Instituto Superior de Engenharia do Porto
Departamento de Engenharia Electrotécnica
Rua Dr. António Bernardino de Almeida 431, 4200-072 Porto

Transacção de Componentes Multimédia Suportada por Agentes

Tese/Dissertação do Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações

Bruno Miguel Delindro Veloso

Orientação: Professora Doutora Maria Benedita Campos Neves Malheiro

Ano Lectivo: 2011/2012

Resumo

O objectivo do projecto descrito nesta dissertação é o desenvolvimento da interface entre as empresas e a plataforma *Business-to-Business* (B2B) de negociação automática de anúncios em construção. A plataforma, no seu todo, deve garantir que os intervalos da programação são preenchidos com um alinhamento de anúncios compatível com os interesses expressos e o perfil construído dos espectadores.

A plataforma funciona como um mercado electrónico de negociação automática destinado a agências de publicidade (empresas produtoras) e empresas provedoras de conteúdos e serviços multimédia aos consumidores finais (empresas distribuidoras). As empresas, uma vez registadas na plataforma, passam a ser representadas por agentes que negoceiam automaticamente os itens submetidos com o comportamento especificado. Do ponto de vista da arquitectura, a plataforma consiste num sistema multiagente organizado em três camadas compostas por: (*i*) agentes de interface com as empresas; (*ii*) agentes de modelação das empresas; e (*iii*) agentes delegados, de duração efémera, exclusivamente criados para participar em negociações específicas de conteúdos multimédia.

Cada empresa representada na plataforma possui, para além de um número indeterminado de delegados envolvidos em negociações específicas, dois agentes: (*i*) o agente de interface com a empresa, que expõe um conjunto de operações de interface ao exterior através de um serviço *Web*, localizado na primeira camada; e (*ii*) o agente que modela a empresa na plataforma, que expõe através de um serviço *Web* um conjunto de operações aos agentes das restantes camadas da plataforma, residente na camada intermédia.

Este projecto focou-se no desenvolvimento da camada superior de interface da plataforma com as empresas e no enriquecimento da camada intermédia.

A realização da camada superior incluiu a especificação da parte da ontologia da plataforma que dá suporte às operações de interface com o exterior, à sua exposição como serviços *Web* e à criação e controlo dos agentes de interface. Esta camada superior deve permitir às empresas carregar e descarregar toda informação relevante de e para a plataforma, através de uma interface gráfica

ou de forma automática, e apresentar de forma gráfica e intuitiva os resultados alcançados, nomeadamente, através da apresentação da evolução das transacções.

Em relação à camada intermédia, adicionou-se à ontologia da plataforma a representação do conhecimento de suporte às operações de interface com a camada superior, adoptaram-se taxonomias de classificação de espectadores, anúncios e programas, desenvolveu-se um algoritmo de emparelhamento entre os espectadores, programas e anúncios disponíveis e, por fim, procedeu-se ao armazenamento persistente dos resultados das negociações.

Do ponto de vista da plataforma, testou-se o seu funcionamento numa única plataforma física e assegurou-se a segurança e privacidade da comunicação entre empresa e plataforma e entre agentes que representam uma mesma empresa.

Abstract

The objective of the project described in this dissertation is the development of the interface between companies and the platform (B2B) of automatic trading of ads under construction. The platform as a whole, must ensure that the gaps are filled with programming Ad alignment with the interests expressed and the profile created of viewers.

The platform acts as an electronic marketplace for Automatic trading for the advertising agencies (companies producers) and companies providing content and servicesmultimedia to end consumers (distribution companies). The Companies, once registered on the platform become represented by agents who negotiate automatically the items submitted with the specified behavior.

This platform is arranged in a multiagent system three layers consisting of: (*i*) to interface with agents companies; (*ii*) agents modeling of enterprises; (*iii*) delegated agents of short-lived, created solely to participate in specific negotiations of multimedia content.

Each company has represented the platform, in addition to a indeterminate number of delegates involved in specific negotiations, two agents: (*i*) the agent interface of the company, which exposes a set of interface operations to outside through an Web Service, located in first layer, and (*ii*) the agent that shapes the company's platform, which exposes a Web Service set of operations to agents of the other layers of the platform, residing in the intermediate layer.

This design has focused on development of the topsheet platform interface with business and the enrichment of intermediate layer.

The performance of the topsheet includes a specification of the ontology platform that supports the operations interface with the outside world, to its exposure as Web Services and the creation and control of interface agents. This top layer should allow companies to load and unload all the relevant information to and from the platform, through a graphical interface or automatically, and presenting

intuitive graphical achievements, in particular, through the presentation of the evolution of the transactions.

In relation to the intermediate layer, added to the ontology platform for knowledge representation to support operations interfacing with the topsheet, measures have been taxonomies classification of viewers, ads and programs, developed a pairing algorithm between the audience, programs and advertisements available and, finally, proceeded to the persistent storage of the results of negotiations.

From the viewpoint of the platform, we tested its operation a single physical platform and ensured the safety and privacy of communication between the company and between platform and agents representing the same company.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Lista de Equações

Glossário

Abreviatura	Descrição	Página
ACL	*Agent Communication Language*	6
AJAX	*Asynchronous JavaScript and* XML	18
API	*Application Programming Interface*	12
BPEL	*Business Process Execution Language*	13
BPML	*Business Process Modeling Language*	14
BPR	*Business Process Reengineering*	15
BPSS	*Business Process Specification Schema*	14
B2B	*Business to Business*	iii
CDA	*Continuous Double Auction*	5
CFP	*Call For Proposals*	17
CNIP	*Contract Net Interaction Protocol*	17
CPA	*Collaboration Protocol Agreement*	14
CPI	*Continuous Process Improvement*	15
CPP	*Collaboration Protocol Profile*	14
DAML-S	DARPA *Agent Markup Language for Services*	15
DS	*Description Scheme*	16
DTD	*Document Type Definition*	12
ebMS	ebXML *Messaging Service*	14
ebXML	*electronic business* XML	13
ebXML-RS	ebXML *Registry Service*	14
ebXML-RIM	ebXML *Registry Information Model*	14
EER	*Enhanced Entity Relationship*	27
FPSB	*First-price Sealed-bid*	5
FICNIP	*Fixed Iterated Contract Net Interaction Protocol*	17
FIPA	*Foundation for Intelligent Physical Agents*	6
GUI	*Graphic User Interface*	3
HTML	*HyperText Markup Language*	18
HTTP	*HyperText Transfer Protocol*	40
ICNIP	*Iterated Contract Net Interaction Protocol*	17
IDE	*Integrated Development Environment*	25
ISEP	Instituto Superior de Engenharia do Porto	xix

Abreviatura	Descrição	Página
JAAS	*Java Authentication and Authorization Service*	17
JADE	*Java Agent DEvelopment Framework*	6
JAXB	*Java Architecture for* XML *Binding*	12
JAX-WS	*Java API for* XML *Web Services*	38
JBOSS	*Java Bean Open Source Software*	22
JDK	*Java Development Kit*	25
JESS	*Java Expert System Shell*	22
JRE	*Java Runtime Environment*	25
JVM	*Java Virtual Machine*	17
JSA	JADE *Semantics Add-on*	6
JSF	*JavaServer Faces*	18
JSP	*JavaServer Pages*	18
JSTL	*JavaServer Tag Library*	19
J2SE	*Java 2 Standard Edition*	25
MIME	*Multipurpose Internet Mail Extensions*	13
MIT	*Massachusetts Institute of Technology*	6
MPEG-4	*Moving Pictures Experts Group 4*	16
MPEG-7	*Moving Pictures Experts Group 7*	16
MPEG-7-MDS	MPEG-7 *Multimedia Description Schemes*	16
MVC	*Model-View-Controller*	18
NAICS	*North American Industry Classification System*	12
OASIS	*Organization for the Advancement of Structured Information Standards*	14
OWL	*Web Ontology Language*	16
OWL-S	*Web Ontology Language for Services*	15
PaaS	*Platform as a Service*	76
PVM	*Parallel Virtual Machine*	6
P2P	*Peer to Peer*	13
RNIF	*RosettaNet Implementation Framework*	14
RNTD	*RosettaNet Technical Dictionary*	15
SGML	*Standard Generalized Markup Language*	12
SOA	*Service Oriented Architecture*	7
SOAP	*Simple Object Access Protocol*	11
SSL	*Secure Sockets Layer*	11
SWS	*Semantic Web Services*	13
UBL	*Universal Business Language*	8
UDDI	*Universal Description Discovery and Integration*	7
UNSPSC	*United Nations Standard Products and Services Code*	15
URL	*Uniform Resource Locator*	12
USPSC	*Universal Standard Products and Services Classification*	12
WAR	*Web ARchive*	25
WSDC	*Web Services Dynamic Client*	25
WSDL	*Web Services Description Language*	11

Abreviatura	Descrição	Página
WSIG	*Web Services Integration Gateway*	25
WSS	*Web Services Security*	11
W3C	*World Wide Web Consortium*	16
XML	*eXtensible Markup Language*	12
XSD	XML *Schema Definition*	12

Agradecimentos

Considerando que esta dissertação é o culminar da minha caminhada no Instituto Superior de Engenharia do Porto (ISEP), agradecer a todos os que para ela contribuíram positivamente não é tarefa fácil.

Para não incorrer em injustiça agradeço de antemão a todos, sem excepção, que se cruzaram comigo durante este percurso académico. Evidencio, particularmente, a Professora Doutora Benedita Malheiro pela sua extraordinária sensibilidade que a diferencia como orientadora. São vários os adjectivos que a valorizam quer pelo seu elevado grau de profissionalismo, quer pela dedicação que demonstrou. Deixo aqui também um especial agradecimento aos meus pais, irmão e namorada porque foram eles que me apoiaram incondicionalmente durante o desenvolvimento deste projecto.

Capítulo 1

Introdução

Neste capítulo apresenta-se a contextualização do projecto, o problema que se pretende resolver, os objectivos a alcançar, o planeamento do projecto e a estrutura deste documento.

1.1 Contexto

Este projecto surgiu no âmbito da plataforma B2B de transacção automática de componentes multimédia em construção que é, por sua vez, um dos componentes do sistema de personalização de conteúdos baseado no perfil dos espectadores em desenvolvimento. O domínio de aplicação é a personalização dos intervalos publicitários, *i.e.*, a criação de um alinhamento de anúncios compatível com os interesses expressos e o perfil construído dos espectadores. Prevê-se que, no futuro, este tipo de plataformas venham a ser utilizadas pelas empresas de publicidade para, em conjunto com os fornecedores de conteúdos multimédia, conseguirem atingir segmentos específicos do público.

A plataforma funciona como um mercado electrónico de negociação automática destinado a agências de publicidade (empresas produtoras) e empresas provedoras de conteúdos e serviços multimédia aos consumidores finais (empresas distribuidoras). As empresas, uma vez registadas na plataforma, passam a ser representadas por agentes que negoceiam automaticamente os itens submetidos com o comportamento especificado.

A arquitectura global da plataforma consiste num sistema multiagente organizado em três camadas compostas por: (*i*) agentes de interface com as empresas; (*ii*) agentes modelação das empresas; e (*iii*) agentes delegados dedicados às negociações específicas de bens ou serviços.

A programação baseada em agentes é uma tecnologia popular na área dos mercados electrónicos. Os agentes são módulos computacionais que podem trabalhar em paralelo, realizando tarefas especificas em representação de alguma entidade, apresentando graus de autonomia muito diversos e interagindo com o utilizador e com outros agentes ou sistemas computacionais. Destacam-se nas áreas da inteligência artificial, da computação na nuvem e da computação em grelha. Uma das grandes vantagens deste tipo de agentes é a sua capacidade de execução abstraindo-se do sistema operativo, o que assegura a portabilidade, mobilidade e interoperabilidade.

1.2 Problema

O desafio consiste no desenvolvimento da interface entre as empresas e a plataforma B2B de transacção de componentes multimédia. Do lado da plataforma, o trabalho vai-se centrar no desenvolvimento de funcionalidades para as camadas superior e intermédia. Do lado das empresas, pretende-se desenvolver um conjunto mínimo de funcionalidades que permitam o armazenamento persistente assim como o carregamento e descarregamento automático de dados de e para a plataforma.

A camada superior da plataforma é composta por agentes de interface com as empresas. A cada empresa registada na plataforma corresponde um agente de interface dedicado que expõe um conjunto de operações através de uma interface *Web* padrão. Esta interface destina-se exclusivamente à empresa em causa e permite-lhe carregar para a plataforma as características dos anúncios ou intervalos a transaccionar, os perfis de negociação a adoptar, *etc.* e descarregar os resultados das negociações realizadas.

Na camada intermédia encontram-se os agentes que modelam cada uma das empresas registadas na plataforma. Estes agentes oferecem, através de uma interface serviço *Web*, um conjunto de operações, mas, desta vez, apenas aos agentes da plataforma. Os agentes da camada superior invocam estas operações para submeter os dados recebidos das empresas e obter os resultados das negociações, respectivamente.

1.3 Motivação

A motivação para a elaboração desta tese adveio da perspectiva futura de implementação num ambiente real, nomeadamente entre operadores de *video on demand* e agências de publicidade, a grande diversidade de temas abrangidos, que exigiu um elevado empenho e capacidade de trabalho, e o conjunto de objectivos bem definidos e claros, que permitiu planear e desenvolver o projecto de forma organizada e faseada.

1.4 Objectivos

Com esta tese pretende-se provar a adequação do paradigma dos mercados electrónicos suportados por sistemas multiagente para, em tempo quase real, criar um alinhamento publicitário personalizado.

Este projecto pretende conceber e desenvolver a interface entre as empresas e a plataforma B2B de transacção automática de componentes multimédia em construção. Dada a sua complexidade, o trabalho foi dividido no seguinte conjunto de objectivos parcelares:

- Estudo do estado da arte e da plataforma B2B de negociação automática de anúncios em construção [1];
- Concepção e realização da camada de interface, incluindo os agentes de interface;
- Desenvolvimento dos serviços *Web* dos agentes de interface e de modelação das empresas necessários à carga e descarga de informação entre empresas e plataforma;
- Desenvolvimento da *Graphic User Interface* (GUI) dos agentes de interface;
- Gestor automático de perfil de mercado;
- Realização e validação de testes.

1.5 Planeamento do Projecto

O projecto foi dividido num conjunto de doze tarefas de forma a estabelecerem-se metas e prazos para a sua execução. A primeira tarefa consistiu no estudo e pesquisa do conjunto de tópicos abordados. Dada a variedade de temas a abordar, subdividiu-se esta tarefa no estudo dos sistemas pré-existentes (de forma a conhecer as tecnologias utilizadas) e, na segunda fase, já de mais rigor e com uma ideia definida das tecnologias a utilizar, aprofundou-se o seu estudo. A segunda tarefa consistiu na elaboração da dissertação que foi subdividida em seis partes que correspondem a cada capítulo do relatório. A terceira tarefa teve como objectivo instalar, preparar e explorar as ferramentas de trabalho a utilizar. A quarta tarefa, que é a mais prolongada, consistiu no desenvolvimento da plataforma. A quinta tarefa consistiu na depuração, teste e validação do protótipo. Por fim, na sexta tarefa efectuou-se uma discussão dos resultados obtidos. Na Figura 1.1 apresenta-se a calendarização do projecto.

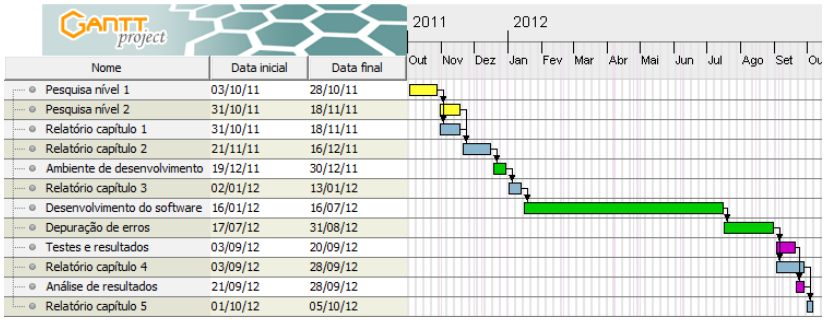


Figura 1.1: Calendarização

1.6 Estrutura da Dissertação

A dissertação é composta por seis capítulos. No Capítulo 1 efectua-se a contextualização e a apresentação do projecto. No Capítulo 2 descrevem-se os sistemas congéneres mais representativos e comparam-se com o sistema em construção. No Capítulo 3 são apresentadas as tecnologias de suporte ao desenvolvimento do projecto. No Capítulo 4 apresenta-se o ambiente de desenvolvimento adoptado. No Capítulo 5 descreve-se o desenvolvimento das aplicações de interface com as empresas e da plataforma (camadas de interface e intermédia). No Capítulo 6 detalha-se o conjunto de testes efectuados ao funcionamento da plataforma e ao desempenho do sistema. No Capítulo 7 apresentam-se as principais conclusões e as perspectivas de futuros desenvolvimentos. Em anexo, apresentam-se os diagramas de sequência das principais interacções com plataforma, os casos de uso relativos as acções que cada agente disponibiliza, o modelo Enhanced Entity Relatition (EER), contendo todas as tabelas da base de dados e, por fim, as três ontologias utilizadas.

Capítulo 2

Plataformas de Negociação Automática

Neste capítulo é apresentado o estado da arte dos sistemas de negociação automática mais representativos e é efectuada uma análise comparativa das suas principais características face ao sistema em desenvolvimento.

2.1 Soluções Analisadas

Actualmente existe um conjunto de sistemas com objectivos semelhantes aos propostos nesta dissertação. De entre o vasto conjunto de sistemas disponíveis na literatura da área, seleccionaram-se três plataformas multiagente de negociação automática e três plataformas *Business-to-Business* (B2B).

2.1.1 Sistemas Multiagente de Comércio Electrónico

No domínio dos sistemas multiagente de comércio electrónico escolheram-se o AuctionBot [2], o Fishmarket [3] e o Kasbah [4].

AuctionBot é um sistema multiagente desenvolvido na Universidade do Michigan que gere múltiplos leilões em paralelo. Implementa diversos tipos de leilões: (*i*) leilões de perfil único, *i.e.*, onde os apostadores apenas podem ser compradores ou vendedores, *e.g.*, *First-price Sealed-bid* (FPSB) *Vickrey Auction*, *Dutch Auction* e *English Auction*; e (*ii*) leilões de perfil duplo onde existe a possibilidade dos intervenientes serem compradores e vendedores ao mesmo tempo, *e.g.*, *Continuous Double Auction* (CDA) e *Call Market*. O AuctionBot organiza as negociações activas num catálogo de forma hierárquica, sendo possível criar subcategorias de negociação. A arquitectura

é composta por: (*i*) um conjunto de interfaces da plataforma que interagem directamente com uma base de dados; (*ii*) um módulo que permite agendar leilões e enviar os parâmetros do leilão para os agentes de negociação; e (*iii*) os agentes de negociação [2].

Fishmarket é um sistema desenvolvido no Artificial Intelligence Research Institute da Universidade Politécnica da Catalunha que, tal como AuctionBot, permite múltiplos leilões em simultâneo e utiliza um conjunto de agentes com regras pré-definidas para simplificar a sua classificação. Foi desenvolvido utilizando *Parallel Virtual Machine* (PVM) como *middleware* que assegura uma grande mobilidade e a possibilidade de configuração de máquinas virtuais, criando um sistema do tipo *sand-box* com a possibilidade de expansão da plataforma distribuída. O tipo de leilão é baseado no leilão da lota do peixe, apresentando as propostas por ordem decrescente [3].

Kasbah é um sistema desenvolvido no Massachusetts Institute of Technology (MIT) que utiliza o *Continuous Double Auction* (CDA) e tem como objectivo que os agentes negoceiem de forma autónoma. Os agentes vendedores são classificados como anunciantes, sendo definido um produto para cada agente. O sistema contém três tipos de adaptação do preço ao longo do tempo: linear, quadrático e cúbico. Este mecanismo define a estratégia de negociação do agente no mercado. Este e outros parâmetros podem ser alterados em qualquer momento da negociação. Uma característica interessante do Kasbah é o *Better Business Bureau* que permite, após conclusão da negociação, avaliar qualitativamente a entidade envolvida no negócio de forma a usar posteriormente esta informação para definir a credibilidade da entidade envolvida na negociação [4]. Existem alguns projectos utilizando o Kasbah como, por exemplo, um trabalho realizado pela Universidade de Campinas que utiliza o Kasbah com um sistema de catálogo para as negociações [5].

2.1.2 Plataformas B2B

No domínio das plataformas B2B seleccionaram-se a JIA Mediation Platform [6], a B2B E-procurement Platform [7], a B2B E-Commerce Model (MAECS) Platform [8] e a Plataforma B2B de Personalização de Anúncios [9] em construção.

JIA Mediation Platform é uma plataforma B2B de mediação automática constituída por um sistema multiagente. A plataforma foi desenvolvida usando Java Agent DEvelopment Framework (JADE) e o *plug-in* JADE Semantics Add-on (JSA) que consiste num motor de regras para formalizar a semântica das mensagens padrão *Agent Communication Language* (ACL) da Foundation for Intelligent Physical Agents (FIPA). Nesta plataforma existem três

tipos de agentes mediadores: (*i*) agentes mediadores da negociação com os agentes das empresas; (*ii*) mediadores de todos os agentes da plataforma; e (*iii*) agentes de mediação institucionais. Os agentes mediadores, ao invés de implementarem um protocolo de negociação específico, implementam um conjunto de regras de interacção – *rules of exchange* – com o objectivo de assegurar a interoperabilidade entre agentes clientes e fornecedores e evitar bloqueios [10].

E-procurement Platform é uma plataforma B2B de aquisição electrónica baseada em agentes e serviços *Web*. A plataforma foi desenhada com base no paradigma *Service Oriented Architecture* (SOA) de forma a disponibilizar as suas acções sob a forma de serviços prontos a consumir. A sua arquitectura conta com um agente de mercado que pesquisa no *Universal Description Discovery and Integration* (UDDI) os fornecedores de serviços e inclui uma camada intermédia de negociação e uma base dados para guardar a informação das empresas [7].

Multi-Agent E-Commerce Model System (MAECS) é um sistema multiagente destinado ao comércio electrónico entre empresas. Neste caso, os agentes estão organizados numa estrutura federativa que contém agentes facilitadores com a missão de disponibilizar informação e estabelecer a ponte entre o distribuidor e o produtor, uma vez que os agentes de duas entidades distintas não podem comunicar directamente. Os agentes comunicam trocando mensagens FIPA-ACL [8].

Plataforma B2B de Personalização de Anúncios é um sistema multiagente destinado à transacção de componentes multimédia entre empresas em desenvolvimento [11] [12]. A arquitectura da plataforma está organizada em três camadas: (*i*) camada de interface com as empresas; (*ii*) camada de modelação das empresas; (*iii*) camada de mercado onde apenas existem agentes delegados dos agentes da camada intermédia a negociar os produtos. A comunicação entre os agentes de diferentes camadas é efectuada através de serviços *Web*. Os agentes da camada de mercado comunicam entre si trocando mensagens FIPA-ACL e implementam os protocolos de negociação FIPA-ICNIP, o FIPA-CNIP e o *Fixed* ICNIP(FICNIP). Todos os serviços *Web* expostos pelos agentes da plataforma são publicados no UDDI para permitir a sua descoberta e consumo [9].

2.2 Análise Comparativa

As soluções apresentadas vão ser analisadas em relação às seguintes características:

- Arquitectura do sistema – se é composto por um ou mais agentes / módulos organizados numa ou em múltiplas camadas;
- Negociação do sistema – se dispõe de um ou vários protocolos de negociação;
- Escalabilidade do sistema – se permite aumentar a capacidade de processamento, *i.e.*, o número de agentes / módulos, de forma significativa sem afectar o desempenho global;
- Interoperabilidade do sistema – se permite interagir com outros agentes, módulos ou aplicações de forma automática independentemente da plataforma e da linguagem de desenvolvimento adoptada;
- Abertura do sistema – se agentes / módulos podem ser criados, ficar isolados ou ser eliminados sem afectar o funcionamento dos restantes componentes.

Na Tabela 2.1 comparam-se a arquitectura, os protocolos de negociação, a escalabilidade, a interoperabilidade e a abertura dos sistemas de comércio electrónico apresentados.

Tabela 2.1: Comparação dos sistemas de comércio electrónico

	AuctionBot	FishMarket	Kasbah
Arquitectura	Camada única	Camada única	Camada única
Negociação	Leilão de Vickrey Leilão inglês Leilão holandês Leilão duplo contínuo Call Market	Leilão holandês	Directa
Escalabilidade	Não	Sim	Não
Interoperabilidade	Sim	Sim	Sim
Abertura	Não	Não	Não

A Tabela 2.2 apresenta a comparação do tipo de arquitectura, protocolos de negociação, escalabilidade, interoperabilidade e abertura dos sistemas B2B descritos. A JIA Mediation Platform não foi incluída nesta tabela porque modela a fase das transacções B2B que decorre após a etapa de negociação. Assim, ao invés de um protocolo de negociação adopta uma abordagem baseada na troca de documentos electrónicos de acordo com a especificação dos processos de negócio efectuada em *Universal Business Language* (UBL).

2.3 Conclusão

Neste capítulo apresentaram-se e compararam-se as principais características de sistemas congéneres e do sistema em desenvolvimento. Esta análise permite con-

Tabela 2.2: Comparação dos sistemas B2B

	E-Proc.	MAECS	MMB
Arquitectura	Org. em camadas	Camada única	Org. em camadas
Negociação	Não explicitado	Leilão proprietário	CNIP e ICNIP
Escalabilidade	Não	Sim	Sim
Interoperabilidade	Sim	Sim	Sim
Abertura	Não	Não	Sim

cluir que a plataforma B2B de personalização de anúncios em construção apresenta um conjunto de características promissor face aos sistemas congéneres.

No capítulo seguinte identificam-se as principais tecnologias de suporte ao desenvolvimento da plataforma B2B de personalização de anúncios.

