INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Plataforma de Competição de Veleiros Autónomos

BRUNO MIGUEL FERREIRA ALVES
novembro de 2016

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Electrotécnica
Rua Dr. António Bernardino de Almeida, 431, P-4249-015 Porto

Plataforma de Competição de Veleiros Autónomos

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização em Telecomunicações

Bruno Miguel Ferreira Alves

Orientação Científica: Professora Maria Benedita Campos Neves Malheiro

Ano Letivo: 2015-2016

Resumo

Esta dissertação apresenta o desenvolvimento de uma plataforma de competição de veleiros autónomos. O objetivo da plataforma é suportar competições em ambientes reais ou simulados, constituindo assim uma ferramenta de avaliação de algoritmos de navegação.

A plataforma é um sistema aberto, escalável, modular e distribuído que adota o paradigma dos Sistemas Multi-Agente (SMA) para a modelação dos veleiros e uma interface *Web* para visualização e acompanhamento das competições reais ou simuladas. Do ponto de vista da arquitetura, a plataforma encontra-se dividida em duas camadas: (*i*) *Front-End* e (*ii*) *Back-End*. O *Back-End* é constituído por um servidor *Web*, um servidor *JavaScript Object Notation* (JSON), um servidor de base de dados relacionais, um servidor de base de dados não relacionais e uma plataforma de execução de agentes que expõe uma *Application Programming Interface* (API) através de serviços *Web* do tipo *Representational State Transfer* (REST) – RESTful. O *Front-End* contempla dois tipos de clientes: os navegadores que disponibilizam a interface *Web* e os agentes de *software* que representam os veleiros e interagem com a plataforma de execução de agentes através da API RESTful.

Cada competição representada na plataforma possui três tipos de agentes: (*i*) o agente administrador da competição, que controla o cumprimento das regras por parte dos veleiros; (*ii*) o agente meteorológico, responsável pela representação das condições meteorológicas da competição e (*iii*) os agentes veleiro, que realizam a modelação física de um veleiro.

Por último, a modelação física dos veleiros contempla: (*i*) a massa; (*ii*) a área vélica; (*iii*) o ângulo da vela e do leme; (*iv*) a velocidade e a direção do vento e (*v*) a posição e velocidade do casco.

Abstract

This dissertation presents the development of a competition platform for autonomous sailboats. The platform's goal is to support competition in real or simulated environments, thus providing an assessment tool of navigation algorithms.

The platform is an open, scalable, modular and distributed system that adopts the paradigm of Multi-Agent Systems (MAS) for the modelling of sailboats and a Web interface for the viewing and monitoring of real or simulated competitions. From an architectural point of view, the platform is divided into two components: (*i*) Front-End and (*ii*) Back-End. The Back-End consists of a Web server, a JavaScript Object Notation (JSON) server, a relational database server, a non-relational database server and an agent execution platform that exposes an Application Programming Interface (API) via RESTful Web service. The Front End includes two types of customers: browsers that provide the Web interface and software agents that represent the sailboats and interact with the agents execution platform through the RESTful API.

Each competition represented in the platform has three types of agents: (*i*) the manager agent of the competition, which monitors the compliance with the rules by sailboats; (*ii*) the meteorological agent, responsible for providing the weather conditions during the competition and (*iii*) the sailboat agents that perform the physical modeling of the sailboats.

Finally, the physics of sailboats comprises: (*i*) weight; (*ii*) sail area; (*iii*) angle of the sail and rudder; (*iv*) velocity and direction of the wind and (*v*) position and velocity of the hull.

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

Abreviatura	Descrição	Página
AID	*Agent Identifier*	37
AMS	*Agent Management System*	37
API	*Application Programming Interface*	iii
B2B	*Business-to-Business*	11
B2C	*Business-to-Consumer*	11
CAD	*Computer Aided Design*	25
CGI	*Common Gateway Interface*	16
DF	*Directory Facilitator*	37
EER	*Enhanced Entity Relationship*	4
FIPA	*Foundation for Intelligent Physical Agents*	19
GAMA	*Gis & Agent-based Modelling Architecture*	19
GAML	*Uniform Resource Identifier*	21
GIS	*Geographic Information Systems*	20
GNSS	*Global Navigation Satellite System*	52
GUI	*Graphical User Interface*	20
IRSC	*International Robotic Sailing Conference*	5
ISEP	Instituto Superior de Engenharia do Porto	xiii
JADE	*Java Agent Development Framework*	19
JDK	*Java Development Kit*	23
JRE	*Java Run-Time Environment*	22
JSON	*JavaScript Object Notation*	iii
JSR	*Java Specification Request*	22
MASON	*Multi-Agent Simulator Of Neighborhoods*	19
NoSQL	*Not only SQL*	39
ODE	*Open Dynamics Engine*	13
REST	*Representational State Transfer*	iii
RMA	*Remote Management Agent*	37
SCRC	*Simulated Car Racing Championship*	11
SMA	Sistema Multi-Agente	iii
SPADES	*System for Parallel-Agent, Discrete-Event Simulation*	13

Abreviatura	Descrição	Página
SQL	*Structured Query Language*	39
TAC	*Trading Agent Competition*	11
TCP	*Transmission Control Protocol*	15
TORCS	*The Open Racing Car Simulator*	14
UDP	*User Datagram Protocol*	12
URI	*Uniform Resource Identifier*	23
URL	*Uniform Resource Locator*	42
VRML	*Virtual Reality Markup Language*	12
WRSC	*World Robotic Sailing Championship*	5

Agradecimentos

No âmbito desta dissertação, não poderia deixar de agradecer a todos aqueles que acreditaram que seria possível. Sem eles, não teria o apoio, o estímulo, a disponibilidade e o conhecimento indispensável para atingir mais uma etapa de formação importante na minha vida.

À Professora Doutora Benedita Malheiro, exprimo o meu mais sincero agradecimento pela sua orientação, colaboração, dedicação e disponibilidade que sempre demonstrou. Ao Eng.º Bruno Veloso, este agradecimento também lhe dedico, pela sua transmissão de conhecimentos científicos e pelo o apoio que sempre me disponibilizou. Para eles vai um muito obrigado.

Agradeço a todos os meu amigos, em especial ao Nuno Cruz, Diogo Cunha, Carlos Bessa e Pedro Brito, pelos momentos de descontração, mas sobretudo, pelos momentos em que me deram força e me ajudaram a ultrapassar as dificuldades nas noites passadas no Instituto Superior de Engenharia do Porto (ISEP) para o desenvolvimento deste projeto.

À minha namorada agradeço por me ter acompanhado nesta caminhada longa e difícil, com paciência, compreensão e orgulho, mesmo quando havia a necessidade de sacrificar dias, noites e fins-de-semana em prol da realização deste projeto. Soube sempre respeitar o quanto importante era para mim este marco na minha vida. Obrigado por teres sido e seres o meu porto de abrigo, acreditares nas minhas capacidades e por todo o amor e dedicação.

Por último, quero agradecer à minha família pelo o apoio incondicional que me deram, especialmente aos meus pais com os quais tenho uma dívida de gratidão. A eles deixo um grande obrigado pela exigência, preocupação, dedicação, amor, educação e valores que me transmitem todos os dias. Sei que o término desta fase na minha vida é um orgulho e uma felicidade para eles.

Capítulo 1

Introdução

Neste capítulo é apresentado o problema que se pretende resolver, a motivação, os objetivos a alcançar, o planeamento do projeto e a estrutura deste documento.

1.1 Problema

Nos últimos anos tem-se assistido a um aumento significativo de adeptos e de participantes nas competições de veleiros autónomos, conduzindo a um desenvolvimento acentuado das tecnologias relacionadas. Cada vez mais, as equipas têm procurado melhorar os seus sistemas de navegação e utilizado ferramentas que as ajudem a analisar e a testar os algoritmos de controlo de forma a garantir o melhor desempenho possível nas provas. Outra necessidade, é a falta de ferramentas que permitam comparar o que foi planeado e simulado com a realidade, ou seja, ferramentas de seguimento e recolha de dados das competições reais.

1.2 Motivação

A motivação para o desenvolvimento desta dissertação deveu-se ao interesse em alargar os conhecimentos relativos a SMA, ao desenvolvimento de uma plataforma escalável e modular adaptável a outras competições que não apenas às de veleiros autónomos e ao desafio de aprender conceitos de navegação à vela.

1.3 Objetivos

O objetivo fundamental desta dissertação passa por conceber, desenvolver e depurar uma plataforma de teste e de seguimento de competições reais e simuladas

de veleiros autónomos. A plataforma deverá aplicar o conceito SMA para a modelação dos diversos veleiros e apresentar uma interface *Web* intuitiva para a configuração das competições, equipas e embarcações.

O desenvolvimento da plataforma começou pela definição dos casos de uso, requisitos e testes funcionais.

1.3.1 Casos de Uso

Os casos de uso foram determinados com base nas funcionalidades principais que a plataforma deve disponibilizar. Existem, de uma forma resumida, cinco funcionalidades distintas que a plataforma deve oferecer: (*i*) Funcionalidade de Teste – Oferecer a possibilidade de testar algoritmos de navegação; (*ii*) Funcionalidade de Seguimento – Permitir acompanhar competições reais de veleiros autónomos; (*iii*) Funcionalidade de Simulação – Disponibilizar a possibilidade de criar e acompanhar competições simuladas de veleiros autónomos; (*iv*) Funcionalidade de Utilizador – Oferecer a possibilidade de definir preferências de utilização e autenticação na plataforma e (*v*) Funcionalidade de Administração – Permitir a gestão e o controlo da plataforma.

No Anexo A são apresentados os diagramas de casos de uso que identificam com maior detalhe as funcionalidades que devem ser contempladas.

1.3.2 Requisitos

Como requisitos básicos, a plataforma de competição de veleiros autónomos deverá ser aberta, escalável, modular e suportada por tecnologias *open source*. Deverá permitir, em modo simulado, testar e comparar algoritmos de navegação e, em modo real, acompanhar competições de veleiros autónomos.

Em relação à interface, a plataforma deverá oferecer uma API via serviço *Web* RESTful e uma aplicação *Web* de seguimento de veleiros em competição 2D/3D. O *Back-End* da plataforma deverá modelar as embarcações através de um SMA.

1.3.3 Testes Funcionais

Os testes funcionais da plataforma encontram-se organizados em dois cenários: (*i*) Teste de Funcionalidades e (*ii*) Teste da Modelação Física. O primeiro cenário consiste no teste das funcionalidades principais do sistema para comprovar o correto funcionamento do seguimento das competições simuladas e das competições reais. No segundo cenário comprova-se que a modelação física adotada para os veleiros autónomos é coerente com a navegação à Bolina, ao Largo, à Popa e a Través.

1.4 Planeamento do Projeto

O projeto foi dividido em três etapas de execução. A primeira etapa consistiu no estudo e levantamento do estado da arte no domínio dos sistemas de simulação e competição, em particular através de SMA, e na familiarização com as tecnologias envolvidas. A segunda etapa foi dedicada à especificação das características do protótipo e à escolha das linguagens, tecnologias e ferramentas de desenvolvimento a adotar. A terceira etapa residiu no desenvolvimento, teste e depuração do protótipo. Todas as etapas foram redigidas nos respetivos capítulos da dissertação.

A calendarização destas etapas e das suas respetivas tarefas encontram-se no cronograma que é apresentado na Figura 1.1.

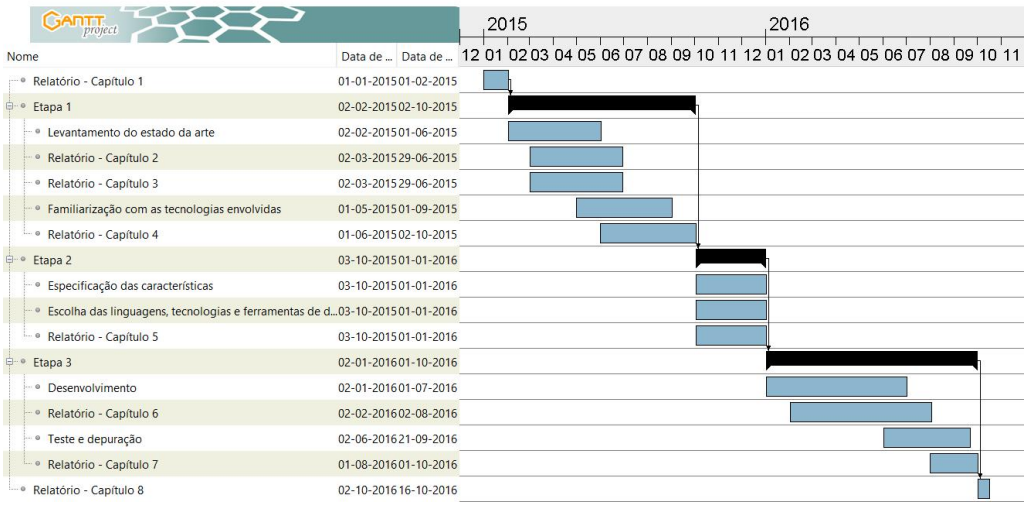

Figura 1.1: Calendarização do projeto.

1.5 Estrutura da Dissertação

Esta dissertação encontra-se dividida em oito capítulos: (*i*) Capítulo 1 – introduz o problema, a motivação, os objetivos e o planeamento do projeto; (*ii*) Capítulo 2 – apresenta as principais competições de veleiros autónomos; (*iii*) Capítulo 3 – expõe os conceitos e o estado da arte dos SMA de competição simulada; (*iv*) Capítulo 4 – explica as tecnologias e plataformas de suporte contempladas para a realização do projeto; (*v*) Capítulo 5 – apresenta o ambiente de desenvolvimento adotado; (*vi*) Capítulo 6 – descreve o desenvolvimento do projeto; (*vii*) Capítulo 7 – apresenta os testes efetuados e (*viii*) Capítulo 8 – apresenta as principais conclusões e as perspetivas de futuros desenvolvimentos.

Em anexo incluem-se os diagramas de casos de uso, o modelo *Enhanced Entity Relationship* (EER) da base de dados, os diagramas de sequência, os fluxogramas referentes à modelação dos agentes e das regras de navegação, o perfil dos clientes navegadores registados para testes e os ficheiros de criação das competições.

Capítulo 2

Competições de Veleiros Autónomos

Neste capítulo são apresentadas as competições mais relevantes de veleiros autónomos existentes atualmente, que são a Microtransat, o World Robotic Sailing Championship e a Sailbot Competition.

2.1 The Microtransat Challenge

A *Microtransat Challenge* [13], que foi concebida em 2005 por Mark Neal da Universidade de Aberystwyth (Reino Unido) e Yves Brière do Instituto Superior de Aeronáutica e do Espaço de Toulouse (França), consiste numa competição transatlântica de barcos totalmente autónomos. Nesta competição, existem duas classes de barcos participantes: (*i*) veleiros, impulsionados exclusivamente pelo vento, com um comprimento de casco, em relação à linha de água, até 4 m e (*ii*) barcos a motor, sem qualquer outra fonte de propulsão, até 2 m de comprimento de casco em relação à linha de água.

2.2 World Robotic Sailing Championship

O *World Robotic Sailing Championship* (WRSC) começou em 2008 em conjunto com a *International Robotic Sailing Conference* (IRSC) e, desde então, tornou-se um duplo evento anual. O WRSC/IRSC é dedicado à avaliação e desenvolvimento de veleiros autónomos não tripulados [14]. Em 2016 [15], os participantes foram divididos em duas classes: (*i*) a classe Micro-veleiro para barcos com comprimento total inferior a 1,5 m e uma massa até 100 kg e (*ii*) a classe de veleiros até 4 m de comprimento total e menos de 300 kg de massa. Em termos de fontes de

propulsão, os veleiros só podem utilizar a energia do vento ou das ondas. Em relação ao casco, os barcos podem usar qualquer tipo de casco e qualquer tipo e número de velas.

2.3 Sailbot Competition

Sailbot [16] é uma competição de veleiros que começou em 2006 na Universidade de Queen, Canadá. A competição internacional tem lugar na América do Norte (EUA ou Canadá) e envolve equipas de estudantes de diferentes universidades de todo o mundo. A competição inclui classes para veleiros com comprimento máximo de 1 m, 2 m e 4 m.

2.4 Conclusão

Existe um conjunto de competições internacionais de veleiros autónomos com um historial consolidado, regras e classes bem definidas. Em 2011, R. Stelzer e K. Jafarmadar [1] realizaram um levantamento do número de veleiros que participou nestas competições de 2006 a 2010, que demonstra um interesse crescente por parte da comunidade de investigação (ver Figura 2.1).

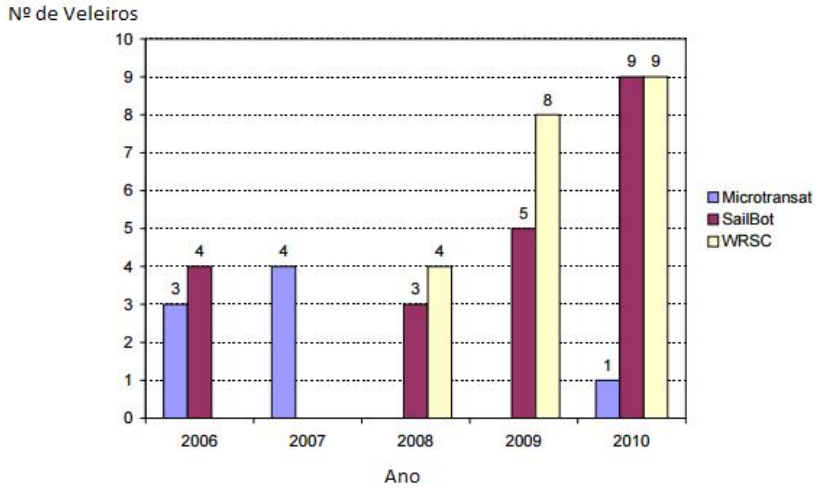

Figura 2.1: Número de veleiros participantes nas competições Microtransat, WRSC e Sailboat entre 2006 e 2010 [1].

Um dos objetivos da plataforma a desenvolver é contribuir, através do modo de simulação de algoritmos de navegação, para a preparação da participação das equipas neste tipo de competições e, através do modo de seguimento, para o acompanhamento das competições reais e simuladas.

Capítulo 3

Sistemas Multi-Agente de Competição Simulada

Neste capítulo são apresentados os conceitos e o estado da arte dos SMA de competição simulada mais representativos. Por último, é realizado uma análise comparativa dos sistemas apresentados.

3.1 Sistemas Multi-Agente

Um SMA [17] é um sistema que habita num dado ambiente e que é composto por dois ou mais agentes que interagem entre si para atingir objetivos pessoais e/ou coletivos. Os conceitos existentes na construção destes sistemas são: (*i*) Agentes; (*ii*) Ambiente; (*iii*) Interação e (*iv*) Organização. Os SMA constituem um modelo versátil e adequado para a modelação de sistemas distribuídos complexos, podendo ser constituídos por conjuntos de agentes reativos, deliberativos ou híbridos e adota diversos tipos de mecanismos de interação como planeamento, coordenação, leilões ou negociação.

3.1.1 Agentes

Um agente, segundo Michael Wooldrige e Nicholas Jennings [18], é um sistema capaz de ser autónomo ("Autonomia"), de comunicar e de partilhar informações ("Habilidade Social"), de responder às alterações do ambiente ("Reativo") e de apresentar comportamentos em função de instruções ("Pro-ativo") (ver Figura 3.1). Esta definição, que não é única nem consensual dentro da comunidade científica, adequa-se ao domínio dos sistemas de simulação baseados em agentes.

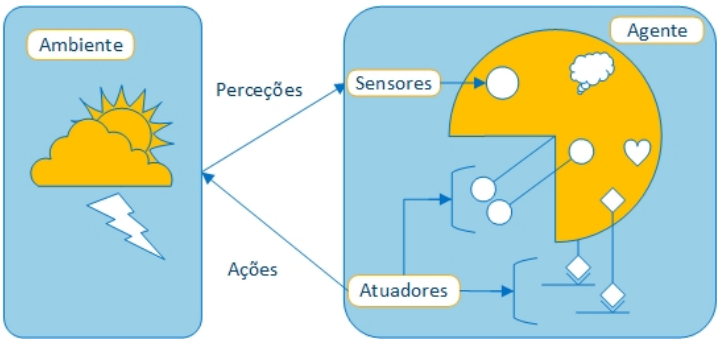

Figura 3.1: Agente a interagir com o ambiente através de sensores e atuadores [2].

Em 1996, Hyacinth Nwana [3] definiu uma tipologia de classificação de agentes baseada em três atributos – autonomia, cooperação e aprendizagem – representada na Figura 3.2:

- Agentes colaborativos com aprendizagem – Agentes com capacidade de cooperar com outros agentes e de aprender;
- Agentes colaborativos – Agentes com elevado nível de autonomia e com capacidade de cooperar com outros agentes;
- Agentes de interface – Agentes com capacidade de aprendizagem e que dependem de si próprios para agir (autonomia);
- Agentes inteligentes – Agentes que têm a capacidade de cooperar, aprender e que possuem um elevado grau de autonomia nas suas ações.

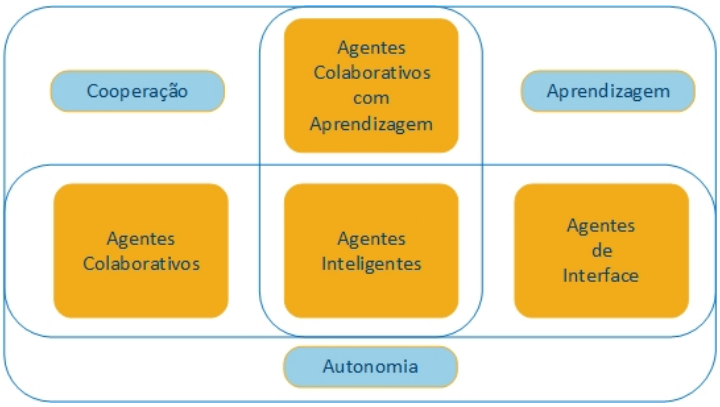

Figura 3.2: Tipologia de Hyacinth Nwana [3].

