceptar a bola. (*moveToBall*) **Transição:** • O robot tem a posse de bola. (*heldBall*)		**Pré-Condições:** • A bola foi interceptada com sucesso. (*hasBallPossession*) **Acção:** Comunicar sucesso ao robot *ZoneMarker*	**Pré Condições:** • A bola não se movimentou em tempo útil. **Acção:** • Comunica insucesso ao *ZoneMarker*.
Robot ZoneMarker		Aceita o compromisso.	**Pré-Condições:** • Aceitação do compromisso. **Acções:** • Movimenta-se para a posição de partida. (*moveToSpot*) **Transição:** • A bola está na sua zona de marcação. (*ballInRange*)		**Pré-Condições:** A bola está na sua zona de marcação (*ballInRange*) **Acção:** Tenta interceptar a bola (moveToBall) **Transição:** Interceptou a bola com sucesso (*heldBall*)	**Pré-Condições:** • A bola foi interceptada com sucesso. (*hasBallPossession*) • Recebeu a informação que o robot chaser recebeu a bola com sucesso. **Acção:** • Assinalar como sucesso a execução do comportamento à BC.	**Pré Condições:** • Recebeu informação de insucesso do companheiro de equipa. • Não obteve a posse de bola em tempo útil. **Acção:** • Assinala como insucesso a execução do comportamento à BC.

Tabela A.2.4 – Descrição do comportamento *Defensive Kickoff*

A.2.8 Implementação do comportamento relacional *Defensive Kickoff*

Este comportamento foi plenamente implementado utilizando a sua concepção original, dado que esta não exigiria alterações e inovações profundas e custosas no código da GT. Os resultados dos testes feitos a este comportamento poderão ser vistos no capítulo 6.

Anexo B – Propriedades das redes de Petri

Como foi mencionado no capítulo 2, as redes de Petri que modelam comportamentos relacionais ou individuais deverão obedecer a certas propriedades para produzirem acções correctas nos agentes.

As propriedades que são essenciais para uma rede de Petri modelar correctamente um comportamento são:

- Vivacidade
- 1-Limitada

Ao contrário do sugerido em [8], não se julga essencial uma rede de Petri ser estritamente conservativa (manter sempre o mesmo número de *tokens*), pois pode dar-se o caso de uma acção de comunicação ser realizada em simultâneo com uma acção de movimento, 'criando' assim mais um *token*, que previsivelmente desaparecerá com o disparar da transição seguinte.

Tomemos então a seguinte definição de vivacidade [12]

- Uma transição *t*, diz-se viva se, para qualquer marcação *m*, atingível a partir de *m'*, atingível a partir de m_0 se *t* for 'disparável' em *m'*.
- Se todas as transições forem vivas, a rede diz-se viva.

E uma definição de limitação:

- Uma rede de Petri diz-se k-limitada se para cada marcação m, atingível a partir de um estado inicial m_0 nunca existe mais do que k *tokens* em cada lugar.

Se tomarmos as redes de Petri apresentadas no anexo anterior e as integrarmos com o mecanismo de controlo de comportamentos apresentado em [8], poderemos calcular as seguintes propriedades:

Propriedade / Comportamento	1-Limitada	Viva
Maradona	Sim	Sim
Clear the ball	Sim	Sim
Sweep	Sim	Sim
Attacking Throw In	Sim	Sim
Defensive Throw In	Sim	Sim
Attacking Kickoff	Sim	Sim
Defensive Kickoff	Sim	Sim

Tabela B.1 – Propriedades das redes de Petri representativas de comportamentos

Ou seja, todos os modelos de comportamentos apresentados têm as propriedades que se considera essenciais para a sua execução correcta.

Bibliografia:

[1] B. Vecht. *Behaviour Coordination for Cooperative Multi-Robot Systems*, pp. 5, Instituto Superior Técnico. (2004)

[2] P. Cohen, H. Levesque. *Teamwork*, *Nous*, Vol. 35, pg. 487-512. (1991)

[3] P. Lima, L. Custódio. *Multi-Robot Systems,* pp. 6 - 13 , Capítulo sobre Innovations in *Robot Mobility and Control*, a publicar por Springer-Verlag. (2005)

[4] *Sony Four Legged Robot Football League Rule Book*, RoboCup Technical Committee. (2004)

[5] T. Röfer *et al*, *German Team – RoboCup 2004*, Center for Computing Technology, Universität Bremen, pp. 23-116. (2004)

[6] B. Ottens, *Aibo Project 2004 - German Team Report*, Universiteit van Amsterdam, pp 1-11. (2004)

[7] F. Wang *et al*, *A Petri-Net Coordination Model for an Intelligent Mobile Robot*, IEEE *Transactions on Robotics and Automation*, Vol. 9, No. 3, pp. 257--262. (1993)

[8] J. Milhinhos e G. Vaz, *Comportamentos Relacionais para Robots Futebolistas,* Instituto Superior Técnico, pp. 29-47. (2004)

[9] Diversos vídeos de jogos de competições de futebol robótico.

[10] Robotics, http://en.wikipedia.org/wiki/Robotics, Wikipedia.org

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

[12] Reisig, W. (1985). Petri Nets: An Introduction. Springer, Berlin.