Capítulo 3

Tecnologias de Suporte

Neste capítulo referem-se as principais tecnologias envolvidas ou de relevância para a realização deste projecto. Descrevem-se os serviços Web, as especificações de transacções B2B, as abordagens de representação de conhecimento, a computação baseada em agentes, a determinação da similaridade e as tecnologias Web utilizados do lado do cliente e do lado do servidor.

3.1 Serviços *Web*

Os serviços *Web* são baseados em quatro tecnologias: XML, *Simple Object Access Protocol* (SOAP), *Web Services Description Language* (WSDL) e UDDI. Esta conjugação permite criar sistemas distribuídos vocacionados para a comunicação intra e interplataforma de acordo com o paradigma SOA que asseguram um elevado nível de interoperabilidade [13]. O UDDI permite localizar e consumir serviços dinamicamente, podendo incluir mecanismos de segurança ao nível da troca das mensagens SOAP. A utilização de *Secure Socket Layer* (SSL) garante protecção ao nível da camada de transporte, mas não cifra os dados. Podem ser adoptados outros mecanismos de segurança como o XML *Encryption* e o XML *Signature*. O primeiro cifra os dados a partir de uma chave privada e o segundo caso utiliza assinaturas digitais para autenticação. Por último, o *Web Services Security* (WSS) é composto por três mecanismos de segurança: (*i*) o primeiro verifica a integridade da mensagem SOAP; (*ii*) o segundo cifra a mensagem SOAP para assegurar confidencialidade; e (*iii*) o terceiro anexa *tokens* de segurança para verificação da identidade junto do receptor. A desvantagem deste mecanismo é o aumento significativo do tamanho (*overhead*) das mensagens SOAP [14].

3.1.1 Extensible Markup Language

A tecnologia *eXtensible Markup Language* (XML) é uma meta-linguagem que permite a especificação de novas linguagens ou dialectos válidos e bem formados. Desenvolveu-se a partir da *Standard Generalized Markup Language* (SGML) com o propósito de permitir armazenar dados de forma semi-estruturada, recorrendo a anotações, e separar o conteúdo da formatação do documento. A especificação de um novo dialecto ou linguagem baseia-se na definição e armazenamento da estrutura do novo dialecto XML num ficheiro que é posteriormente utilizado para validar os documentos do dialecto. Este papel é desempenhado pelo ficheiro *Document Type Definition* (DTD) ou do *XML Schema Definition* (XSD). Como o DTD não utiliza a sintaxe do XML é preferível optar pelo XSD. O ficheiro XSD define o conjunto e a estrutura de anotações do dialecto XML, nomeadamente, o nome (anotação), o tipo de dados e a cardinalidade de cada elemento em XML [15] [16].

Dado que a plataforma é desenvolvida em Java, utilizou-se a *Java Architecture for XML Binding* (JAXB) *application programming interface* (API) que permite, de forma simples e com base no XSD ou DTD, o empacotamento (*marshalling*) de objectos Java em representações XML e o desempacotamento (*unmarshalling*) dos dados de um ficheiro XML nos objectos Java correspondentes. O processo de empacotamento consiste em três passos essenciais: (*i*) validação dos objectos Java para garantir a validade dos dados; (*ii*) conversão dos dados para documentos XML; e (*iii*) armazenamento do documento XML resultante. O processo de desempacotamento também pode ser descrito em três passos essenciais: (*i*) construção do documento XML para desempacotar em objectos Java; (*ii*) conversão dos dados do documento XML em instâncias de classes Java; e (*iii*) armazenamento dos objectos resultantes [15] [16] [17] [18].

3.1.2 UDDI

O serviço de registos UDDI permite definir o modelo dos dados em XML e utilizar as API de SOAP para o registo e pesquisa de informação das empresas e dos seus serviços. De uma forma resumida, o UDDI funciona como as páginas amarelas onde as empresas publicam os seus dados e serviços, incluindo o *Uniform Resource Locator* (URL) dos ficheiros WSDL de descrição dos serviços.

As empresas e os serviços afixados no UDDI podem ser classificados de acordo com taxonomias padrão como, por exemplo, a *North American Industry Classification System* (NAICS) e a *Universal Standard Products and Services Classification* (USPSC), respectivamente. O UDDI utiliza internamente estruturas designadas `tModels` para representar as taxonomias. A utilização da taxonomia NAICS permite efectuar pesquisas e obter correspondências directas como, por

exemplo, a pesquisa de uma empresa ou de um `tModel`, mas, também, a pesquisa por categorias de empresas ou de serviços [19].

O UDDI apresenta dois pontos fracos: (*i*) a sua dependência de um nó central; e (*ii*) a ausência de suporte para *Semantic Web Services* (SWS). Enquanto a primeira fragilidade poderia ser resolvida através da adopção de uma solução dinâmica do tipo *Peer-to-Peer* (P2P), garantindo maior escalabilidade e robustez [20], a segunda impede que os serviços afixados no UDDI sejam, à imagem do que acontece com os SWS, definidos através de uma ontologia de representação de serviços que assegure a interoperabilidade entre plataformas e a integração do domínio de conhecimento.

3.1.3 Pilha Protocolar

Os serviços *Web* utilizam a pilha protocolar da Figura 3.1. Na primeira camada, a camada de transporte, é utilizado habitualmente o protocolo HTTP. O protocolo permite a transferência de dados através da Internet, incluindo mensagens do tipo *Multipurpose Internet Mail Extensions* (MIME) [21]. Na segunda camada designada XML *Messaging* utiliza-se XML e SOAP. Este protocolo destina-se a transportar conteúdos XML sem necessidade de alterar a sua semântica ou estrutura [22]. Na terceira camada encontra-se a linguagem de descrição de serviços WSDL. Esta linguagem baseada em XML permite especificar toda a informação necessária para aceder e utilizar os serviços *Web* [23]. Por fim, na quarta camada encontram-se as especificações UDDI que incluem um serviço de registo de serviços *Web*. O serviço de registo UDDI é ele próprio um serviço *Web* que permite a publicação dos meta-dados de classificação das empresas e dos seus serviços assim como os URL dos ficheiros WSDL de descrição dos serviços publicados [24].

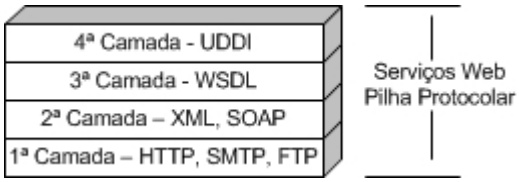


Figura 3.1: Serviços *Web* - pilha protocolar

3.2 Transacções Electrónicas B2B

A *Organization for the Advancement of Structured Information Standards* (OASIS) apresenta duas especificações de modelação de transacções electrónicas entre empresas: *electronic business* XML (ebXML) e o *Business Process Execution Language* (BPEL).

3.2.1 Electronic Business XML

A *Organization for the Advancement of Structured Information Standards* (OASIS) definiu um conjunto de padrões de interface entre processos de negócio de onde se destaca o ebXML, que consiste num padrão de interacção entre plataformas B2B. As mensagens e anexos ebXML foram definidos em SOAP, adicionando qualidade de serviço, segurança e garantia de troca de mensagens. A sua arquitectura é centrada na troca de documentos que vai de encontro ao paradigma dos serviços *Web* orientados para B2B [25].

A finalidade dos serviços *Web* e do ebXML são distintas: os serviços *Web* expõem aplicações que qualquer entidade pode invocar – abordagem SOA; o ebXML destina-se à troca entre parceiros empresariais de documentos bem definidos segundo um processo de negócio pré-acordado e definido – abordagem baseada em relações contratuais [26]. O ebXML descreve a coreografia de um processo electrónico de negócio através de um *Business Process Specification Schema* (BPSS).

O ebXML, tal como o UDDI, disponibiliza um serviço de registo de serviços para as empresas e os seus serviços. No entanto, os dois serviços de registo diferem na forma como os registos são classificados. Enquanto o UDDI utiliza `tModels` para representar uma taxonomia, o ebXML usa uma hierarquia de classificações baseada no registo de itens.

O protocolo estipula que cada empresa cria o seu *Collaboration Protocol Profile* (CPP). Da relação entre dois CPP de duas empresas diferentes resulta a criação de um *Collaboration Protocol Agreement* (CPA) que descreve a relação formal entre as duas entidades. De forma a assegurar a integridade dos dados de ambas as empresas, o ebXML define a utilização do ebXML *Messaging Service* (ebMS).

No domínio semântico, o ebXML inclui duas normas: ebXML *Registry Service* (ebXML-RS) e o ebXML *Registry Information Model* (ebXML-RIM) [27] [28].

A organização RosettaNet, que desenvolve padrões para a cadeia de distribuição global, tem vindo a convergir no sentido da utilização do ebXML, mais concretamente no RosettaNet *Implementation Framework* (RNIF) [29]. Esta decisão da RosettaNet constitui um apoio importante para um alargamento da utilização do ebXML como o padrão no comércio electrónico mundial.

3.2.2 Business Process Execution Language

A BPEL é um linguagem padrão da OASIS para a especificação da execução dos processos de negócio que é suportada pela tecnologia dos serviços *Web*, *i.e.*, utiliza SOAP, WSDL e UDDI.

A BPEL em conjunto com a *Business Process Modelling Language* (BPML) constituem outra abordagem à especificação das transacções electrónicas entre

empresas. Para a BPEL, a execução de uma sequência de processos electrónicos de negócio entre duas empresas é encarada como uma orquestração [30]. Assim a BPEL permite descrever a orquestração da execução de um processo de negociação assim como definir a coreografia de um protocolo de negociação [31]. Inclui ainda dois mecanismos que permitem a alteração dos fluxos de trabalho: o *Business Process Reengineering* (BPR) e o *Continuous Process Improvement* (CPI). Enquanto o BPR analisa e redesenha periodicamente o processo de negócio, o CPI realiza um melhoramento contínuo incorporando e coordenando pequenas alterações [32].

3.3 Representação de Conhecimento

No âmbito deste projecto a representação do conhecimento do domínio suportou-se em taxonomias e ontologias. As ontologias foram utilizadas para representar a estrutura da informação da plataforma assim como os perfis dos espectadores, programas e anúncios. As taxonomias são usadas no âmbito do serviço de registo UDDI.

3.3.1 Taxonomias

A adopção de sistemas de classificação padrão promove a interoperabilidade entre plataformas. As taxonomias são sistemas simples de classificação de conhecimento organizados em árvore. De entre as inúmeras taxonomias existentes, salientam-se três sistemas padrão de classificação para comércio electrónico: (*i*) o *North American Industry Classification System* (NAICS), que permite a classificação de empresas [33]; (*ii*) o *United Nations Standard Products and Services Code* (UNSPSC), que permite a classificação de produtos e serviços [34]; e (*iii*) o *RosettaNet Technical Dictionary* (RNTD), que define um vocabulário para comércio electrónico de bens e serviços [35].

Da comparação entre os sistemas UNSPSC e RosettaNet, ressalta a diferença do nível de detalhe: o UNSPSC possui cerca de 21 000 classes e o RosettaNet aproximadamente 800 classes. Esta diferença pode causar problemas de compatibilidade semântica nos casos em que as duas taxonomias estejam presentes em simultâneo [36] [37].

3.3.2 Ontologias

As ontologias são sistemas de representação da estrutura do conhecimento num dado domínio (conceitos, acções, predicados, *etc.*) e das suas relações. Existem diversas linguagens de definição de ontologias como, *e.g.*, DARPA *Agent Markup Language for Services* (DAML-S) e *Web Ontology Language for Services* (OWL-S). Esta última, inspirou-se na primeira e apresenta como vantagem ser definida

em *Web Ontology Language* (OWL), um padrão do *World Wide Web Consortium* (W3C) para definição de ontologias.

Neste projecto definiram-se ontologias em OWL e em Protégé Frame, um formato proprietário do editor de ontologias Protégé.

Paralelamente, está a decorrer um trabalho vocacionado para o *streaming* de video com anotação semântica. As tecnologias utilizadas são o *Moving Pictures Experts Group 4* (MPEG-4) para a transmissão e compressão do sinal e o *Moving Pictures Experts Group 7* (MPEG-7) para especificação dos metadados. O formato MPEG-4 foi desenvolvido com o objectivo de se obter uma melhor taxa de compressão. No entanto, durante o processo de normalização do MPEG-4, surgiu a necessidade de representação da descrição dos conteúdos multimédia, dos seus produtores e dos seus distribuidores [38]. O MPEG-7 foi desenvolvido para os produtores de conteúdos multimédia terem à sua disposição uma norma de descrição de meta-dados do conteúdo, estrutura e observações que sejam relevantes para o autor [39]. Estas duas normas permitem manipular e enviar objectos multimédia e os seus meta-dados. Para criar modelos semânticos dos conteúdos multimédia codificados em MPEG-4 e anotados em MPEG-7, utilizam-se os MPEG-7-*Multimedia Description Schemes* (MPEG-7-MDS) para criar *Description Schemes* (DS) associados a uma dada ontologia [40]. Neste contexto, existe um conjunto de ontologias baseadas em MPEG-7 que implementam características diversas desta especificação, *e.g.*, a ontologia Harmony, a ontologia aceMedia, a ontologia SmartWeb, a ontologia BOEMIE, a ontologia Rhizomik, a ontologia DS-MIRF ou a ontologia COMM [41].

3.4 Computação Baseada em Agentes

A computação baseada em agentes é o paradigma de programação adoptado neste projecto. Apesar de não existir um consenso sobre a definição de agente, Wooldridge e Jennings afirmam que "*An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.*" [42]. Os agentes podem também ser definidos através das suas características como agranularidade, o grau de autonomia, capacidade de adaptação e de aprendizagem, flexibilidade, tipo de comunicação e de inferência. Os agentes, enquanto unidades autocontidas de código com conhecimento, capacidade de inferência e de comunicação, são particularmente adequados para modelar problemas de natureza distribuída, nomeadamente, o comércio electrónico de bens e serviços.

Do ponto de vista tecnológico, os agentes desenvolvidos em Java apresentam um conjunto acrescido de propriedades e mecanismos. Neste contexto, um sistema multiagente torna-se bastante versátil quer em termos de mobilidade de recursos, quer em relação à migração para a sua rede de processamento na *cloud*. Os

agentes podem ser criados e ficar alojados numa plataforma local onde cada agente é uma *thread* da *Java Virtual Machine* (JVM), explorando as capacidades de processamento paralelo; podem ser criados localmente e movidos para uma plataforma de agentes remota; ou cada agente pode ser criado e alojado numa plataforma física distinta sem recorrer a uma plataforma centralizada de agentes [43]. Em termos de segurança os agentes desenvolvidos em Java dispõem de uma API de segurança chamada *Java Authentication and Authorization Service* (JAAS) que permite a autenticação de agentes numa plataforma centralizada. Outra opção para prevenir a emissão de ataques a uma plataforma de agentes passa pela definição de políticas de acesso aos agentes, *e.g.*, através da definição de uma ontologia com políticas de acesso que, em conjunto com o JAAS, dão uma maior robustez à plataforma [44]. Se os agentes forem publicados na *Web* sob a forma de serviços, pode-se ainda implementar um mecanismo de *tokens* para controlar o acesso às acções dos agentes.

3.4.1 Protocolos de Negociação

As plataformas multiagente de negociação adoptam diversos tipos de protocolos de negociação, incluindo protocolos de negociação de um para um, de um para muitos e de muitos para muitos. A negociação directa ou *bargaining*, os leilões e a rede contratual ou *contract net* são alguns dos tipos de protocolos de negociação mais comuns neste contexto.

Neste projecto utiliza-se o FIPA *Contract Net Interaction Protocol* (CNIP), o FIPA *Iterated Contract Net Interaction Protocol* (ICNIP) e o *Fixed* ICNIP (FICNIP), que é uma variante do FIPA-ICNIP que foi implementada. O FIPA-CNIP apresenta o seguinte funcionamento:

1. O agente que inicia a negociação endereça um pedido de envio de propostas – *Call For Proposals* (CFP) – a um conjunto de agentes.

2. Os agentes visados avaliam o pedido. Se não estão interessados, recusam participar com *refuse*. Caso considerem que se trata de uma oportunidade de negócio, remetem as suas propostas individuais.

3. Quando termina o tempo de recepção de propostas, o agente que iniciou a negociação avalia todas as propostas recebidas e escolhe uma ou nenhuma. Informa cada agente participante do resultado, enviando *accept-proposal* ou *reject-proposal* conforme aceita ou recusa a proposta desse agente.

O funcionamento do FIPA-ICNIP é semelhante ao FIPA-CNIP apenas diferindo no facto de o agente que inicia a negociação repetir o processo descrito até encontrar uma proposta que lhe agrade ou desistir [45] [46]. O FICNIP, que é idêntico

ao FIPA-ICNIP, repete o processo do FIPA-CNIP um número fixo de iterações, escolhendo no final a melhor proposta de todo o conjunto de propostas recebidas.

3.5 Tecnologias *Web*

No âmbito do desenvolvimento da aplicação Web utilizaram-se o Asynchronous JavaScript and XML (AJAX), JavaServer Faces (JSF), incluindo o JavaServer Pages (JSP) e o PrimeFaces.

3.5.1 Asynchronous JavaScript and XML

Inicialmente, a tecnologia *HyperText Markup Language* (HTML) foi pensada para produzir páginas *Web* estáticas. Para permitir a criação de páginas interactivas do lado do cliente foi então desenvolvida uma linguagem de *scripting*: o JavaScript [47].

Na aplicação *Web* utiliza-se o AJAX. O AJAX consiste num conjunto de três tecnologias que permite de um modo assíncrono efectuar pedidos a um servidor. Os pedidos são efectuados em *background* com recurso ao JavaScript e as respostas permitem efectuar actualizações de pequenas porções da página *Web*, evitando o carregamento completo da página. Os principais componentes que permitem esta dinâmica são o JavaScript, que facilita o acesso ao *Document Object Model* (DOM) da página, e o objecto `XMLHttpRequest` que permite estabelecer uma ligação assíncrona ao servidor[48].

3.5.2 JavaServer Faces

O JSF é uma tecnologia que permite o desenvolvimento de aplicações *Web* com recurso a tecnologia Java. O JSF utiliza como padrão de desenvolvimento o *Model-View-Controller* (MVC). O MVC organiza o desenvolvimento da aplicação em três componentes: *Model*, *View* e *Controller*. O *Model* representa os dados de entrada e as operações da aplicação *Web*; o *View* está encarregue da saída; e o *Controller* faz a gestão do *Model*, sempre que recebe um pedido, e altera o *View*, sempre que envia uma resposta [49].

3.5.3 JavaServer Pages

O JSP é uma tecnologia de desenvolvimento de aplicações *Web* do lado do servidor. Permite efectuar a separação entre a lógica da aplicação e da lógica da apresentação. A primeira é realizada por um conjunto de classes de Java designadas JavaBeans e a segunda consiste num conjunto de modelos de páginas *Web* enriquecidas com *scriptlets* de Java. Estes *scriptlets* consistem em chamadas ao código implementado nos JavaBeans. Quando invocadas, os documentos JSP

executam os *scriptlets*, substituindo-os pelos resultados e devolve o documento HTML resultante.

A JavaServer Pages Standard Tag Library (JSTL) é uma biblioteca de anotações para JSP. A JSTL permite efectuar uma programação de JavaServer Pages mais elegante, substituindo *scriptlets* por conjuntos de anotações de mais fácil leitura e assegurando uma melhor organização. Esta tecnologia utiliza o *Expression Language* (EL) que permite aceder às variáveis instânciadas no âmbito dos *scriptlets* JSP [50].

3.5.4 PrimeFaces

A PrimeFaces é uma biblioteca de componentes gráficos que se destina a ser utilizada em conjunto com o JSF. Contém diversos componentes gráficos que funcionam em conjunto com a tecnologia AJAX permitindo criar aplicações *Web* atraentes e interactivas do lado do cliente [51].

3.6 Determinação da Similaridade

Neste projecto é necessário determinar a similaridade entre vectores de características relativos a anúncios, programas e perfis dos utilizadores. Esta técnica, que é habitual nos sistemas de recomendação, é utilizada para efectuar sugestões baseadas nas características do perfil do utilizador e do universo de produtos ou serviços disponíveis. Para tal, tem de estabelecer o grau de emparelhamento ou de afinidade entre as características de um e de outros. Nos sistemas de recomendação utilizam-se frequentemente técnicas baseadas em filtros colaborativos, contextos e preferências expressas. Em alguns casos, recorrem, inclusive, à informação proveniente de redes sociais e das *folksonomies* associadas, levando em conta o conjunto de anotações – *tag clouds* – dos utilizadores [11].

No caso da determinação da similaridade entre o perfil de um espectador e um ou mais anúncios é necessário efectuar um mapeamento prévio das ontologias dado que utilizam ontologias parcialmente sobrepostas, *i.e.*, possuem características sem correspondência directa. A Figura 3.2 ilustra o mapeamento entre a ontologia dos anúncios (Ads) e a ontologia dos programas da `BBC` usada para caracterizar um intervalo.

3.6.1 Similaridade dos Cossenos

Um dos métodos mais conhecidos de comparação de vectores é a determinação da similaridade dos cossenos [52]. *i.e.*, calcula-se o cosseno do ângulo entre os dois vectores. O resultado varia entre 1, quando os vectores são colineares, e 0, quando os vectores são ortogonais. Neste projecto vamos aplicar esta métrica para

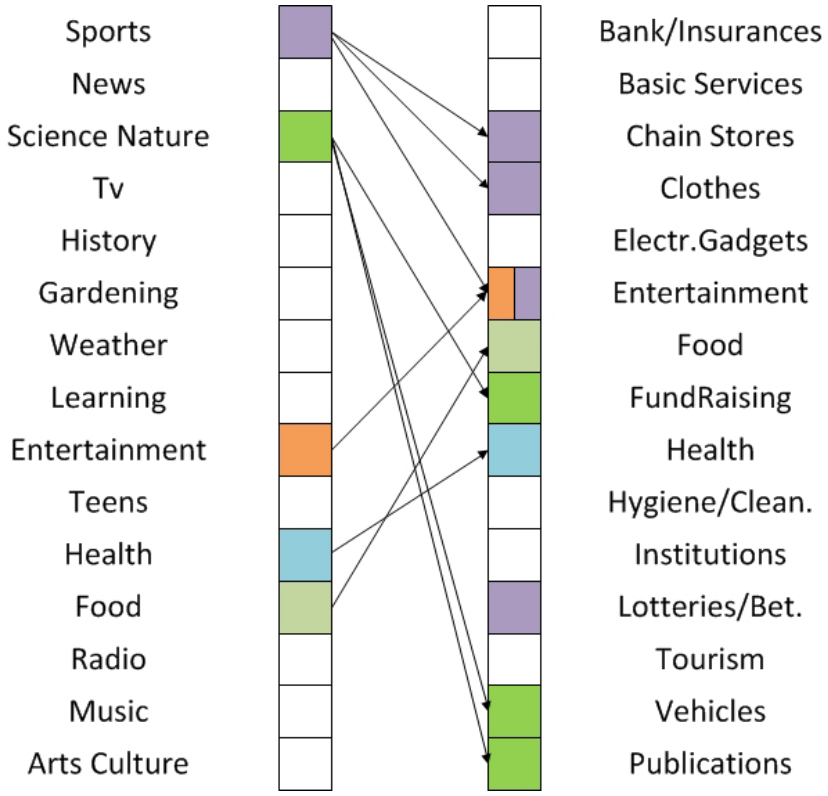


Figura 3.2: Mapeamento entre as ontologias `BBC` e `Ads`

comparar os perfis dos espectadores, dos programas e dos anúncios. A Figura 3.3 ilustra a determinação da similaridade de dois vectores com três características.

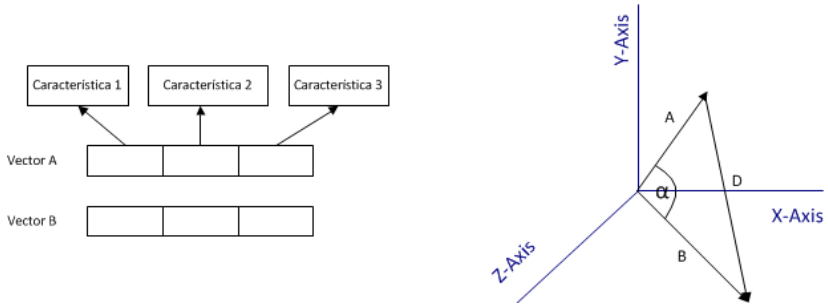


Figura 3.3: Similaridade de cossenos

Num primeiro passo, depois do vector dos anúncios ter sido mapeado para a ontologia dos programas, os vectores de características do anúncio e do intervalo a comparar são normalizados através da aplicação da Equação 3.1, onde V é o vector original, $|V|$ representa a sua norma e $\hat{V}$ é o vector normalizado resultante.

Equação 3.1: Normalização

$$\hat{V} = \frac{V}{|V|} \tag{3.1}$$

Num segundo passo, aplica-se a Equação 3.2 que determina a similaridade entre dois vectores de características $\hat{A}$ e $\hat{B}$.

Equação 3.2: Similaridade dos cossenos

$$\cos\alpha = \frac{\hat{A}\cdot\hat{B}}{|\hat{A}||\hat{B}|} \equiv \frac{\sum_{j=1}^{n}\hat{A}_j\hat{B}_j}{\sqrt{\sum_{j=1}^{n}\hat{A}_j^2}\sqrt{\sum_{j=1}^{n}\hat{B}_j^2}} \tag{3.2}$$

onde $\hat{A}$ e $\hat{B}$ são os vectores normalizados, j identifica a característica e n representa a dimensão dos vectores.

No entanto, este cálculo da similaridade através da Equação 3.2 ignora as magnitudes dos vectores [53]. Assim, quando é necessário escolher um anúncio de entre um conjunto anúncios candidatos, calcula-se a similaridade entre o perfil de cada anúncio e o perfil do espectador. Se ocorrerem empates entre os valores de similaridade resultantes, é necessário incluir um terceiro e último passo: determinar a distância euclidiana dos vectores através da Equação 3.3.