3.1.2 Ambiente

As características do ambiente envolvente de um agente permitem definir a sua arquitetura e o seu modo de funcionamento. Russell [2] identificou os seguintes tipos de ambiente:

- Acessível *vs.* Inacessível – Se um ambiente é acessível significa que um agente pode, através de sensores, aceder a todos os aspetos que são relevantes para uma determinada ação;
- Determinístico *vs.* Não determinístico – Um agente ao executar uma ação num ambiente determinístico tem apenas um único resultado possível;
- Episódico *vs.* Não episódico – Um agente num ambiente episódico tem uma experiência dividida em episódios estanques, ou seja, em cada episódio o agente perceciona e, em seguida, age, sendo os resultados dos episódios independentes;
- Estático *vs.* Dinâmico – Num ambiente estático o agente não se preocupa com o passar do tempo, pois o ambiente só se altera quando este realiza uma ação;
- Discreto *vs.* Contínuo – Um ambiente discreto oferece um número limitado de perceções a agentes com ações definidas.

Na Tabela 3.1 encontram-se exemplos de ambientes típicos.

Tabela 3.1: Exemplos de ambientes envolventes de um agente [2].

Ambiente	**Acessível**	**Determin.**	**Episódico**	**Estático**	**Discreto**
Xadrez	Sim	Sim	Não	Sim	Sim
Poker	Não	Não	Não	Sim	Sim
Sistema de diagnóstico médico	Não	Não	Não	Não	Não
Sistema de análise de imagem	Sim	Sim	Sim	Sim/Não	Sim

3.1.3 Arquitetura

Um SMA pode ser composto por diversos tipos de agentes, resultando em diferentes organizações e estabelecendo interações distintas durante a execução das tarefas no ambiente. Michael Wooldrige e Nicholas Jennings [19], classificam as arquiteturas de SMA em três grupos:

- Reativas – os agentes têm comportamentos baseados num conjunto de regras bem definidas, desta forma um modelo estímulo-resposta (ver Figura 3.3);

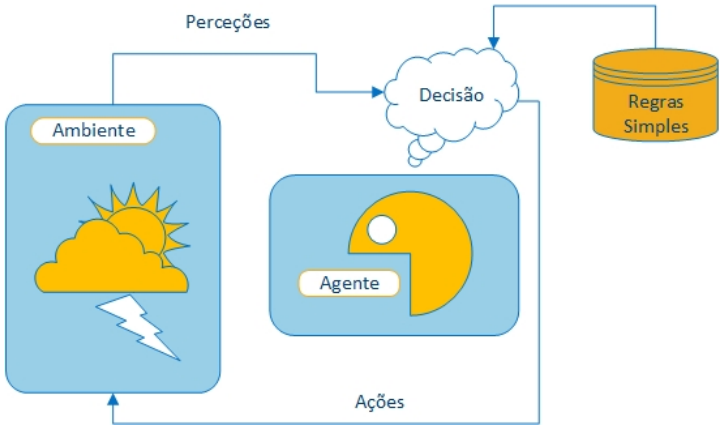

Figura 3.3: Arquitetura reativa.

- Deliberativas – os comportamentos dos agentes são definidos através de um raciocínio lógico, *i.e.*, com esta arquitetura o agente é obrigado a possuir traduções simbólicas do mundo real e, sobretudo, inteligência para tomada de decisões (ver Figura 3.4);

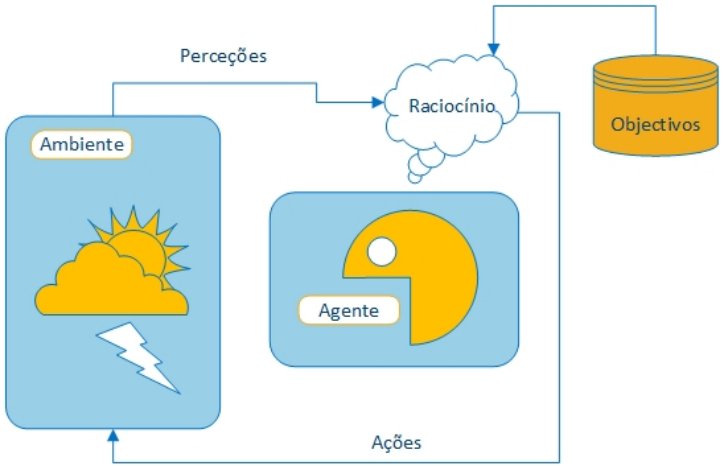

Figura 3.4: Arquitetura deliberativa.

- Híbridas – Estas arquiteturas existem para minimizar as dificuldades e as falhas existentes numa arquitetura puramente reativa ou deliberativa, ou seja, tenta juntar o melhor das duas. Uma arquitetura puramente reativa tem a dificuldade de tomar uma decisão com base num objetivo, enquanto

que uma arquitetura puramente deliberativa tem dificuldade de tomar uma decisão imediata devido ao seu raciocínio complexo e demorado (ver Figura 3.5).

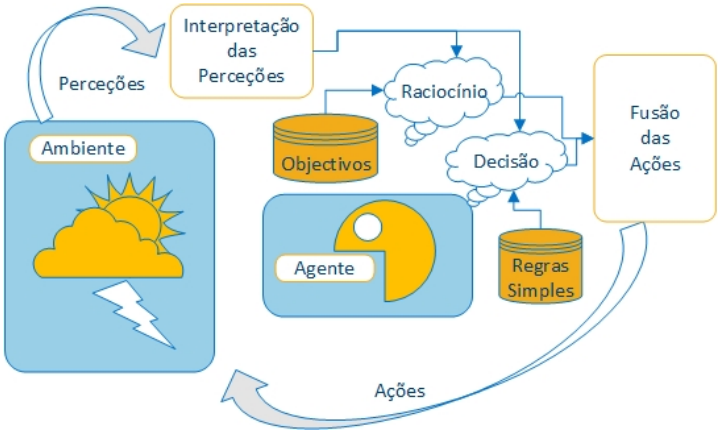

Figura 3.5: Arquitetura híbrida.

3.2 Aplicações

Ao longo dos últimos anos, o número de aplicações suportadas por SMA tem vindo a aumentar de forma exponencial. A Tabela 3.2 ilustra essa diversidade.

Tabela 3.2: Exemplos de aplicações de SMA [2].

Aplicações	**Exemplos**
Industriais	Distribuição de energia elétrica Sistemas de transportes
Pesquisa de Informação	Filtragem de notícias Sistemas de recomendação
Comércio Electrónico	*Business-to-Consumer* (B2C) *Business-to-Business* (B2B)
Simulação	Simuladores de condução Simuladores de produção
Competições Científicas	Futebol Robótico (RoboCup) *Trading Agent Competition* (TAC)

3.3 Sistemas Multi-Agente de Competição Simulada

No âmbito dos sistemas de competição e simulação baseados em SMA estudou-se o RoboCup, o *Simulated Car Racing Championship* (SCRC) e a TAC.

3.3.1 RoboCup

A RoboCup [20] é uma competição científica internacional que incentiva a investigação nas áreas da robótica e da inteligência artificial. Foi criada em 1997 com o objetivo de promover a criação de uma equipa de futebol robótico capaz de vencer em 2050 a seleção de futebol humana campeã do mundo [21].

A Figura 3.6 apresenta a arquitetura proposta para a plataforma de simulação desta competição. O simulador da RoboCup é composto por três grandes módulos [22]: (*i*) cliente; (*ii*) servidor e (*iii*) visualização.

O módulo cliente é responsável pela dinâmica dos jogadores da sua equipa e comunica com o módulo servidor através de um *socket*, enviando mensagens (Mensagens) de controlo das ações dos jogadores através do protocolo da camada de transporte *User Datagram Protocol* (UDP). O módulo servidor é responsável pela simulação física da competição, *i.e.* pela movimentação dos objetos (bola e jogadores) no ambiente simulado bem como pelo cálculo em tempo real das alterações do ambiente (Simulador do Ambiente/Mundo). O servidor suporta a execução de múltiplos jogos em simultâneo, controlando cada jogo de acordo com regras pré-definidas (Regras). O módulo de visualização permite acompanhar visualmente a competição, a pontuação e o comportamento dos jogadores (ver Figura 3.7). Este módulo inclui dois subsistemas: o 3D *Motion Data Generation* e o servidor *Virtual Reality Modeling Language* (VRML). O 3D *Motion Data Generation* disponibiliza em tempo real informação de alta resolução para projeção em telas de grandes dimensões ou para radiodifusão televisiva da informação de um jogo. O servidor VRML disponibiliza essa mesma informação 3D para clientes *Web* (navegadores).

A simulação da competição, tem como base o SimSpark. O SimSpark [4] é

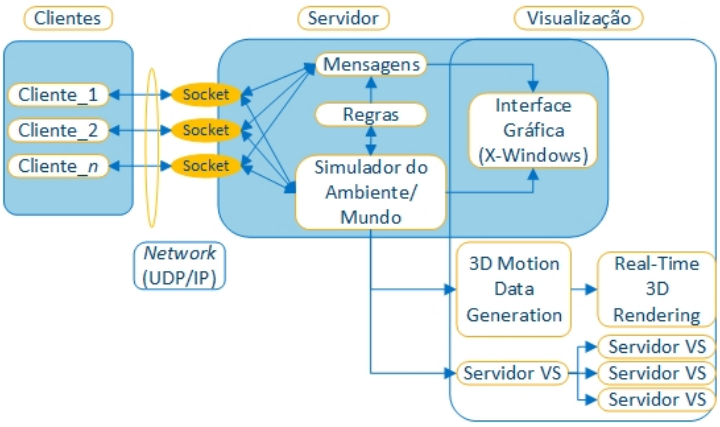

Figura 3.6: Arquitetura da RoboCup.

Figura 3.7: Interface visual da competição RoboCup [4].

um simulador *multi-robot* que utiliza os componentes do SMA de simulação físico designado por Spark. Este é constituído por três importantes partes (ver Figura 3.8): (*i*) motor físico; (*ii*) gestor de objetos/memória e (*iii*) motor de simulação. O motor físico implementado no SimSpark é o *Open Dynamics Engine* (ODE). O ODE oferece simulações de corpo rígido, deteção de colisões e estruturas para utilização de corpos articulados. A gestão dos objetos/memória é realizada através de uma plataforma designada por Zeitgeist. Zeitgeist [4] define, através de coleções de objetos, as classes abstratas e as interfaces de dados para criar e armazenar as respetivas instâncias/objetos (memória). Assim, comporta-se como uma memória partilhada, onde as instâncias são armazenadas numa estrutura em árvore (sistema de ficheiros virtual). Cada nó da árvore contém o seu próprio caminho dentro da árvore e as referências ao nó pai e aos nós filhos. O armazenamento do caminho em cada nó permite, em tempo de execução, encontrar facilmente serviços e objetos. As instâncias/objetos são nós filhos das classes abstratas respetivas. As coleções de objetos estão armazenadas em locais pré-definidos no sistema de ficheiros virtuais, permitindo criar objetos de classes desconhecidos em tempo de compilação, tornando o sistema extensível porque permite adicionar novos sensores, atuadores ou modelos de robôs. O *core* da simulação, responsável pela união dos processos como a calendarização, gestão de eventos e comunicação com processos externos, contém dois modos de execução do simulador [4]. Um dos modos é relativamente simples pois de uma forma direta realiza as ações solicitadas pelos agentes. O outro modo é mais elaborado pois utiliza o *System for Parallel-Agent, Discrete-Event Simulation* (SPADES).

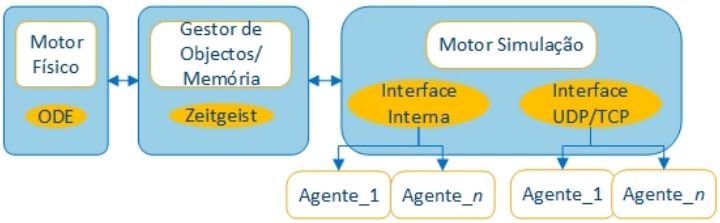

Figura 3.8: Arquitectura do SimSPark [4].

3.3.2 Simulated Car Racing Championship

A plataforma do campeonato simulado de automóveis de corrida – SCRC [5] – é uma competição baseada no *The Open Racing Car Simulator* (TORCS). O TORCS [5, 23, 24] fornece aos clientes uma forma de criar, testar e utilizar os seus automóveis simulados (BOT). Para controlar e limitar os comportamentos dos BOT é facultado ao cliente a possibilidade de interagir com o servidor. Cada BOT em execução é constituído por um servidor e um cliente correspondente. Todas as comunicações são realizadas através do protocolo da camada de transporte UDP. O servidor BOT envia a cada 20 ms o estado dos sensores (perceções obtidas do ambiente simulado) ao cliente. O cliente, em seguida, tem 10 ms para indicar ao servidor BOT que ações deve tomar. Esta interação entre o servidor BOT e o cliente é realizada através de um protocolo bem definido. O cliente desde que siga este protocolo pode desenvolver o seu algoritmo de controlo sem limitações de escolha de linguagem de programação, *i.e.* existe uma dissociação entre o cliente e o servidor BOT. A arquitetura da plataforma SCRC mencionada encontra-se ilustrada na Figura 3.9.

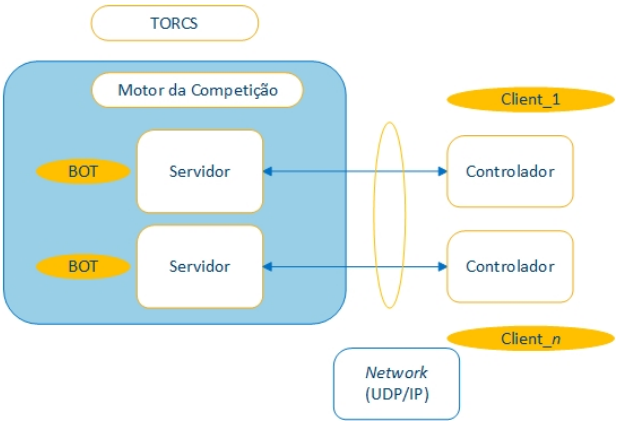

Figura 3.9: Arquitetura TORCS [5].

A competição oferece aos clientes um mecanismo de simulação física bastante desenvolvido bem como um acompanhamento visual em 3D (ver Figura 3.10). Por outro lado, faculta aos clientes mecanismos de controlo das configurações das corridas, tais como, condições ambientais, número de voltas e tipo de saída de dados do simulador. Estes mecanismos são desenvolvidos para permitir aos clientes treinar e testar os seus controladores sob condições definidas.

Figura 3.10: Interface TORCS [5].

3.3.3 Trading Agent Competition

A competição de negociação entre agentes – TAC [6] – é constituída por oito agentes que competem entre si no mercado através de leilões. O objetivo principal dos agentes (agências de viagens) é criar pacotes de viagem de acordo com as preferências dos respetivos clientes. Um pacote de viagem é constituído por: (*i*) um voo de ida e volta (entre TAC-Town e Boston) com uma duração de cinco dias; (*ii*) uma estadia no mesmo hotel entre a data de entrada e a data de saída e (*iii*) entretenimento durante a viagem de forma a satisfazer os clientes (aumentar a utilidade). Cada agente comunica com o TAC-Server através de mensagens sobre *Transmission Control Protocol* (TCP). O servidor oferece uma biblioteca de interface específica – API – que permite obter informação do mercado e submeter propostas. Um jogo dura 12 min e envolve vinte e oito leilões.

O servidor da plataforma TAC baseia-se no *Michigan Internet AuctionBot* [7], que é uma plataforma leiloeira segundo o modelo cliente-servidor. A Figura 3.11 mostra o diagrama do servidor e dos seus componentes. Os módulos API, Escalonamento, Leiloeiras e Base de Dados executam as operações centrais do AuctionBot, enquanto que os restantes módulos executam funções especificas da plataforma TAC. O AuctionBot permite a criação, configuração e licitação em

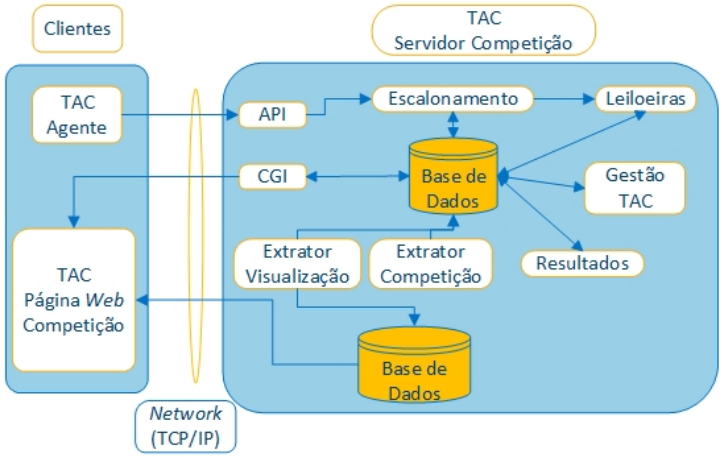

Figura 3.11: Arquitectura da plataforma TAC [6].

leilões através do navegador *Web* e através de agentes [6]. No primeiro caso, o utilizador pode, através de uma interface *Web* (TAC *Game Page*), submeter ofertas e acompanhar os leilões em curso. Os dados enviados são submetidos do lado do servidor numa aplicação *Web* que implementa a especificação *Common Gateway Interface* (CGI). No segundo caso, a plataforma oferece a API que permite a agentes interagir com o servidor de leilões. A API apresenta um protocolo de comunicação cliente/servidor que pode ser desenvolvido pelo utilizador em qualquer plataforma e linguagem de programação. O agente liga-se e autentica-se usando a API disponibilizada. Assim que a ligação é estabelecida, são apresentadas ao agente todas as propostas disponíveis. O agente mantém o controlo do estado da oferta e submete novas ofertas sempre que ache conveniente. Na Figura 3.12 encontra-se ilustrado o protocolo de comunicação com AuctionBot.

Figura 3.12: Representação do protocolo de comunicação com AuctionBot [7].

A competição de negociação é dividida em três momentos [6]: (*i*) antes; (*ii*) durante e (*iii*) depois. Antes da competição o gestor da competição (Gestão TAC) permite aos utilizadores agendarem no módulo Escalonamento novas instâncias da competição e inscreverem-se em competições futuras. A API tem a funcionalidade de informar os agentes da sua próxima competição, permitindo assim que o agente execute um *loop* contínuo e desperte para a competição quando é avisado. Para iniciar cada competição, o gestor da competição valida se todos os leilões da competição anterior foram terminados de forma correta e seleciona a próxima competição a ser executada. Em seguida, cria os leilões no módulo Leiloeiras e executa-os. Por último, lança todos os vendedores TAC da competição e, se o número de agentes participantes for inferior a oito, lança também agentes fictícios. Durante o decorrer da competição, o extrator de visualização envia todos os dados necessários para todas as ferramentas de visualização. Após a competição, o extrator de dados cria ficheiros de texto que descrevem todos os eventos relevantes. Esses ficheiros de texto são utilizados para posterior recuperação de resultados através da interface *Web*.

3.3.4 Comparação

A comparação dos sistemas de competição simulada apresentados foi efetuada em termos da arquitetura (modelo, comunicação e modelação dos concorrentes), características do ambiente simulado (perceção e atuação, acessibilidade, determinismo, dinamismo e modelo temporal) e mecanismos de coordenação entre concorrentes (cooperação ou competição). A Tabela 3.3 apresenta os resultados da comparação.

Tabela 3.3: Comparação das plataformas de simulação.

	RoboCup	**SCRC**	**TAC**
Arquitectura			
Sistema distribuído	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor
Protocolo de transporte	UDP/IP	UDP/IP	TCP/IP
Modelação dos concorrentes	Agentes	Agentes	Agentes
Ambiente			
Perceção/ação	Síncronas	Síncronas	Assíncronas
Acessível	Sim	Sim	Sim
Determinístico	Não	Não	Sim
Estático	Não	Não	Não
Discreto	Sim	Sim	Sim
Coordenação	Cooperação Competição	Competição	Competição

3.4 Conclusão

Neste capítulo foram apresentados os conceitos básicos associados aos SMA e os SMA de competição simulada mais representativos. Este estudo serviu de ponto de partida para a conceção da arquitetura da plataforma de competição de veleiros autónomos. Em particular, adotou-se o modelo do agente cliente que envia mensagens de controlo através de um protocolo bem definido e do servidor responsável pela simulação física da competição através de uma plataforma SMA. Esta opção permite que as equipas mantenham a execução dos algoritmos de navegação em plataformas das próprias (veleiros reais ou agentes de *software*).

No capítulo seguinte são identificadas as principais tecnologias e plataformas de suporte ao desenvolvimento da plataforma de competição de veleiros autónomos.

Capítulo 4

Tecnologias e Plataformas de Suporte

Neste capítulo são apresentadas as tecnologias e plataformas open source de desenvolvimento contempladas para a realização deste projeto, designadamente, as principais plataformas de desenvolvimento de Sistemas Multi-Agente (SMA) e ferramentas de criação de interfaces padrão do tipo serviço Web, interfaces Web 3D e de desenvolvimento de Front-Ends Web.

4.1 Plataformas de Desenvolvimento de SMA

Neste contexto analisaram-se as plataformas *Java Agent Development Framework* (JADE), *Gis & Agent-based Modelling Architecture* (GAMA), NetLogo e *Multi-Agent Simulator Of Neighborhoods* (MASON).

4.1.1 JADE

JADE [9, 25] é uma plataforma desenvolvida em linguagem Java, *open-source*, com múltiplos *add-ons* e uma comunidade alargada de utilizadores. Pode, ainda, ser usada em dispositivos com recursos limitados dado que requer apenas a instalação prévia da versão 1.2 da linguagem Java. O objetivo da plataforma é simplificar a implementação de SMA através de um *Middleware* que obedece às especificações da *Foundation for Intelligent Physical Agents* (FIPA). A FIPA [9, 26] é uma organização que se dedica à promoção da indústria dos agentes inteligentes através da definição de especificações de desenvolvimento de aplicações baseadas em agentes e de promoção da interoperabilidade. Atualmente, o projeto

JADE é supervisionado pela Telecom Italia, Motorola, Whitestein Technologies AG, Profactor GmbH e France Telecom R&D.

A plataforma JADE [27] pode ser distribuída entre várias máquinas e a sua configuração poder ser controlada através de uma *Graphical User Interface* (GUI) remota (Figura 4.1). A configuração pode ser alterada durante a execução, recorrendo a agentes que se deslocam entre máquinas sempre que necessário.