Equação 3.3: Distância euclidiana

$$d = |\hat{A} - \hat{B}| = \sqrt{\sum_{j=1}^{n}|\hat{A}_j - \hat{B}_j|^2} \tag{3.3}$$

onde $\hat{A}$ e $\hat{B}$ são os vectores normalizados, j o índice da característica e n a dimensão dos vectores.

O nível de satisfação final é calculado através de regras de inferência específicas.

3.6.2 Similaridade da Característica Dominante

A determinação da similaridade do anúncio baseada na característica dominante do perfil do intervalo efectua-se em três passos. O primeiro passo consiste na realização do mapeamento entre as ontologias do anúncio e do intervalo referido na Figura 3.2. No segundo passo identifica-se a característica dominante *cd* do vector A e determina-se a diferença dos valores da característica dominante do vector A e do vector B através da Equação 3.4, onde *cd* identifica a característica dominante do vector A e d_{cd} representa a distância da característica dominante.

Equação 3.4: Diferença da característica dominante

$$d_{cd} = A[cd] - B[cd] \tag{3.4}$$

Por fim, verifica-se se o resultado pertence ao intervalo definido na Equação 3.5, onde d_{cd} é a distância da característica dominante obtida através da Equação 3.4, $A[cd]$ é o valor da característica dominante do vector A e D_{ref} é a distância de referência definida.

Equação 3.5: Pertence ao intervalo de interesse

$$d_{cd} \in [A[cd] - D_{ref}, A[cd] + D_{ref}] \tag{3.5}$$

Teoricamente, $D_{ref} \in [0, 9]$, mas na prática utiliza-se $D_{ref} \in [0, 2]$ para definir quatro regiões distintas no intervalo de interesse que correspondem aos quatro níveis de satisfação possíveis.

Se d_{cd} pertence ao intervalo de interesse, a determinação do nível de satisfação é calculado através de regras de inferência específicas. No caso contrário, o nível de satisfação é zero (0).

3.6.3 Algoritmo Rete

Este projecto utiliza a plataforma de inferência Drools, e consequentemente o algoritmo Rete, para representar as regras de determinação do grau de afinidade entre as características de anúncios, programas e perfis dos utilizadores

O algoritmo Rete, que foi desenvolvido por Charles Forgy [54], é um algoritmo de apoio a sistemas de inferência guiados por eventos, *i.e.*, determina em tempo real e em função dos dados disponíveis no momento, qual o conjunto de regras de inferência passíveis de aplicação. Este tipo de sistemas de inferência guiados por eventos, por oposição aos sistemas de inferência guiados por objectivos, permitem uma mais rápida assimilação da informação recém-chegada [55]. O algoritmo pode ser dividido em três partes: representação, gestão e utilização de conhecimento [56]. O *Java Expert System Shell* (JESS) e o Drools do *Java Bean Open Source Software* (JBOSS) são exemplos de plataformas de inferência que implementam o Rete. Sendo um algoritmo utilizado em processos guiados por eventos, quando o número de regras de inferência aumenta, requer poder computacional e capacidade memória elevados [57].

3.7 Conclusão

Neste capítulo descreveram-se as tecnologias que vão ser utilizadas, nomeadamente, a computação baseada em agentes e os protocolos de negociação adopta-

dos, as interfaces do tipo serviço *Web* e o serviço de registo UDDI, as ontologias para a representação de conhecimento, as abordagens de determinação da similaridade entre o universo de candidatos e um perfil alvo e as tecnologias de desenvolvimento da aplicação *Web*.

Foram ainda apresentadas as linguagens ebXML e BPEL de especificação das transacções electrónicas entre empresas. Apesar de não terem sido adoptadas, foram estudadas e contempladas dado o nível de interoperabilidade que asssseguram.

No capítulo seguinte detalha-se a instalação do ambiente de desenvolvimento deste projecto que envolve as tecnologias seleccionadas.

Capítulo 4

Ambiente de Desenvolvimento

Neste capítulo descreve-se a instalação do ambiente de desenvolvimento utilizado.

4.1 Instalação do Ambiente de Desenvolvimento

A plataforma de desenvolvimento adoptada foi um computador portátil com o sistema operativo Windows 7. Nesta plataforma instalou-se o ambiente de desenvolvimento padrão de Java, o ambiente de desenvolvimento de sistemas baseados em agentes e os servidores de aplicações e serviços *Web*, o servidor de bases de dados, o serviço de registo UDDI, o editor de ontologias e um conjunto de API adicionais.

4.1.1 Ambiente de Desenvolvimento de Sistemas Multi-Agente

O Java Agent DEvelopment Framework (JADE) foi o ambiente de desenvolvimento de sistemas baseados em agentes escolhido. Trata-se de um sistema desenvolvido em linguagem Java, sendo por esta razão necessário começar pela instalação do *Java 2 Standard Edition* (J2SE), que inclui o *Java Runtime Environment* (JRE) e o *Java Development Kit* (JDK), e do *Integrated Development Environment* (IDE) NetBeans da Oracle. As versões utilizadas corresponderam ao JDK 7u9, JRE 7u9 e ao NetBeans 7.2 [58]. Após a instalação do J2SE, procedeu-se à definição de algumas variáveis de ambiente do Microsoft Windows, *e.g.*, a `%JAVA_HOME%`, `%CLASSPATH%`, *etc.* [59].

Prosseguiu-se com a instalação do JADE 4.2 e de dois *plug-ins*: o *Web Services Integration Gateway* (WSIG) e o *Web Services Dynamic Client* (WSDC). O WSIG permite aos agentes da plataforma JADE criar interfaces do tipo serviço *Web* e, assim, expor serviços *Web* que podem ser consumidos por terceiros [60] [61] [62]. Para instalar o WSIG é necessário gerar o ficheiro *Web Archive* (WAR)

do WSIG, compilando o código fonte com a ferramenta Apache Ant 1.8.4 [63] e colocá-lo na pasta `webapps` do Apache Tomcat. O WSDC permite aos agentes da plataforma JADE consumir directamente serviços *Web* através da criação automática de *stubs* e *skeletons* de interacção com serviços *Web*. O JADE e o WSDC instalam-se no NetBeans como bibliotecas da aplicação.

No NetBeans instalaram-se ainda como bibliotecas da aplicação os *plug-ins* Drools e JFreeChart. O Drools 5.4.0, desenvolvido pela *Java Bean Open Source Software* (JBOSS), permite a definição e utilização de regras de inferência [64]. O JFreeChart 1.0.14 é uma API de criação de gráficos personalizados [65].

Por último, deve-se definir a variável ambiente do sistema operativo `JADE_HOME` e, no NetBeans, especificar nas propriedades da aplicação a `main.Class` como `jade.Boot` com os argumentos `-gui -platform-id Tedi Auto:AgentAuto`.

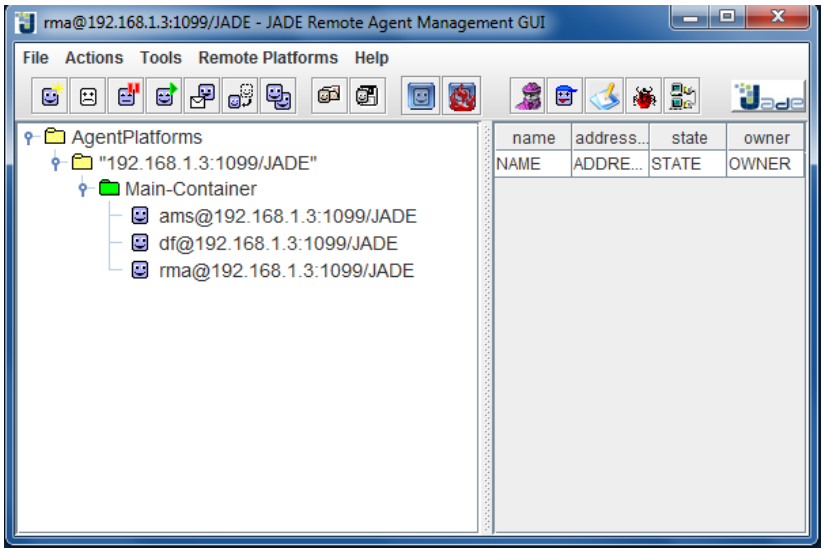


Figura 4.1: JADE

4.1.2 Ambiente de Desenvolvimento de Aplicações *Web*

No NetBeans instalaram-se ainda como bibliotecas da aplicação os *plug-ins* PrimeFaces, JSTL e JSF. O PrimeFaces 3.4, desenvolvido pela PrimeFaces, permite a definição e utilização de componentes desenvolvidos com recurso ao AJAX [51]. O JSTL 1.2.1 é uma API que permite a utilização de anotações nos documentos JSP [66]. Por seu lado, o JSF 2.1.14 é uma API de representação e manutenção do estado de componentes gráficos, incluindo o processamento de eventos, validação do lado do servidor, conversão de dados, navegação, *etc.* [67].

4.1.3 Servidor de Aplicações e Serviços *Web*

Instalou-se o servidor de aplicações *Web* Apache Tomcat e o *plug-in* Apache Axis2/Java para o desenvolvimento e *deployment* dos serviços *Web* expostos pela plataforma JADE e a sua afixação no serviço de registos UDDI. A versão do Apache Tomcat foi a 7.0.32 [68] e a do Apache Axis2/Java foi a 1.6.2. Para instalar o *plug-in* Apache Axis2/Java basta copiar para a pasta `webapps` do Apache Tomcat o ficheiro WAR do Apache Axis2/Java [69].

Para tornar mais cómodo o desenvolvimento efectuou-se um conjunto de configurações para permitir ao NetBeans controlar o Apache Tomcat. Foi necessário no ficheiro de autenticação dos utilizadores `tomcat-users.xml` os papéis `standard,manager,manager-script e admin` e no menu `Tools` do NetBeans seleccionar `servers` e adicionar o servidor Apache Tomcat, especificando os dados de autenticação de administração.

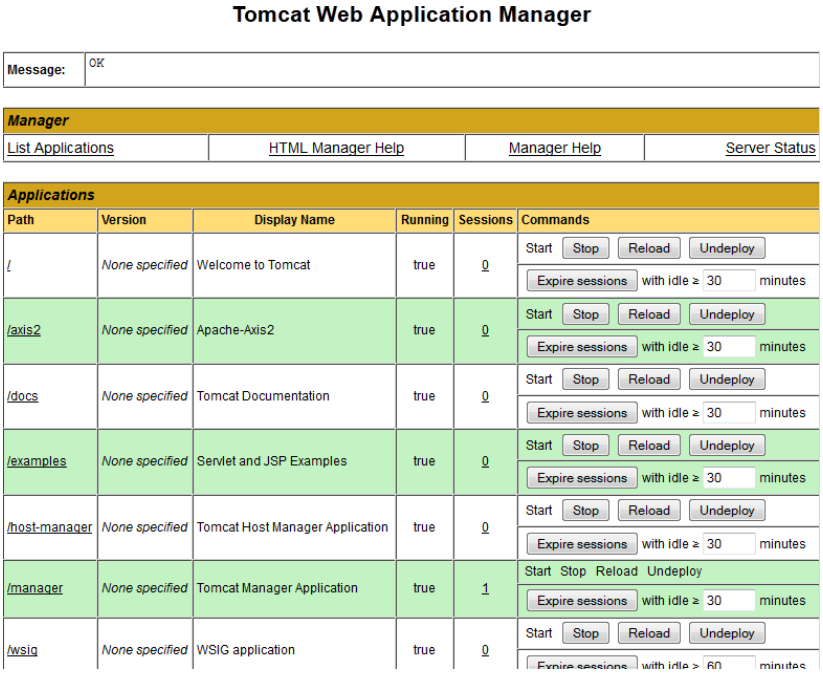


Figura 4.2: Apache Tomcat e Apache Axis2/Java

4.1.4 Servidor de Bases de Dados

Procedeu-se à instalação do servidor de base de dados MySQL Community Server 5.5 para armazenar de forma persistente a informação das empresas e da plataforma. Incluiu-se o MySQL Workbench 5.2 que fornece um ambiente gráfico para a interacção com o servidor e que inclui, entre outras ferramentas, a possibilidade de criação de modelos *Enhanced Entity Relationship* (EER) [70].

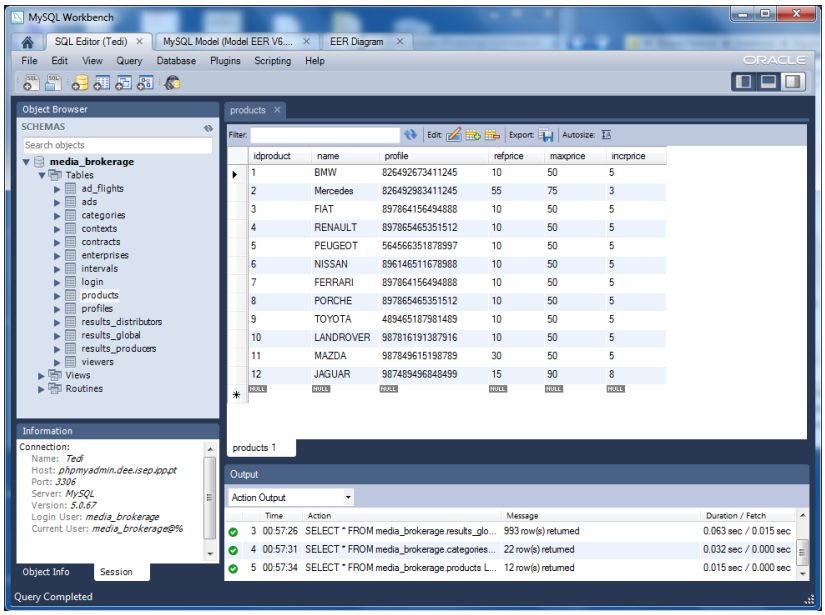


Figura 4.3: MySQL Workbench

4.1.5 Serviço de Registos UDDI

O serviço de registos UDDI escolhido foi o Apache jUDDI, tendo sido instalada a versão 2.0.1. O jUDDI foi configurado para utilizar como servidor de bases de dados o MySQL e como servidor de aplicações e serviços *Web* o Apache Tomcat. Desta forma, a base de dados de suporte ao serviço de registos UDDI, que inclui os dados das empresas e dos serviços registados, ficou alojada no MySQL e o jUDDI foi colocado na pasta `webapps` do Apache Tomcat. Com estes passos o serviço de registos UDDI ficou a funcionar como se ilustra na Figura 4.4.

4.1.6 Editor de Ontologias

Para representar o conhecimento do domínio optou-se pela especificação de ontologias, tendo sido seleccionado o editor de ontologias Protégé que permite desenvolver e guardar ontologias em diversos formatos. Instalou-se o Protégé 3.4.8 e um *plug-in* adicional o BeanGenerator 3.2 que permite converter as ontologias criadas no Protégé no conjunto de classes de Java de representação dos tipos de dados definidos [71] [72]. Estas classes são posteriormente reutilizadas pelo sistema multiagente para representar esses dados.

Um exemplo deste procedimento ocorre com a ontologia da plataforma de transacção de componentes multimédia – `BrokeragePlatformOntology` – desenvolvida. Esta ontologia inclui a definição dos serviços *Web* expostos pelos agentes da plataforma, *i.e.*, as interfaces do tipo serviço *Web* criadas. Para pu-

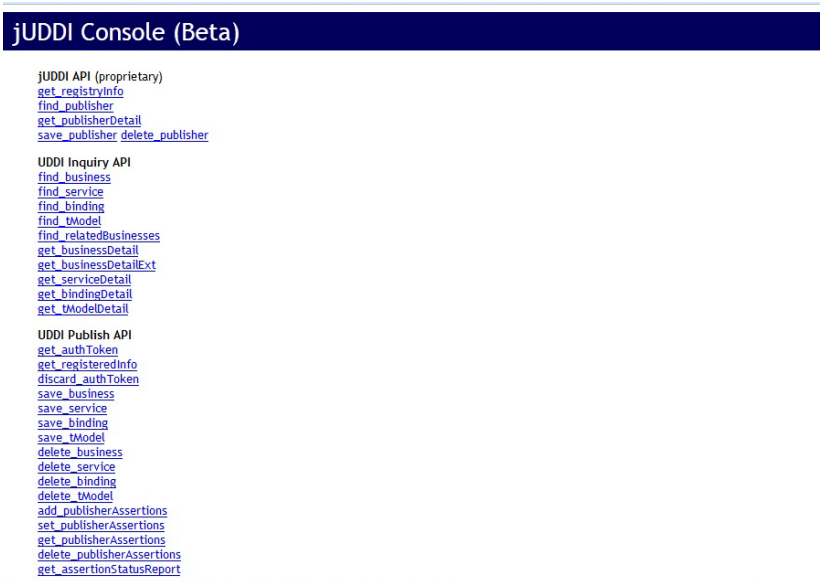


Figura 4.4: jUDDI Console

blicar estes serviços *Web* no UDDI é necessário compilar as classes da ontologia `BrokeragePlatformOntology` (`BrokeragePlatformOntology.jar`), colocar o ficheiro `BrokeragePlatformOntology.jar` na pasta "`webapps\wsig\Web-INF\lib`" do WSIG e definir no ficheiro de configuração do WSIG localizado "`\Tomcat 7.0\webapps\wsig\conf`" a localização do servidor UDDI, as credenciais da empresa que publica os serviços e, por fim, a designação da ontologia utilizada.

Para suportar o processamento de ontologias armazenadas em OWL é necessário adicionar ao NetBeans como bibliotecas da aplicação alguns *plug-ins* que se encontram na subpasta `plug-ins` do Protégé.

4.2 Resultado da Instalação

No final deste processo de instalação e configuração pode-se testar pela primeira vez o ambiente de desenvolvimento da plataforma. Na Figura 4.6 apresenta-se a interface *Web* do WSIG e a GUI do JADE.

Na Figura 4.7 apresenta-se o ambiente de desenvolvimento do NetBeans em modo de execução. A Figura 4.7 – 1 apresenta o servidor Apache Tomcat, a Figura 4.7 – 2 a plataforma JADE em modo de execução e a Figura 4.7 – 3 os `logs` do Apache Tomcat.

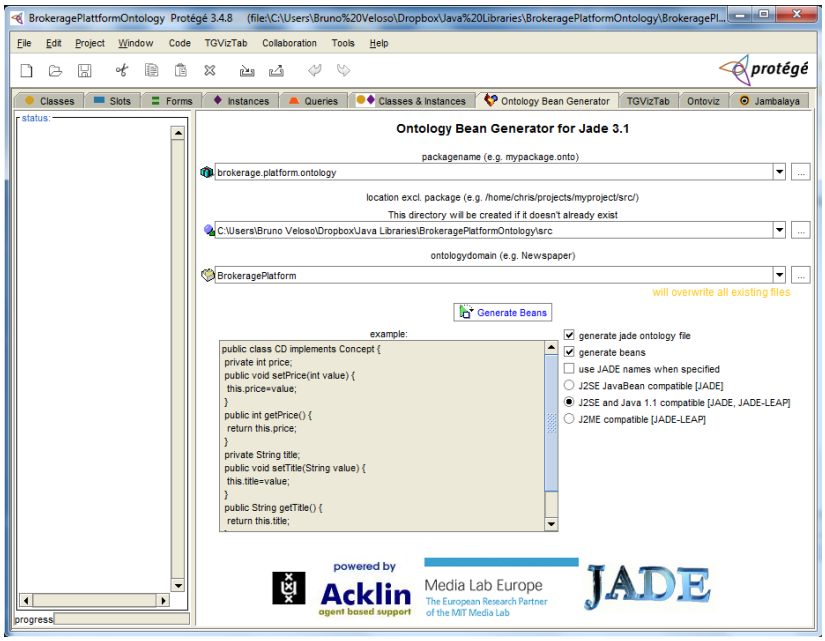


Figura 4.5: Protégé

4.3 Conclusão

A instalação do ambiente de desenvolvimento é complexa e extensa dada a multiplicidade de tecnologias de suporte incorporadas. Apresenta como vantagens recorrer exclusivamente a soluções *open source*, ser independente da plataforma porque é baseada em ambientes, ferramentas e bibliotecas desenvolvidas em Java, e assegurar um elevado nível de interoperabilidade através da adopção de interfaces do tipo serviço *Web*.

A adopção de uma plataforma Windows para o desenvolvimento apresenta a desvantagem de não permitir a criação de um *link* simbólico do ficheiro compilado da ontologia para a pasta do WSIG. Esta desvantagem é facilmente ultrapassável através da migração para uma plataforma Linux.

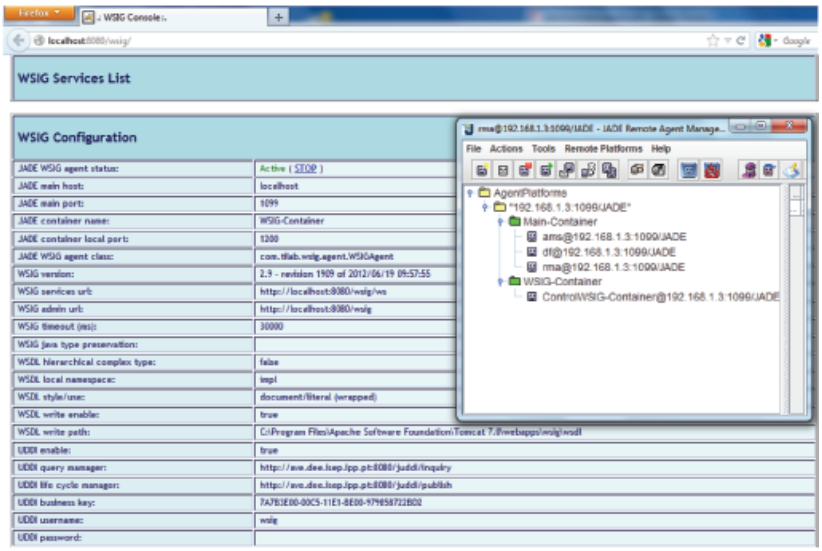


Figura 4.6: WSIG e JADE

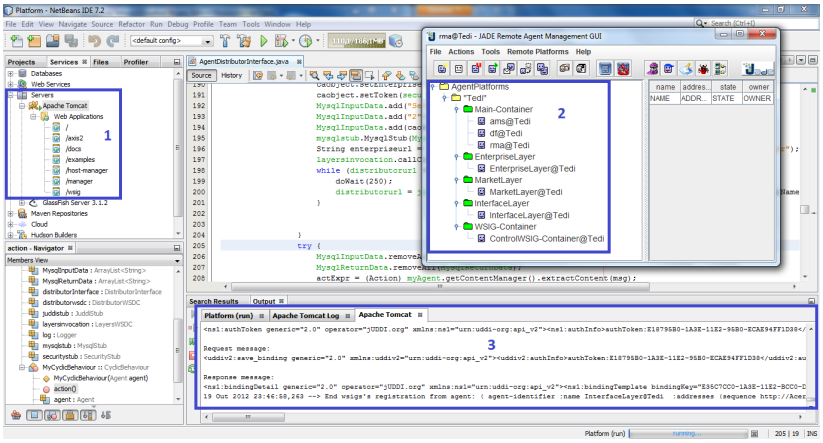


Figura 4.7: NetBeans

Capítulo 5

Desenvolvimento do Sistema

Neste capítulo descreve-se o desenvolvimento do projecto, referindo-se a representação de conhecimento, a arquitectura e as funcionalidades dos diferentes componentes (aplicações externas, camadas da plataforma e serviços de suporte).

5.1 Representação de Conhecimento

A representação do conhecimento do domínio foi efectuada através de ontologias. Foram criadas duas ontologias relativas aos anúncios e aos programas e foi completada a ontologia da plataforma com os conceitos de suporte à interface da plataforma com o exterior.

5.1.1 Ontologia da Plataforma

De forma a assegurar a integridade dos dados utilizados pela plataforma, definiu-se uma ontologia de representação da estrutura do conhecimento do domínio da plataforma – a `BrokeragePlatformOntology`. A ontologia foi criada com o Protégé e armazenada no formato Protégé Frame, gerando um ficheiro `Pont` com a ontologia. Neste projecto, a `BrokeragePlatformOntology` foi enriquecida com a representação da estrutura do conhecimento de suporte ao funcionamento da camada de interface e da interacção entre esta e a camada intermédia.

Na Figura 5.1 estão apresentados os principais conceitos da ontologia. Os conceitos representados estão agrupados por tipos de agente (`AgentType`), dados dos agentes (`AgentData`), acções dos agentes (`AgentAction`), itens multimédia completos (`MultimediaItem`) e em versão reduzida (`ReducedMultimediaItem`), produtos (`Product`), regras de inferência de (`rulesProfile`) e protocolos de negociação (`Negotiate`).

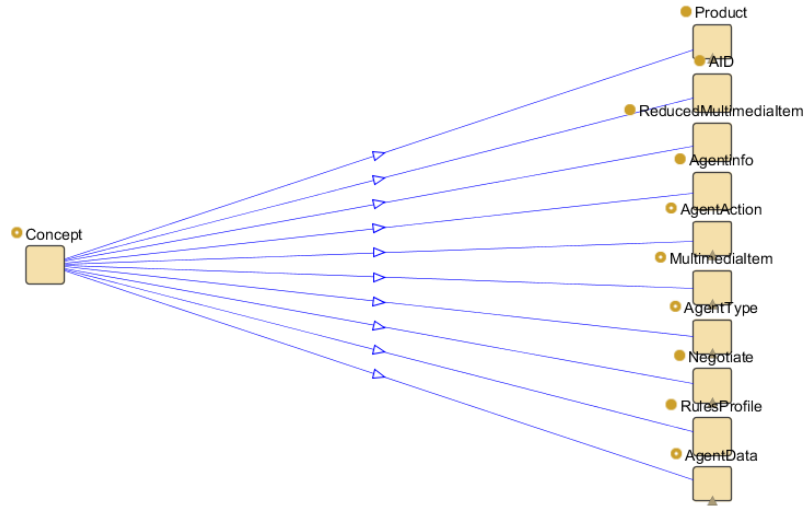


Figura 5.1: `BrokeragePlatformOntology: Concept`

O conceito `AgentType` agrupa os tipos de agente da plataforma. Os agentes foram estruturados por tipo e camada como se apresenta na Figura 5.2.

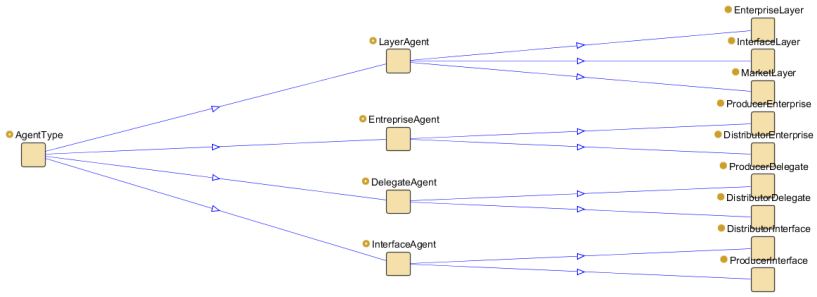


Figura 5.2: `BrokeragePlatformOntology: AgentType`

A Figura 5.3 contém a definição do conjunto das acções dos agentes, *i.e.*, as acções dos serviços *Web* que os diferentes tipos de agente expõem. Este conjunto está agrupado sob o conceito genérico `AgentAction`.

Na Figura 5.4 apresentam-se os conceitos representados e processados pelos agentes e que estão agrupados sob o conceito genérico `AgentData`.

O conceito item multimédia – `MultimediaItem` – agrupa todo o tipo de componente multimédia, incluindo imagens, vídeos e áudios. Anúncios e intervalos são itens multimédia. A Figura 5.5 apresenta a estrutura deste conceito.

Por fim, na Figura 27 do Anexo D apresenta-se uma vista completa da ontologia da plataforma.

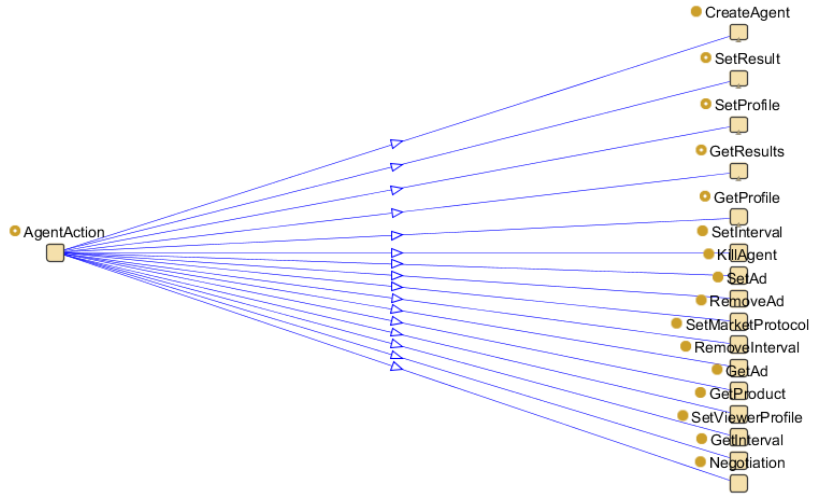


Figura 5.3: `BrokeragePlatformOntology: AgentAction`

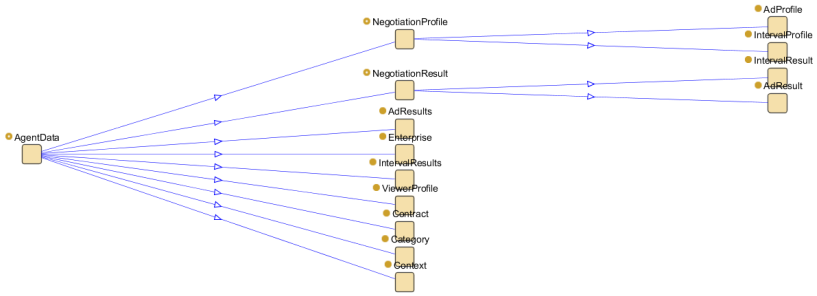


Figura 5.4: `BrokeragePlatformOntology: AgentData`

5.1.2 Ontologia dos Programas

A ontologia dos programas inspirou-se na estrutura de classificação dos programas adoptada pelo *site* da BBC [73]. A ontologia foi criada no Protégé no formato OWL. Foram seleccionadas quinze categorias principais que estão representadas na Figura 5.6. Cada categoria principal contém as sub-categorias representadas na Figura 28 do Anexo E.

5.1.3 Ontologia dos Anúncios

A ontologia dos anúncios foi baseada nas categorias dos classificados das Páginas Amarelas [74]. A ontologia foi especificada no Protégé e armazenada no formato OWL. Na Figura 5.7 apresentam-se as principais classes resultantes. Contudo, nem todas as categorias definidas nesta ontologia tem mapeamento directo para as categorias da ontologia dos programas. Na Figura 29 do Anexo F apresenta-se

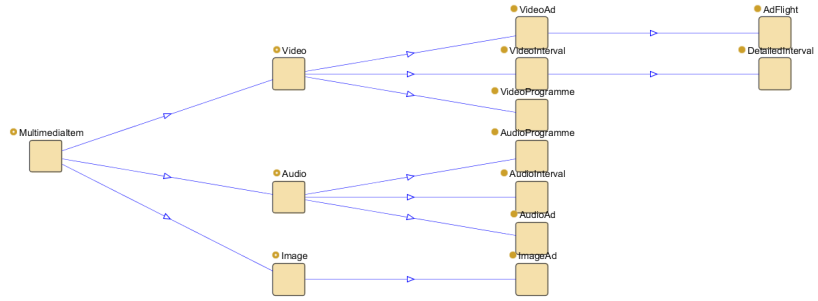


Figura 5.5: `BrokeragePlatformOntology: MultimediaItem`

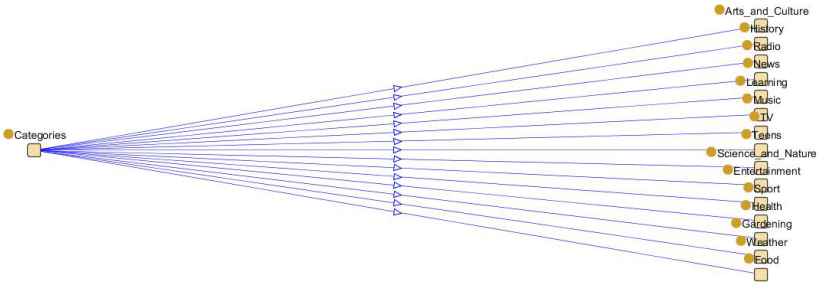


Figura 5.6: `BBC: Categories`

esta ontologia.

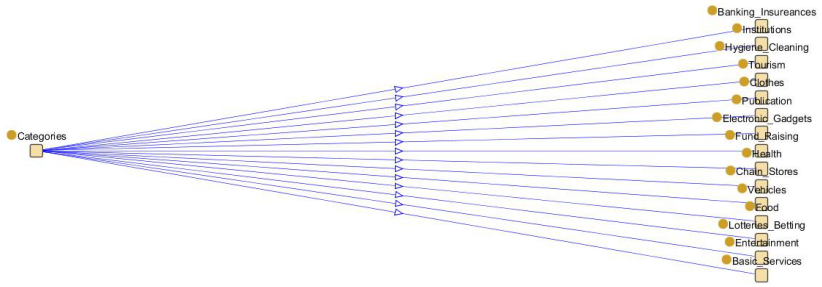


Figura 5.7: `Ads: Categories`

5.1.4 Ontologia dos Espectadores

Esta ontologia é similar à ontologia dos programas porque o perfil do espectador é dinamicamente construído com base no histórico de visualização de conteúdos. Na página 41 apresenta-se na Figura 5.11 a interpretação gráfica do perfil de um espectador.

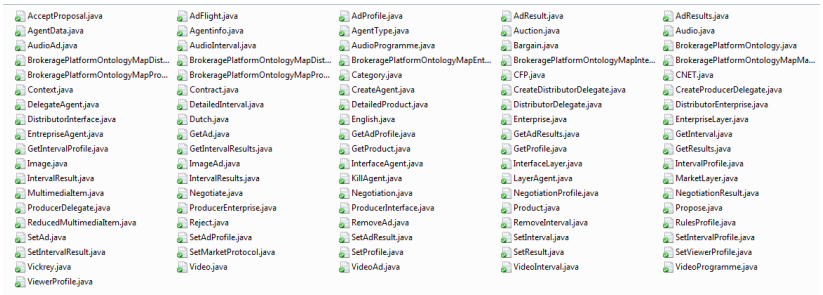


Figura 5.8: Classes geradas pelo BeanGenerator

5.1.5 Aplicação das Ontologias

A ontologia da plataforma, que se encontra armazenada no formato Protégé Frame, é convertida no conjunto de classes de Java que representam a informação do domínio através do recurso ao *plug-in* BeanGenerator do Protégé. A Figura 5.8 lista as noventa e uma classes resultantes deste processo. A plataforma utiliza este conjunto de classes para representar o conhecimento do domínio. A classe mais importante do conjunto é a `BrokeragePlatformOntology` que especifica os conceitos da ontologia, o vocabulário utilizado e os resultados da invocação das diferentes acções. A Figura 5.9 apresenta um excerto da classe `BrokeragePlatformOntology`.

As restantes ontologias, dado que estão no formato OWL, são directamente processáveis pela plataforma.

5.2 Arquitectura

A arquitectura foi desenvolvida tendo em conta quatro objectivos: (*i*) a modularidade do sistema, *i.e.*, a facilidade de incluir novos protocolos, tácticas ou métricas; (*ii*) a interoperabilidade do sistema, *i.e.*, a adopção de interfaces padrão independentes da plataforma ou linguagem; (*iii*) a escalabidade do sistema, *i.e.*, a capacidade de migração futura dos agentes das empresas para plataformas internas das empresas, mantendo apenas no exterior a camada de mercado; e (*iv*) a abertura e robustez do sistema, *i.e.*, a capacidade de criação e eliminação de representantes das empresas sem afectar o desempenho global, nem requer a reiniciação da platforma.

A arquitectura do sistema é composta pela plataforma de transacção de componentes multimédia e pelas aplicações externas de interface com a plataforma desenvolvidas – ver Figura 5.10. A plataforma é, por sua vez, composta por três camadas distintas: a camada de interface, a camada de modelação de empresas e o mercado. As duas aplicações externas de interface com a plataforma – a

BrokeragePlatformOntology.java

```
238     public static final String INTERFACELAYER="InterfaceLayer";
239     public static final String ADPROFILE_ADID="adId";
240     public static final String ADPROFILE="AdProfile";
241     public static final String VICKREY="Vickrey";
242     public static final String AUDIOINTERVAL="AudioInterval";
243     public static final String CONTRACT_PRODUCERINFO="producerInfo";
244     public static final String CONTRACT_CONTRACTID="contractId";
245     public static final String CONTRACT_DISTRIBUTORINFO="distributorInfo";
246     public static final String CONTRACT="Contract";
247
248   /**
249    * Constructor
250   */
251   private BrokeragePlatformOntology(){
252     super(ONTOLOGY_NAME, BasicOntology.getInstance());
253     try {
254
255     // adding Concept(s)
256     ConceptSchema contractSchema = new ConceptSchema(CONTRACT);
257     add(contractSchema, brokerage.platform.ontology.Contract.class);
258     ConceptSchema audioIntervalSchema = new ConceptSchema(AUDIOINTERVAL);
259     add(audioIntervalSchema, brokerage.platform.ontology.AudioInterval.class);
260     ConceptSchema vickreySchema = new ConceptSchema(VICKREY);
261     add(vickreySchema, brokerage.platform.ontology.Vickrey.class);
262     ConceptSchema adProfileSchema = new ConceptSchema(ADPROFILE);
263     add(adProfileSchema, brokerage.platform.ontology.AdProfile.class);
264     ConceptSchema interfaceLayerSchema = new ConceptSchema(INTERFACELAYER);
265     add(interfaceLayerSchema, brokerage.platform.ontology.InterfaceLayer.class);
266     ConceptSchema agentinfoSchema = new ConceptSchema(AGENTINFO);
267     add(agentinfoSchema, brokerage.platform.ontology.Agentinfo.class);
268     ConceptSchema imageSchema = new ConceptSchema(IMAGE);
269     add(imageSchema, brokerage.platform.ontology.Image.class);
270     ConceptSchema marketLayerSchema = new ConceptSchema(MARKETLAYER);
271     add(marketLayerSchema, brokerage.platform.ontology.MarketLayer.class);
```

Figura 5.9: Conteúdo da classe `BrokeragePlatformOntology`

aplicação *stand-alone* e a aplicação *Web* – são clientes dos serviços *Web* expostos pelos agentes da camada de interface.

5.2.1 Aplicação *Stand-alone* de Interface com a Plataforma

As empresas usam esta aplicação designada SAApp para comunicar com a plataforma e gerir os seus agentes. A comunicação da aplicação com a plataforma JADE baseia-se no consumo dos serviços *Web* criados na plataforma para o efeito. O estabelecimento desta comunicação pode ser efectuado através de três modalidades distintas: a ferramenta WSDL2Java, a API do *Java API for XML Web Services* (JAX-WS) e o *plug-in* WSDC do JADE.

A primeira abordagem consiste na criação de um cliente estático através da ferramenta WSDL2Java disponibilizada pelo AXIS2 usando o seguinte comando:

```
AXIS2HOME\bin\WSDL2Java -uri http://localhost:8080/wsig/ws/InterfaceLayer?WSDL -p -d adb
-s -o c:\build\clientinterfacelayer
```

Um dos problemas desta abordagem é a necessidade de criação de um cliente estático para cada serviço da plataforma. Este problema pode ser resolvido al-

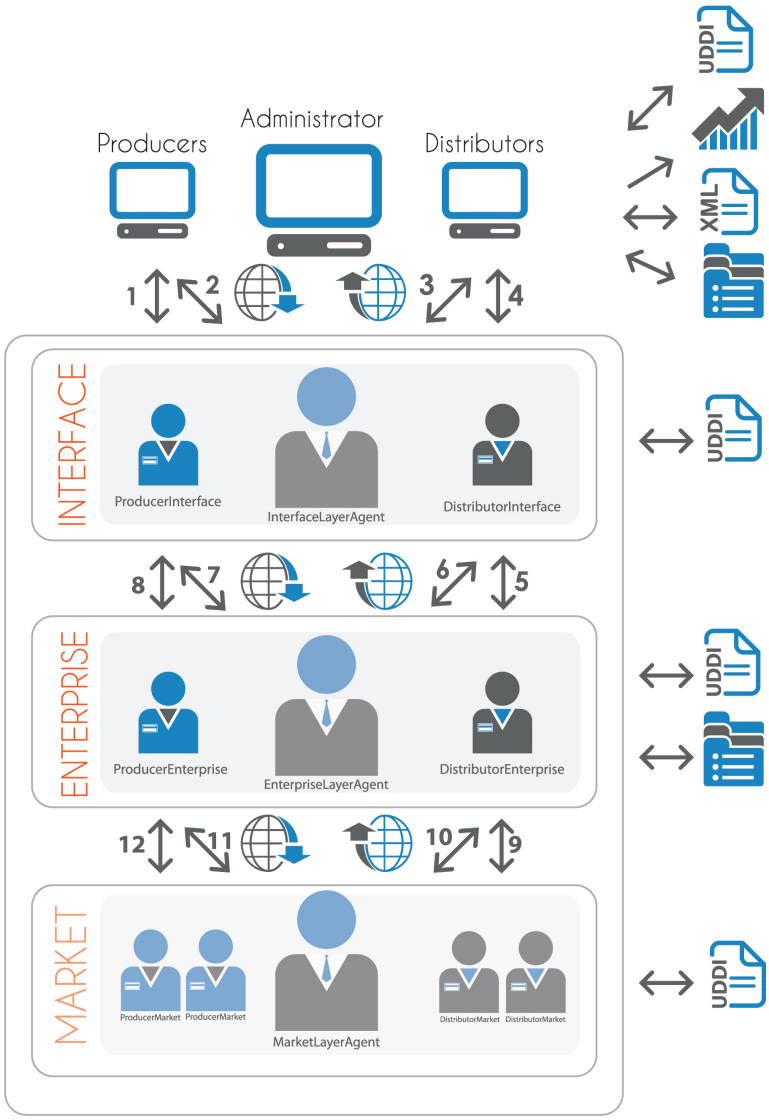


Figura 5.10: Arquitectura da plataforma

terando o código gerado pela ferramenta de forma a receber como parâmetros o URL do serviço pretendido. Esta solução só é aplicável quando se trata de interagir com agentes do mesmo tipo, *e.g.*, só com agentes distribuidores.

A segunda abordagem consiste na criação de clientes estáticos e dinâmicos suportados pelo JAX-WS. É uma abordagem padrão que requer, no caso dos clientes dinâmicos, a criação manual das mensagens a enviar aos serviços *Web*, *i.e.*, há que criar manualmente as mensagens SOAP a enviar, incluindo todos os parâmetros de entrada das acções a invocar.

Por fim, a última abordagem consiste na utilização do WSDC, que é um *plug-in* da plataforma JADE. O WSDC apenas necessita que se forneça o URL da plataforma e os objectos a enviar, encarregando-se de criar automaticamente os *stubs* e *skeletons* para a correcta comunicação com a plataforma. Em termos de segurança este *plug-in* possibilita a utilização de: (*i*) *HyperText Transfer Protocol* (HTTP) *basic authentication*; (*ii*) SSL com ou sem certificados; e (*iii*) WSS.

Foi implementada a possibilidade de carregamento do perfil de um `Ad` ou de um `Interval` através de um ficheiro XML. Utilizou-se a API JAXB para transformar o documento XML, num objecto `Ad` ou `Interval` da ontologia `MediaBrokerageOntology`. Inversamente, o JAXB transforma os objectos `Ad` ou `Interval` num documento XML. Esta API permite, de forma transparente, a conversão entre formatos de armazenamento dos dados.

A aplicação apresenta gráficos relativos a: (*i*) empresas distribuidoras – perfis dos espectadores e programas e conteúdo negociado por intervalo; (*ii*) empresas distribuidoras – perfis de anúncios, evolução dos valores de transmissão negociados por anúncio e respectivos histogramas; e (*iii*) plataforma – volume de negócios efectuado no mercado.

A Figura 5.11 – 1 apresenta o perfil de um espectador, a Figura 5.11 – 2 o perfil do programa que o espectador está a ver e, por fim, a Figura 5.11 – 3 o perfil do intervalo resultante, *i.e.*, o perfil do intervalo a preencher.

O gráfico apresentado na Figura 5.12 destina-se a empresas produtoras e descreve o comportamento de um anúncio no mercado. Apresenta a evolução do valor pago pela empresa por segundo para a transmissão do anúncio ao longo de um período de tempo especificado pelo utilizador. As barras laterais indicam o grau de cumprimento dos objectivos face ao número de visualizações alvo e face ao valor orçamentado para a campanha do anúncio.

As empresas produtoras podem ainda obter o histograma do custo da transmissão de um anúncio desde o início da campanha – ver Figura 5.13.

As empresas distribuidoras podem consultar os conteúdos dos intervalos negociados. A Figura 5.14 apresenta o alinhamento de anúncios do intervalo seleccionado após a conclusão com sucesso de uma ronda negocial.

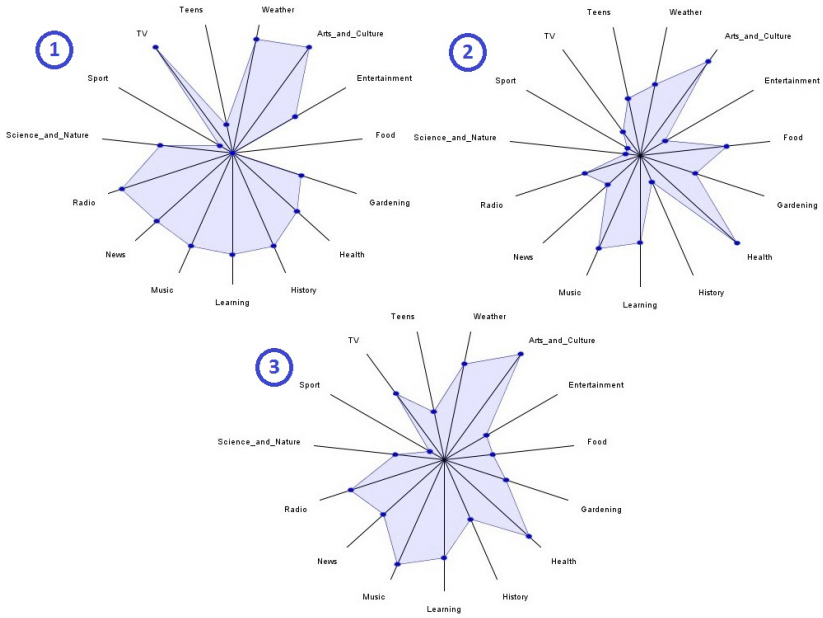


Figura 5.11: Gráficos de perfil: utilizador, programa e intervalo

A Figura 5.15 mostra o histograma do valor das transacções efectuadas no mercado no período de tempo definido pelo utilizador.

5.2.2 Aplicação *Web* de Interface com a Plataforma

A aplicação *Web* – designada WebApp – contém as mesmas funcionalidades que a aplicação de interface *stand-alone*. Contudo, durante o desenvolvimento da aplicação *Web* surgiu um problema inesperado: o JSF não suporta o *plug-in* WSDC. Não querendo abdicar do dinamismo proporcionado pelo *plug-in* WSDC, optou-se por se desenvolver um módulo servidor adicional que serve de *proxy* entre a WebApp e a plataforma MultimediaBrokerage.

Este módulo servidor, desenvolvido em linguagem Java, utiliza o protocolo de transporte TCP para receber objectos da aplicação *Web*, que comportam a informação necessária para interagir com o *plug-in* WSDC, e devolver instâncias da ontologia da plataforma. O servidor, quando recebe um pedido da aplicação *Web*, invoca o serviço *Web* correspondente, utilizando o *plug-in* WSDC.

Na Figura 5.16 representa-se o funcionamento do ProxyServer. A seta bidireccional número 1 assinala a interacção entre a aplicação *Web* e o ProxyServer e a número dois a interacção entre o ProxyServer e a plataforma.

Na Figura 5.17 – 1 apresenta-se a página de carregamento das características dos anúncios no formato XML. Na Figura 5.17 – 2 ilustra-se o carregamento

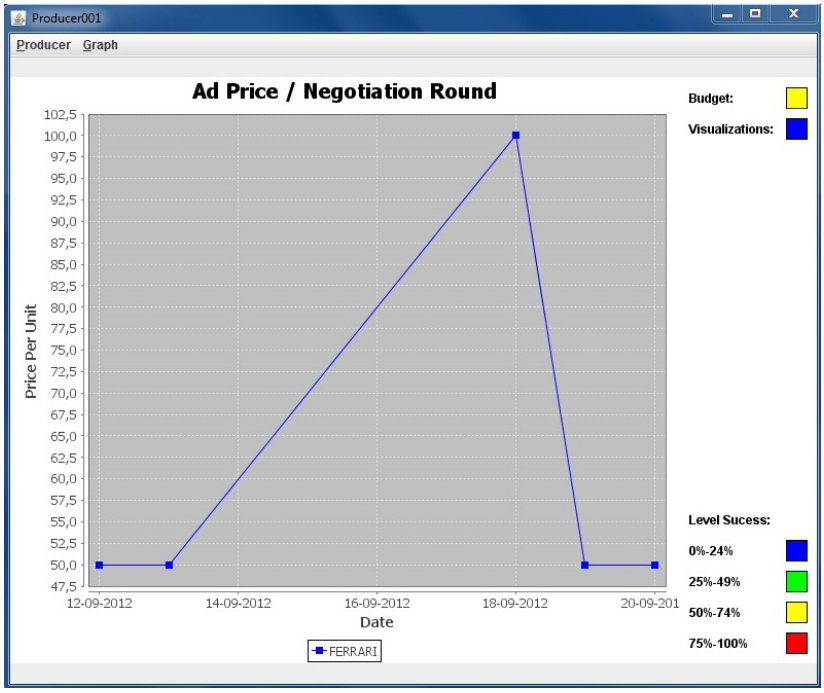


Figura 5.12: Gráfico do anúncio: evolução do custo de transmissão

assíncrono dos campos da página *Web* com os dados provenientes da base de dados.

Na Figura 5.18 – 1 apresenta-se o gráfico do perfil de um espectador e na Figura 5.18 – 2 o formulário de inserção da data, incluindo a validação da data por JavaScript.

5.2.3 Camada Superior de Interface

A camada superior de interface é constituída por agentes de interface que têm um único objectivo: estabelecer a ponte entre a empresa e o agente que a modela na plataforma. O agente de interface de uma empresa expõe um serviço *Web* que oferece um conjunto de acções à aplicação de interface que a empresa utiliza para gerir o seu representante na plataforma. Estes agentes utilizam *ontology mappers* para suprimir todas as acções definidas na ontologia da plataforma que não pretendem disponibilizar. O Extracto de Código 5.1 apresenta um excerto da classe de um *ontology mapper* que utiliza a anotação `@SuppressOperation` para suprimir a operação `CreateDistributorDelegate`.