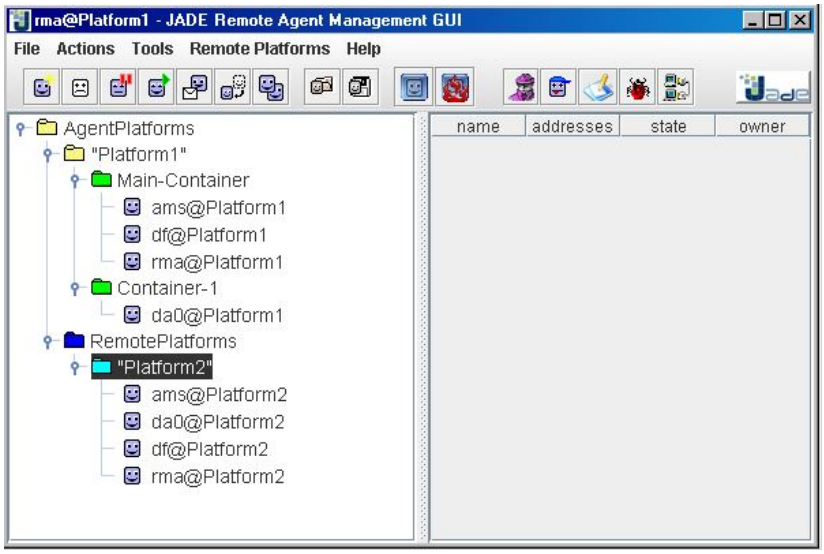

Figura 4.1: GUI da plataforma JADE.

Em termos de popularidade, JADE é a plataforma de desenvolvimento de SMA, compatível com as especificações FIPA, mais popular na comunidade de investigação e indústria. O código desta plataforma *open source* é estável e é disponibilizado pela Telecom Italia.

4.1.2 GAMA

A GAMA [9, 28, 29] é uma plataforma para a realização de simulações e modelações com informação geográfica, permitindo a interação com *Geographic Information Systems* (GIS). O seu principal objetivo é oferecer serviços e operações de índole geográfica, convertendo os objetos geográficos em agentes e associando-lhes a respetiva geometria, estados e comportamentos.

Os utilizadores da plataforma têm a possibilidade de executar simulações, em grande escala, suportadas por uma modelação dos agentes multi-nível, *i.e.*, cada agente é representado por diferentes níveis (tempo, espaço e comportamento), uma visualização instantânea do ambiente em 2D ou 3D (baseado em OpenGL [30]) e uma gestão de ambientes complexos. Tal como a plataforma JADE, é *open*

source, mas utiliza uma linguagem orientada a agentes designada por *Generalized Analytical Markup Language* (GAML) que fornece uma ferramenta gráfica de apoio à construção de modelos SMA espacialmente explícitos.

4.1.3 NetLogo

O NetLogo [9, 31, 32] consiste num ambiente de modelação programável para SMA que permite a exploração de fenómenos naturais e sociais. O seu ambiente fornece uma extensa biblioteca de modelos, incluindo modelos relacionados com várias temáticas, tais como a economia, a biologia, a física, a química e a psicologia. Pode ser usado tanto por professores como por especialistas sem conhecimentos de programação para modelar fenómenos pretendidos. Além da sua biblioteca extensa, os utilizadores podem criar novos modelos bem como editar os existentes. Os utilizadores têm também a possibilidade de, durante a simulação, interagir e dar instruções aos agentes a fim de explorar o comportamento sob várias condições.

O ambiente do NetLogo é escrito em Java e após cinco anos de desenvolvimento é um produto estável e popular na comunidade educativa e de investigação. Ao contrário da plataforma GAMA e JADE, não é *open source*.

4.1.4 MASON

O MASON [9, 33, 34] é uma ferramenta *open source* de simulação de eventos discretos multiagente escritos em Java, que atua como uma plataforma de simulação de agentes. O seu principal objetivo é suportar o desenvolvimento de SMA que vão desde de ambientes robóticos a ambientes sociais.

A ferramenta de simulação é dividida em três camadas: a camada útil, a camada modelo e a camada de visualização. A camada útil consiste nas classes de uso genérico. A camada modelo contém todas as classes relacionadas com a programação discreta de eventos. O utilizador, com apenas as duas camadas anteriores, já consegue desenvolver simulações na linha de comandos. O MASON, sem a opção de visualização, é mais eficiente (em termos de memória), permitindo a simulação com milhões de agentes. A camada de visualização possibilita a visualização através de uma GUI das simulações em 2D e 3D.

4.1.5 Comparação

A comparação das plataformas de desenvolvimento de SMA apresentados foi efetuada em termos do seu licenciamento, usabilidade, operacionalidade e suporte. A Tabela 4.1 apresenta o resultado.

Tabela 4.1: Comparação das plataformas de desenvolvimento de SMA [9].

	JADE	**GAMA**	**NetLogo**	**MASON**
Licenciamento				
Última atualização	06/12/2013	06/10/2013	19/12/2013	01/05/2013
Licença	LGPL v2	GPL	GPL	*Academic Free*
Open Source	Sim	Sim	Não	Sim
Usabilidade				
Simplicidade	Muito Simples	Simples	Simples	Complicado
Apreensibilidade	Fácil	Fácil	Fácil	Médio
Escalabilidade	Alta	Boa	Boa	Média
Compatibilidade	FIPA,CORBA	FIPA,GIS	Desconhecida	Desconhecida
Operacionalidade				
Desempenho	Alto	Bom	Bom	Bom
Estabilidade	Alto	Boa	Boa	Boa
Robustez	Alta	Boa	Média	Média
Linguagens de Programação	Java	GAML	NetLogo	Java
Suporte				
Suporte ao *user*	Alto	Bom	Bom	Médio
Popularidade	Alta	Baixa	Alta	Média

Após a realização das análises às plataformas de desenvolvimento de SMA, a escolha recaiu sobre a plataforma JADE visto ser a mais utilizada (popularidade elevada) na criação de SMA, utilizar as especificações FIPA, ser bastante simples e escalável, oferecer boa documentação (elevado suporte ao *user*) e ser *open source*.

4.2 Serviços Web RESTful

No âmbito do desenvolvimento de serviços *Web* RESTful analisaram-se as plataformas Jersey, RESTlet e RESTEasy.

4.2.1 Jersey

A linguagem Java, através da *Java Specification Request* (JSR) 311 [35], dá suporte a serviços REST. Esta especificação é designada por JAX-RS. A JAX-RS utiliza anotações para associar classes Java a recursos *Web*. A plataforma Jersey [10, 36, 37] de desenvolvimento de serviços *Web* RESTful em Java é uma implementação de referência da especificação JSR 311 *open source*.

A Jersey fornece uma API para implementar serviços *Web* RESTful em contentores do tipo Servlet Java. Do lado do servidor, oferece classes predefinas para

identificar recursos RESTful e uma biblioteca para o cliente comunicar com os serviços *Web* RESTful definidos no servidor. No ficheiro de configuração web.xml são registados os Servlets da aplicação *Web*. Estes Servlets analisam os pedidos HTTP e selecionam a classe e o método correspondente ao pedido. Esta seleção é baseada nas anotações da classe e dos métodos.

4.2.2 RESTlet

RESTlet [10] é uma plataforma *open source* que fornece uma estrutura leve, mas abrangente, para o mapeamento de conceitos REST com classes Java. Pode ser usado na implementação de qualquer tipo de sistema RESTful, não ficando limitado apenas a serviços RESTful para aplicações *Web*.

As aplicações baseadas em RESTlet utilizam uma arquitetura de aplicação *standalone* e um módulo *container*, tal como as aplicações baseadas em Servlets.

A maior parte das aplicações RESTful que adotam a abordagem RESTlet precisam de apenas de duas classes: *Application* e *Resource*. A instância *Application* mapeia o *Uniform Resource Identifier* (URI) para a instância *Resource*. A instância *Resource* é responsável pela manipulação dos comandos CRUD (*Create, Read, Update and Delete*) associados aos pedidos HTTP GET, POST, PUT e DELETE.

A plataforma inclui um servidor *Web* independente que permite o desenvolvimento de aplicações sem a necessidade de instalação de um servidor *Web*.

4.2.3 RESTEasy

A plataforma RESTEasy [38, 10] consiste numa plataforma de serviços *Web* RESTful. Trata-se de uma plataforma de implementação portátil da especificação JAX-RS que corre em qualquer contentor Servlet Java e que é integrável com o *JBoss Application Server*. Adicionalmente, o RESTEasy JAX-RS suporta funções do lado cliente através da plataforma *Client*.

RESTEasy, ao incluir as especificações JAX-RS e uma plataforma cliente, simplifica a escrita de clientes HTTP. Esta plataforma é adicionável a qualquer aplicação servidora desenvolvida com *Java Development Kit* (JDK) 5.0 ou superior.

4.2.4 Comparação

A comparação das plataformas de desenvolvimento de serviços *Web* RESTful apresentadas foi efetuada em termos da comunidade ativa, documentação e o tipo de aplicação do lado do servidor e cliente. A Tabela 4.2 apresenta o resultado.

Tabela 4.2: Comparação das plataformas para serviços *Web* de RESTful [10].

	Jersey	**RESTEasy**	**RESTlet**
Open Source	Sim	Sim	Sim
Comunidade	Ativa	Ativa	Ativa
Documentação	Detalhada	Detalhada	Detalhada
Aplicação *Server*	Servlet Standlone	Contentor Servlet	Servlet Standlone GWT GAE ANdroid OSGi
Aplicação *Client*	API do cliente; Boa capacidade de expansão;	Client Framework é oferecido; O servidor e o cliente podem usar interfaces comuns;	API do cliente disponível com funcionalidades padrão; O servidor e o cliente podem usar interfaces comuns;

Desta análise das plataformas para serviços *Web* RESTful, resultou na escolha da plataforma RESTlet. Esta decisão foi tomada porque se trata de uma plataforma madura, bem documentada e com uma comunidade de utilizadores ativa – considerações importantes quando se escolhe uma tecnologia *open source*. Outra vantagem da escolha desta plataforma é que a maior parte dos componentes requeridos para a construção de um serviço *Web* são fornecidos, ou seja, existe uma redução significativa no tempo de desenvolvimento.

4.3 Bibliotecas de Desenvolvimento Web 3D

Em relação ao desenvolvimento de *Web* 3D analisaram-se as bibliotecas Three.js, Babylon.js e Scene.js.

4.3.1 Three.js

A biblioteca *open source* Three.js [11] foi lançada a primeira vez no GitHub[1] em 2010 e ficou pronta a ser usada em WebGL em 2011. Trata-se de uma biblioteca *cross-browser* em JavaScript para criar gráficos 3D animados. O principal objetivo da biblioteca Three.js é criar uma camada de abstração no desenvolvimento WebGL, permitindo criar um mundo 3D WebGL facilmente. A biblioteca tenta

[1] https://github.com/

não ser um motor de jogo típico, mas sim uma ferramenta de animação de propósito geral. Atualmente, o motor de renderização pode ser utilizado em Canvas do HTML5, WebGL ou SVG.

4.3.2 Babylon.js

Babylon.js [11] foi lançada em 2013 pela Microsoft. Esta biblioteca tem uma abordagem mais centralizada do que a biblioteca Three.js. A Babylon.js centraliza-se no desenvolvimento de jogos com deteção de colisões e *anti-aliasing*. É uma biblioteca *open source* e a sua estrutura suporta um cenário gráfico com luzes, câmaras e materiais. Dispõe ainda de motores de colisões, física, animação e otimizações.

4.3.3 Scene.js

SceneJS [11] é uma biblioteca JavaScript baseada em WebGL para um elevado detalhe de visualização 3D. Esta biblioteca oferece uma API JSON baseada numa interface gráfica WebGL. É uma biblioteca especializada em renderização rápida de um grande número de objetos articulados e focada em *Computer Aided Design* (CAD), *e.g.*, para visualização médica e projetos de engenharia, em vez de se focar em sombras, reflexos e colisões que são características importantes em motores de jogo.

4.3.4 Comparação

A comparação das bibliotecas de desenvolvimento *Web* 3D apresentadas foi efetuada em termos da comunidade ativa, da documentação, do motor físico, do motor de colisões e da possibilidade de importação de modelos 3D externos. A Tabela 4.3 apresenta o resultado.

Tabela 4.3: Comparação das bibliotecas de desenvolvimento *Web* 3D [11].

	Three.js	**Babylon.js**	**Scene.js**
Open source	Sim	Sim	Sim
Comunidade	Ativa	Ativa	Inativa
Documentação	Rica	Fraca	Muito fraca
Motor físico	Não	Sim	Não
Motor de colisões	Não	Sim	Não
Importação de modelos 3D Externos	Sim	Sim	Sim

Perante a análise efetuada em relação às bibliotecas de desenvolvimento *Web* 3D optou-se por dar preferência à Babylon.js. Apesar de não ter uma documentação muito rica, tem motor físico e de colisões. Por outro lado, a sua comunidade encontra-se activa e é *open source*.

4.4 CSS Front-End Frameworks

No âmbito do desenvolvimento *Web* analisaram-se as plataformas Bootstrap, Foundation e Semantic UI.

4.4.1 Bootstrap

Atualmente, a Bootstrap [12] é, indiscutivelmente, a plataforma líder em termos de popularidade. Este fator é a principal força da plataforma, que apesar de não ser necessariamente melhor que as outras, oferece mais recursos como artigos, extensões e *templates* provenientes da sua comunidade ativa.

4.4.2 Foundation

A plataforma Foundation [12] é uma plataforma utilizada por muitos sites de grande relevo, incluindo Facebook, Mozilla, Ebay e National Geographic. Esta plataforma tem uma estrutura profissional com suporte ao negócio, à formação e à consultoria. De acrescentar que oferece muitos recursos de suporte à aprendizagem e utilização.

4.4.3 Semantic UI

Semantic UI [12] suporta a construção de aplicações *Web* com uma abordagem semântica. Para tal, utiliza os princípios da linguagem natural, tornando o código muito mais legível e compreensível. Esta plataforma é atualmente a mais inovadora e possuí bastantes recursos. A estrutura global da plataforma, em termos de lógica e de convenções de nomenclatura, supera as restantes.

4.4.4 Comparação

A comparação das CSS *Front-End Frameworks* apresentadas foi efetuada em termos da popularidade, do tamanho, dos extras, da documentação e dos navegadores suportados. A Tabela 4.4 apresenta a comparação.

Tabela 4.4: Comparação das CSS *Front-End Frameworks* [12] .

	Bootstrap	**Foundation**	**Semantic UI**
Stars no GitHub	75 900	18 000	12 900
Tamanho	145 kB	326 kB	552 kB
Extras/*Add-ons*	Terceiros	Sim	Sim
Documentação	Boa	Boa	Muito boa
Navegadores suportados	Firefox Chrome Safari IE8+	Firefox Chrome Safari IE9+ iOS Android Windows Phone 7+	Firefox Chrome Safari IE10+ Android 4 BlackBerry

Apesar de na análise ser evidente que a plataforma Foundation e Semantic UI apresentam uma melhor documentação e oferecem suporte a um número superior de navegadores, a escolha recaiu em Bootstrap devido à sua elevada comunidade ativa.

4.5 Conclusão

Neste capítulo foram analisadas as tecnologias e plataformas de suporte de maior relevo e foram escolhidas as soluções para o desenvolvimento da plataforma, nomeadamente, a plataforma de desenvolvimento de SMA JADE, o RESTlet para implementação de interfaces do tipo serviço *Web* e a plataforma CSS *Front-End* Bootstrap. Foram ainda apresentadas as principais bibliotecas de desenvolvimento *Web* 3D para desenvolvimentos futuros, que não foram adotadas.

No capítulo seguinte é detalhada a instalação do ambiente de desenvolvimento deste projeto que envolve as tecnologias e plataformas escolhidas.

Capítulo 5

Ambiente de Desenvolvimento

Neste capítulo é relatada a instalação do ambiente de desenvolvimento necessário para a construção da plataforma de competição de veleiros autónomos.

5.1 Instalação do Ambiente de Desenvolvimento

Para desenvolver a plataforma foi necessário escolher: (*i*) um ambiente integrado de desenvolvimento de SMA e de aplicações *Web*; (*ii*) dois servidores *Web* e (*iii*) dois servidores de armazenamento de dados. A instalação do ambiente de desenvolvimento foi efetuada num computador portátil com o sistema operativo Windows 10.

5.1.1 Ambiente Integrado de Desenvolvimento

O ambiente integrado de desenvolvimento escolhido para a construção da plataforma foi o NetBeans IDE versão 8.0.2. A sua escolha deveu-se ao facto de ser *open source*, possuir comandos de execução e depuração flexíveis e, apesar de ser completamente desenvolvido em Java, ser capaz de suportar outras linguagens de programação e tecnologias [39].

Na página oficial do ambiente integrado de desenvolvimento NetBeans IDE[1] é possível descarregar o executável de instalação – netbeans-8.1-windows.exe. Após a instalação, pode iniciar-se o desenvolvimento (ver Figura 5.1).

[1] https://netbeans.org/

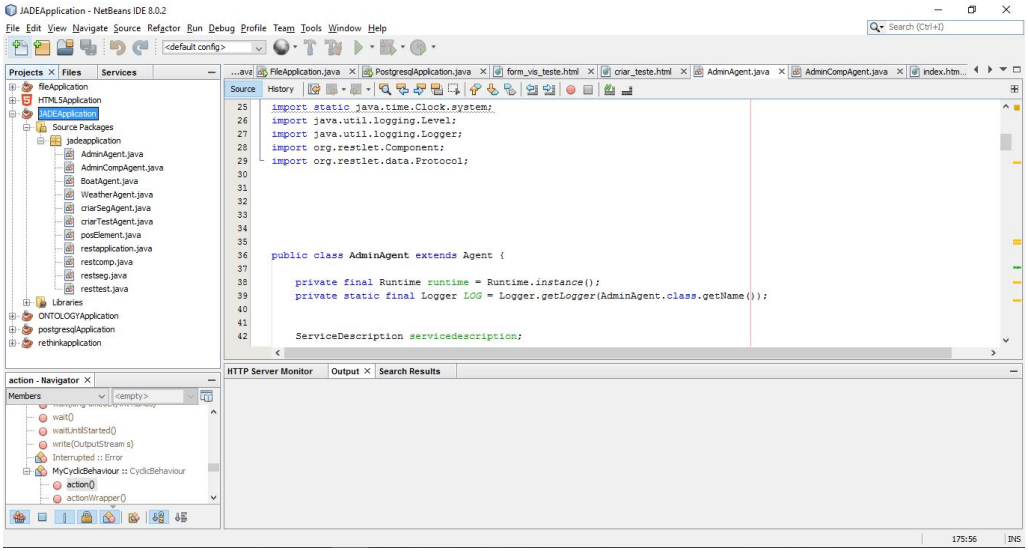

Figura 5.1: Interface NetBeans IDE.

5.1.2 Ambiente de Desenvolvimento de Sistemas Multi-agente

O ambiente de desenvolvimento de SMA escolhido foi a plataforma JADE descrita no capítulo anterior. Dado que se trata de uma plataforma baseada em Java, foi necessário instalar o *Java Run-Time Environment* (JRE) versão 1.8 e o JDK versão 1.8. O código da plataforma JADE (versão 4.3.3) foi descarregado da página oficial[2] e incluiu os seguintes ficheiros:

- jade-bin-4.3.3 – arquivos Java pré-compilados (.jar);
- jade-doc-4.3.3 – documentação para administradores e programadores;
- jade-examples-4.3.3 – código fonte de exemplos;
- jade-src-4.3.3 – código fonte JADE.

A instalação passou pela descompactação dos ficheiros e pela configuração das variáveis de ambiente $JADE_HOME$ e $CLASSPATH$ que correspondem, respetivamente, ao diretório da raiz da plataforma JADE e à biblioteca jade.jar. No NetBeans especificaram-se nas propriedades da aplicação a main.Class e os argumentos (ver Figura 5.2).

De forma a validar a correta instalação do JADE foi executado o seguinte comando na Linha de Comandos do Windows: “java -classpath lib\jade.jar jade.Boot -gui”(ver Figura 5.3).

[2]http://jade.tilab.com/

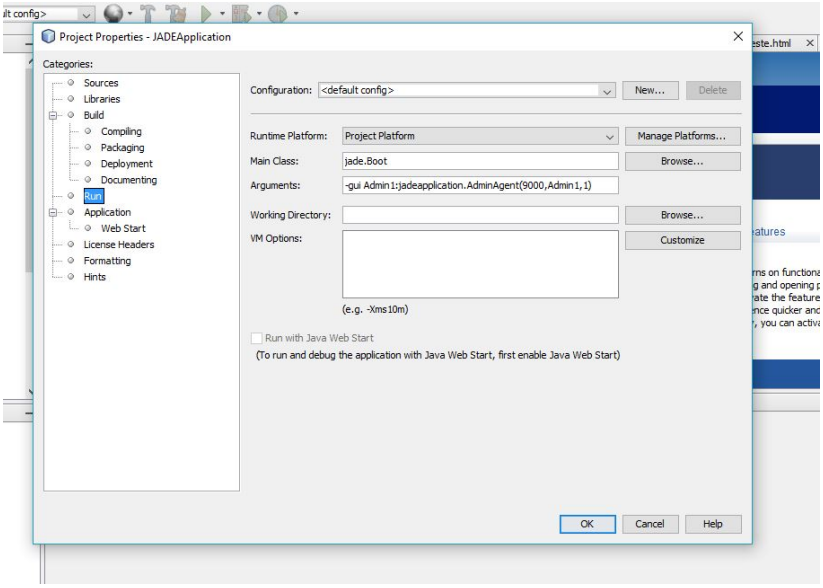

Figura 5.2: Especificação das propriedades da aplicação para executar JADE.

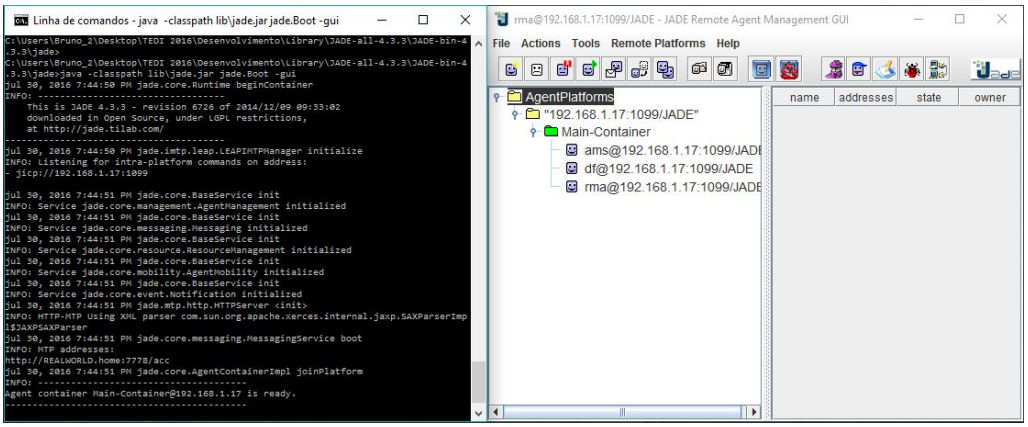

Figura 5.3: Interface JADE.