Para evitar que alguma empresa tente maliciosamente controlar os agentes de outras empresas, foi implementado um mecanismo de *tokens* que verifica na base de dados a autenticidade do representante da empresa *token*. Assim, a

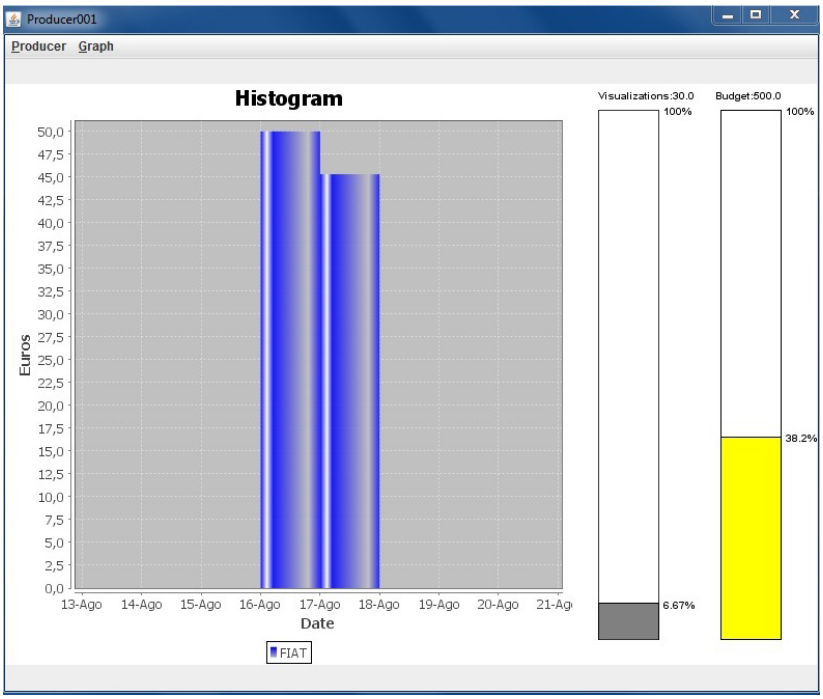


Figura 5.13: Gráfico do anúncio: histograma do custo de transmissão

Excerto de Código 5.1 Excerto de um `mapper`.

```
1     @SuppressOperation
2     public CreateDistributorDelegate toCreateDistributorDelegate() {
3      return null;
4     }
```

invocação de qualquer operação é acompanhada do envio do *token* de identificação do emissor.

5.2.4 Camada Intermédia de Modelação de Empresas

A camada intermédia – Enterprise Layer – destina-se a modelar as empresas, *i.e.*, alberga os agentes que representam as empresas. O agente da empresa armazena a informação submetida pelo utilizador, adopta o comportamento definido e reporta os resultados sempre que solicitado.

Quando se aproxima o intervalo de um espectador, o distribuidor actualiza o perfil do espectador com o perfil do programa que o espectador se encontra a ver, resultando o perfil do intervalo. Dado que as ontologias dos espectadores e dos programas são semelhantes, a determinação do vector de características resultante baseia-se no cálculo da média aritmética de cada característica dos dois vectores. A relevância de uma característica é representada por um valor

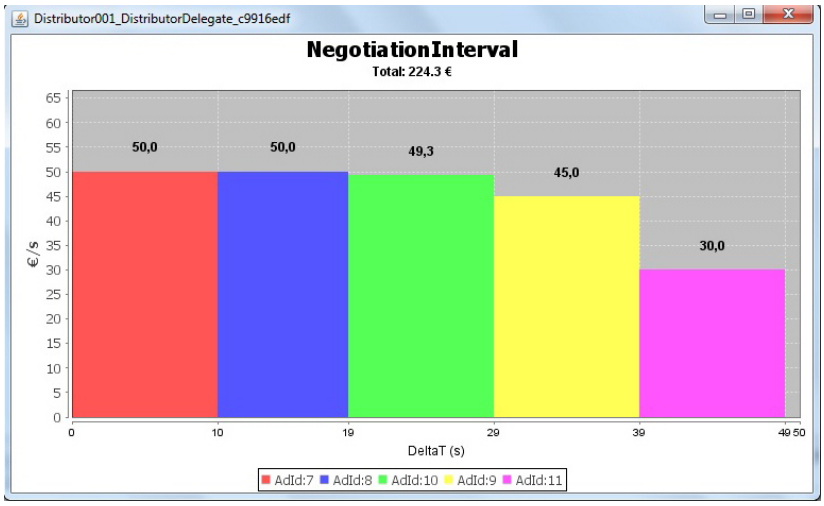


Figura 5.14: Gráfico do intervalo: alinhamento de anúncios negociados

inteiro entre 0 e 9. Por exemplo, dado o vector de características do espectador E e o vector de características do programa P, resulta o vector de características do intervalo I:

$$E = [950567777851928]$$
$$P = [826492673411245]$$
$$I = [833474675631536]$$

Os agentes distribuidores sempre que pretendem personalizar um intervalo, procuram no UDDI os produtores com anúncios compatíveis e convidam-nos a participar numa ronda negocial. Os produtores convidados, determinam a similaridade dos anúncios que possuem em carteira com o perfil do intervalo através de duas abordagens: (*i*) a comparação do valor da característica dominante do espectador no vector do anúncio; e (*ii*) a aplicação sequencial da Equação 3.1 de normalização, da Equação 3.2 da determinação da similaridade e da Equação 3.3 da distância euclidiana apresentadas na página 21. Os agentes produtores, em função dos resultados obtidos, podem ou não decidir participar na ronda negocial. Em caso afirmativo, criam um agente delegado na camada de mercado e delegam nele a responsabilidade de negociar o anúncio com similaridade e distância euclidiana mais elevadas.

A determinação da similaridade entre os perfis de um intervalo e de um anúncio baseada na determinação da similaridade dos cossenos efectua-se em três etapas. Em primeiro lugar é necessário realizar um mapeamento entre a ontologia dos anúncios (`Ads`) e a ontologia dos programas (`BBC`). Por exemplo, dado o vector de características do anúncio A e o mapeamento apresentado na Figura

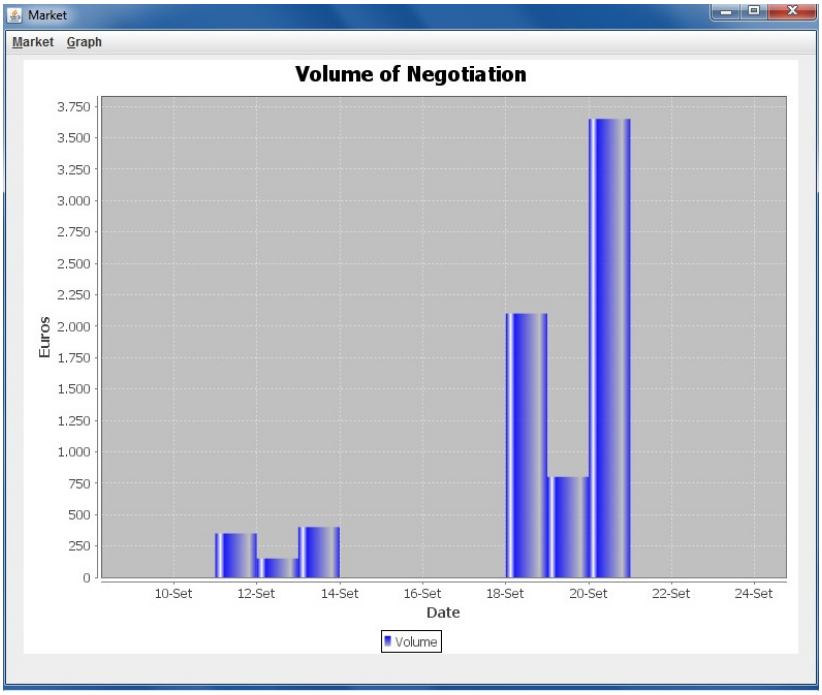


Figura 5.15: Gráfico do mercado: volume de negócios do mercado

3.2 da página 20, resulta o vector de características mapeadas do anúncio A_{map}.

$$A = [987489496848499]$$
$$A_{map} = [094060000000000]$$

O passo seguinte consiste na normalização dos vectores através da Equação 3.1.

$$\hat{A_{map}} = [0.00, 0.78, 0.34, 0.00, 0.52, 0.00, 0.00, 0.00, 0.00, 0.00, \quad (5.1)$$
$$0.00, 0.00, 0.00, 0.00, 0.00]$$
$$\hat{I} = [0.40, 0.15, 0.15, 0.20, 0.35, 0.20, 0.30, 0.35, 0.25, 0.30, \quad (5.2)$$
$$0.15, 0.05, 0.25, 0.15, 0.30]$$

Por último, aplicam-se a Equação 3.2 e a Equação 3.3 apresentadas na página 21 aos vectores A_{map} e I, obtendo-se uma distância de 1,13 e uma similaridade de 0,35 entre vectores:

$$\cos \alpha = 0,35$$
$$d = 1,13$$

Estes valores de similaridade e distância satisfazem as pré-condições da regra de inferência `Rule-04` apresentada no 5.4, resultando um nível de satisfação igual a um (1).

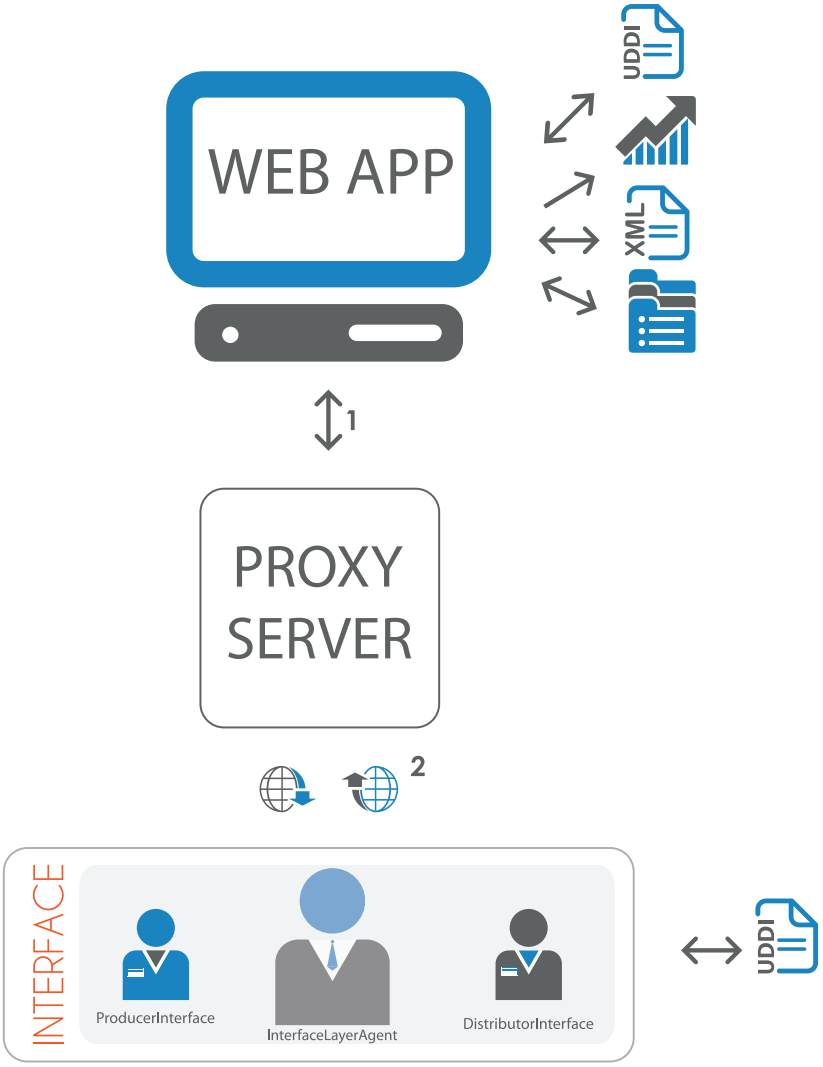


Figura 5.16: Gráfico do perfil do utilizador

A determinação da similaridade do anúncio baseada na característica dominante do intervalo efectua-se em três passos. O primeiro passo consiste na realização do mapeamento entre as ontologias do anúncio e do intervalo referido na Figura 3.2, resultando o vector A_{map}. No segundo passo identifica-se a característica dominante cd e determina-se a diferença entre os valores da característica dominante nos dois vectores $A_{map}[cd]$ e $I[cd]$ através da Equação 3.4 da página 22. Por fim, verifica-se se o resultado está dentro do intervalo dado pela Equação 3.5 da página 22 . Em caso afirmativo, é atribuído um nível de satisfação obtido através das regras de inferência definidas.

Dado o perfil do anúncio mapeado A_{map} e o perfil do intervalo I apresentados,

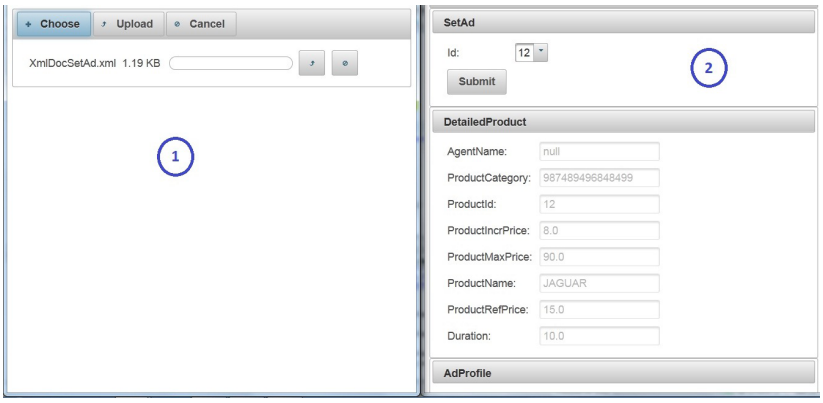


Figura 5.17: WebApp: `SetAd`

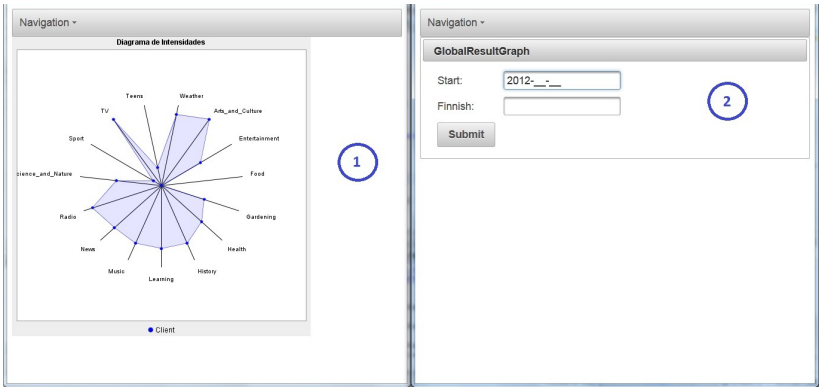


Figura 5.18: WebApp: gráficos

Excerto de Código 5.2 Regra de inferência 04.

```
1    rule "Metric-2 Rule-04"
2    when
3        r24:Metric ((metricType=='2') && similarity<similarityLevel && distance>distanceLevel)
4    then
5        r24.setSatistiedConditionsLevel('1');
6        System.out.println("Executed Rule: Metric-2 Rule-04
7                        \nSatisfiedConditionLevel: "+r24.getSatistiedConditionsLevel());
8    end
```

constata-se que o anúncio não possui a característica dominante do intervalo.

$$I = [833474675631536]$$
$$A_{map} = [094060000000000]$$
$$cd = 0$$
$$I[cd] = 8$$
$$A_{map}[cd] = 0$$
$$d_{cd} = 8$$

Assim, a diferença da característica dominante encontra-se fora do intervalo de interesse da Equação 3.5, resultando num nível de satisfação igual a zero (0).

Por omissão, o agente distribuidor tem um intervalo com um perfil indefinido (`NA`). Quando a empresa adiciona ao seu agente um novo `viewerprofile`, o agente determina o perfil do novo intervalo, calculando a média do perfil do espectador e do perfil do programa, e aplica-o ao intervalo indefinido. De seguida, especifica quatro parâmetros: o algoritmo de similaridade a utilizar ($\{1, 2\}$, onde 1 representa a característica dominante e 2 representa a similaridade dos cossenos), o nível de satisfação aceite ($\{1, 2, 3, 4\}$), o nível de referência de similaridade entre vectores ($[0, 1]$) e o nível de referência da distância entre vectores ($[0, \sqrt{n}]$, onde n representa a dimensão dos vectores).

Nos testes efectuados o algoritmo de similaridade utilizado foi a similaridade dos cossenos (2), o nível de satisfação pré-acordado foi 1 (um), o que significa uma precisão baixa, o nível de referência de similaridade entre vectores foi 0,7 e, por fim, o nível de referência da distância entre vectores foi 0,8.

O distribuidor pesquisa no UDDI por produtores e desencadeia, em relação aos produtores obtidos, as acções representadas na Figura 6.4. O distribuidor anuncia o perfil do intervalo conjuntamente com os parâmetros definidos e os produtores que satisfizerem as condições estabelecidas podem responder enviando propostas relativas a anúncios de produtos compatíveis.

Os produtores aplicam o sistema de regras de inferência para determinar o nível de satisfação dos anúncios (`satisfactionlevel`) que detêm face aos parâmetros definidos pelos distribuidores. No Excerto de Código 5.3 apresenta-se a `Rule-01` que determina que se a similaridade e a distância for inferior a 0,7 e 0,8, respectivamente, é atribuído ao anúncio um nível de satisfação igual a 4.

No caso do algoritmo de similaridade escolhido ser a característica dominante (1), o distribuidor anuncia aos produtores o perfil do intervalo e a distância de referência. Os produtores mapeiam as ontologias dos programas da `BBC` e dos `Ads`, abrindo os ficheiros OWL respectivos e procurando obter uma correspondência entre categorias ou subcategorias. Por exemplo, se a categoria dominante

Excerto de Código 5.3 Regra de inferência 01.

```
1    rule "Metric-2 Rule-01"
2        when
3        r21:Metric ((metricType=='2') && similarity>=similarityLevel && distance<=distanceLevel)
4        then
5        r21.setSatisfactionConditionsLevel('4');
6        System.out.println("Executed Rule: Metric-2 Rule-01 \nSatisfactionConditionLevel: " +
7                           r21.getSatisfactionConditionsLevel());
8    end
```

do intervalo for "desporto", a ontologia dos Ads é percorrida em busca da categoria "`desporto`" e, caso não exista, é efectuada uma segunda pesquisa, desta vez pelas subcategorias. Se, por exemplo, existir uma categoria "`roupa`" e uma subcategoria "`desporto`" então é aplicado o algoritmo da característica dominante à categoria "`roupa`".

Os produtores, caso optem por participar, lançam os seus delegados na camada de mercado e, no final da ronda negocial, armazenam na base de dados da plataforma os resultados da negociação para uma estimativa global do volume de negócios.

Em todas as camadas é utilizado um conjunto de classes comum que permitem estabelecer ligações com o *plug-in* WSDC. Como a inicialização deste *plug-in* é demorada, devido à criação dos *skeletons* e dos *stubs*, cria-se apenas uma instância para cada agente. Em termos de código, este procedimento consiste na chamada do `DynamicClientCache` demonstrado no Excerto de Código 5.4.

Excerto de Código 5.4 Excerto de código da classe que utiliza o `DynamicClientCache`.

```
1    DynamicClientCache dcc;
2    DynamicClient dc;
3    dcc = DynamicClientCache.getInstance();
4    dc = dcc.get(new URI(url_wsdc));
```

Nesta camada é utilizado um sistema de inferência baseado em regras através da utilização do *plug-in* Drools. Esta abordagem permite separar a inferência do restante código.

5.2.5 Camada Inferior de Mercado

Por fim, a última camada tem como objectivo negociar os componentes multimédia. Na camada de mercado estão implementados os protocolos de negociação FIPA-ICNIP, FIPA-CNIP e *Fixed* ICNIP (FICNIP) com diferentes tácticas de adaptação de preço: linear, exponencial, aleatória, quadrática e preço fixo [1]. As mensagens trocadas durante a negociação, são mensagens que implementam a especificação FIPA-ACL, o agente delegado comunica com o agente da empresa

para obter os dados do perfil de negociação do produto ou intervalo e no final para reportar o resultado da negociação. Algumas remodelações tiveram que ser realizadas nesta camada, para a integrar.

A primeira alteração foi efectuada nas mensagens *Fixed* ICNIP (FICNIP) apresentadas na Tabela 5.1 para permitir múltiplas negociações simultâneas, passando o nome dos agentes a incluir o *id* único de cada negociação. A segunda alteração foi efectuada nos gráficos do distribuidor que mudam de aspecto dependendo do anúncio ou não do `suggestedprice`[1]. A terceira alteração foi efectuada aos agentes delegados dos distribuidores, garantindo que só terminam a execução quando preenchem na totalidade o intervalo ou quando não encontram produtores com conteúdos adequados. Por fim, já na fase de testes e depuração de erros, foi detectado um comportamento estranho no protocolo de negociação quando se utilizava a táctica aleatória. Caso a última proposta fosse inferior à melhor proposta recebida até ao momento, o agente delegado do distribuidor não enviava as mensagens ACL `AcceptProposal` ou `RejectProposal` aos delegados dos produtores. Após múltiplos testes e uma análise dos resultados com o auxílio do Java Sniffer e do *debugger* do IDE NetBeans, concluiu-se que o protocolo pressupunha que as propostas fossem sempre de valor crescente. Como a táctica aleatória não satisfaz este pressuposto, provocava o mau funcionamento do protocolo. Para solucionar este problema, o agente delegado do distribuidor passou a armazenar a melhor proposta recebida até ao momento conjuntamente com a identificação do seu proponente.

5.2.6 WSIG

O *plug-in* do JADE WSIG permite disponibilizar funcionalidades dos agentes, sobre a forma de serviços *Web*. O *plug-in* publica os serviços *Web* no servidor de aplicações e serviços Apache Tomcat. A consola do WSIG tem o aspecto referido na Figura 5.19, apresentando os serviços dos agentes de controlo das diferentes camadas. No caso do agente EnterpriseLayer, a Figura 5.19 apresenta os dados referentes ao Mapper que utiliza, a ontologia, a chave do serviço no UDDI, o endereço URL e as operações disponíveis. Para assegurar a segurança da plataforma de comércio electrónico e dada a utilização dos *plug-ins* WSDC e do WSIG existem três mecanismos: HTTP *basic authentication*, SSL e WSS. O HTTP *basic authentication* apresenta como a maior desvantagem o envio das credenciais de acesso em *plaintext*, sendo fácil a sua captura. O SSL é suficiente quando a empresa tem a sua própria plataforma JADE. Gera um certificado privado para comunicação com os seus agentes e utiliza um certificado público para comunicação com a plataforma JADE externa que contém a camada mercado. Por último, o WSS disponibiliza as mesmas funcionalidades do SSL em termos

[1] O agente delegado do distribuidor informa os delegados dos produtores do preço mínimo que aceita.

Tabela 5.1: Estrutura das mensagens FICNIP

Performativa	Conteúdo
CFP	(Negotiation :availableTime :negotiationPrice :intervalId)
PROPOSE	Negotiation :availableTime :negotiationPrice :intervalId) :negotiationId :ReducedMultimediaItem :adId :productName :duration)
ACCEPT-PROPOSAL	(Negotiation :availableTime :negotiationPrice :intervalId)
REJECT-PROPOSAL	(Negotiation :negotiationPrice :availableTime)

de ter chaves privadas e públicas e permite ainda cifrar as mensagens SOAP. No caso do SSL as chaves são utilizadas de modo síncrono enquanto no WSS de forma assíncrona. No WSS, mesmo que sejam capturadas as mensagens SOAP, como estão cifradas garantem a privacidade nos dados das empresas, motivo pelo qual este mecanismo foi o seleccionado para implementação. Para implementar o WSS é apenas necessário utilizar o Excerto de Código 5.5.

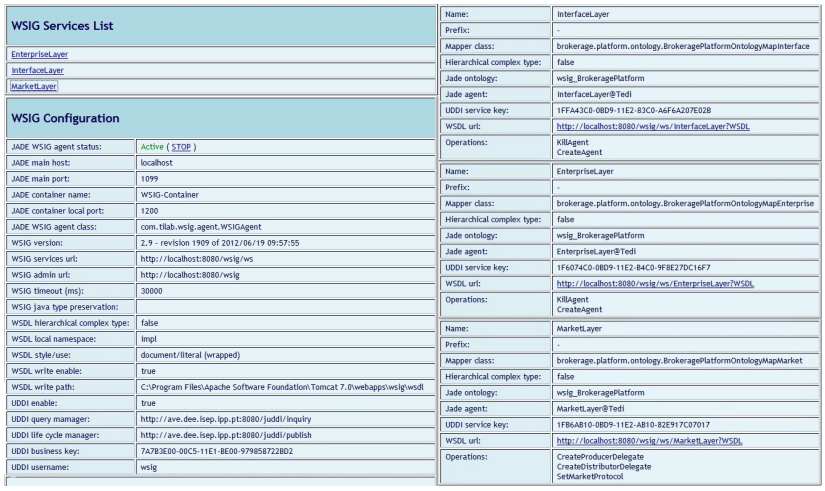
Excerto de Código 5.5 Excerto de código da classe que utiliza o WSDC.

```
1    DynamicClient dc;
2    dc.setDefaultWSSUsername("bruno");
3    dc.setDefaultWSSPassword("veloso");
4    dc.setDefaultWSSPasswordType(SecurityProperties.PW_TEXT);
5    dc.setDefaultWSSTimeToLive(6);
```

5.2.7 UDDI

Os serviços *Web* disponibilizados pelo WSIG são registados no registo UDDI. No Excerto de Código 5.6 ilustra-se a realização de um pedido SOAP para encontrar as empresas registadas no serviço de registos UDDI e a respectiva resposta no Excerto de Código 5.7.

Para se obter a lista detalhada dos serviços expostos pela plataforma especifica-



WSIG Services List

EnterpriseLayer
InterfaceLayer
MarketLayer

WSIG Configuration

JADE WSIG agent status:	Active (STOP)
JADE main host:	localhost
JADE main port:	1099
JADE container name:	WSIG-Container
JADE container local port:	1200
JADE WSIG agent class:	com.tilab.wsig.agent.WSIGAgent
WSIG version:	2.9 - revision 1909 of 2012/06/19 09:57:55
WSIG services url:	http://localhost:8080/wsig/ws
WSIG admin url:	http://localhost:8080/wsig
WSIG timeout (ms):	30000
WSIG java type preservation:	
WSDL hierarchical complex type:	false
WSDL local namespace:	impl
WSDL style/use:	document/literal (wrapped)
WSDL write enable:	true
WSDL write path:	C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\wsig\wsdl
UDDI enable:	true
UDDI query manager:	http://ave.dee.isep.ipp.pt:8080/juddi/inquiry
UDDI life cycle manager:	http://ave.dee.isep.ipp.pt:8080/juddi/publish
UDDI business key:	7A7B3E00-00C5-11E1-BE00-979858722BD2
UDDI username:	wsig

Name:	InterfaceLayer
Prefix:	-
Mapper class:	brokerage.platform.ontology.BrokeragePlatformOntologyMapInterface
Hierarchical complex type:	false
Jade ontology:	wsig_BrokeragePlatform
Jade agent:	InterfaceLayer@Tedi
UDDI service key:	1FFA43C0-0BD9-11E2-83C0-A6F6A207E02B
WSDL url:	http://localhost:8080/wsig/ws/InterfaceLayer?WSDL
Operations:	KillAgent CreateAgent

Name:	EnterpriseLayer
Prefix:	-
Mapper class:	brokerage.platform.ontology.BrokeragePlatformOntologyMapEnterprise
Hierarchical complex type:	false
Jade ontology:	wsig_BrokeragePlatform
Jade agent:	EnterpriseLayer@Tedi
UDDI service key:	1F6074C0-0BD9-11E2-B4C0-9FBE27DC16F7
WSDL url:	http://localhost:8080/wsig/ws/EnterpriseLayer?WSDL
Operations:	KillAgent CreateAgent

Name:	MarketLayer
Prefix:	-
Mapper class:	brokerage.platform.ontology.BrokeragePlatformOntologyMapMarket
Hierarchical complex type:	false
Jade ontology:	wsig_BrokeragePlatform
Jade agent:	MarketLayer@Tedi
UDDI service key:	1FB6AB10-0BD9-11E2-AB10-82E917C07017
WSDL url:	http://localhost:8080/wsig/ws/MarketLayer?WSDL
Operations:	CreateProducerDelegate CreateDistributorDelegate SetMarketProtocol

Figura 5.19: WSIG-Console: lista de serviços

Excerto de Código 5.6 Excerto de código do pedido SOAP `findbusiness`.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <find_business maxRows="100" generic="2.0" xmlns="urn:uddi-org:api_v2">
   <name> \% </name>
  </find_business>
 </soapenv:Body>
</soapenv:Envelope>
```

se a chave da empresa referente ao WSIG. O Excerto de Código 5.8 e o Excerto de Código 5.9 apresentam o pedido e a resposta da lista de serviços da plataforma.

Para se obter a lista de operações de cada serviço tem de se efectuar um pedido especificando a chave do serviço. O Excerto de Código 5.10 apresenta um pedido e o Excerto de Código 5.11 a resposta.

5.2.8 Base de Dados das Empresas

Na Figura 26 do Anexo C apresenta-se o modelo EER da base de dados do sistema que inclui a base de dados das empresas e a tabela *backup* de todas as negociações da plataforma. As relações entre tabelas são do tipo *one-to-many identifying relationship* porque um elemento *child* só pode ser identificado a partir de um elemento *parent*. As relações entre tabelas estão representadas utilizando a notação *Crow's Foot* proposta por Gordon Everest e popularizada por Clive Finkelstein, em que as caixas são as entidades e as linhas entre as caixas são as relações [75]. A base de dados das empresas é composta por treze tabelas, que estão relacionadas entre si, através de *foreign keys* para manter a integridade de dados. A informação relativa às empresas distribuidoras está armazenada nas tabelas

Excerto de Código 5.7 Excerto de código da resposta SOAP `findbusiness`.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body xmlns="urn:uddi-org:api_v2">
  <businessList generic="2.0" operator="jUDDI.org">
   <businessInfos>
    <businessInfo businessKey="7A7B3E00-00C5-11E1-BE00-979858722BD2">
     <name>JADE WSIG add-on, Inc.</name>
     <description>Web Services Interface Gateway (WSIG) is a JADE add-on</description>
     <serviceInfos>
      <serviceInfo businessKey="7A7B3E00-00C5-11E1-BE00-979858722BD2"
                   serviceKey="F0FBA530-0E37-11E2-BEB9-A2B219180914">
       <name>WSIG's businessService for EnterpriseLayer</name>
      </serviceInfo>
      <serviceInfo businessKey="7A7B3E00-00C5-11E1-BE00-979858722BD2"
                   serviceKey="F1544C80-0E37-11E2-BEB9-8015D0108308">
       <name>WSIG's businessService for MarketLayer</name>
      </serviceInfo>
      <serviceInfo businessKey="7A7B3E00-00C5-11E1-BE00-979858722BD2"
                   serviceKey="F19D8A80-0E37-11E2-BEB9-F4941D7A8A70">
       <name>WSIG's businessService for InterfaceLayer</name>
      </serviceInfo>
     </serviceInfos>
    </businessInfo>
   </businessInfos>
  </businessList>
 </soapenv:Body>
</soapenv:Envelope>
```

Excerto de Código 5.8 Excerto de código da resposta SOAP `businessdetail`.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <get_businessDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
   <businessKey>7A7B3E00-00C5-11E1-BE00-979858722BD2</businessKey>
  </get_businessDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

`categories`, `contexts`, `viewers`, `intervals` e `results_distributors`. Os dados das empresas produtoras são comportados pelas tabelas `ads`, `ad_flights`, `products` e `results_producers`. As tabelas que são comuns aos dois tipos de empresa são a `profiles`, `login`, `enterprises` e `contracts`. Na Figura 5.20 são apresentados os *routine groups* e os *stored procedures* criados. Os *stored procedures* foram agrupados de acordo com a sua finalidade: (*i*) suporte à criação de gráficos; (*ii*) armazenamento; e (*iii*) consulta de dados.

5.2.9 Base de Dados da Plataforma

Na Figura 26 da apresenta-se a tabela `results_global` que armazena o conjunto mínimo de dados que permitem à plataforma recuperar os resultados de qualquer negociação realizada. Para manter a integridade entre os identificadores das tabelas de resultados (`results_distributors` e `results_producers`) e da tabela da plataforma `results_global` foram estabelecidas *foreign keys*. Existem dois *stored procedures* relacionados com esta tabela: o `SetResultsGlobal`, que arma-

Excerto de Código 5.9 Excerto de código da resposta SOAP `businessdetail`.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body xmlns="urn:uddi-org:api_v2">
  <businessDetail generic="2.0" operator="jUDDI.org">
   <businessEntity authorizedName="WSIG Publisher"
    businessKey="7A7B3E00-00C5-11E1-BE00-979858722BD2" operator="jUDDI.org">
    <businessServices>
     <businessService businessKey="7A7B3E00-00C5-11E1-BE00-979858722BD2"
                      serviceKey="F0FBA530-0E37-11E2-BEB9-A2B219180914">
      <name>WSIG's businessService for EnterpriseLayer</name>
      <bindingTemplates>
       <bindingTemplate bindingKey="F1193F50-0E37-11E2-BEB9-A7B744B29B0A"
                        serviceKey="F0FBA530-0E37-11E2-BEB9-A2B219180914">
        <accessPoint URLType="http">http://localhost:8080/wsig/ws</accessPoint>
        <tModelInstanceDetails>
         <tModelInstanceInfo tModelKey="uuid:F0E3D770-0E37-11E2-BEB9-BBB400E5CA19"/>
        </tModelInstanceDetails>
       </bindingTemplate>
      </bindingTemplates>
      <categoryBag>
       <keyedReference keyName="fipaServiceName" keyValue="EnterpriseLayer"
                       tModelKey="UUID:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
       <keyedReference keyName="KillAgent" keyValue="KillAgent"
                       tModelKey="UUID:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
       <keyedReference keyName="fipaServiceName" keyValue="EnterpriseLayer"
                       tModelKey="UUID:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
       <keyedReference keyName="CreateAgent" keyValue="CreateAgent"
                       tModelKey="UUID:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
      </categoryBag>
     </businessService>
    </businessServices>
   </businessEntity>
  </businessDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

Excerto de Código 5.10 Excerto de código do pedido SOAP `servicedetail`.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <get_serviceDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
   <serviceKey>F0FBA530-0E37-11E2-BEB9-A2B219180914</serviceKey>
  </get_serviceDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

zena os dados de negociação, e o `Graph_globalresults`, que retorna os dados necessários para a criação do gráfico dos resultados da plataforma – ver Figura 5.20.

5.3 Descrição de Funcionamento

Na Figura 5.10 da página 39 detalha-se a interacção entre os diversos componentes desenvolvidos (agentes da plataforma e aplicações de interface externas). Toda a comunicação entre componentes é efectuada através da exposição e consumo de serviços *Web*. Neste contexto, o serviço de registo UDDI é um elemento fundamental para a pesquisa e descoberta dos serviços requeridos pelos utilizadores da

Excerto de Código 5.11 Excerto de código da resposta SOAP `servicedetail`.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body xmlns="urn:uddi-org:api_v2">
  <serviceDetail generic="2.0" operator="jUDDI.org">
   <businessService businessKey="7A7B3E00-00C5-11E1-BE00-979858722BD2"
                    serviceKey="F0FBA530-0E37-11E2-BEB9-A2B219180914">
    <name>WSIG's businessService for EnterpriseLayer</name>
    <bindingTemplates>
     <bindingTemplate bindingKey="F1193F50-0E37-11E2-BEB9-A7B744B29B0A"
                      serviceKey="F0FBA530-0E37-11E2-BEB9-A2B219180914">
      <accessPoint URLType="http">http://localhost:8080/wsig/ws</accessPoint>
      <tModelInstanceDetails>
       <tModelInstanceInfo tModelKey="uuid:F0E3D770-0E37-11E2-BEB9-BBB400E5CA19"/>
      </tModelInstanceDetails>
     </bindingTemplate>
    </bindingTemplates>
    <categoryBag>
     <keyedReference keyName="fipaServiceName" keyValue="EnterpriseLayer"
                     tModelKey="UUID:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
     <keyedReference keyName="KillAgent" keyValue="KillAgent"
                     tModelKey="UUID:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
     <keyedReference keyName="fipaServiceName" keyValue="EnterpriseLayer"
                     tModelKey="UUID:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
     <keyedReference keyName="CreateAgent" keyValue="CreateAgent"
                     tModelKey="UUID:A035A07C-F362-44dd-8F95-E2B134BF43B4"/>
    </categoryBag>
   </businessService>
  </serviceDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

plataforma.

Após o arranque, a plataforma contém os agentes responsáveis de cada camada: o InterfaceLayerAgent, EnterpriseLayerAgent e MarketLayerAgent. Os dois primeiros agentes oferecem os mesmos tipos de serviços – *Web* ver Tabela 5.2 e Tabela 5.3. A operação `CreateAgent` permite a criação dos agentes de Interface das empresas registadas e autenticadas assinalada na Figura 5.21 com as setas número dois e três. No caso da operação `KillAgent` apenas o responsável pela camada a pode invocar.

Tabela 5.2: Operações disponibilizadas pelos agentes Interface

	Operações
Producer Agent	RemoveAd SetAdProfile SetAd GetAdResults
Distributor Agent	SetViewerProfile RemoveInterval SetInterval GetIntervalResults SetIntervalProfile
Interface Layer Agent	KillAgent CreateAgent

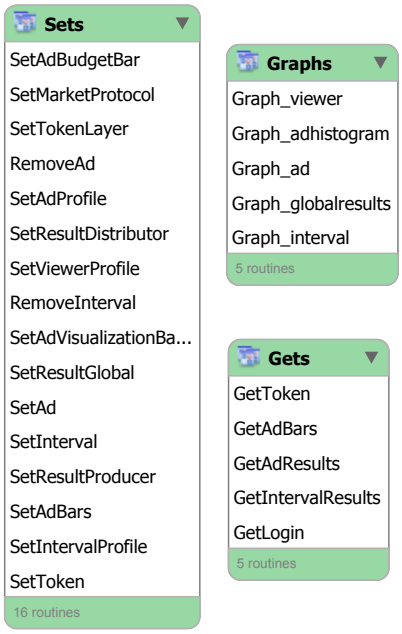


Figura 5.20: Base de dados - *Stored Procedures*

Uma vez criado o agente interface de uma empresa na camada Interface, a comunicação das aplicações externas de interface efectua-se directamente com o respectivo agente (ProducerInterface ou DistributorInterface). Caso se trate de uma empresa produtora, passa a comunicar directamente com o seu agente de interface como se ilustra através da seta número um da Figura 5.21. O agente de interface fornece quatro acções: `RemoveAd` que permite remover um `Ad` a partir do *id*, `SetAdProfile` que define o perfil de um `Ad` com um determinado *id*, `SetAd` permite definir um `Ad` e `GetAdResults` permite retornar todos os resultados da plataforma num `ArrayList` – ver Tabela 5.2. O agente de interface, caso seja a primeira vez que comunica com o seu agente Enterprise, pede ao agente EnterpriseLayerAgent para criar o agente representante da sua empresa na camada Enterprise como se ilustra com a seta número sete da Figura 5.22. A partir desse momento, os dois agentes passam a comunicar directamente como se representa através na seta número oito da Figura 5.22.

Caso se trate da primeira vez que um distribuidor se autentica na plataforma, requer ao agente InterfaceLayerAgent a criação do seu agente interface. Este pedido encontra-se representado pela seta número 3. A partir deste momento a comunicação entre a empresa e a plataforma é efectuada através do agente de interface do distribuidor – ver seta número 4 da Figura 5.21. Este agente expõe um serviço *Web* com as seguintes operações: `SetViewerProfile` permite definir um perfil de um espectador; `RemoveInterval` que remove um intervalo

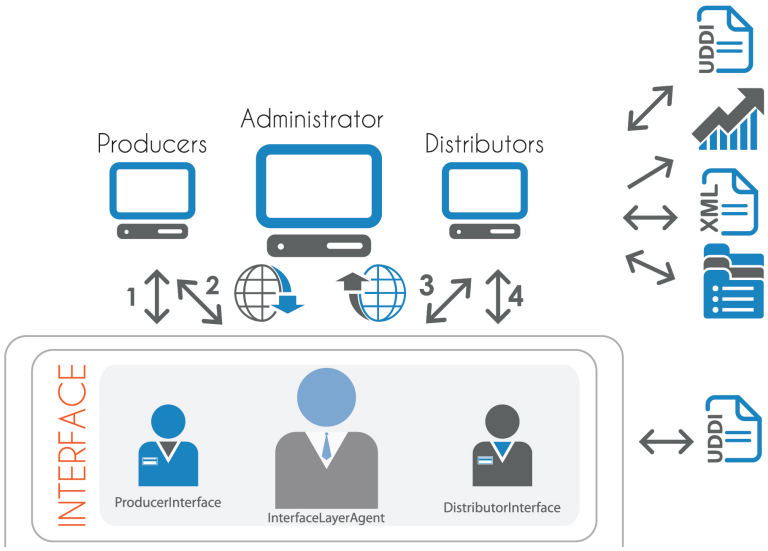


Figura 5.21: Arquitectura da plataforma (parte 1)

da lista de intervalos activos através do `id`, `SetInterval` que define um intervalo, `GetIntervalResults` que retorna um ArrayList de todos os resultados e, por fim `SetIntervalProfile` que define um perfil de um intervalo mediante a especificação do `id` do intervalo – ver Tabela 5.2.

Caso seja a primeira vez que o agente de interface comunica com a camada Enterprise, solicita ao agente responsável pela camada a criação do agente representante da sua empresa (ProducerEnterprise ou DistributorEnterprise). Este

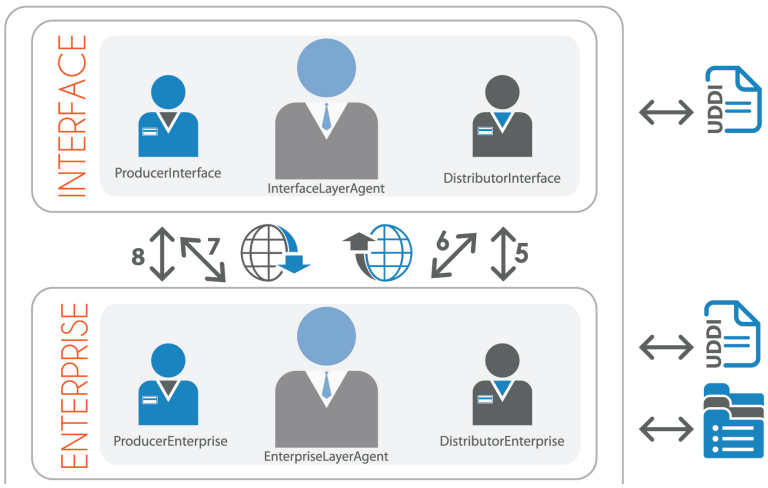


Figura 5.22: Arquitectura da plataforma (parte 2)

pedido está identificado através da seta número 6 na Figura 5.22. A partir deste momento o agente de interface passa a comunicar directamente com o agente que representa a sua empresa na plataforma – ver seta número 4 da Figura 5.22. Os agentes de interface apenas disponibilizam serviços para o exterior, *i.e.*, são intermediários na transferência de informação entre o utilizador e o agente que modela a empresa e vice-versa.

O agente ProducerEnterprise tem um conjunto de acções que disponibiliza especificamente aos seus agentes delegados da camada de mercado. Estas operações são: `GetAdProfile` para que o agente delegado obtenha o perfil de um anúncio pelo seu `id`, `SetAdResult` que guarda os resultados da negociação de um anúncio na camada de mercado, `GetAd` que permite ao agente delegado obter as características do anúncio que vai negociar e, por fim, `GetProducts` para devolver aos distribuidores a lista de anúncios de produtos que satisfazem os requisitos especificados. Na Tabela 5.3 listam-se as acções disponíveis.

Tabela 5.3: Operações disponibilizadas pelos agentes Enterprise

	Operações
Producer Agent	GetAdProfile
	RemoveAd
	SetAdProfile
	SetAdResult
	SetAd
	GetAd
	GetAdResults
	GetProduct
Distributor Agent	GetIntervalProfile
	SetViewerProfile
	SetIntervalResult
	RemoveInterval
	SetInterval
	GetIntervalResults
	GetInterval
	SetIntervalProfile
Enterprise Layer Agent	KillAgent
	CreateAgent

Se o agente ProducerEnterprise responder a um pedido `GetProducts` de um distribuidor enviando as características de um anúncio de um produto, cria automaticamente um agente delegado na camada mercado para negociar o anúncio em questão. Este procedimento consiste no pedido de criação de um agente delegado com o `id` do anúncio que vai ser negociado ao agente MarketLayer – assinalado pela seta número onze da Figura 5.22. De seguida o agente delegado na camada de mercado solicita, através da operação `GetAd`, a informação relativa ao anúncio - assinalado pela seta número cinco na Figura 5.22.

O agente DistributorEnterprise expõe um conjunto de acções que disponibiliza especificamente aos seus agentes delegados da camada de mercado: `SetIntervalResult` que guarda os resultados da negociação de um intervalo provenientes da camada de mercado, `GetIntervalProfile` permite ao agente delegado obter o perfil do intervalo mediante o `id` e, por fim, o `GetInterval` fornece ao agente delegado as características do intervalo a negociar. O agente DistributorEnterprise solicita a criação de um agente MarketLayer assinalada com a seta número dez na Figura 5.23. Uma vez criado o delegado, a comunicação passa a ser directa como se representa através da seta número doze da Figura 5.23.

O agente responsável pela camada de mercado fornece três acções: `CreateProducerDelegate` que permite criar um agente delegado de um produtor, `CreateDistributorDelegate` que cria um agente delegado de um distribuidor e `SetMarketProtocol` que define o protocolo da camada de mercado - ver a Tabela 5.4.

Tabela 5.4: Operações disponibilizadas pelos agentes Market

	Operações
Market Layer Agent	CreateProducerDelegate CreateDistributorDelegate SetMarketProtocol

Os agentes delegados da camada de mercado comunicam entre si utilizando o *Fixed* ICNIP (FICNIP) para negociar o alinhamento de anúncios dos intervalos

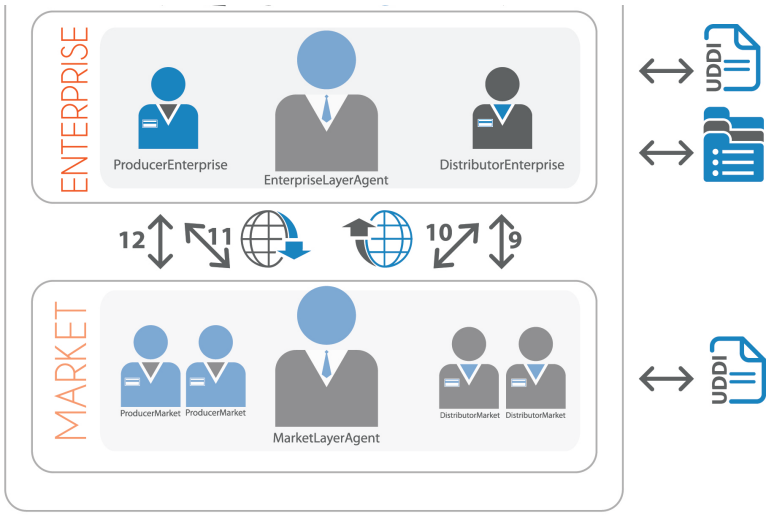


Figura 5.23: Arquitectura da plataforma (parte 3)

e apresentam os respectivos gráficos de negociação - ver Figura 6.6 e Figura 6.7.

5.4 Conclusão

A plataforma permite carregar e descarregar dados das empresas através das aplicações de interface desenvolvidas e dos agentes da camada de interface. Internamente, os agentes produtores e distribuidores encontram-se através de pesquisas no UDDI, interagem consumindo os serviço *Web* criados e negoceiam no mercado através da criação de agentes delegados de granularidade fina.

A selecção dos anúncios a negociar é efectuada pelos produtores com base no mecanismo de emparelhamento desenvolvido.

No capítulo seguinte detalha-se o conjunto de testes efectuado às funcionalidades assim como ao desempenho do sistema e analisam-se os resultados obtidos.

Capítulo 6

Testes e Resultados

Neste capítulo descrevem-se os testes de funcionamento da plataforma e de desempenho do sistema e, por último, analisam-se os resultados obtidos.

6.1 Funcionalidades da Plataforma

O teste das funcionalidades da plataforma encontra-se organizado em três cenários: (*i*) teste da selecção de anúncios; (*ii*) teste das tácticas de negociação; e (*iii*) teste do protocolo de negociação. O primeiro cenário ilustra a selecção do anúncio que os agentes produtores fazem quando são convidados por um distribuidor a negociar. O segundo cenário envolve um distribuidor e cinco produtores com anúncios compatíveis com o intervalo de um espectador. Os vários produtores dispõem-se a pagar valores mínimos e máximos idênticos pela transmissão dos respectivos anúncios, mas possuem tácticas de negociação distintas. No terceiro cenário lançam-se na plataforma um distribuidor e dois produtores. Cada produtor possui um anúncio compatível com o perfil do espectador. Os dois produtores dispõem-se a pagar valores máximos e mínimos diferentes pela transmissão dos respectivos anúncios e possuem a mesma táctica de negociação.

A plataforma arranca apresentando os agentes InterfaceLayer, EnterpriseLayer e MarketLayer de controlo das camadas e os agentes de suporte da plataforma JADE – ver a Figura 6.1 – 1.

Existem duas possibilidades de adicionar empresas à plataforma desenvolvida. A primeira consiste na utilização da aplicação GUI *stand-alone* e a segunda na WebApp. Independentemente da aplicação escolhida, as funcionalidades da interface disponibilizadas são as mesmas. O primeiro passo consiste na autenticação do utilizador. A autenticação com sucesso do utilizador permite criar na plataforma representantes de empresas produtoras, distribuidoras ou mercados. O

pedido de criação do conjunto dos representantes de uma empresa na plataforma desencadeia o processo referido na Secção 5.2.3. O conjunto de operações *Web* envolvidas dependem do tipo de empresa e estão apresentadas na Tabela 5.2. O segundo passo consiste no envio para a plataforma dos dados referentes aos anúncios e intervalos conforme se trate de produtores ou distribuidores. Existem duas modalidades de realização desta tarefa: (*i*) a invocação do serviço *Web* do agente de interface correspondente, passando-lhe o conjunto de dados pretendido; e (*ii*) o carregamento de um ficheiro XML com os dados. Neste segundo caso, o ficheiro XML é interpretado e os seus dados são mapeados através da utilização do JAXB para objectos da ontologia MultiMediaBrokerage que, de seguida, são transmitidos utilizando a primeira modalidade.

6.1.1 Teste de Selecção de Anúncios

Este teste foi definido de forma a avaliar as duas métricas de selecção de anúncios implementadas: o emparelhamento baseado na comparação da característica dominante e o emparelhamento baseado no cálculo da similaridade.

Neste teste foram criadas uma empresa produtora e uma empresa distribuidora. À empresa produtora são adicionados dois anúncios com perfis diferentes. À empresa distribuidora é adicionado um intervalo relativo a um espectador com um dado perfil e contexto (programa).

Foram utilizados os dados de autenticação apresentados na Tabela 6.1 e os perfis de anúncio da Tabela 6.2.

Tabela 6.1: Cenário 1: dados de autenticação

Tipo de Empresa	Nome da Empresa	Nome do Utilizador	Senha
Producer	Prod001	marco	veloso
Distributor	Dist001	paulo	veloso

Tabela 6.2: Cenário 1: características dos anúncios

Anúncio	Perfil
Ferrari	897864156494888
Jaguar	987489496848499

A Tabela 6.3 contém as características do intervalo a preencher.

Tabela 6.3: Cenário 1: características do intervalo

Duração (s)	Empresa	Perfil	Preço Ref. (€)	Preço Max. (€)
10	Dist001	415321202322464	25	90

Na Tabela 6.4 apresentam-se o perfil e contexto do espectador. Este cenário

Tabela 6.4: Cenário 1: perfil e contexto do espectador

Empresa	Perfil	Canal	Programa	Perfil Programa
Dist001	104351267334794	Discovery	MythBusters	826492673411245

foi utilizado para testar as duas métricas.

Da aplicação da Equação 3.1 e da Equação 3.2 aos dois anúncios da empresa produtora resultam os valores de similaridade apresentados na Tabela 6.5. A empresa produtora face a estes resultados escolhe negociar o anúncio da Ferrari.