5.1.3 Ambiente de Desenvolvimento para Web

Em relação ao ambiente de desenvolvimento de aplicações *Web*, foi utilizada a *framework* Bootstrap. No NetBeans foi criado um novo projeto, incluído na categoria de aplicações HTML5 (configurado para HTML5, CSS e JavaScript), e definido, como *template*, o "Initializr: Bootstrap"(ver Figura 5.4).

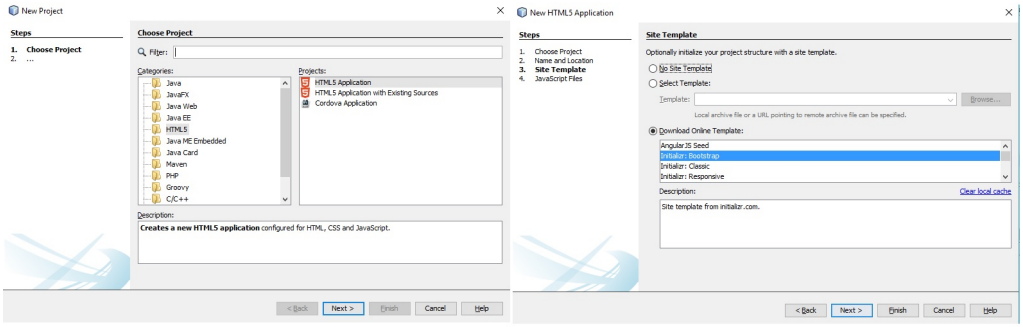

Figura 5.4: Instalação do template Bootstrap no NetBeans.

5.1.4 Servidores Web

No NetBeans foi configurado o servidor *Web* Apache Tomcat versão 8.0.15.0 para a disponibilização de aplicações *Web* (ver Figura 5.5). Devido à necessidade de utilizar um servidor que, para além da disponibilização de aplicações *Web*, permitisse também uma comunicação através de WebSockets, foi necessário incluir um servidor NodeJS.

Para se desenvolver um ambiente NodeJS foi necessário selecionar um editor de texto, tendo sido escolhido o Notepad++ versão 6.9.2. Os ficheiros criados no editor de texto com a extensão .js contêm o código-fonte JavaScript das aplicações que serão compilados e executados pelo NodeJS.

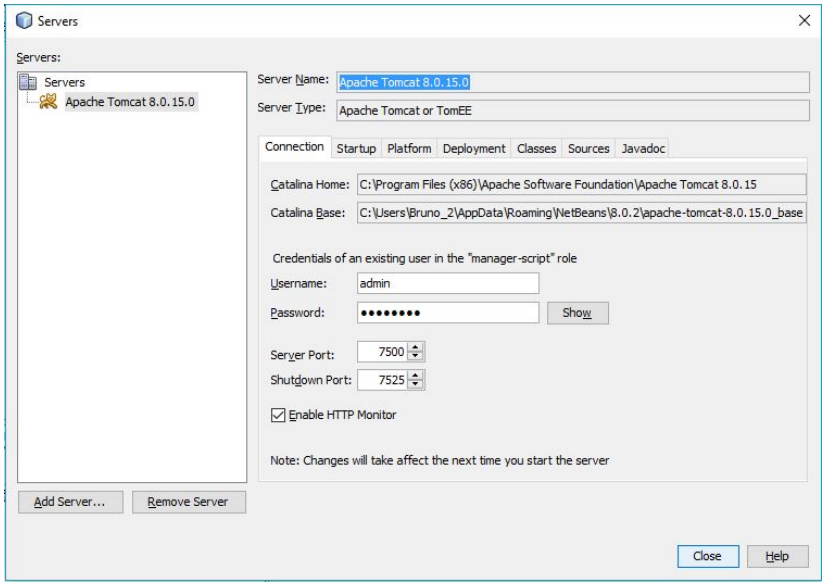

Figura 5.5: Configuração do servidor *Web* Apache Tomcat versão 8.0.15.0.

Na página oficial do NodeJS[3] é possível descarregar o ficheiro MSI – node-v4.3.2-x64.msi – e efetuar, de seguida, a instalação. Para verificar a correta instalação do NodeJS foram realizados os seguintes testes (os resultados dos testes podem ver visualizados na Figura 5.6):

- Validação da versão instalada – na Linha de Comandos do Windows executou-se o comando: "node -v" – Resultado esperado: "v4.3.2";
- Criação e execução de um código-fonte de teste – criou-se o ficheiro helloworld.js com o editor de texto, adicionou-se a linha de código "console.log('TEDI 2016 – NodeJS')" e executou-se "node helloworld.js" na Linha de Comandos do Windows – Resultado esperado: "TEDI 2016 – NodeJS".

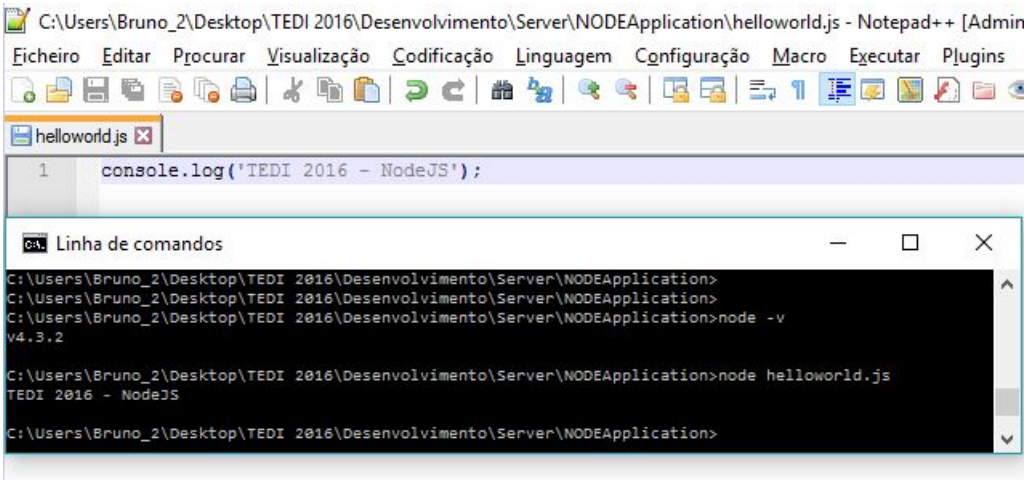

Figura 5.6: Teste de instalação do servidor NodeJS.

5.1.5 Servidores de Armazenamento de Dados

Foram instalados dois servidores para armazenamento de dados. O servidor PostgreSQL versão 9.5 foi o escolhido para o armazenamento de dados de forma persistente da plataforma e para a transmissão de dados em tempo real selecionou-se o servidor RethinkDB versão 2.2.4.

Na página oficial do PostgreSQL[4] é possível descarregar o ficheiro de instalação – postgresql-9.5.0-win64.exe – e realizar, de seguida, a respetiva instalação.

[3] https://nodejs.org/en/
[4] https://www.postgresql.org/

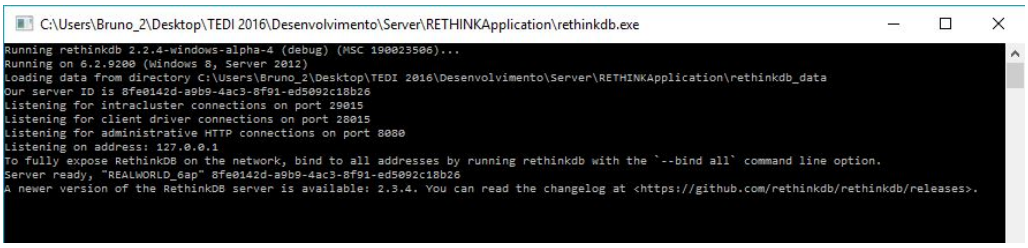

Figura 5.7: Execução do servidor RethinkDB.

Em relação ao RethinkDB, não foi necessário efetuar uma instalação, mas apenas descarregar um executável da página oficial do RethinkDB[5]. Esse executável deverá ser executado sempre que se pretenda ativar o servidor (ver Figura 5.7).

5.2 Conclusão

Neste capítulo foi apresentada a instalação do ambiente de desenvolvimento do projeto suportado exclusivamente por soluções *open source*.

No próximo capítulo é descrito o desenvolvimento da plataforma de competição de veleiros autónomos com este ambiente de desenvolvimento.

[5]https://www.rethinkdb.com/

Capítulo 6

Desenvolvimento da Plataforma

Neste capítulo é apresentado o desenvolvimento da plataforma de competição de veleiros autónomos.

6.1 Arquitetura da Plataforma

A arquitetura da plataforma, representada na Figura 6.1, encontra-se dividida em dois grandes blocos: *Front-End* e *Back-End*. O *Back-End* inclui, em termos de componentes, um servidor *Web* (Apache), um servidor NodeJs, dois servidores de bases de dados (PostgreSQL e RethinkDB), uma plataforma de execução de agentes (JADE) e uma plataforma RESTful API (RESTlet). O *Front-End* contempla dois tipos de clientes: os navegadores e os agentes de *software*.

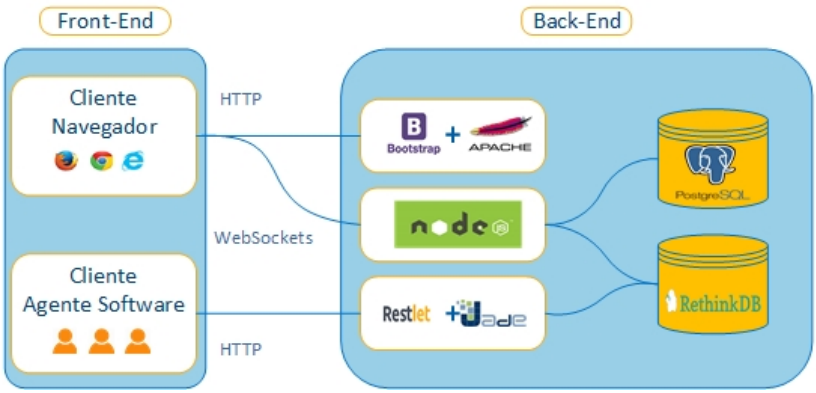

Figura 6.1: Arquitetura da plataforma de competição de veleiros autónomos.

6.1.1 JADE e Restlet

A plataforma JADE, juntamente com a API RESTlet, corresponde ao núcleo do sistema, *i.e.*, onde as competições reais e simuladas são representadas. Cada competição é modelada através de um SMA, que corresponde a um contentor da plataforma JADE, e onde os veleiros participantes (simulados ou reais) são modelados através de agentes. Ao nível da arquitetura, a plataforma JADE é constituída por diversos contentores que comportam coleções de agentes.

Neste sistema foram considerados três tipos de contentores (ver Figura 6.2):

- Contentor Principal – constituído pelos agentes principais da plataforma JADE;
- Contentor RethinkDB – representado pelos agentes que ficam à escuta de eventos na base de dados RethinkDB;
- Contentor Competição – constituído pelo agente representante da competição (real ou simulada) e pelos agentes representantes das embarcações participantes na competição. O sistema possuí um contentor Competição por cada competição ativa.

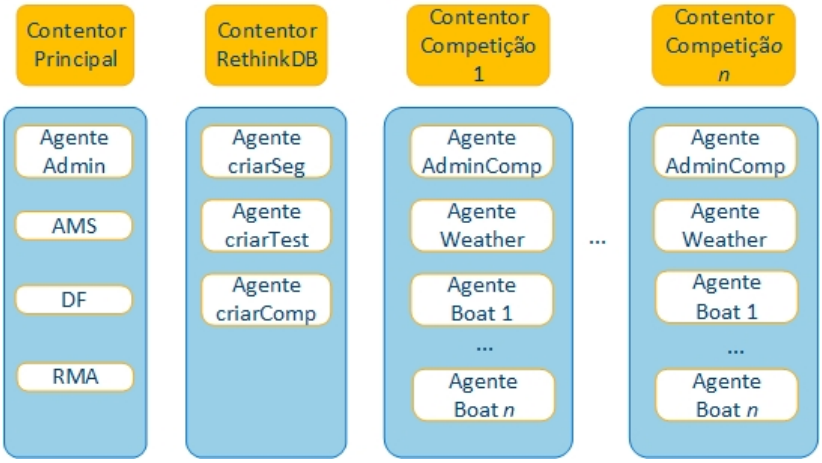

Figura 6.2: Arquitetura da plataforma JADE.

Em relação aos agentes, na plataforma JADE foram considerados dez tipos de agentes:

- Agente AdminComp – implementa o papel de júri das competições. Este agente é responsável por controlar o tempo das provas, validar a posição dos participantes, a passagem dos participantes pelas bóias e verificar a existência de colisões;

- *Agent Management System* (AMS) – gere a plataforma e oferece um serviço de páginas brancas onde todos os agentes se registam e lhes é atribuído um *Agent Identifier* (AID);

- *Directory Facilitator* (DF) – oferece um serviço de páginas amarelas onde os agentes registam os detalhes dos serviços que proporcionam;

- *Remote Management Agent* (RMA) – disponibiliza uma interface gráfica para a administração da plataforma e dos restantes agentes;

- Agente Admin – implementa a interface RESTlet na plataforma JADE. No arranque do sistema, é responsável por criar o contentor RethinkDB e os respetivos agentes;

- Agente criarSeg – monitora o estado da base de dados RethinkDB para verificar se é necessário criar uma competição do tipo Seguimento. O contentor Competição, juntamente com os agentes, é criado pelo agente;

- Agente criarTest – monitora o estado da base de dados RethinkDB para verificar se é necessário criar uma competição do tipo Teste. O contentor Competição, juntamente com os agentes, é criado pelo agente;

- Agente criarComp – monitora o estado da base de dados RethinkDB para verificar se é necessário criar uma competição do tipo Simulação. O contentor Competição, juntamente com os agentes, é criado pelo agente;

- Agente Weather – responsável pela modelação das condições meteorológicas da competição (velocidade e direção do vento);

- Agente Boat – realiza a modelação dos veleiros autónomos na plataforma. A modelação foi baseada na existente no *software open source* TrackSail e contempla: (*i*) a posição e a velocidade do veleiro; (*ii*) a velocidade e a direção do vento; (*iii*) a direção da vela e (*v*) a direção do leme. A esta modelação foi acrescentada a influência da massa e da área vélica do veleiro. Tal como no TrackSail [40], a modelação não contempla a influência das ondas nem das correntes.

A interação entre a plataforma JADE e os agentes de *software* é realizada através da interface REST resultante da integração da plataforma RESTlet com a plataforma JADE. A plataforma JADE é responsável por introduzir todos os dados das competições na base de dados RethinkDB.

6.1.2 NodeJS

O NodeJS [41] é um servidor criado sobre o interpretador V8 JavaScript da Google que adota um procedimento de execução baseado em eventos não bloqueantes para tornar as aplicações mais velozes e escalonáveis. Este procedimento permite atualizações automáticas das aplicações *Web*. A comunicação com os navegadores clientes utiliza o protocolo WebSockets implementado pela biblioteca Socket.IO. A Socket.IO [42] é uma biblioteca em JavaScript que permite desenvolver aplicações com aptidão para comunicações bidirecionais cliente/servidor e em tempo real.

No servidor NodeJS foram criados quatro tipos de instância servidor (ver Figura 6.3):

- Login.js – responsável pelas funcionalidades de pedidos de inscrição e de acesso à plataforma;
- AdministrativoAdmin.js – implementa as funcionalidades associadas à administração da plataforma (criar e listar simulações, criar e listar seguimentos e validar pedidos de inscrição);
- AdministrativoUser.js – oferece as funcionalidades afetas aos utilizadores da plataforma (criar e listar testes, listar seguimentos e listar simulações);
- Comp.js – responsável pelas funcionalidades de suporte à visualização das competições. Estas funcionalidades atualizam as páginas com o estado da competição (estado meteorológico e posição dos veleiros) e apresentam os eventos (início da competição, passagem de veleiros pelas bóias, colisões e fim da competição). Sempre que é solicitada a criação de uma competição, os agentes criarSeg, criarTest ou criarComp da plataforma JADE são responsáveis pela respetiva instância servidor Comp.js, ou seja, por cada competição ativa irá existir uma instância servidor do tipo Comp.js.

Figura 6.3: Arquitetura do servidor NodeJS.

De referir que o servidor NodeJS, para oferecer as funcionalidades apresentadas, nas instâncias referidas, realiza leituras e escritas nas base de dados PostgreSQL e RethinkDB (ver Figura 6.1). A plataforma JADE apenas realiza leituras e escritas na base de dados RethinkDB.

6.1.3 PostgreSQL

PostgreSQL [43] é um sistema de gestão de base de dados relacional, ou seja, permite a definição de bases de dados constituídas por múltiplas tabelas que podem estar relacionadas. Este sistema é *open source* e, em termos de linguagem de interrogação, adota a *Structured Query Language* (SQL).

No Anexo B, pode ser visualizado o modelo *Enhanced Entity Relationship* (EER) da base de dados PostgreSQL criada e constituída por sete tabelas:

- Tabela Login – contém os dados de acesso dos utilizadores registados;
- Tabela Comps – possuí a informação das competições do tipo Simulação;
- Tabela Tests – armazena os dados das competições do tipo Teste;
- Tabela Segs – representa a informação das competições do tipo Seguimento;
- Tabela Teams – contém os dados das equipas registadas na plataforma;
- Tabela Boats – armazena os dados dos veleiros referentes às equipas registadas na plataforma;
- Tabela Ports – tabela de gestão de portos do servidor criada devido à necessidade de cada instância servidor NodeJS e cada agente JADE ocupar um porto individual no servidor principal da plataforma. Foi definido que do porto 7010 ao 7499 ficaria reservado para os agentes JADE e do porto 7500 ao 7999 ficaria para as instâncias servidor NodeJS.

6.1.4 RethinkDB

RethinkDB [44] é uma base de dados JSON *open source*, *Not only* SQL (NoSQL) e escalável para aplicações *Web* com necessidades de execução de funcionalidades em tempo real. Enquanto que a base de dados PostgreSQL armazena a informação de cariz mais estático, a base de dados RethinkDB suporta a atualização automática das páginas *Web* das competições através do armazenamento da informação dinâmica das competições em tempo real.

A estrutura de base de dados representada na Figura 6.4 e implementada na plataforma contém sete tipos de documentos de armazenamento de dados.

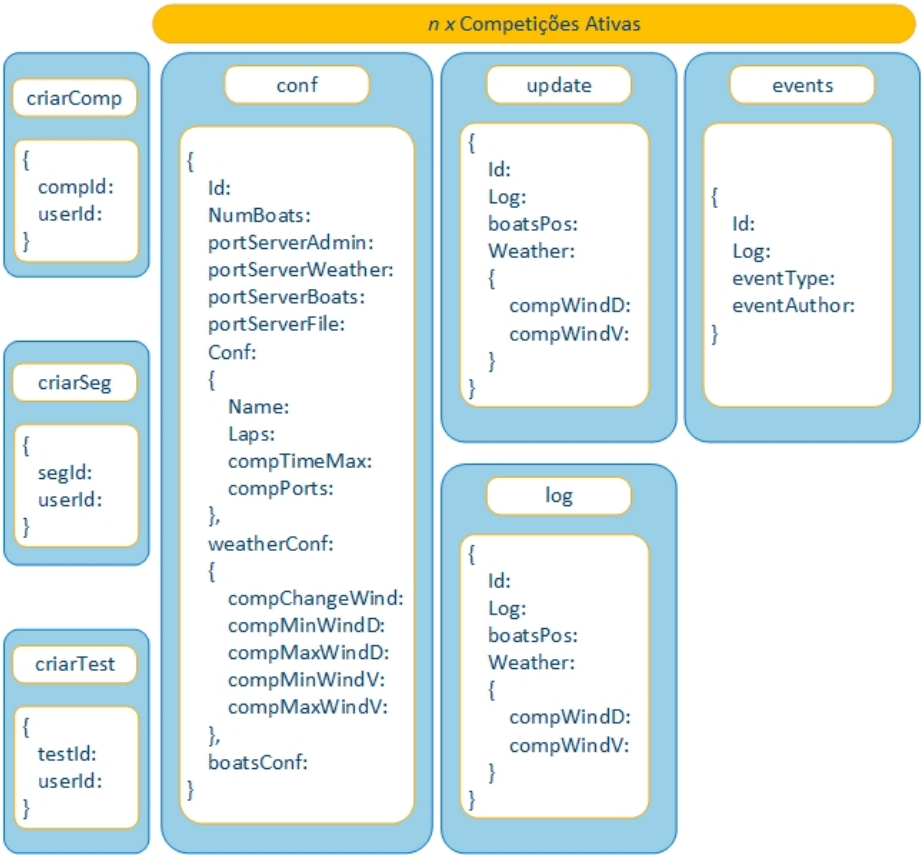

Figura 6.4: Estrutura de dados RethinkDB.

Os documentos criarComp, criarSeg e criarTest são responsáveis por armazenar os pedidos de criação de competição que irão originar a criação dos contentores das respetivas competições na plataforma JADE. O restante grupo de documentos é gerado individualmente, para cada competição ativa e armazena os dados referentes às configurações (documento Conf), ao histórico (documento Log), às atualizações (documento Update) e aos eventos (documento Events) das respetivas competições.

6.1.5 Agentes de Software

Os agentes de *software* interagem com a plataforma através da API do tipo serviço *Web* RESTful criada através plataforma RESTlet. Cada agente de *software*, que implementa do lado do cliente um determinado algoritmo de navegação, comunica através da API as ações a realizar pelo seu homónimo na plataforma e recebe o estado da sua competição.

Nas Tabelas 6.1, 6.2 e 6.3 encontram-se enumerados os recursos da plataforma.

Tabela 6.1: Recursos REST para competições do tipo Seguimento.