Tabela 6.5: Cenário 1: resultados da determinação da similaridade

	Similaridade	Distância	Nível Satisfação
Ferrari	0.431	1.066	1
Jaguar	0.379	1.114	1

Da comparação entre o valor da característica dominante do intervalo e os valores correspondentes dos anúncios da empresa produtora, resultam os valores apresentados na Tabela 6.6. Neste caso, a empresa produtora escolhe negociar o anúncio da Jaguar.

Tabela 6.6: Cenário 1: resultados da comparação da característica dominante

	Viewer Dom. Char.	Ad Dom. Char.	Nível Satisfação
Ferrari	7	6	2
Jaguar	7	8	3

Se da aplicação das métricas resultasse um empate entre anúncios proceder-se-ia à escolha aleatória de um anúncio.

As duas métricas obtiveram resultados diferentes. A similaridade dos cossenos é mais robusta do que abordagem baseada na característica dominante devido a inexistência de um histórico que permita representar adequadamente as características do espectador.

6.1.2 Teste de Tácticas de Negociação

Criam-se seis instâncias de agentes de interface correspondentes a uma empresa distribuidora e cinco empresas produtoras com os dados de autenticação e de empresa, apresentados na Tabela 6.7.

Os agentes de interface encarregam-se, por sua vez, de solicitar ao agente responsável pela segunda camada a criação dos respectivos agentes de empresa com as características definidas.

Tabela 6.7: Cenário 2: dados de autenticação

Tipo de Empresa	Nome da Empresa	Nome do Utilizador	Senha
Producer	Prod001	marco	veloso
Producer	Prod002	miguel	veloso
Producer	Prod003	tiago	veloso
Producer	Prod004	flavio	veloso
Producer	Prod005	nuno	veloso
Distributor	Dist001	paulo	veloso

Cada empresa produtora submete via agente de interface ao seu agente de empresa um anúncio com os parâmetros apresentados na Tabela 6.8.

Tabela 6.8: Cenário 2: dados dos anúncios

Produto	Ferrari	Porsche	Toyota	LandRover	Mazda
Empresa	Prod001	Prod002	Prod003	Produ004	Prod005
Preço Ref. (€)	10	10	10	10	30
Preço Max. (€)	50	50	50	50	50
Protocolo	FICNIP	FICNIP	FICNIP	FICNIP	FICNIP
Táctica	Quadratic	Exponential	Linear	Random	None
Duração (s)	10	9	10	10	10

A empresa distribuidora envia ao seu agente de empresa, através do seu agente de interface, os dados relativos ao intervalo e aos perfis do espectador e do seu contexto (programa). Esta informação é apresentada na Tabela 6.10 e na Tabela 6.9. Na Figura 6.1 – 2 apresenta-se o conjunto de agentes criados da plataforma.

Tabela 6.9: Cenário 2: dados do intervalo

Duração (s)	Empresa	Perfil	Preço Ref. (€)	Preço Max. (€)
50	Dist001	833474675631536	25	90

Tabela 6.10: Cenário 2: dados do espectador

Empresa	Perfil	Canal	Programa	Perfil Programa
Dist001	950567777851928	Discovery	MythBusters	826492673411245

Os resultados da negociação de cada anúncio são apresentados na Tabela 6.11 e a Figura 6.3 e a Figura 6.2 apresentam os respectivos gráficos. Estes resultados demonstram o correcto funcionamento do protocolo de negociação.

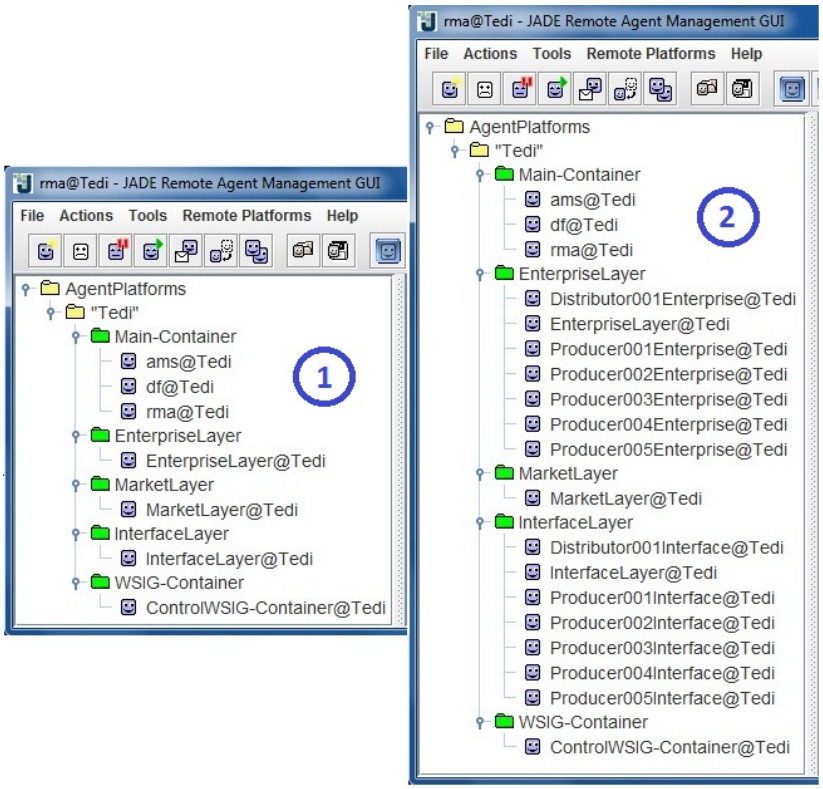


Figura 6.1: Plataforma Jade: agentes registados na plataforma.

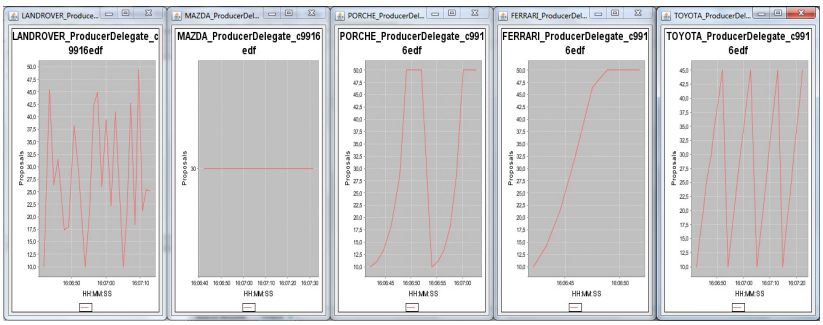


Figura 6.2: Gráficos da negociação dos produtores.

Tabela 6.11: Cenário 2: resultados da negociação

	Ferrari	Porsche	Toyota	LandRover	Mazda
Preço Negociado (€)	50.0	50.0	45.0	49.3	30.0

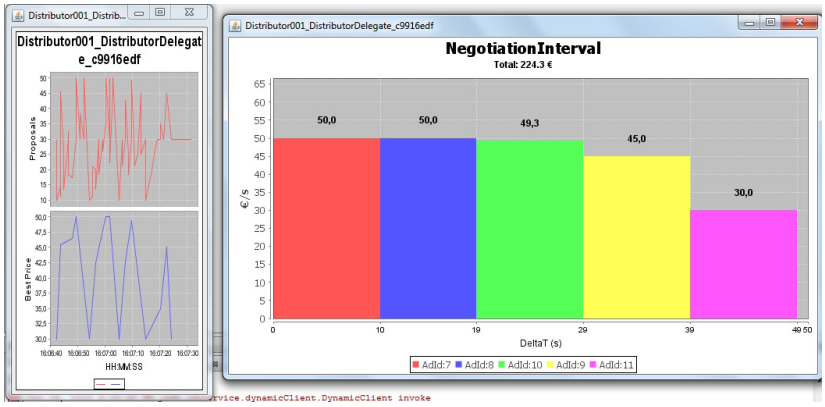


Figura 6.3: Plataforma Jade: gráficos referentes à negociação de um intervalo.

O produtor que oferece 50 € pela transmissão do seu anúncio da Ferrari é o primeiro vencedor devido à sua táctica de ajustamento de preço ser do tipo *quadratic*. Logo, o seu delegado é o primeiro que se desregista da plataforma e o primeiro a preencher parte do intervalo disponível. O segundo produtor a colocar um anúncio no intervalo é o produtor que utiliza uma táctica de negociação do tipo *exponential* e que se dispõe a pagar 50 € pela transmissão do anúncio da Porsche. A terceira táctica de negociação a conseguir ocupar uma parte do intervalo foi a do produtor do anúncio da Toyota que atingiu um preço de 45 €. O quarto produtor a conseguir uma parte do intervalo foi o que fornece o anúncio da LandRover. Este produtor adoptou uma táctica de negociação *random* oferecendo 49.3 €. Por fim o produtor com um anúncio da Mazda ocupou o quinto lugar no intervalo com um valor oferecido de 30 €.

6.1.3 Teste do Protocolo de Negociação

Foi efectuado um segundo teste com o objectivo de analisar o comportamento da plataforma com dois anúncios com a mesma táctica de negociação e com preços limite diferentes. O sistema arranca com os agentes responsáveis pelas camada InterfaceLayer, EnterpriseLayer, MarketLayer e com os agentes de suporte da plataforma JADE. De seguida, criaram-se três instâncias das aplicações externas de interface com a plataforma, especificando-se os dados de utilizador e de empresa apresentados na Tabela 6.12.

Cada produtor adicionou ao seu agente representante na plataforma um anún-

Tabela 6.12: Cenário 3: dados de autenticação

Tipo de Empresa	Nome da Empresa	Nome Utilizador	Palavra Chave
Producer	Prod001	marco	veloso
Producer	Prod002	miguel	veloso
Distributor	Dist001	paulo	veloso

cio com os parâmetros apresentados na Tabela 6.13.

Tabela 6.13: Cenário 3: dados dos anúncios

Produto	Jaguar	Porsche
Empresa	Prod001	Prod002
Preço Ref. (€)	15	10
Preço Max. (€)	90	50
Protocolo	FICNIP	FICNIP
Táctica	Exponential	Exponential
Duração (s)	10	10

O distribuidor adicionou ao seu agente representante na plataforma um intervalo e um espectador com os perfis da Tabela 6.15 e da Tabela 6.14.

Tabela 6.14: Cenário 3: dados do intervalo

Duração (s)	Empresa	Perfil	Preço Ref. (€)	Preço Max. (€)
50	Dist001	833474675631536	25	90

Tabela 6.15: Cenário 3: dados do espectador

Empresa	Perfil	Canal	Programa	Perfil Programa
Dist001	950567777851928	Discovery	MythBusters	826492673411245

O agente distribuidor procura no UDDI agentes produtores para negociar o preenchimento do intervalo e encontra cinco candidatos. De seguida, solicita-lhes anúncios com as características do intervalo, dando origem ao procedimento de selecção de anúncios descrito na secção anterior. Os cinco produtores respondem favoravelmente, lançando no mercado os seus delegados. Por sua vez, o distribuidor cria o seu delegado na camada de mercado, dando início à negociação.

Na Figura 6.5 está apresentado um diagrama de sequência que ilustra uma negociação. Neste exemplo, apresentam-se dois produtores e duas iterações no protocolo FICNIP. O primeiro passo consiste no envio pelo distribuidor de uma

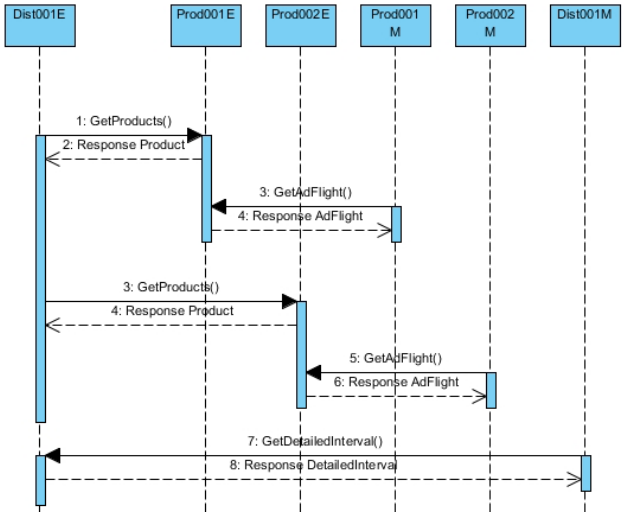


Figura 6.4: Diagrama de sequência referente à negociação (parte 1).

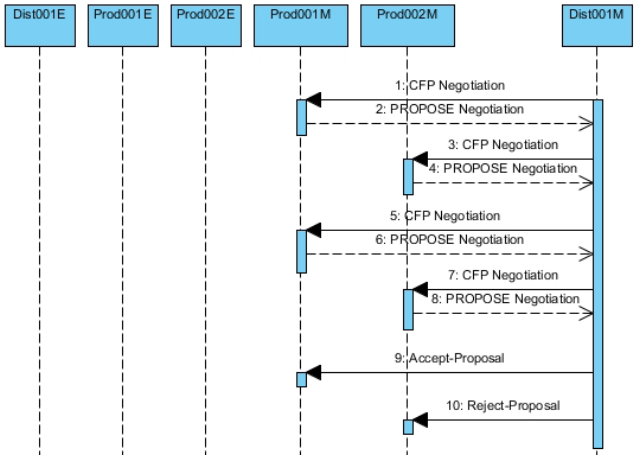


Figura 6.5: Diagrama de sequência referente à negociação (parte 2).

mensagem de CFP aos produtores. Os produtores respondem com `Proposal Negotiation` e, no final das iterações previstas, é enviado um `accept-proposal` ao vencedor da negociação e um `reject-proposal` ao produtor perdedor.

Os resultados da negociação para cada anúncio estão apresentados na Tabela 6.16 e na Figura 6.7 e na Figura 6.6 apresentam-se os respectivos gráficos. Estes resultados demonstram o correcto funcionamento do protocolo de negociação quando existem produtores com a mesma táctica e com preços limite distintos.

O produtor do anúncio da Jaguar é o primeiro a oferecer 90 €. Por fim, o produtor com o anúncio da Porsche apesar de oferecer 50 € pela transmissão

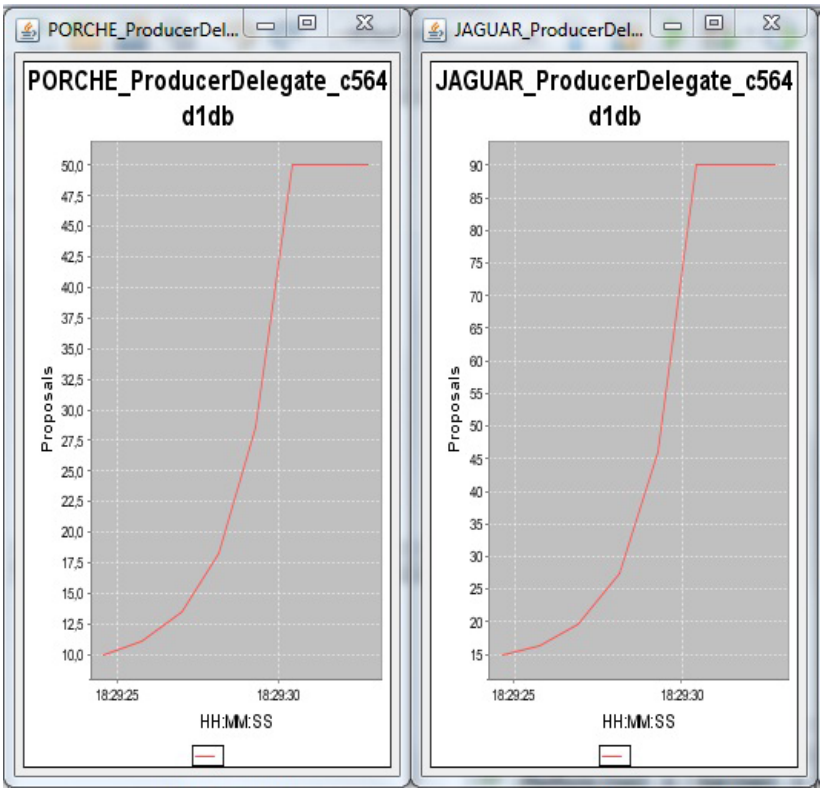


Figura 6.6: Gráficos da negociação dos produtores.

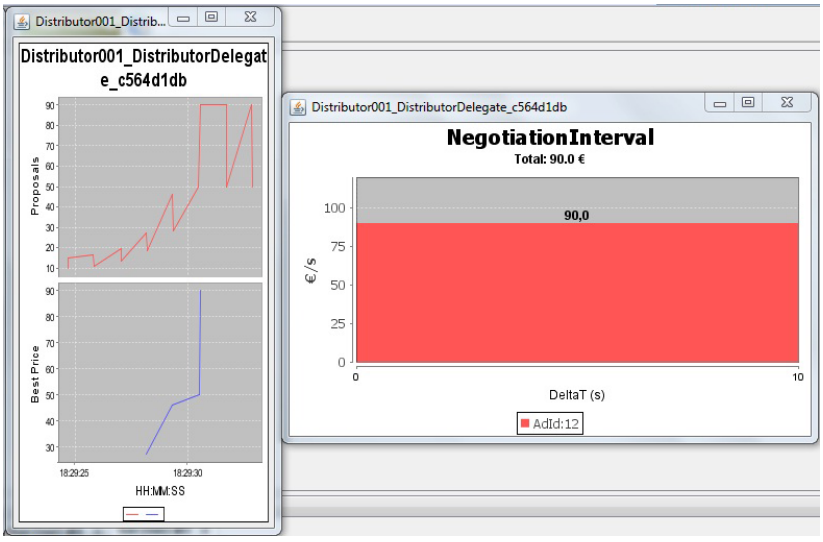


Figura 6.7: Plataforma Jade: gráficos referentes à negociação dos distribuidores.

Tabela 6.16: Cenário 3: resultados da negociação

	Preço Negociado (€)
Jaguar	90.0
Porsche	00.0

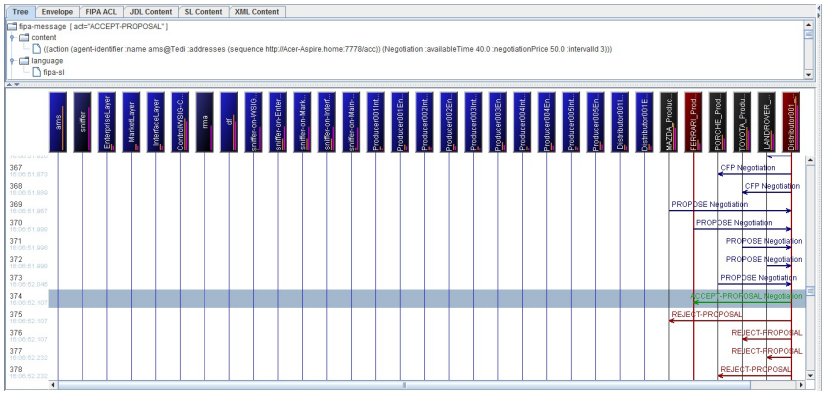


Figura 6.8: Java Sniffer: protocolo de negociação.

não conseguiu colocar o anúncio porque o intervalo apenas continha um *timeslot* disponível de 10 s.

6.1.4 Depuração

A depuração é um processo demorado e trabalhoso que se destina a encontrar e solucionar problemas de codificação. Para se realizar a depuração da plataforma adoptaram-se apenas duas ferramentas: o modo Debug do IDE NetBeans e a aplicação Java Sniffer v2.7 desenvolvida pela Rockwell Automation que se apresenta na Figura 6.8. Através destas duas ferramentas verificou-se, por um lado que os objectos da ontologia são adequados à negociação e, por outro lado, que o protocolo de negociação funciona correctamente.

Quando se procedeu à integração do código da plataforma realizado no âmbito de [1], surgiram alguns problemas. Após várias execuções com as mesmas condições iniciais, foi detectado um problema associado à táctica de negociação *random*. Se o agente com táctica *random* tiver efectuado a melhor proposta até ao momento e na última iteração efectuar uma proposta de valor inferior, as mensagens finais do protocolo não são correctamente enviadas (`accept-proposal` e `reject-proposal`). Para ultrapassar este problema foi necessário enviar uma mensagem de `reject-proposal` aos agentes bloqueados, libertando-os para continuar a negociar.

Figura 6.9: Desempenho da plataforma: *Hot spots*

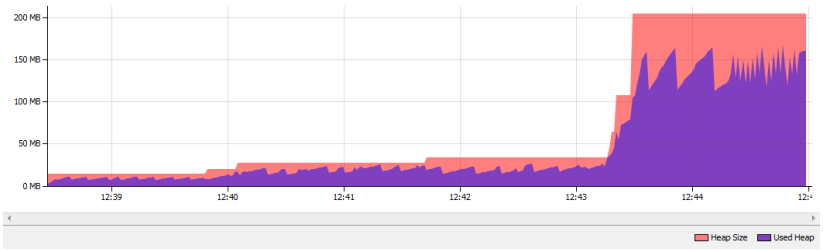


Figura 6.10: Desempenho da plataforma: JVM *Memory Heap*

6.2 Desempenho da Plataforma

Para analisar o desempenho da plataforma em tempo de execução foi criado um *profile* da plataforma. Esta operação permite analisar diversos aspectos do desempenho da plataforma. Em primeiro lugar identificaram-se os *hot spots* da plataforma. Na Figura 6.9 verifica-se que o sistema tem um *hot spot* associado ao `java.lang.Object.wait`. Este *hot spot* é provocado pelos tempos de espera introduzidos pelas seguintes operações: *(i)* registo de um agente na camada inferior no UDDI (250 ms); e *(ii)* comunicação entre agentes através do WSIG.

Em segundo lugar analisou-se o comportamento da pilha da memória (memory heap) da Java Virtual Machine. Esta análise permite detectar a existência de *memory leaks* e verificar a taxa de utilização de memória durante o curso de uma negociação envolvendo seis agentes em simultâneo. Na Figura 6.10 constatase que entre o instante 00:01:43 e o instante 00:01:45 existiu um acréscimo substancial de utilização de memória. O principal motivo foi a criação de uma JFrame/agente para apresentar o gráfico da negociação. Isto é, foram criadas um total de seis janelas seguidas de uma janela adicional contendo o gráfico do preenchimento do intervalo. Caso ocorresse um aumento da memória utilizada estando a plataforma parada, isso indicaria a existência de um *memory leak*.

Em terceiro lugar verificou-se o correcto funcionamento do Java *garbage collector*. Este teste permite verificar a eficácia na desalocação de memória de ob-

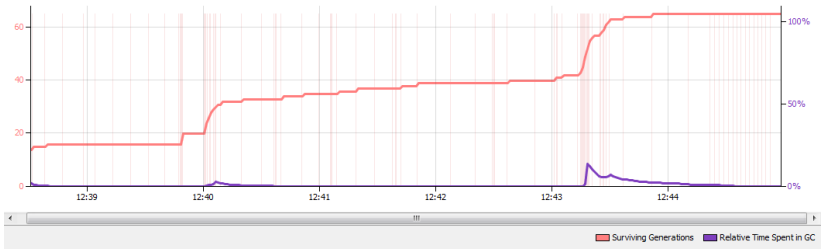


Figura 6.11: Desempenho da plataforma: *Memory Garbage Collection*

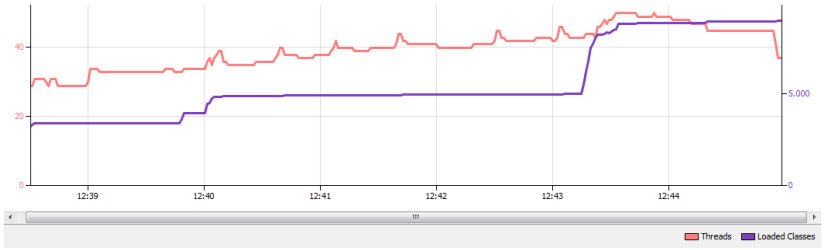


Figura 6.12: Desempenho da plataforma: *Threads/Classes*

jectos obsoletos. Em sistemas onde o número de componentes pode variar muito rapidamente, *e.g.*, de alguns agentes para várias centenas ou milhares de agentes, a correcta libertação da memória é crítica. Na Figura 6.11 verifica-se que entre o instante 00:01:43 e o instante 00:01:44 ocorre um aumento do tempo utilizado pelo *garbage collector*. Este facto coincide com a libertação da memória dos objectos criados pelo Drools para executar as regras de inferência.

Por último verificou-se o volume de *threads* criadas e de classes carregadas. Na Figura 6.12 constata-se que são carregadas mais de cinco mil classes e são criadas cerca de quarenta *threads*. No final da negociação, à medida que são eliminados os agentes delegados, verifica-se uma diminuição do número de *threads*. Na Figura 6.13 detalham-se as *threads* relacionadas com os agentes delegados da camada mercado. Como se pode observar, a maior parte do tempo as *threads* estão em modo *wait* (cor amarela), estando durante alguns momentos em modo de execução (cor verde) ou em modo de monitorização (cor vermelha).

6.3 Conclusão

Neste capítulo foram descritos três cenários de teste que ilustram o correcto funcionamento da plataforma: a selecção de anúncios, a aplicação de tácticas e a implementação do protocolo de negociação.

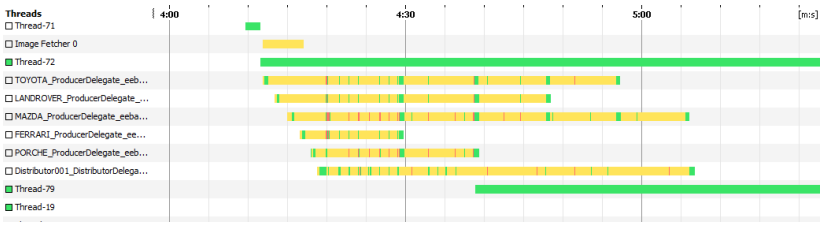


Figura 6.13: Desempenho da plataforma: *Threads*

Por último, foi efectuada uma análise ao desempenho da plataforma em relação a pontos críticos, à utilização e libertação de memória e à execução de *threads*.

No capítulo seguinte apresenta-se o balanço do projecto realizado, identificam-se as suas limitações e sugerem-se eventuais soluções.