	Recurso Seg 1
Descrição	Definir a posição do Agente Boat e saber o estado da competição
Método	GET
URI	/client-agent/seg/boat/{idSeg}/{boatNome}/{posLat}/{posLon}
Parâmetros	idSeg – identificação da competição ; boatNome – nome do Agente Boat ; posLat – posição nova do Agente Boat em relação à latitude (°); posLon – posição nova posição do Agente Boat em relação à longitude (°);
Resposta	/{boatNome}/{posLat}/{posLon}/{velW}/{dirW}
Parâmetros	boatNome – nome do Agente Boat; posLat – posição atual do Agente Boat em relação à latitude (°); posLon – posição atual do Agente Boat em relação à longitude (°); velW – velocidade do vento do Agente Weather (km/h); dirW – direção do vento do Agente Weather (°);
	Recurso Seg 2
Descrição	Definir a direção e a velocidade do vento do Agente Weather
Método	GET
URI	/client-agent/seg/weather/{idSeg}/{compWindV}/{compWindD}
Parâmetros	idSeg – identificação da competição; compWindV – velocidade nova do vento do Agente Weather (km/h); compWindD – direção nova do vento do Agente Weather (°);
Resposta	/{idSeg}/{compWindV}/{compWindD}
Parâmetros	idSeg – identificação da competição; compWindV – velocidade atual do vento do Agente Weather (km/h); compWindD – direção atual do vento do Agente Weather (°);

Tabela 6.2: Recursos REST para competições do tipo Teste.

	Recurso Test 1
Descrição	Definir posição dos atuadores e saber o estado da competição
Método	GET
URI	/client-agent/test/boat/{idTest}/{boatNome}/{leme}/{vela}
Parâmetros	idTest – identificação da competição; boatNome – nome do Agente Boat; leme – direção nova do leme do Agente Boat (°); vela – direção nova da vela do Agente Boat (°);
Resposta	/{boatNome}/{posX}/{posY}/{leme}/{vela}/{velW}/{dirW}
Parâmetros	boatNome – nome do Agente Boat; posX – posição atual do Agente Boat em relação ao eixo do X (m); posY – posição atual do Agente Boat em relação ao eixo do Y (m); leme – direção do leme do Agente Boat (°); vela – direção da vela do Agente Boat (°); velW – velocidade do vento do Agente Weather (km/h); dirW – direção do vento do Agente Weather (°);

Tabela 6.3: Recursos REST para competições do tipo Simulação.

	Recurso Sim 1
Descrição	Definir posição dos atuadores e saber o estado da competição
Método	GET
URI	/client-agent/comp/boat/{idComp}/{boatNome}/{leme}/{vela}
Parâmetros	idComp – identificação da competição; boatNome – nome do Agente Boat; leme – direção nova do leme do Agente Boat (°); vela – direção nova da vela do Agente Boat (°);
Resposta	/{boatName}/{posX}/{posY}/{leme}/{vela}/{velW}/{dirW}
Parâmetros	boatNome – nome do Agente Boat; posX – posição atual do Agente Boat em relação ao eixo do X (m); posY – posição atual do Agente Boat em relação ao eixo do Y (m); leme – direção do leme do Agente Boat (°); vela – direção da vela do Agente Boat (°); velW – velocidade do vento do Agente Weather (km/h); dirW – direção do vento do Agente Weather (°);

No Anexo C podem ser visualizados os diagramas de sequência que ilustram a interação com os recursos REST identificados.

6.1.6 Clientes Navegadores

Na Figura 6.5 é apresentado o mapa de navegação da interface *Web* disponibilizada aos navegadores.

A página de entrada é apresentada aos utilizadores não autenticados e é constituída pelo nome da plataforma (Plataforma de Competição de Veleiros Autónomos), por uma pequena apresentação dos autores (Instituição, Estudante, Orientadora e Projeto) e oferece as seguintes funcionalidades:

- Autenticação – autenticação do utilizador composta pelo nome de utilizador e pela senha;
- Registo – pedido de acesso dos utilizadores composto pelos campos referentes à autenticação (nome de utilizador e senha), à equipa (nome da equipa, país, cidade, instituição/empresa, *Uniform Resource Locator* (URL) e *email*) e ao veleiro autónomo (nome do veleiro, massa (kg), boca (m), calado (m), comprimento (m), área vélica (m^2), fontes de alimentação, atuadores, sensores, tipos de comunicações e computadores).

O menu principal, dependendo do tipo de utilizador autenticado, será o menu participante (utilizadores registados e autenticados como participantes) ou o

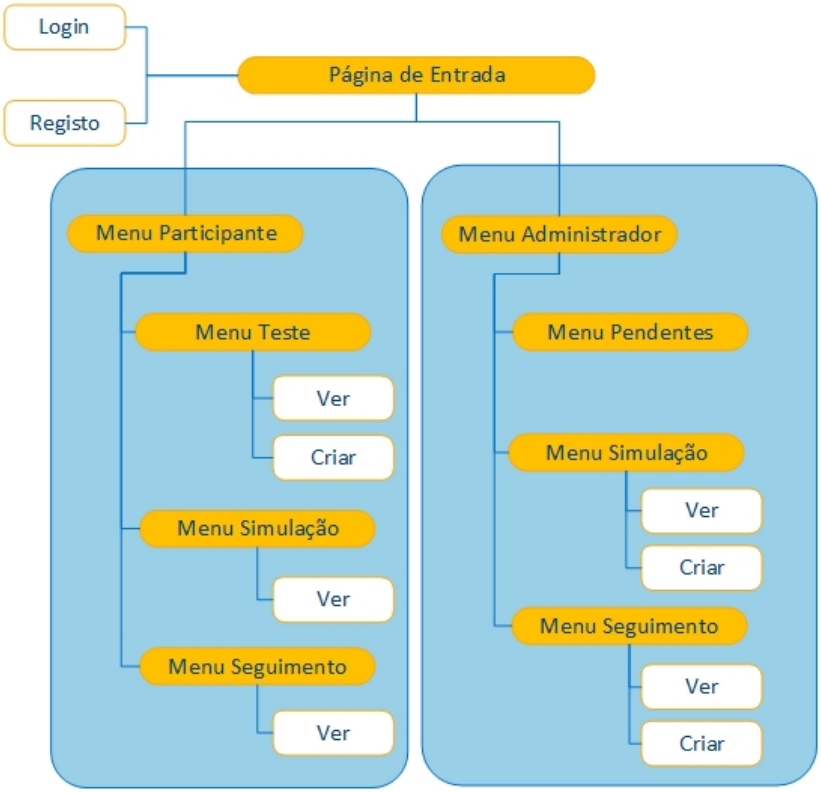

Figura 6.5: Mapa de navegação.

menu administrador (utilizadores autenticados como administradores). O menu participante disponibiliza três tipos de funcionalidades:

- Menu Teste – o utilizador pode criar uma competição do tipo teste através de um *upload* de um ficheiro de extensão .txt (ver Figura 6.6) e visualizar o decorrer da competição do tipo teste criada;
- Menu Simulação – permite ao utilizador selecionar e observar uma competição do tipo simulação criada por um utilizador administrador;
- Menu Seguimento – o utilizador pode escolher e visualizar uma competição do tipo simulação criada por um utilizador administrador.

Em relação ao menu administrador, são oferecidos três tipos de funcionalidades:

- Menu Pendentes – o utilizador valida os utilizadores participantes que solicitaram a autorização para utilizar a plataforma e se achar que os dados

introduzidos durante o registo se encontram corretos aprova o pedido. Após a validação e aprovação do registo, o utilizador participante passa a ter permissão de autenticação na plataforma;

- Menu Simulação – permite ao utilizador criar uma competição simulação através de um *upload* de um ficheiro de extensão .txt (ver Figura 6.6) e visualizar o decorrer da competição do tipo simulação criada;
- Menu Seguimento – o utilizador pode criar uma competição do tipo seguimento através de um *upload* de um ficheiro de extensão .txt (ver Figura 6.6) e observar o desenrolar da competição criada.

Relativamente à componente de visualização dos percursos realizados pelos veleiros nas competições, a interface *Web* foi desenvolvida com recurso à biblioteca Leaflet. A Leaflet [45] é uma biblioteca JavaScript *open source* para a criação de mapas interativos compatíveis com dispositivos móveis.

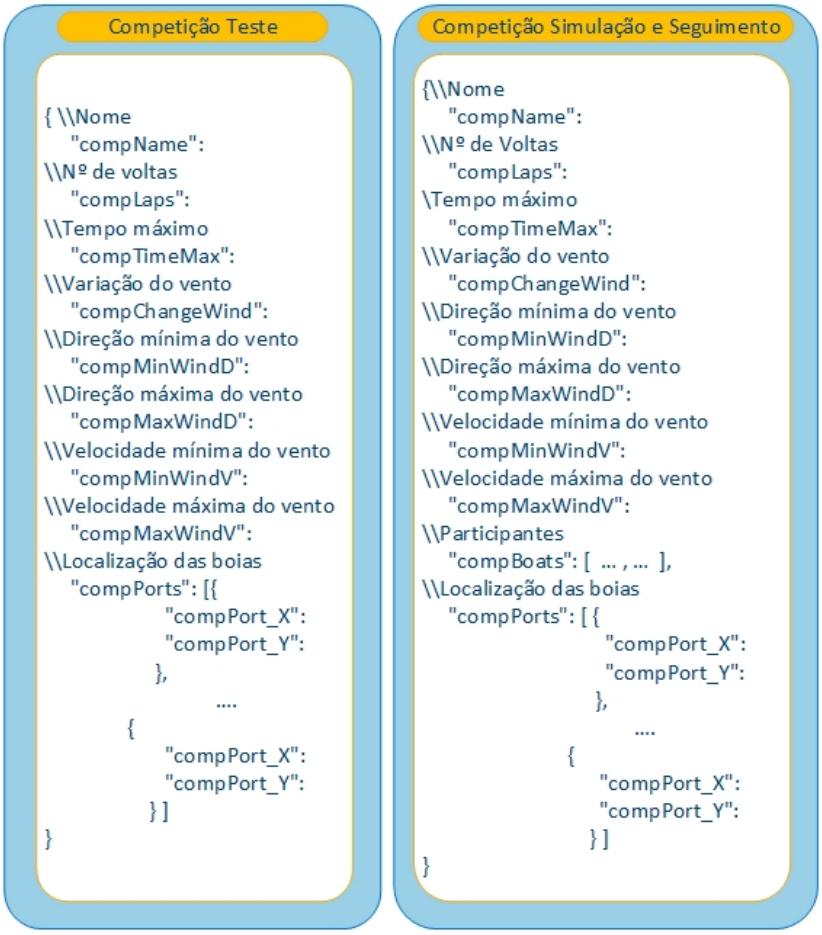

Figura 6.6: Estrutura dos ficheiros para criação das competições.

6.2 Modelação dos Agentes

A modelação dos agentes Boat e Weather foi baseada na modelação existente no *software open source* TrackSail. Ambos os agentes apresentam um comportamento do tipo TickerBehaviour (comportamento periódico) na plataforma JADE, *i.e.*, são executados com uma dada frequência. O tempo considerado foi 1 s.

O fluxograma representativo do comportamento cíclico do Agente Boat encontra-se representado no Anexo D. Trata-se de uma modelação física simples que calcula a velocidade e a posição de um veleiro através da direção e da velocidade do vento, da direção do próprio veleiro e da direção da vela.

No Anexo D encontra-se também representado o fluxograma referente ao comportamento cíclico do Agente Weather. A modelação do Agente Weather altera aleatoriamente, mas sem exceder a gama de variação especificada, a velocidade e a direção do vento. A variação aleatória serve para dar realismo à simulação e é definida através dos seguintes parâmetros existentes no ficheiro de criação submetido pelo utilizador: (*i*) compMinWindV – Velocidade mínima do vento (km/h); (*ii*) compMaxWindV – Velocidade máxima (km/h) (*iii*) compMinWindD – Direção mínima do vento (°); (*iv*) compMaxWindD – Direção máxima do vento (°) e (*v*) compChangeWind – Variação máxima da velocidade e da direção do vento.

6.3 Regras de Navegação

A verificação do cumprimento das regras de navegação de uma competição é efetuada pelo agente AdminComp. O agente AdminComp apresenta dois tipos de comportamento na plataforma JADE: (*i*) OneShotBehaviour e (*ii*) TickerBehaviour. O comportamento OneShotBehaviour efectua o processamento numa única fase de execução, *i.e.*, é construído e executado sempre que existe uma alteração de posicionamento num veleiro da competição. Tal como os agentes Boat e Weather, o comportamento TickerBehaviour é executado com uma frequência de 1 Hz.

No Anexo E encontram-se representados os fluxogramas dos dois comportamentos. O AdminComp controla o tempo das provas, valida a posição dos participantes, o cumprimento do percurso e deteta a existência de colisões entre veleiros.

6.4 Conclusão

Neste capítulo, reservado ao desenvolvimento da plataforma de competição de veleiros autónomos, foi descrita a arquitetura e os seus módulos constituintes que permitiram cumprir com os objetivos propostos inicialmente, *i.e.*, cumpriu com os requisitos determinados e os casos de uso propostos.

No próximo capítulo é detalhado os testes realizados com base nos cenários identificados nos testes funcionais e são analisados os resultados obtidos.

Capítulo 7

Testes e Resultados

Neste capítulo são descritos os testes funcionais e de modelação física e, por último, são analisados os resultados obtidos.

7.1 Teste Funcional

Este teste encontra-se subdividido em cinco testes: (*i*) Página de Entrada – visualização da página; (*ii*) Autenticação – registo, aprovação e autenticação de um utilizador participante; (*iii*) Competição Teste – criação e visualização de uma competição do tipo teste; (*iv*) Competição Simulação – criação e visualização de uma competição do tipo simulação e (*v*) Competição Seguimento – criação e visualização de uma competição do tipo seguimento.

7.1.1 Página de Entrada

O utilizador não autenticado que acede à plataforma através do navegador obtém a página de entrada representada na Figura 7.1. Na página de entrada encontra-se o nome da plataforma, a apresentação dos autores, um botão para aceder ao formulário de pedido de acesso e o formulário de autenticação com: (*i*) um campo de texto para introdução do nome de utilizador; (*ii*) um campo de texto para inserção da senha e (*iii*) um botão de submissão das credenciais.

Figura 7.1: Página de entrada.

7.1.2 Autenticação

Para se autenticar, o utilizador necessita previamente de preencher o formulário de pedido de acesso ilustrado na Figura 7.2, introduzindo os dados de autenticação (nome de utilizador e senha), da equipa (nome da equipa, país, cidade, instituição, URL e *email*) e do veleiro autónomo (nome do veleiro, massa (kg), boca (m), área vélica (m^2), calado (m), comprimento (m), fontes de alimentação, atuadores, sensores, tipos de comunicações e computadores) e aguardar que o administrador da competição valide e aprove a sua participação na plataforma.

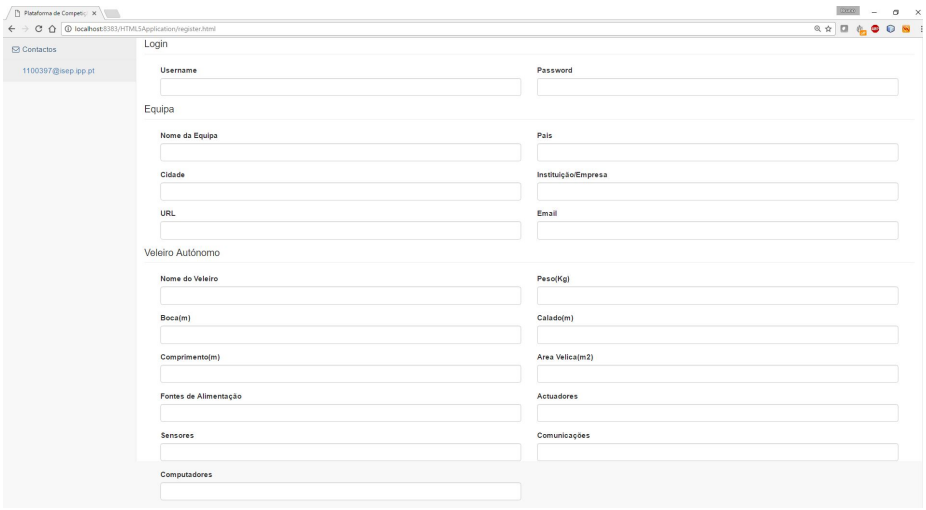

Figura 7.2: Página de registo.

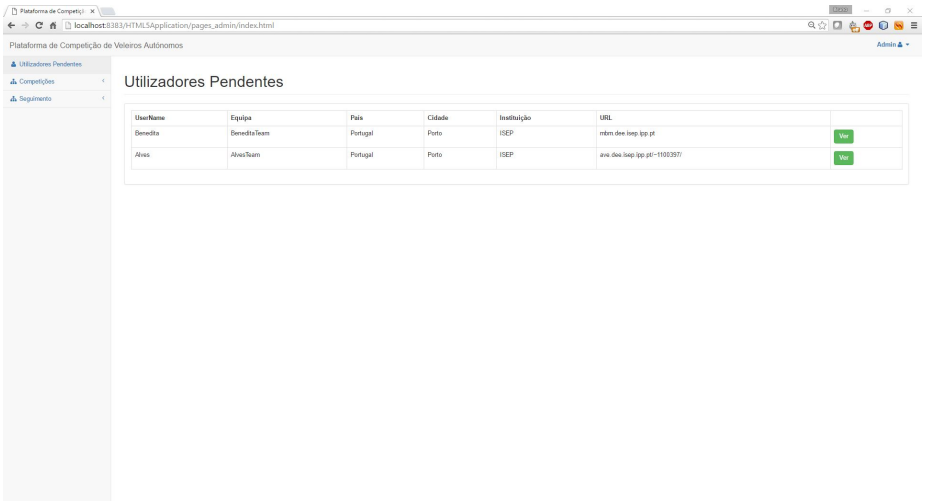

Figura 7.3: Página de pedidos pendentes.

O administrador, através do menu pendentes, visualiza a lista de utilizadores pendentes de aprovação (ver Figura 7.3) e aprova ou rejeita os pedidos de acesso à plataforma.

Para a realização dos restantes testes foram criados dois clientes navegadores participantes com os dados descritos na Anexo F. Estes clientes irão ser utilizados para simular duas equipas nos três tipos de competição que a plataforma oferece (teste, simulação e seguimento).

7.1.3 Competição de Teste

A competição do tipo teste foi criada pelo cliente navegador Alves através do ficheiro que se encontra no Anexo G. Durante a criação do teste, a plataforma JADE criou o contentor da competição (TEST-437) com os agentes Admin (AdminComp-437), Boat (Boat-AlvesBoat-437) e Weather (Weather-437).

Na Figura 7.4 é apresentado o aspeto visual oferecido ao cliente navegador com a informação da competição (id da competição, número de voltas, tempo máximo da prova (s), número de bóias e direção (°) e velocidade do vento (km/h)), a listagem dos eventos, a representação das bóias, a informação de todos os pontos do trajeto realizado pelo veleiro (nome do veleiro, ângulo do leme (°) e da vela (°), velocidade (kn) e as coordenadas de posição (eixo dos X (m) e eixo dos Y (m))) e a posição atual do veleiro.

Na Tabela 7.1 encontra-se um pedido e resposta REST da competição de teste.

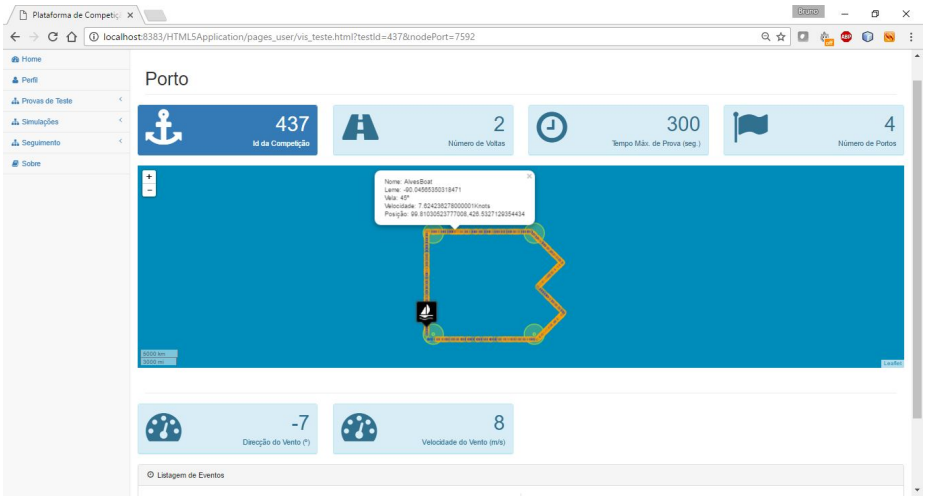

Figura 7.4: Página de uma competição do tipo teste.

Tabela 7.1: Recurso REST da competição do tipo Teste – AlvesBoat.

Pedido	/client-agent/test/boat/437/AlvesBoat/-90/45
Parâmetros	Identificação da competição: 437; Nome do veleiro: AlvesBoat; Direção do leme: -90°; Direção da vela: 45°;
Resposta	/AlvesBoat/99/426/-90/45/8/-7
Parâmetros	Nome veleiro: AlvesBoat; Posição do veleiro em relação ao eixo do X: 99 m; Posição do veleiro em relação ao eixo do Y: 426 m; Direção do veleiro: -90°; Direção da vela: 45°; Velocidade do vento: 8 km/h; Direção do vento: -7°;

7.1.4 Competição de Simulação

A competição do tipo simulação foi testada com dois veleiros simulados dos clientes navegadores Alves e Benedita. No Anexo H encontra-se o ficheiro utilizado pelo administrador da plataforma para a criação da competição. A plataforma JADE criou o contentor da competição (COMP-120) com os agentes Admin (AdminComp-120), Boat (Boat-AlvesBoat-120 e Boat-BeneditaBoat-120) e Weather (Weather-120).

O aspeto visual da competição está representado na Figura 7.5, sendo ainda disponibilizada aos clientes navegadores a informação da competição (id da com-

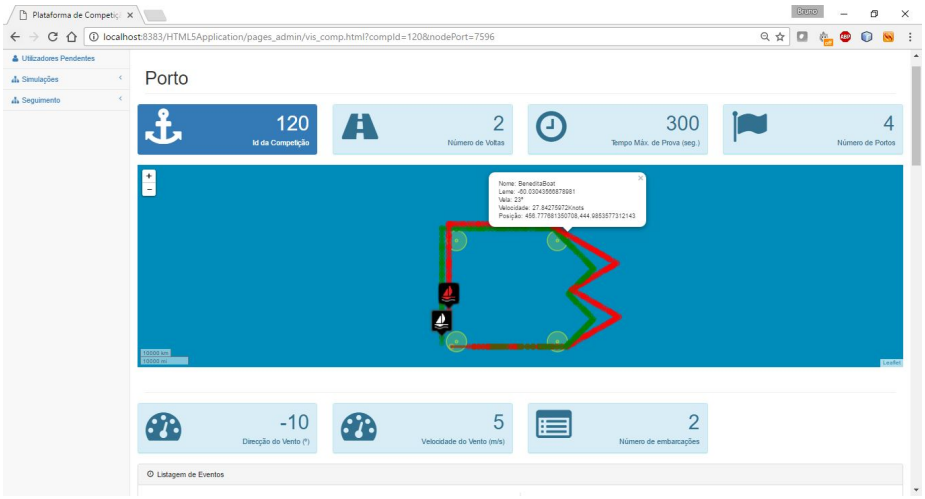

Figura 7.5: Página de uma competição do tipo simulação.

petição, número de voltas, tempo máximo da prova (s), número de bóias, direção (°) e velocidade do vento (km/h) e número de embarcações), a listagem dos eventos, a representação das bóias, a informação de todos os pontos do trajeto realizado por cada veleiro (nome do veleiro, ângulo do leme (°) e da vela (°), velocidade (kn) e coordenadas de posição (eixo dos X (m) e eixo dos Y (m))) e a posição atual dos veleiros.

Nas Tabelas 7.2 e 7.3 encontra-se um pedido e resposta REST relativo aos veleiros AlvesBoat e BeneditaBoat.

Tabela 7.2: Recurso REST da competição do tipo Simulação – AlvesBoat.

Pedido	/client-agent/comp/boat/120/AlvesBoat/180/-30
Parâmetros	Identificação da competição: 120; Nome do veleiro: AlvesBoat; Direção do leme: 180°; Direção da vela: -30°;
Resposta	/AlvesBoat/72/468/-90/45/5/-10
Parâmetros	Nome veleiro: AlvesBoat; Posição do veleiro em relação ao eixo do X: 72 m; Posição do veleiro em relação ao eixo do Y: 468 m; Direção do veleiro: -90°; Direção da vela: 45°; Velocidade do vento: 5 km/h ; Direção do vento: -10°;

Tabela 7.3: Recurso REST da competição do tipo Simulação – BeneditaBoat.

Pedido	/client-agent/comp/boat/120/BeneditaBoat/-60/23
Parâmetros	Identificação da competição: 120; Nome do veleiro: BeneditaBoat; Direção do leme: -60°; Direção da vela: 23°;
Resposta	/BeneditaBoat/456/444/-60/23/5/-10
Parâmetros	Nome veleiro: BeneditaBoat; Posição do veleiro em relação ao eixo do X: 456 m; Posição do veleiro em relação ao eixo do Y: 444 m; Direção do veleiro: -60°; Direção da vela: 23°; Velocidade do vento: 5 km/h; Direção do vento: -10°;

7.1.5 Competição de Seguimento

Para o teste da competição do tipo seguimento foi feito um percurso terrestre com dois veículos automóveis. Foi instalado um recetor *Global Navigation Satellite System* (GNSS) da Holux[1], modelo GM-210, para fazer o seguimento. Os dados foram tratados num computador local e enviados através da utilização dos recursos REST para a plataforma.

No Anexo I encontra-se o ficheiro utilizado pelo administrador da plataforma

[1]http://www.holux.com/

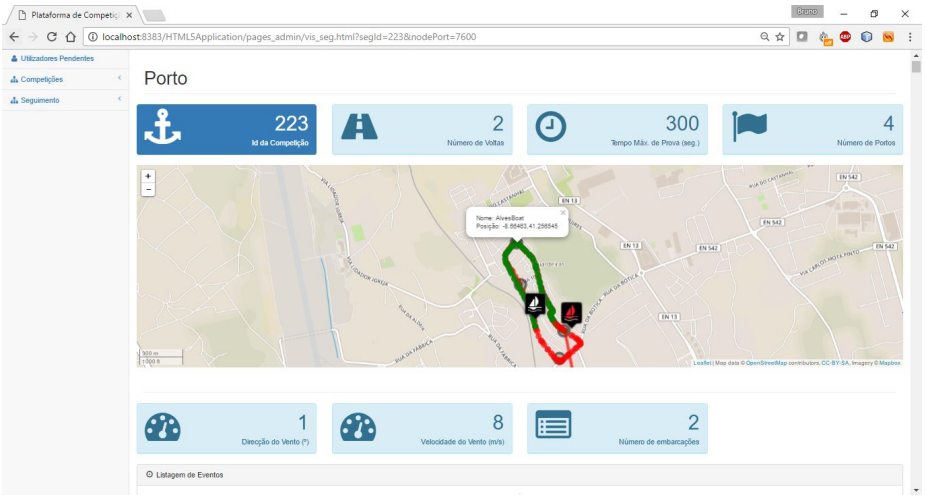

Figura 7.6: Página de uma competição do tipo seguimento.

para a criação da competição. A plataforma JADE criou o contentor da competição (SEG-223) com os agentes Admin (AdminSeg-223), Boat (Boat-AlvesBoat-223 e Boat-BeneditaBoat-223) e Weather (Weather-223).

Na Figura 7.6 é ilustrada a interface oferecida aos clientes navegadores com a informação da competição (id da competição, número de voltas, tempo máximo da prova (s), número de bóias, direção (°) e velocidade do vento (km/h) e número de embarcações), a listagem dos eventos, a representação das bóias, a informação de todos os pontos do trajeto realizado por cada veleiro (nome do veleiro e coordenadas de posição (lat. (°) e long. (°))) e a posição atual dos veleiros.

Tabela 7.4: Recurso REST da competição do tipo Seguimento – AlvesBoat.

Pedido	/client-agent/seg/boat/223/AlvesBoat/-8.66308/41.25151
Parâmetros	Identificação da competição: 223; Nome do veleiro: AlvesBoat; Latitude do veleiro: -8.66308°; Longitude do veleiro: 41.25151°;
Resposta	/AlvesBoat/-8.6630/41.25151/8/1
Parâmetros	Nome veleiro: AlvesBoat; Latitude do veleiro: -8.6630°; Longitude do veleiro: 41.25151°; Velocidade do vento: 8 km/h; Direção do vento: 1°;

Tabela 7.5: Recurso REST da competição do tipo Seguimento – BeneditaBoat.

Pedido	/client-agent/seg/boat/223/BeneditaBoat/-8.6622/41.2501
Parâmetros	Identificação da competição: 223; Nome do veleiro: BeneditaBoat; Latitude do veleiro: -8.6622°; Longitude do veleiro: 41.2501°;
Resposta	/BeneditaBoat/-8.6622/41.2501/8/1
Parâmetros	Nome veleiro: BeneditaBoat; Latitude do veleiro: -8.6622°; Longitude do veleiro: 41.2501°; Velocidade do vento: 8 km/h ; Direção do vento: 1°;

Tabela 7.6: Recurso REST da competição do tipo Seguimento – Weather.

Pedido	/client-agent/seg/weather/223/8/1
Parâmetros	Identificação da competição: 223; Nome do veleiro: BeneditaBoat; Velocidade do vento: 8 km/h; Direção do vento: 1°;
Resposta	/223/8/1
Parâmetros	Identificação da competição: 223; Velocidade do vento: 8 km/h; Direção do vento: 1°;

Nas Tabelas 7.4, 7.5 e 7.6 é possível visualizar um pedido e resposta REST relativo aos veleiros AlvesBoat e BeneditaBoat e ao Agente Weather.

7.2 Teste de Modelação

Para comprovar o modelo físico de navegação implementado na plataforma foram realizados testes com competições do tipo simulação com dois veleiros (AlvesBoat e BeneditaBoat). Nestes testes foram contemplados os seguintes modos de navegação:

- Bolina – mareação em que os veleiros navegam contra a direção do vento;
- Popa – mareação em que os veleiros navegam com o vento traseiro direto;
- Través – mareação perpendicular à direção do vento;
- Largo – mareação em que os veleiros navegam entre a Popa e Través;

Na Figura 7.7 são ilustrados os 4 modos de navegação indicados e onde se verifica que é impossível avançar contra o vento. É importante referir que em qualquer tipo de mareação deve de existir um compromisso entre a direção e a velocidade [46], como se pode comprovar com os resultados obtidos.

Nos testes de modelação foi também testada a influência que a área vélica e a massa do veleiro têm na direção e na velocidade do veleiro.

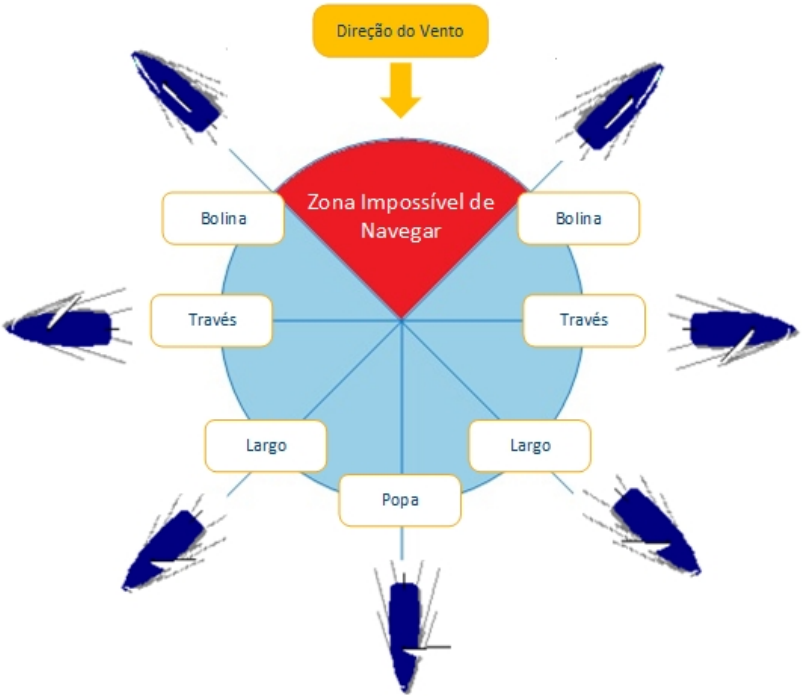

Figura 7.7: Agentes de uma competição do tipo simulação [8].

7.2.1 Bolina

No teste da navegação à Bolina, representado na Figura 7.8, foi colocado o veleiro AlvesBoat a realizar uma navegação com um ângulo de 45° em relação ao vento enquanto que o veleiro BeneditaBoat realizou uma trajetória mais arribada (ângulo de bolina de 60° em relação ao vento).

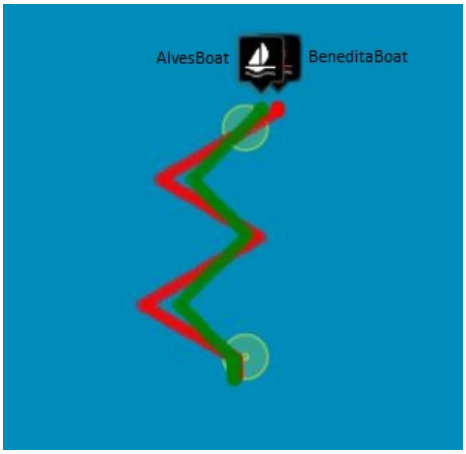

Figura 7.8: Teste da navegação à Bolina.

Na Tabela 7.7, que apresenta o tempo gasto e a velocidade média do percurso (da bóia 1 à bóia 2) em navegação à bolina, é possível de verificar que o veleiro AlvesBoat é mais rápido do que o veleiro BeneditaBoat apesar da sua velocidade média ser inferior. Esta situação é originada pelo o facto do percurso realizado ser inferior, o que comprova que deverá de existir sempre um compromisso entre a direção e a velocidade.

Tabela 7.7: Resultado do teste da navegação à Bolina.

Veleiro	Duração de Prova (s)	Velocidade Média (kn)
AlvesBoat	68	7
BeneditaBoat	110	12

7.2.2 Popa e Largo

Na Figura 7.9 é representado o teste da navegação à Popa, realizado pelo veleiro AlvesBoat com um ângulo de 180º em relação à direção do vento, e o teste de navegação ao Largo, realizado pelo veleiro BeneditaBoat com ângulos de 135º em relação à direção do vento.

Na Tabela 7.8 é possível verificar que os veleiros atingem maior velocidade nestes dois modos de navegação do que no modo de navegação à Bolina. Por outro lado, é importante de referir que a navegação à Popa, apesar do veleiro atingir maior velocidade e realizar o percurso mais rapidamente, é instável e perigosa

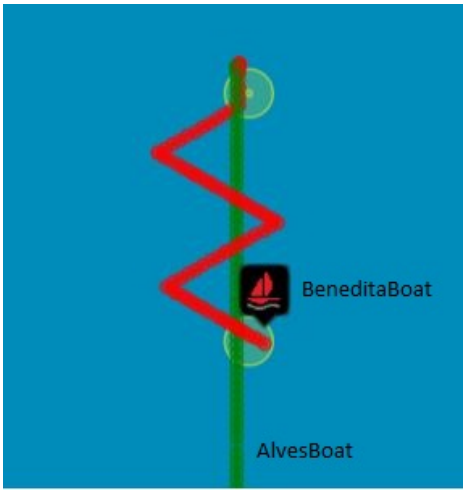

Figura 7.9: Teste da navegação à Popa e ao Largo.

Tabela 7.8: Resultado do teste da navegação à Popa e ao Largo.

Veleiro	Tempo de Prova (s)	Velocidade Média (kn)
AlvesBoat	32	16
BeneditaBoat	70	14

devido ao risco da cambadela[2] ou da orçadela[3] violenta.

A cambadela pode ser perigosa caso a parte superior do patilhão[4] esteja a obstruir a passagem da retranca[5] , o que originará a sua danificação. Em relação à orçadela violenta, esta poderá provocar o tombar do veleiro. Ambos os fenómenos não são contemplados na modelação [47].

7.2.3 Través

O teste da navegação a Través, representado na Figura 7.10, serviu fundamentalmente para validar a influência que o ângulo da vela tem na velocidade do veleiro. Ambos os veleiros (AlvesBoat e BeneditaBoat) realizam a sua navegação perpendicularmente à direção do vento mas com ângulos da vela diferentes.

Os resultados podem ser visualizados na Tabela 7.9, em que se verifica o quanto a vela não deve de ser demasiado cassada neste modo de navegação.

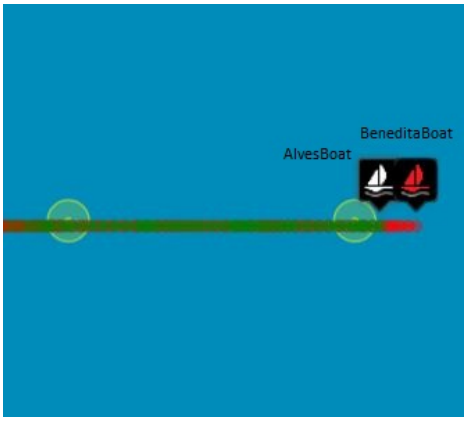

Figura 7.10: Teste da navegação a Través.

[2]Cambar – mudar a direção do veleiro, mudando as velas de lado.
[3]Orçar – aproximar a parte da frente de um veleiro da direção do vento
[4]Patilhão – estrutura em forma de barbatana, fixada na parte de baixo do casco do veleiro, que aumenta a estabilidade e a resistência ao abatimento.
[5]Retranca – perfil horizontal usado para prender e estender a esteira da vela grande

Tabela 7.9: Teste da navegação a Través.

Veleiro	Ângulo da Vela (°)	Tempo de Prova (s)	Velocidade Média (kn)
AlvesBoat	30	35	14
BeneditaBoat	45	32	16

7.2.4 Influência da Massa e da Área Vélica

A influência da massa e da área vélica na velocidade de um veleiro foi testada através da criação de diagramas polares gerados em função da modelação implementada.

Na Figura 7.11 encontra-se representado o diagrama polar da velocidade de dois veleiros com área vélica igual (1,9 m^2) e massa diferente (AlvesBoat com 10 kg e BeneditaBoat com 14 kg). O veleiro mais leve atinge velocidades superiores em relação ao veleiro mais pesado.

No diagrama polar da velocidade de dois veleiros com massa igual (10 kg) e com área vélica diferente (AlvesBoat com 1,4 m^2 e BeneditaBoat com 1,9 m^2), que se encontra representado na Figura 7.12, conclui-se que quanto maior for a área vélica maior será a velocidade do veleiro.

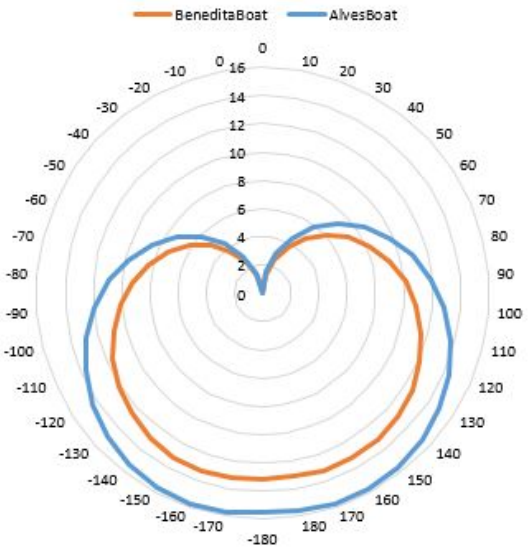

Figura 7.11: Diagrama polar da velocidade de dois veleiros em função do rumo – Teste massa.

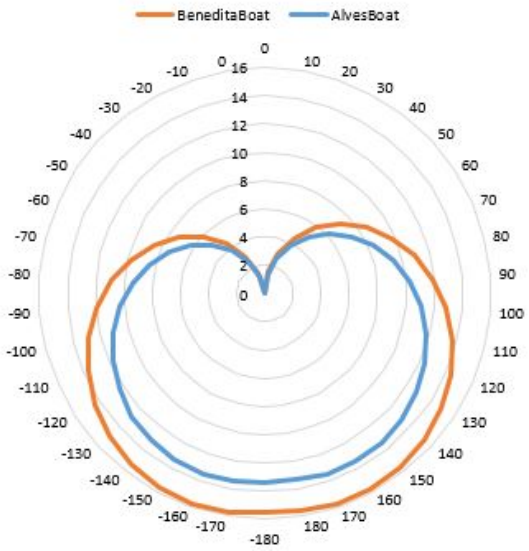

Figura 7.12: Diagrama polar da velocidade de dois veleiros em função do rumo – Teste área vélica.

7.3 Conclusão

Neste capítulo foram apresentados os testes que ilustram o correto e expectável funcionamento da plataforma: (i) teste funcional e (ii) teste da modelação física.

No próximo capítulo é apresentado o balanço do projeto, identificam-se as suas limitações e são sugeridos desenvolvimentos futuros.

Capítulo 8

Conclusões

Neste capítulo é apresentado o balanço do trabalho desenvolvido e os possíveis desenvolvimentos futuros.

8.1 Balanço

No âmbito desta dissertação concebeu-se, desenvolveu-se e testou-se uma plataforma que permite validar algoritmos de navegação, acompanhar competições reais de veleiros autónomos e criar/acompanhar competições simuladas de veleiros autónomos. Aos clientes é oferecida uma API do tipo serviço *Web* RESTful e uma aplicação *Web* 2D para acompanhamento e gestão das competições. A plataforma tem a capacidade de modelação física dos veleiros devido ao SMA implementado. Apesar de ser uma modelação baseada num software já existente TrackSail, foram adicionados dois novos parâmetros – massa e área vélica do veleiro.

O balanço do desenvolvimento do projeto é bastante positivo, tendo em conta, que todos os objetivos e requisitos propostos inicialmente foram cumpridos. De acrescentar, que a plataforma se encontra preparada para ser utilizada para as equipas melhorarem os seus algoritmos de navegação e seguirem as suas participações em provas reais e simuladas.

Todo o desenvolvimento foi suportado por tecnologias *open source* e pensado de forma a tornar a plataforma escalável e modular, *i.e.*, permite a sua evolução.

8.2 Desenvolvimentos Futuros

A passagem da aplicação *Web* 2D para 3D seria um grande avanço pois permitia que o cliente pudesse aumentar o realismo da competição real ou simulada e ter

uma melhor perceção das animações físicas dos veleiros em tempo real.

A modelação física deve ser aperfeiçoada. Deverá, por exemplo, ser contemplado o comprimento da linha de água do veleiro bem como a influência da ondulação, das marés e da corrente.

Um teste que poderá ser realizado no futuro é a comparação de uma navegação real com uma navegação simulada de forma a validar a discrepância e o que deve de ser melhorado na modelação.

Bibliografia

[1] R. Stelzer and K. Jafarmadar, *History and Recent Developments in Robotic Sailing*, pp. 3–23. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. [citado na p. v, 6]

[2] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995. [citado na p. v, ix, 8, 9, 11]

[3] H. S. Nwana, "Software agents: An overview," *Knowledge Engineering Review*, vol. 11, pp. 205–244, 1996. [citado na p. v, 8]

[4] J. Boedecker and M. Asada, "Simspark – concepts and application in the 3d soccer simulation league," in *Workshop Proceedings*, SIMPAR 2008, (Venice,Italy), pp. 174–181, 2008. [citado na p. v, 12, 13, 14]

[5] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship: Competition software manual," *CoRR*, vol. abs/1304.1672, 2013. [citado na p. v, 14, 15]

[6] T. Team, M. P. Wellman, P. R. Wurman, K. O'Malley, R. Bangera, S.-d. Lin, D. Reeves, and W. E. Walsh, "Designing the market game for a trading agent competition," *IEEE Internet Computing*, vol. 5, pp. 43–51, Mar. 2001. [citado na p. v, 15, 16, 17]

[7] J. Hu, D. Reeves, and H. shan Wong, "Agent service for online auctions," in *in: Proceedings of the AAAI-99 Workshop on AI for Electronic Commerce*, AAAI Press, 1999. [citado na p. v, 15, 16]

[8] L. Gomes, "eventos 3 – desenvolvimento de controlador difuso para navegação autónoma de veleiro," Master's thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Lisboa, Portugal, 09 2015. [citado na p. vi, 55]

[9] K. Kravari and N. Bassiliades, "A survey of agent platforms," *Journal of Artificial Societies and Social Simulation*, vol. 18, no. 1, p. 11, 2015. [citado na p. ix, 19, 20, 21, 22]

[10] M. Fischer, K. Képes, and A. Wassiljew, "Vergleich von frameworks zur implementierung von rest-basierten anwendungen," Master's thesis, Institute of Architecture of Application Systems, Stuttgart, Deutschland, 06 2013. [citado na p. ix, 22, 23, 24]

[11] L. M. Hestman, "The potential of utilizing bim models with the webgl technology for building virtual environments-a web-based prototype within the virtual hospital field," Master's thesis, Department of Computer and Information Science, NTNU, Trondheim, Norway, 7 2015. [citado na p. ix, 24, 25]

[12] N. Jain, "Review of different responsive css front-end frameworks," *Journal of Global Research in Computer Science*, vol. 5, no. 11, pp. 5–10, 2015. [citado na p. ix, 26, 27]

[13] microtransat.org, "The microtransat challenge." `http://www.microtransat.org/`, 2016. Online; Acedido em Junho 2016. [citado na p. 5]

[14] A. A. for Innovative Computer Science, "World robotic sailing championship / international robotic sailing conference (wrsc/irsc)." `http://www.roboticsailing.org/`, 2016. Online; Acedido em Junho 2016. [citado na p. 5]

[15] W. . O. Committee, "World robotic sailing championship." `http://web.fe.up.pt/~jca/wrsc2016.com/competition.html`, 2016. Online; Acedido em Junho 2016. [citado na p. 5]

[16] sailbot.org, "Sailbot." `http://sailbot.org/`, 2016. Online; Acedido em Junho 2016. [citado na p. 6]

[17] L. Reis, *Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico*. PhD thesis, FEUP - Faculdade de Engenharia da Universidade do Porto, Porto, Portugal, 7 2003. [citado na p. 7]

[18] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Engineering Review*, vol. 10, pp. 115–152, 1995. [citado na p. 7]

[19] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. [citado na p. 9]

[20] T. R. Federation, "About robocup: What is robocup?." `http://www.robocup.org/about-robocup/`, 05 2015. Online; Acedido em Maio 2016. [citado na p. 12]

[21] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the First International Conference on Autonomous Agents*, AGENTS '97, (New York, NY, USA), pp. 340–347, ACM, 1997. [citado na p. 12]

[22] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawai, and H. Matsubara, *RoboCup: A challenge problem for AI and robotics*, pp. 1–19. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. [citado na p. 12]

[23] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié, C. Guionneau, and R. Coulom, "TORCS, the open racing car simulator." `http://www.cse.chalmers.se/~chrdimi/papers/torcs.pdf`, 03 2015. Online; Acedido em Maio 2016. [citado na p. 14]

[24] S. C. Corneliussen and M. Westergaard, "Combining offline and online learning in developing an adaptive controller for a simulated car racing environment," Master's thesis, Department of Computer and Information Science, NTNU, Trondheim, Norway, 6 2011. [citado na p. 14]

[25] F. Bellifemine, A. Poggi, and G. Rimassa, "Jade: A fipa2000 compliant agent development environment," in *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, (New York, NY, USA), pp. 216–217, ACM, 2001. [citado na p. 19]

[26] S. Poslad and P. Charlton, "Standardizing agent interoperability: The fipa approach," in *Selected Tutorial Papers from the 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School on Multi-Agent Systems and Applications*, EASSS '01, (London, UK, UK), pp. 98–117, Springer-Verlag, 2001. [citado na p. 19]

[27] J. Site, "Java agent development framework is an open source platform for peer-to-peer agent based applications." `http://jade.tilab.com/`, 2016. Online; Acedido em Janeiro 2016. [citado na p. 20]

[28] A. Grignard, P. Taillandier, B. Gaudou, D. A. Vo, N. Q. Huynh, and A. Drogoul, *GAMA 1.6: Advancing the Art of Complex Agent-Based Modeling and Simulation*, pp. 117–131. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. [citado na p. 20]

[29] A. Drogoul, E. Amouroux, P. Caillou, B. Gaudou, A. Grignard, N. Marilleau, P. Taillandier, M. Vavasseur, D.-A. Vo, and J.-D. Zucker, *GAMA: A Spatially Explicit, Multi-level, Agent-Based Modeling and Simulation Platform*, pp. 271–274. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. [citado na p. 20]

[30] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 3rd ed., 1999. [citado na p. 20]

[31] U. Wilensky, "Netlogo." `https://ccl.northwestern.edu/netlogo/`, 2016. Online; Acedido em Janeiro 2016. [citado na p. 21]

[32] S. Tisue, "Netlogo: Design and implementation of a multi-agent modeling environment," in *In Proceedings of Agent 2004*, 2004. [citado na p. 21]

[33] MASON, "Mason features." `https://cs.gmu.edu/~eclab/projects/mason/`, 2015. Online; Acedido em Dezembro 2015. [citado na p. 21]

[34] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "Mason: A multiagent simulation environment," *Simulation*, vol. 81, pp. 517–527, July 2005. [citado na p. 21]

[35] C. D. of Java Technology Specifications, "Jsr 311: Jax-rs: The javatm api for restful web services." `https://jcp.org/en/jsr/detail?id=311`, 2011. Online; Acedido em Dezembro 2015. [citado na p. 22]

[36] O. Corporation, "About." `https://jersey.java.net/`, 2016. Online; Acedido em Fevereiro 2016. [citado na p. 22]

[37] M. Hadley, S. Pericas-Geertsen, and P. Sandoz, "Exploring hypermedia support in jersey," in *Proceedings of the First International Workshop on RESTful Design*, WS-REST '10, (New York, NY, USA), pp. 10–14, ACM, 2010. [citado na p. 22]

[38] RESTEasy, "Resteasy." `http://resteasy.jboss.org/`, 2016. Online; Acedido em Fevereiro 2016. [citado na p. 23]

[39] O. Corporation, "Netbeans ide features." `https://netbeans.org/features/index.html`, 2016. Online; Acedido em Fevereiro 2016. [citado na p. 29]

[40] T. K. e Stefan Brockmann, "Tracksail." `http://tracksail.sourceforge.net/`, 2002. Online; Acedido em Junho 2016. [citado na p. 37]

[41] N. Foundation, "About node.js." `https://nodejs.org/en/`, 2016. Online; Acedido em Março 2016. [citado na p. 38]

[42] SocketIO, "Socketio." `http://socket.io/`, 2016. Online; Acedido em Março 2016. [citado na p. 38]

[43] T. P. G. D. Group, "*About PostgreSQL*." `https://www.postgresql.org/about/`, 2016. Online; Acedido em Julho 2016. [citado na p. 39]

[44] Rethinkdb, "What is rethinkdb?." `https://rethinkdb.com/faq/`, 2016. Online; Acedido em Julho 2016. [citado na p. 39]

[45] V. Agafonkin, "Leafletjs." `http://leafletjs.com/`, 2016. Online; Acedido em Julho 2016. [citado na p. 44]

[46] N. B. e Sá, "Física elementar da navegação à vela," *Gazeta de Física - Sociedade Portuguesa de Física*, vol. 30, no. 2, pp. 5–14, 2007. [citado na p. 54]

[47] T. Rosa, "Iniciação ao treino da vela," tech. rep., Universidade de Trás-os-Montes e Alto Douro, Vila Real, Portugal, 04 2003. [citado na p. 57]

Anexo A

Diagramas de Casos de Uso - Funcionalidades

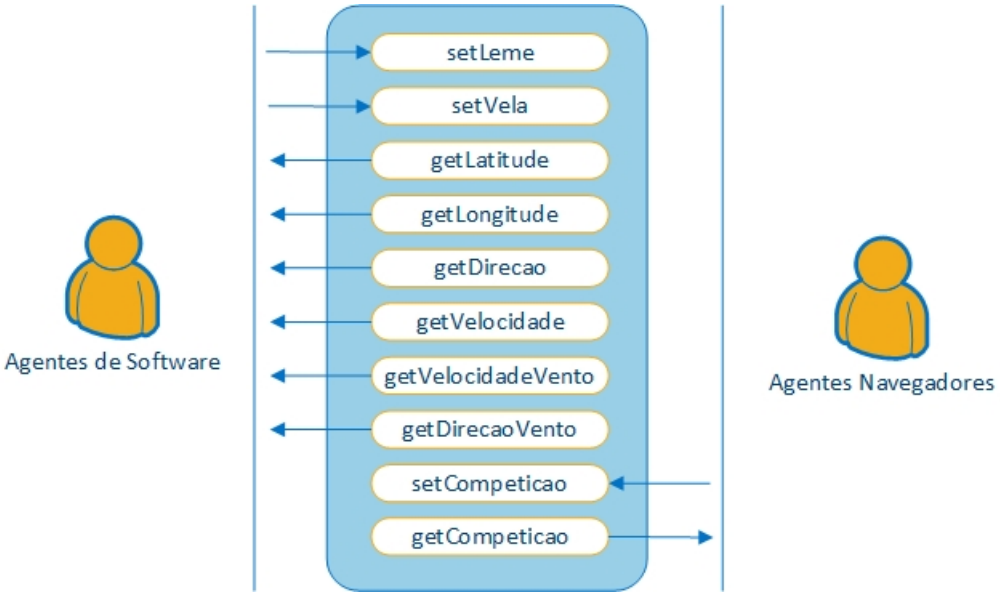

Figura A.1: Diagrama de caso de uso: Funcionalidade de Teste.

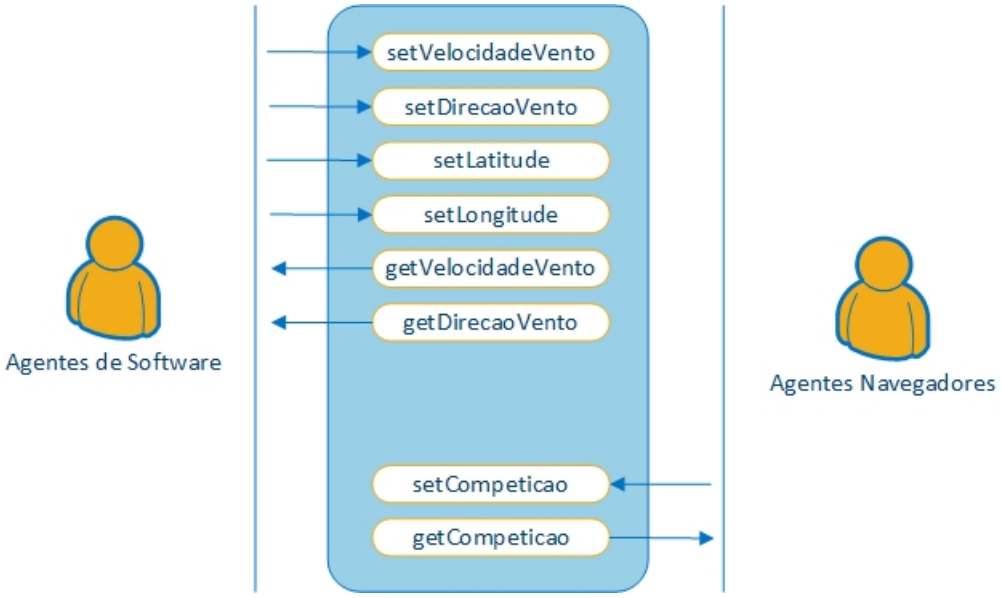

Figura A.2: Diagrama de caso de uso: Funcionalidade de Seguimento.

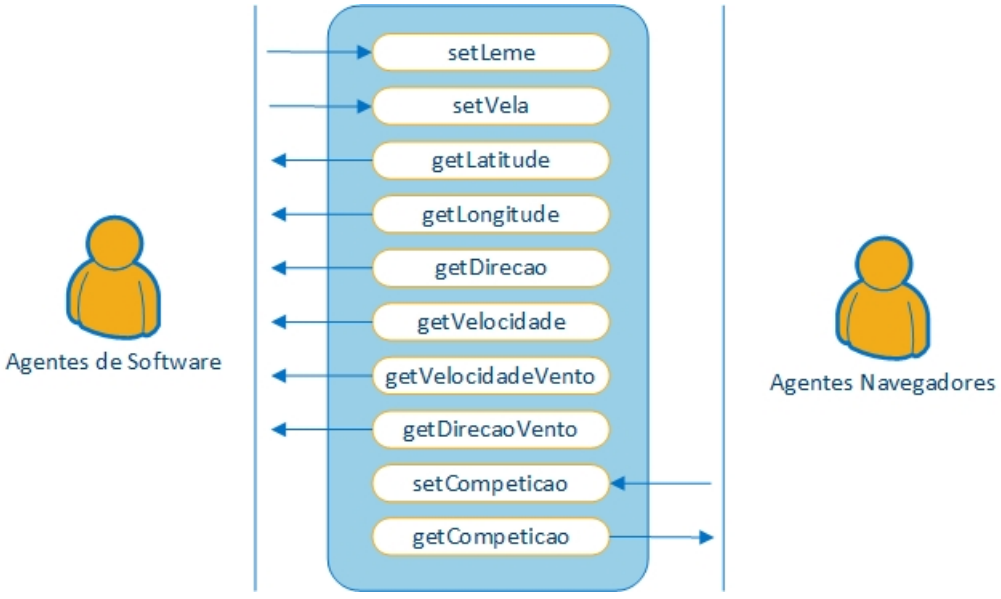

Figura A.3: Diagrama de caso de uso: Funcionalidade de Simulação.

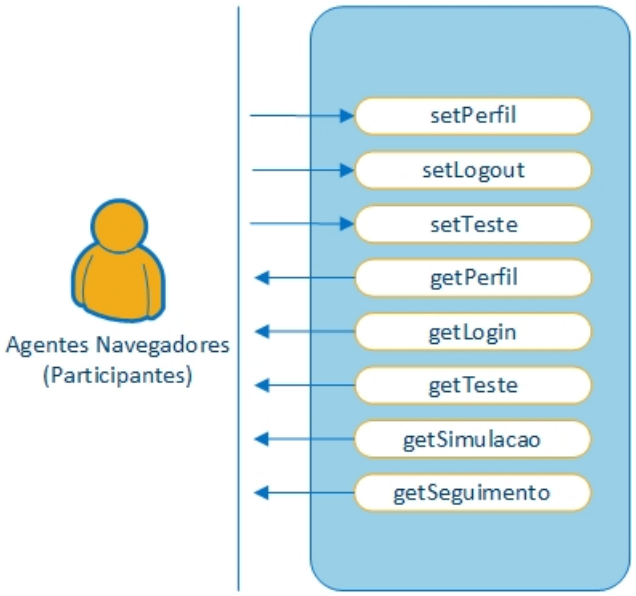

Figura A.4: Diagrama de caso de uso: Funcionalidade de Utilizador.

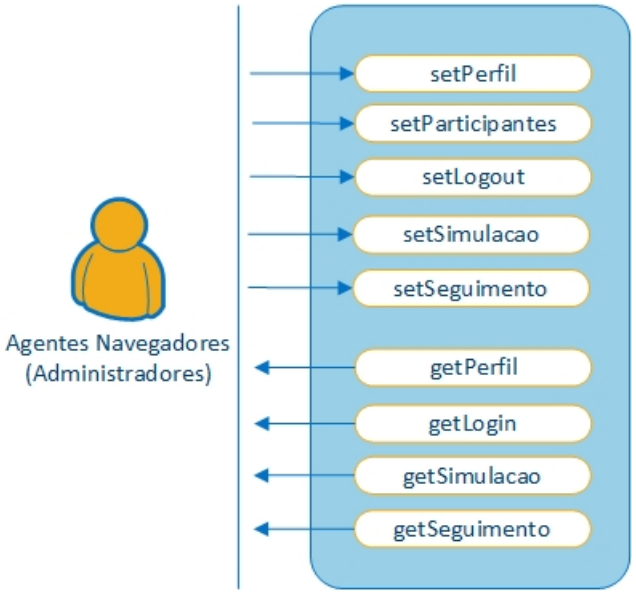

Figura A.5: Diagrama de caso de uso: Funcionalidade de Administrativo.

Anexo B

Modelo EER da Base de Dados

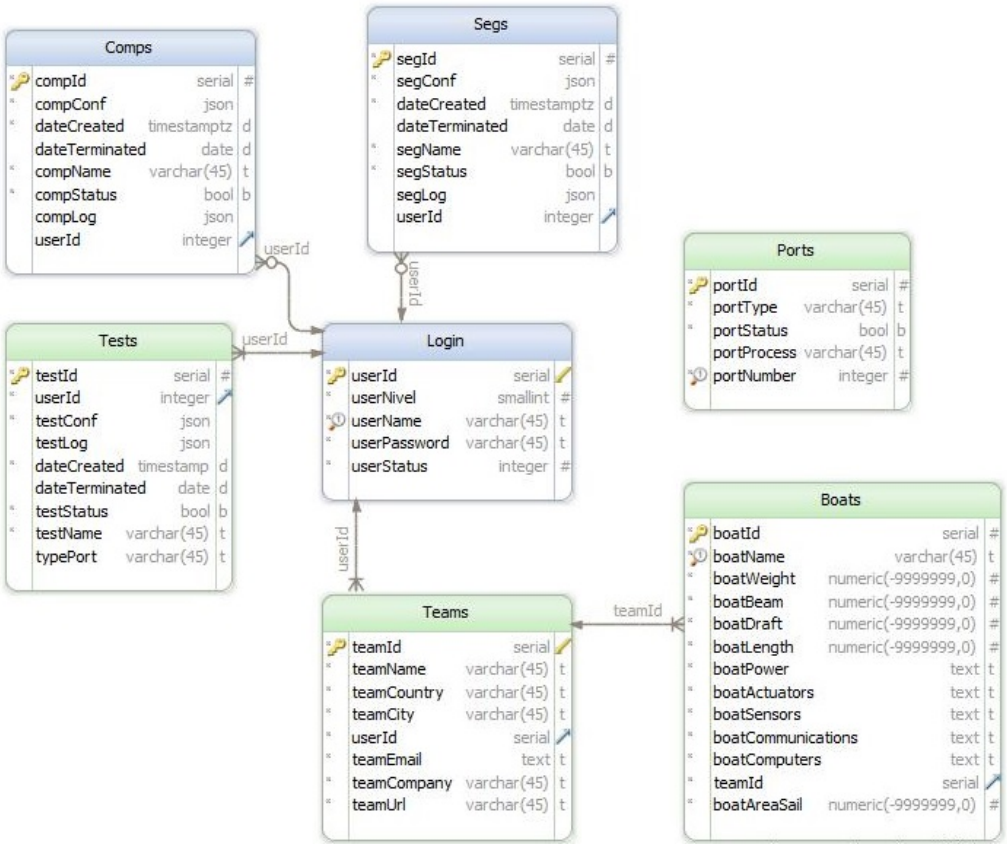

Figura B.1: Base de dados - Modelo EER.

Anexo C

Diagramas de Sequência - Recursos REST

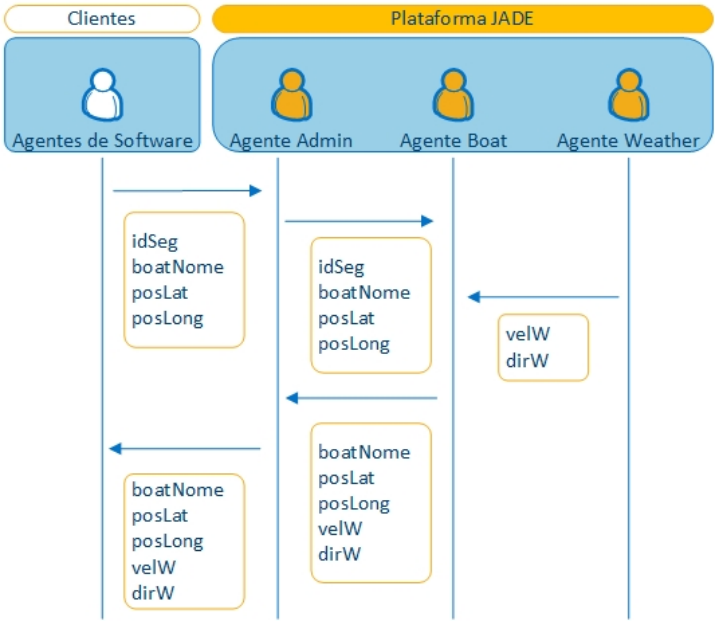

Figura C.1: Diagrama de sequência: Recurso Seg 1

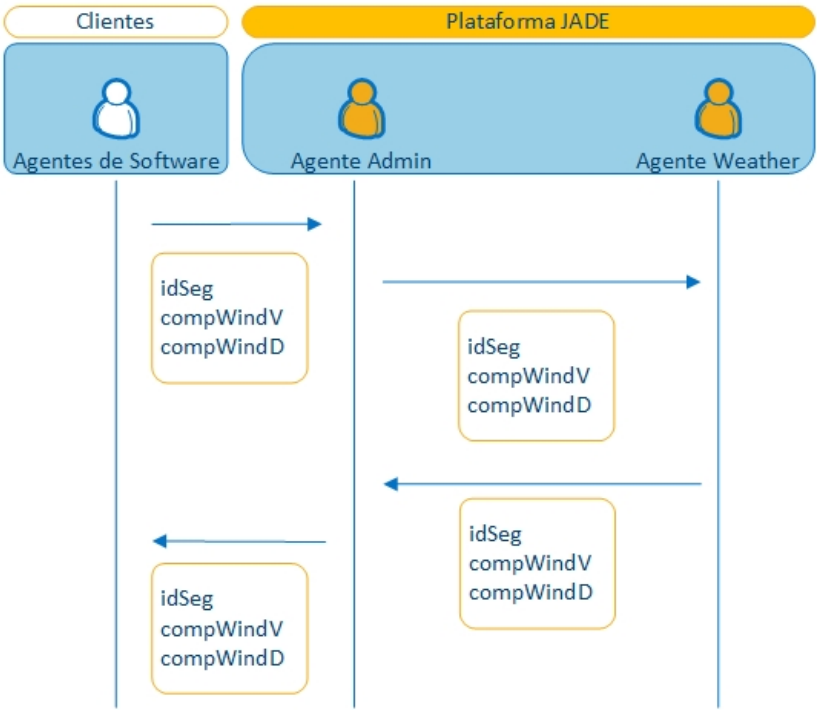

Figura C.2: Diagrama de sequência: Recurso Seg 2

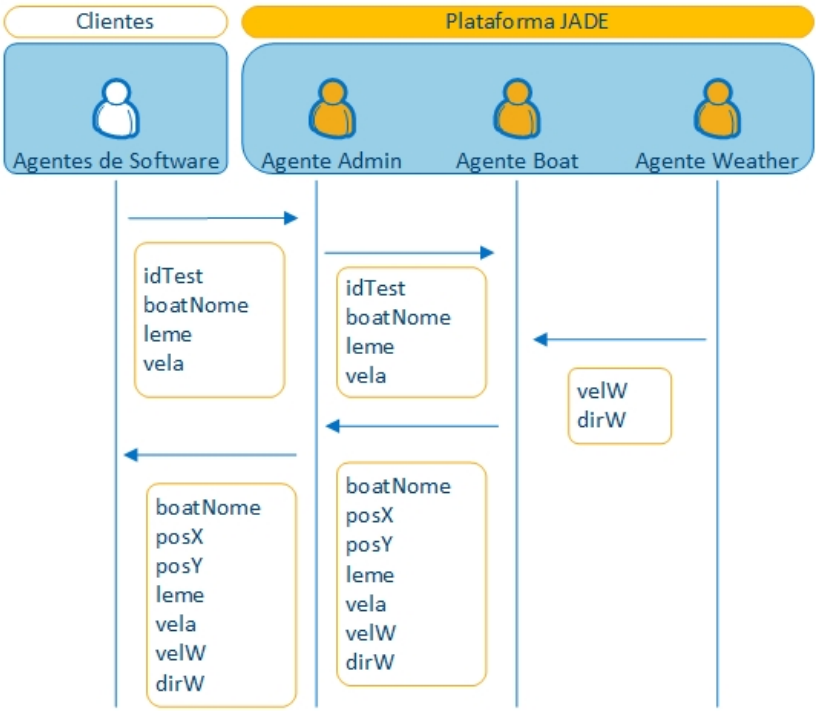

Figura C.3: Diagrama de sequência: Recurso Test 1

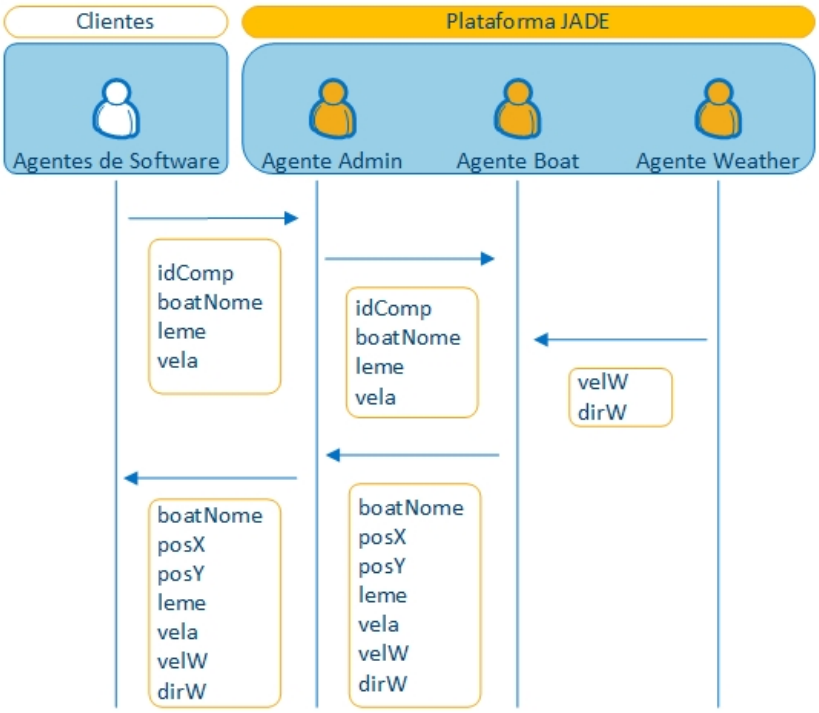

Figura C.4: Diagrama de sequência: Recurso Sim 1

Anexo D

Fluxogramas - Modelação dos Agentes

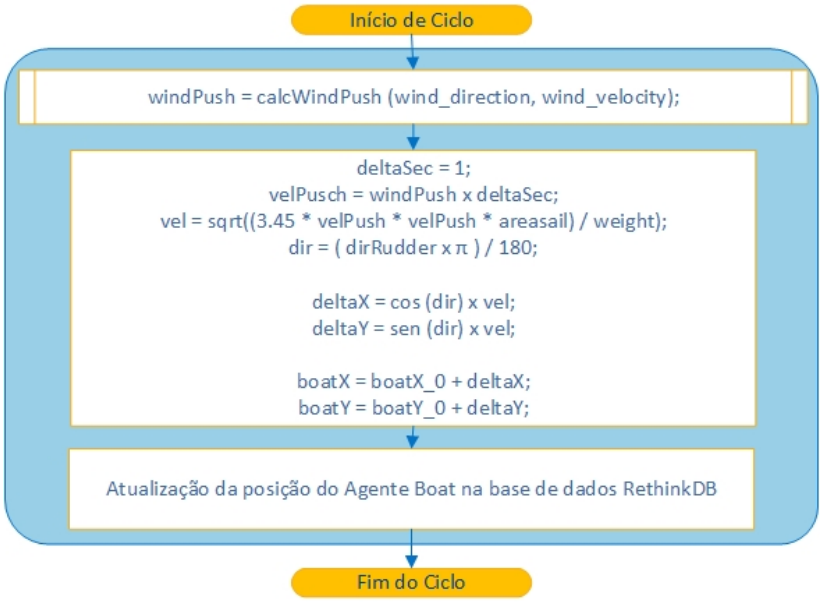

Figura D.1: Fluxograma - Modelação do agente Boat - Main()

Tabela D.1: Descrição das variáveis representadas no fluxograma da Figura D.1

windPush	Força exercida pelo vento
wind_direction	Direção do vento
wind_velocity	Velocidade do vento
deltaSec	Variação de tempo
areasail	Área do veleiro
weight	Massa do veleiro
vel	Velocidade do veleiro
dir	Direção do veleiro
dirRudder	Direção do leme
deltaY	Variação em Y do veleiro
deltaX	Variação em X do veleiro
boatX	Nova posição em X do veleiro
boatX_0	Inicial posição em X do veleiro
boatY	Nova posição em Y do veleiro
boatY_0	Inicial posição em Y do veleiro

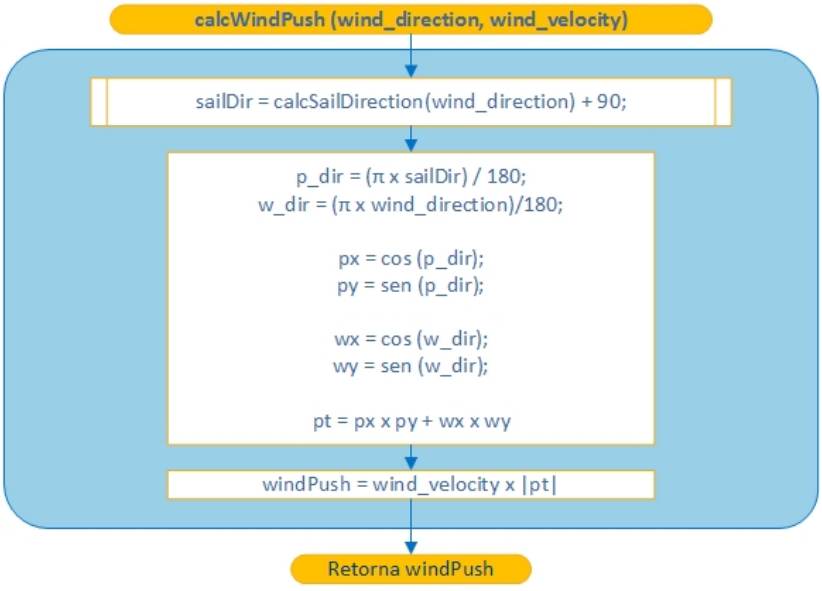

Figura D.2: Fluxograma - Modelação do agente Boat - windPush()

Tabela D.2: Descrição das variáveis representadas no fluxograma da Figura D.2

sailDir	Direção da vela
wind_direction	Direção do vento
p_dir	Direção da vela em radianos
w_dir	Direção do vento em radianos
px e py	Vetor da vela
wx e wy	Vetor do vento
pt	Produto escalar (vela e vento)
windPush	Força exercida pelo vento
wind_velocity	Velocidade do vento

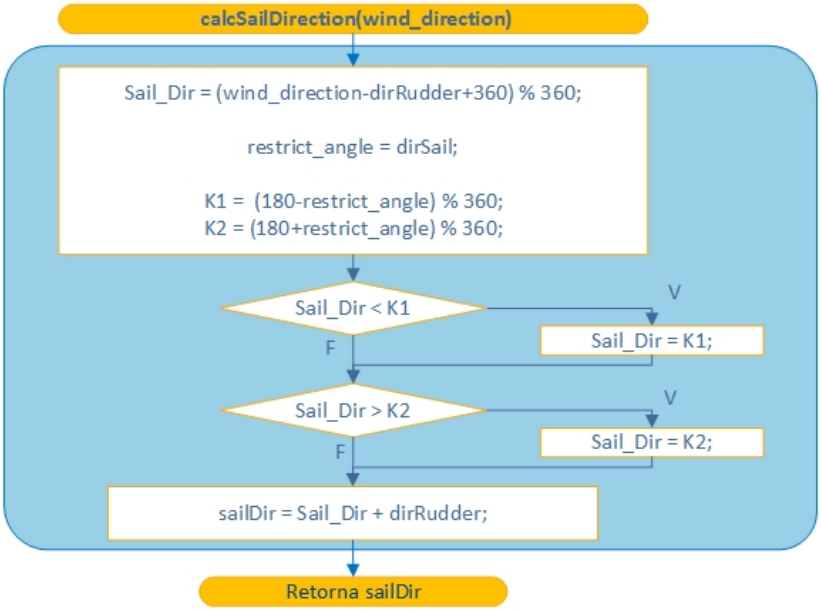

Figura D.3: Fluxograma - Modelação do agente Boat - sailDirection()

Tabela D.3: Descrição das variáveis representadas no fluxograma da Figura D.3

sailDir	Direção da vela
wind_direction	Direção do vento
dirRudder	Direção do leme
K1 e K2	Mín. e máx. da direção da vela

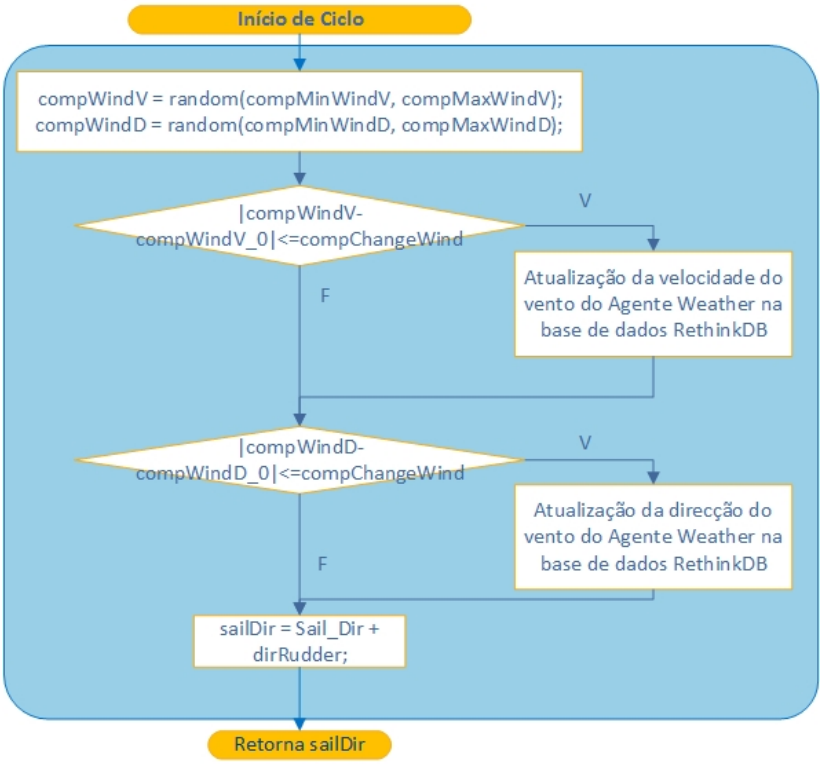

Figura D.4: Fluxograma - Modelação do agente Weather

Tabela D.4: Descrição das variáveis representadas no fluxograma da Figura D.4

compWindV	Velocidade do vento
compWindD	Direção do vento
compMinWindV	Velocidade mín. do vento
compMaxWindV	Velocidade máx. do vento
compMinWindD	Direção mín. do vento
compMaxWindD	Direção máx. do vento
compChangeWind	Variação máx. do vento

Anexo E

Fluxogramas - Regras de Navegação

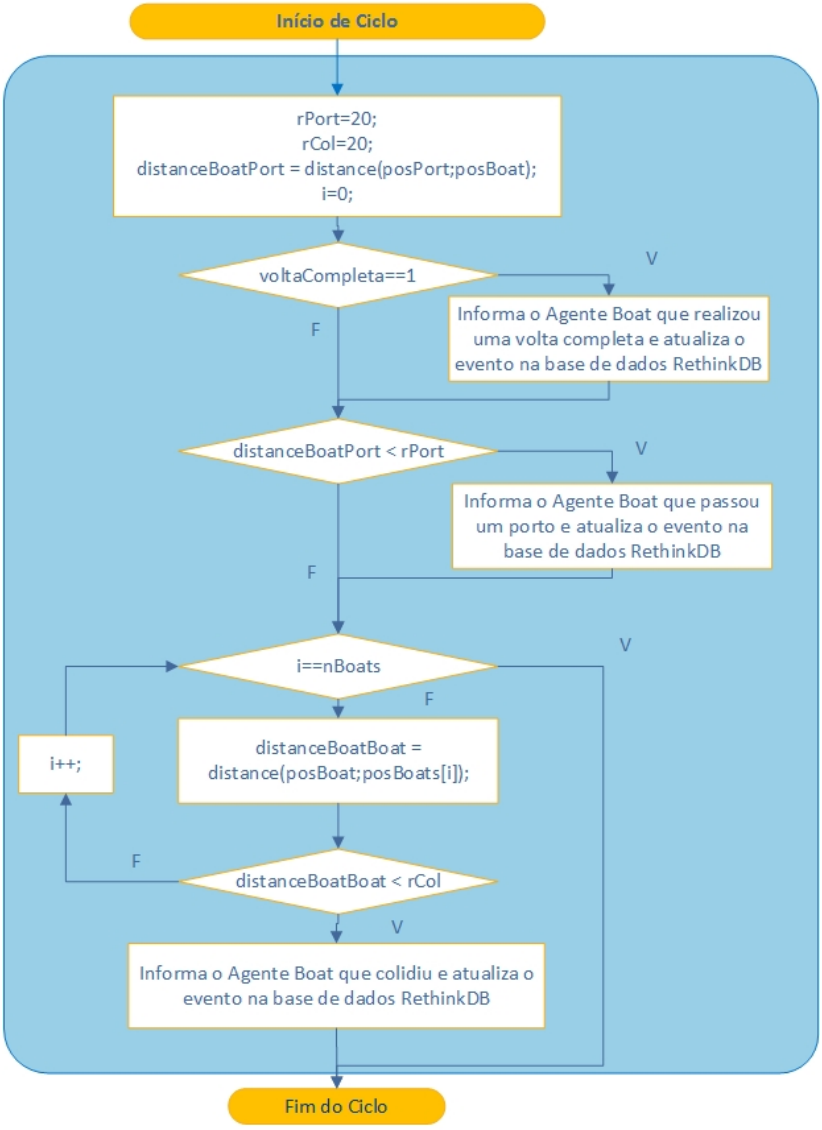

Figura E.1: Fluxograma - Regras de navegação

Tabela E.1: Descrição das variáveis representadas no fluxograma da Figura E.1

rPort	Raio mín. para passagem no porto
rCol	Raio mín. para colisão
posPort	Posição do próximo porto
posBoat	Posição do veleiro
distanceBoatPort	Distância entre veleiro e porto
voltaCompleta	Flag de deteção de volta
distanceBoatBoat	Distância entre veleiros

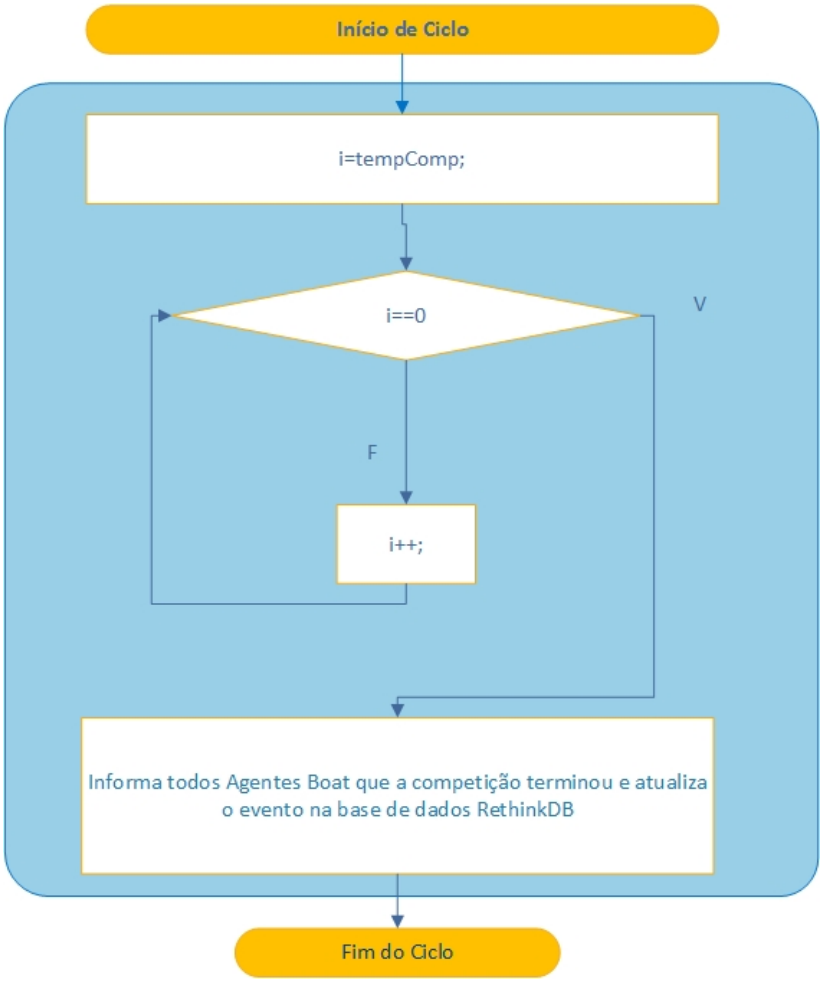

Figura E.2: Fluxograma - Regras de navegação - Fim de prova

Anexo F

Clientes navegadores registados para testes

Tabela F.1: Lista de dados de clientes navegadores registados para testes

Nome de Utilizador	Benedita	Alves
Equipa		
Nome da Equipa	BeneditaTeam	AlvesTeam
País	Portugal	Portugal
Cidade	Porto	Porto
Instituição	ISEP	ISEP
URL	ave.dee.isep.ipp.pt/ mbm/	ave.dee.isep.ipp.pt/ 1100397/
Email	mbm@isep.ipp.pt	1100397@isep.ipp.pt
Veleiro Autónomo		
Nome do Veleiro	BeneditaBoat	AlvesBoat
Massa (kg)	10	10
Boca (m)	0.4	0.3
Calado (m)	0.41	1
Área Velica(m^2)	1.4	1.4
Comprimento(m)	1.26	1.4
Fontes de Alimentação	2 Bat. - 3V7 6Ah LiPo	2 Bat. - 3V7 6Ah LiPo
Atuadores	Leme e Vela	Leme e Vela
Sensores	Anemómetro e Cata-vento	Anemómetro e Cata-vento
Comunicações	3G	4G
Computadores	ARM 64	Arduino

Anexo G

Ficheiro de Criação de uma Competição de Teste

Ficheiro .txt

```
{
"compName":"Porto",
"compLaps":2,
"compTimeMax":300,
"compChangeWind":5,
"compMinWindD":-10,
"compMaxWindD":10,
"compMinWindV":5,
"compMaxWindV":15,
"compPorts": [
        {"compPort_X":20, "compPort_Y":20},
        {"compPort_X":420, "compPort_Y":20},
        {"compPort_X":420, "compPort_Y":420},
        {"compPort_X":20, "compPort_Y":420}
        ]
}
```

Figura G.1: Ficheiro de criação de uma competição teste

Anexo H

Ficheiro de Criação de uma Competição de Simulação

Ficheiro .txt

```
{
"compName":"Porto",

"compLaps":2,

"compTimeMax":300,

"compChangeWind":5,

"compMinWindD":-10,

"compMaxWindD":10,

"compMinWindV":5,

"compMaxWindV":15,

"compBoats": [ "AlvesTeam", "BeneditaTeam" ],

"compPorts": [

        {"compPort_X":20, "compPort_Y":20},

        {"compPort_X":420, "compPort_Y":20},

        {"compPort_X":420, "compPort_Y":420},

        {"compPort_X":20, "compPort_Y":420}
        ]
}
```

Figura H.1: Ficheiro de criação de uma competição simulação

Anexo I

Ficheiro de Criação de uma Competição de Seguimento

Ficheiro .txt

```
{
"compName":"Porto",

"compLaps":2,

"compTimeMax":300,

"compChangeWind":5,

"compMinWindD":-10,

"compMaxWindD":10,

"compMinWindV":5,

"compMaxWindV":15,

"compBoats": [ "AlvesTeam", "BeneditaTeam" ],

"compPorts": [

{"compPort_X":-8.66051, "compPort_Y":41.250915},

{"compPort_X":-8.664548, "compPort_Y":41.256508 },

{"compPort_X":-8.664228, "compPort_Y":41.253827},

{"compPort_X":-8.660915, "compPort_Y":41.249126}

 ]
}
```

Figura I.1: Ficheiro de criação de uma competição seguimento