Capítulo 7

Conclusões

Neste capítulo é apresentado o balanço do trabalho desenvolvido, os possíveis desenvolvimentos futuros e as conclusões finais.

7.1 Balanço

O balanço do projecto desenvolvido no âmbito desta tese é muito positivo dado que os objectivos propostos foram todos alcançados e, em alguns casos, até superados. Neste projecto foi adoptada uma perspectiva *top-down* uma vez que o desenvolvimento da camada de mercado estava em curso [1]. Esta estratégia funcionou bem durante dois terços do desenvolvimento. Apenas no momento da integração desta camada na plataforma surgiram questões de índole estrutural, de representação de conhecimento e ao nível do protocolo de negociação. Foi então necessário proceder a uma adaptação do código cedido para ultrapassar os problemas identificados.

Foram criadas ontologias de representação dos perfis do espectador e programas – a ontologia BBC – inspirada na classificação de programas adoptada no *site* da BBC e dos anúncios – a ontologia Ads – baseada nos classificados das Páginas Amarelas. A ontologia da plataforma foi completada com todos os conceitos de suporte à interface externa com a plataforma, à representação de perfis e aos novos tipos de agentes criados.

No âmbito da plataforma concebeu-se e implementou-se a camada de interface e enriqueceu-se a camada intermédia com o conjunto de funcionalidades de suporte ao carregamento e descarregamento de informação de e para as empresas assim como a determinação da similaridade entre perfis. A determinação da similaridade baseada na característica do intervalo é uma contribuição original deste trabalho. Na camada de modelação de empresas foi utilizado um sistema de

regras de inferência com base na API Drools que utiliza como suporte o algoritmo Rete. Esta decisão permitiu separar a lógica da inferência da similaridade entre perfis do restante código.

Por último foram desenvolvidas duas aplicações externas a SAApp e a WebApp de interface com a plataforma destinadas às empresas registadas.

7.2 Desenvolvimentos Futuros

A migração da plataforma para um serviço do tipo *Platform-as-a-Service* (PaaS) é uma evolução que aumentaria a escalabilidade e a robustez do sistema. Dado que a plataforma já utiliza interfaces do tipo serviço *Web* esta operação deverá ser transparente.

A nível das aplicações de interface, a carga e descarga de documentos XML contendo listas de anúncios, intervalos e resultados agilizaria estas operações de interface.

Apesar de estar fora do âmbito desta tese, a modelação dos espectadores dos programas e dos anúncios pode ser aperfeiçoada. As ontologias definidas para representar estas entidades deverão ser refinadas para contemplar novos aspectos. Por exemplo, a construção do perfil do espectador deverá levar em conta o histórico das interacções com a empresa distribuidora, incluindo os programas vistos (tipo e duração), os *sites Web* visitados, dados de contexto do espectador (contexto pessoal, temporal, cultural e de localização) e preferências expressas. Também os programas podem ser classificados colaborativamente pelos espectadores de forma explicita ou através de comentários nas redes sociais.

O mapeamento realizado entre a ontologia dos anúncios e a ontologia dos programas pode ser melhorado. Actualmente, efectua-se o mapeamento directo entre características das duas ontologias e o mapeamento entre características dos anúncios e subcaracterísticas dos programas. Neste último caso o mapeamento é efectuado entre a característica do anúncio e a subcaracterística correspondente com valor mais elevado. Alternativamente, poder-se-ia atribuir o valor médio de todas as subcaracterísticas correspondentes.

No contexto da determinação da similaridade entre perfis de intervalos e anúncios pode ser introduzido um mecanismo de escolha inesperada de conteúdos (*serendipity*), provocando uma sensação de novidade e de imprevisibilidade ao espectador. No caso da determinação da similaridade baseada na característica dominante, pode-se seleccionar não apenas a característica dominante mas, *e.g.*, o conjunto das cinco principais características do intervalo, permitindo escolher anúncios mais conformes com o perfil do espectador.

O agrupamento de espectadores em *clusters* permitiria libertar recursos e melhorar o tempo de resposta da plataforma através da reutilização dos intervalos entre espectadores do mesmo *cluster*.

O mecanismo de preenchimento dos intervalos suportado pela camada de mercado não garante o preenchimento da totalidade dos intervalos. Uma sugestão seria adoptar modelos genéricos de intervalo ou implementar heurísticas que assegurem o preenchimento na totalidade dos intervalos.

7.3 Conclusão

Embora se tenha obtido sucesso no cumprimento de todos os objectivos estabelecidos e superado alguns pontos, existe ainda um extenso trabalho a realizar para tornar esta plataforma mais versátil e mais completa em termos de funcionalidades.

O cumprimento integral dos objectivos estabelecidos permitiu desenvolver um protótipo funcional que prova a tese inicial. No entanto, dada a complexidade do domínio de aplicação existem múltiplas oportunidades de melhoria. A construção da plataforma de personalização de anúncios prossegue, desenvolvendo-se actualmente esforços no âmbito da criação e manutenção automática dos perfis dos utilizadores incluindo informação contextual.

Bibliografia

[1] L. M. G. V. de Sousa, "Plataforma multiagente para transacção de componentes multimédia," Master's thesis, Instituto Superior de Engenharia do Porto, Instituto Politécnico do Porto, Novembro 2012. [citado na p. 3, 49, 70, 75]

[2] P. R. Wurman, M. P. Wellman, and W. E. Walsh, "The michigan internet auctionbot: a configurable auction server for human and software agents," in *Proceedings of the second international conference on Autonomous agents*, ser. AGENTS '98. New York, NY, USA: ACM, 1998, pp. 301–308. [Online]. Available: http://doi.acm.org/10.1145/280765.280847 [citado na p. 5, 6]

[3] C. Di Napoli, C. Sierra, M. Giordano, P. Norlega, and M. Furnari, "A pvm implementation of the fishmarket multiagent system," in *ISAI/IFIS 1996. Mexico-USA Collaboration in Intelligent Systems Technologies. Proceedings*, nov. 1996, pp. 68 –76. [citado na p. 5, 6]

[4] K. Smith, *Kasbah@MIT: A Real-world Agent Mediated Electronic Marketplace*. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1998. [Online]. Available: http://books.google.pt/books?id=gPkkOAAACAAJ [citado na p. 5, 6]

[5] J. Ueyama and E. Madeira, "An automated negotiation model for electronic commerce," in *Autonomous Decentralized Systems, 2001. Proceedings. 5th International Symposium on*, 2001, pp. 29 –36. [citado na p. 6]

[6] S. Picant, F. Bourge, and A. Mouaddib, "Towards a multi-agent platform for automatic b2b exchanges," in *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, vol. 1, oct. 2010, pp. 473 –480. [citado na p. 6]

[7] Z. Dongsheng and L. Lancui, "A framework for b2b e-procurement platform based on agents and web services," in *Computational Intelligence and Soft-*

ware Engineering, 2009. CiSE 2009. International Conference on, dec. 2009, pp. 1 –5. [citado na p. 6, 7]

[8] W. Dan and M. Song, "Multi-agent systems set for b2b e-commerce systems," in *Management of e-Commerce and e-Government, 2009. ICMECG '09. International Conference on*, sept. 2009, pp. 3 –8. [citado na p. 6, 7]

[9] L. V. de Sousa, B. Malheiro, and J. Foss, "Negotiation platform for personalised advertising," in *Fifth International European Conference on the Use of Modern Information and Communication Technologies (ECUMICT 2012)*, L. de Strycker, Ed. Department of Engineering Technology of KAHO-SL, March 2012, pp. 361–373. [citado na p. 6, 7]

[10] C. Adam, V. Louis, F. Bourge, and S. Picant, "A multi-agent mediation platform for automated exchanges between businesses," in *Agent-Based Technologies and Applications for Enterprise Interoperability*, ser. Lecture Notes in Business Information Processing, K. Fischer, J. P. Müller, R. Levy, W. Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, and C. Szyperski, Eds. Springer Berlin Heidelberg, 2012, vol. 98, pp. 170–190, 10.1007/978-3-642-28563-9_10. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28563-9_10 [citado na p. 7]

[11] J. D. Foss, B. Malheiro, and J.-C. Burguillo, "Personalised placement in networked video," in *Proceedings of the 21st international conference companion on World Wide Web*, ser. WWW '12 Companion. New York, NY, USA: ACM, 2012, pp. 959–968. [Online]. Available: http://doi.acm.org/10.1145/2187980.2188229 [citado na p. 7, 19]

[12] B. Malheiro, J. Foss, J. Burguillo, A. Peleteiro, and F. Mikic, "Dynamic personalisation of media content," in *Semantic Media Adaptation and Personalization (SMAP), 2011 Sixth International Workshop on.* IEEE, 2011, pp. 21–26. [Online]. Available: http://dx.doi.org/10.1109/SMAP.2011.17 [citado na p. 7]

[13] K. Mockford, "Web services architecture," *BT Technology Journal*, vol. 22, pp. 19–26, 2004, 10.1023/B:BTTJ.0000015492.03732.a6. [Online]. Available: http://dx.doi.org/10.1023/B:BTTJ.0000015492.03732.a6 [citado na p. 11]

[14] N. Nordbotten, "Xml and web services security standards," *Communications Surveys Tutorials, IEEE*, vol. 11, no. 3, pp. 4 –21, quarter 2009. [citado na p. 11]

[15] B. McLaughlin, *Java & XML Data Binding*, ser. Java Series. O'Reilly Media, Incorporated, 2002. [Online]. Available: http://books.google.pt/books?id=Y_PHyCK5d6EC [citado na p. 12]

[16] W3C. Xml technology. [Online]. Available: http://www.w3.org/standards/xml/ [citado na p. 12]

[17] Java.net. Project jaxb. [Online]. Available: http://jaxb.java.net/ [citado na p. 12]

[18] Oracle. Java architecture for xml binding (jaxb). [Online]. Available: http://www.oracle.com/technetwork/articles/javase/index-140168.html [citado na p. 12]

[19] X. Dong, W. Tong, and J. Feng, "Using categorization to further enhance the utilization of semantic web in uddi," in *Computer and Computational Sciences, 2006. IMSCCS '06. First International Multi-Symposiums on*, vol. 2, june 2006, pp. 475 –479. [citado na p. 13]

[20] X. Ren and R. Hou, "Extend uddi using ontology for automated service composition," in *Mechanic Automation and Control Engineering (MACE), 2011 Second International Conference on*, july 2011, pp. 298 –301. [citado na p. 13]

[21] W3C. Hypertext transfer protocol – http/1.1. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html [citado na p. 13]

[22] W3C. Xml messaging specification. [Online]. Available: http://www.w3.org/TR/xmsg [citado na p. 13]

[23] W3C. Web services description language (wsdl) 1.1. [Online]. Available: http://www.w3.org/TR/wsdl [citado na p. 13]

[24] OASIS. Oasis uddi specification tc. [Online]. Available: http://www.oasis-open.org/committees/uddi-spec/faq.php [citado na p. 13]

[25] E. Newcomer, *Understanding Web Services: Xml, Wsdl, Soap, and Uddi*, ser. Independent Technology Guides. Addison-Wesley, 2002. [Online]. Available: http://books.google.pt/books?id=M1rADzqjo2cC [citado na p. 14]

[26] P. Gerstbach, "ebxml vs. web services comparison of ebxml and the combination of soap/wsdl/uddi/bpel," Online, 2009. [Online]. Available: http://classes.soe.ucsc.edu/ism211/Winter09/ebxml-ws.pdf [citado na p. 14]

[27] A. Dogac, Y. Kabak, and G. Laleci, "Enriching ebxml registries with owl ontologies for efficient service discovery," in *Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications, 2004. Proceedings. 14th International Workshop on*, march 2004, pp. 69 – 76. [citado na p. 14]

[28] Q. Wen, "Ebxml-based application integration in service-oriented business," in *Services Computing, 2006. APSCC '06. IEEE Asia-Pacific Conference on*, dec. 2006, pp. 589 –592. [citado na p. 14]

[29] A. Pereira, F. Cunha, P. Malheiro, and A. Azevedo, "Ebxml-based application integration in service-oriented business," in *Innovation in Manufacturing Networks*, vol. 266, 2008, pp. 127 –136. [citado na p. 14]

[30] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46 – 52, oct. 2003. [citado na p. 15]

[31] J.-H. Kim and C. Huemer, "From an ebxml bpss choreography to a bpel-based implementation," *SIGecom Exch.*, vol. 5, no. 2, pp. 1–11, Nov. 2004. [Online]. Available: http://doi.acm.org/10.1145/1120687.1120689 [citado na p. 15]

[32] P. A. Buhler and J. M. Vidal, "Towards adaptive workflow enactment using multiagent systems," *Information Technology and Management*, vol. 6, pp. 61–87, 2005, 10.1007/s10799-004-7775-2. [Online]. Available: http://dx.doi.org/10.1007/s10799-004-7775-2 [citado na p. 15]

[33] NAICS. North american industry classification system. [Online]. Available: http://www.statcan.gc.ca/subjects-sujets/standard-norme/naics-scian/2007/list-liste-eng.htm [citado na p. 15]

[34] UNSPSC. The united nations standard products and services code. [Online]. Available: http://www.unspsc.org/ [citado na p. 15]

[35] RosettaNet. Rosettanet technical dictionary. [Online]. Available: http://www.rosettanet.org/TheStandards/RosettaNetStandards/Dictionaries/tabid/477/Default.aspx [citado na p. 15]

[36] J. Leukel, V. Schmitz, and F.-D. Dorloff, "A modeling approach for product classification systems," in *Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on*, sept. 2002, pp. 868 – 874. [citado na p. 15]

[37] M. Hepp, J. Leukel, and V. Schmitz, "Content metrics for products and services categorization standards," in *e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on*, march-1 april 2005, pp. 740 – 745. [citado na p. 15]

[38] J. Watkinson, *The MPEG Handbook*. Taylor & Francis, 2012. [Online]. Available: http://books.google.pt/books?id=6f_gov_-CuAC [citado na p. 16]

[39] P. Angelides and H. Agius, *The Handbook of MPEG Standards and Applications*. John Wiley & Sons, 2011. [Online]. Available: http://books.google.pt/books?id=jc8EZwEuEIoC [citado na p. 16]

[40] L. Bai, S. Lao, G. Jones, and A. Smeaton, "Video semantic content analysis based on ontology," in *Machine Vision and Image Processing Conference, 2007. IMVIP 2007. International*, sept. 2007, pp. 117 –124. [citado na p. 16]

[41] S. Dasiopoulou, V. Tzouvaras, I. Kompatsiaris, and M. G. Strintzis, "Enquiring mpeg-7 based multimedia ontologies," *Multimedia Tools Appl.*, vol. 46, no. 2-3, pp. 331–370, Jan. 2010. [Online]. Available: http://dx.doi.org/10.1007/s11042-009-0387-4 [citado na p. 16]

[42] M. Wooldridge, *An Introduction to MultiAgent Systems*. Wiley, 2009. [Online]. Available: http://books.google.pt/books?id=X3ZQ7yeDn2IC [citado na p. 16]

[43] C. Panayiotou, G. Samaras, E. Pitoura, and P. Evripidou, "Parallel computing using java mobile agents," in *EUROMICRO Conference, 1999. Proceedings. 25th*, vol. 2, 1999, pp. 430 –437 vol.2. [citado na p. 17]

[44] G. Tonti, R. Montanari, J. Bradshaw, L. Bunch, R. Jeffers, N. Suri, and A. Uszok, "Automated generation of enforcement mechanisms for semantically-rich security policies in java-based multi-agent systems," in *Multi-Agent Security and Survivability, 2004 IEEE First Symposium on*, aug. 2004, pp. 11 – 20. [citado na p. 17]

[45] F. T. Communication, "Fipa contract net interaction protocol specification," FIPA TC Communication, http://www.fipa.org/specs/fipa00029/SC00029H.pdf, Standard 29, 12 2002. [citado na p. 17]

[46] F. T. Communication, "Fipa iterated contract net interaction protocol specification," FIPA TC Communication, http://www.fipa.org/specs/fipa00030/SC00030H.pdf, Standard 30, 12 2002. [citado na p. 17]

[47] D. Gosselin, *JavaScript*, ser. The Web Technologies Series. Course Technology, 2010. [Online]. Available: http://books.google.pt/books?id=wvsO38OnZ9QC [citado na p. 18]

[48] B. Brinzarea-Iamandi and C. Darie, *AJAX and PHP: Building Modern Web Applications*, ser. From technologies to solutions. Packt Pub., 2009. [Online]. Available: http://books.google.pt/books?id=aB6jWdpdcZgC [citado na p. 18]

[49] Harwani, *Java Server Faces*. Prentice-Hall Of India Pvt. Ltd., 2009. [Online]. Available: http://books.google.pt/books?id=wLbw97cDJeAC [citado na p. 18]

[50] J. Heaton, *Jstl: JSP Standard Tag Library Kick Start*, ser. Kick Start Series. Sams, 2002. [Online]. Available: http://books.google.pt/books?id=nKpvsy8mAZMC [citado na p. 19]

[51] PrimeFaces. Primefaces. [Online]. Available: http://primefaces.org/ [citado na p. 19, 26]

[52] A. Madylova and S. Oguducu, "A taxonomy based semantic similarity of documents using the cosine measure," in *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on*, sept. 2009, pp. 129 –134. [citado na p. 19]

[53] M. Bjelica, "Towards tv recommender system: experiments with user modeling," *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 3, pp. 1763 –1769, aug. 2010. [citado na p. 21]

[54] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, vol. 19, no. 1, pp. 17–37, 1982. [citado na p. 22]

[55] M. B. C. N. Malheiro, "Metodologias de revisão de crenças em sistemas multi-agente," Ph.D. dissertation, Faculdade de Engenharia da Universidade do Porto, 1999. [citado na p. 22]

[56] E. Tyugu and Ė. Tyugu, *Algorithms and Architectures of Artificial Intelligence*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2007. [Online]. Available: http://books.google.pt/books?id=o2s83NuChn8C [citado na p. 22]

[57] Z. Ren and D. Wang, "The improvement research on rule matching algorithm rete in electronic commerce application systems," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, oct. 2008, pp. 1 –4. [citado na p. 22]

[58] Oracle. Netbeans. [Online]. Available: http://netbeans.org/ [citado na p. 25]

[59] Oracle. Java development kit. [Online]. Available: http://www.oracle.com/technetwork/java/javase/downloads/jdk-7u4-downloads-1591156.html [citado na p. 25]

[60] T. I. SpA. Java agent development framework. [Online]. Available: http://jade.tilab.com/ [citado na p. 25]

[61] T. I. SpA. Web services dynamic client. [Online]. Available: http://jade.tilab.com/ [citado na p. 25]

[62] T. I. SpA. Jade web services integration gateway. [Online]. Available: http://jade.tilab.com/ [citado na p. 25]

[63] T. A. S. Foundation. Apache ant. [Online]. Available: http://ant.apache.org/ [citado na p. 26]

[64] JBOSS. Drools - the business logic integration platform. [Online]. Available: http://www.jboss.org/drools/ [citado na p. 26]

[65] D. Gilbert. Jfreechart. [Online]. Available: http://www.jfree.org/jfreechart/ [citado na p. 26]

[66] java.net. Jsp standard tag libray. [Online]. Available: http://jstl.java.net/ [citado na p. 26]

[67] java.net. Oracle mojarra javaserver faces. [Online]. Available: http://javaserverfaces.java.net/ [citado na p. 26]

[68] T. A. S. Foundation. Apache tomcat. [Online]. Available: http://tomcat.apache.org/ [citado na p. 27]

[69] T. A. S. Foundation. Apache axis2. [Online]. Available: http://axis.apache.org/axis2/java/core/ [citado na p. 27]

[70] Oracle. Mysql. [Online]. Available: http://www.mysql.com/ [citado na p. 27]

[71] S. C. for Biomedical Informatics Research. Protege. [Online]. Available: http://protege.stanford.edu/ [citado na p. 28]

[72] S. C. for Biomedical Informatics Research. Protege ontology bean generator. [Online]. Available: http://protegewiki.stanford.edu/wiki/OntologyBeanGenerator_4.1 [citado na p. 28]

[73] B. B. C. (BBC). Program categories. [Online]. Available: http://www.bbc.co.uk/a-z/ [citado na p. 35]

[74] P. Amarelas. Categorias de publicidade. [Online]. Available: http://www.pai.pt [citado na p. 35]

[75] C. Finkelstein, *An Introduction to Information Engineering: From Strategic Planning to Information Systems.* Addison-Wesley, 1989. [citado na p. 52]

Anexos

Anexo A Diagramas de Sequência

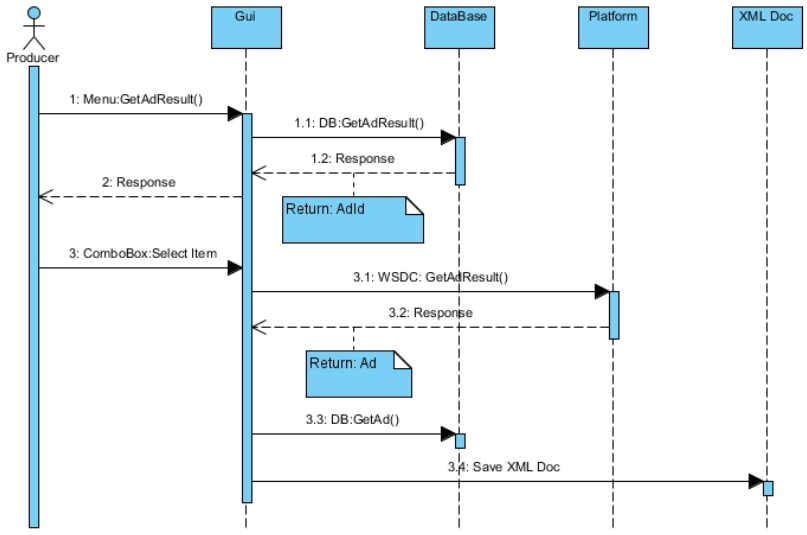


Figura 1: Diagrama de sequência: `GetAdResult`

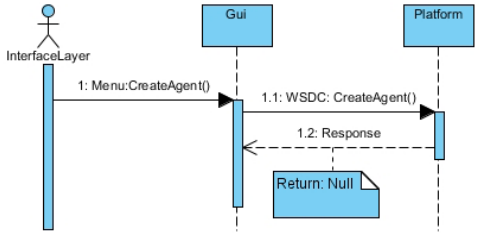


Figura 2: Diagrama de sequência: `CreateAgent`

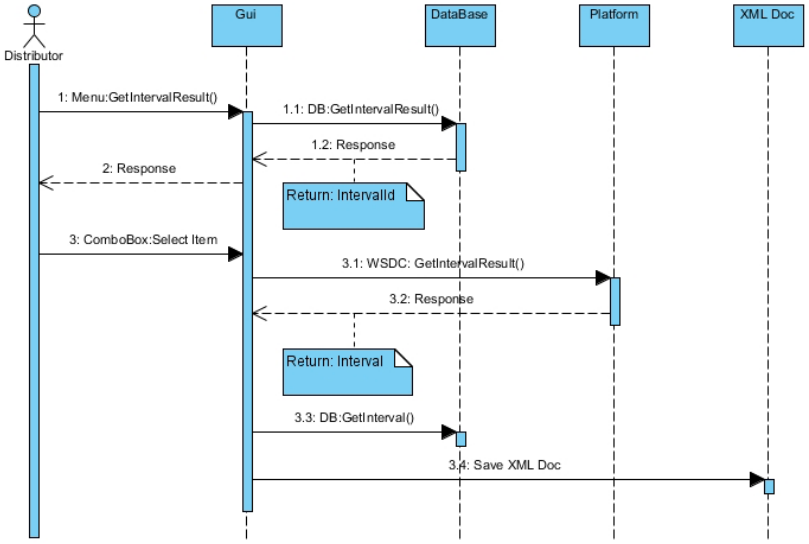


Figura 3: Diagrama de sequência: `GetIntervalResult`

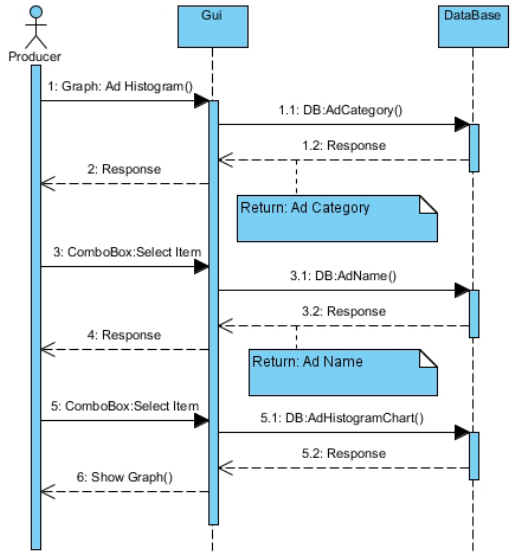


Figura 4: Diagrama de sequência: `GraphAdHistogram`

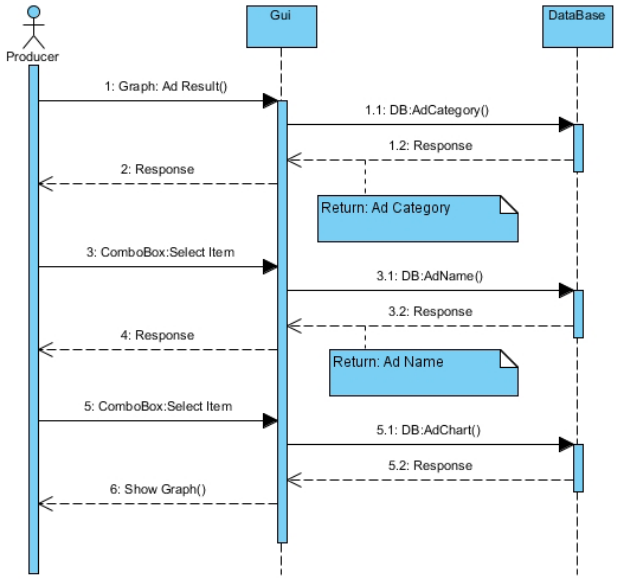


Figura 5: Diagrama de sequência: `GraphAdResult`

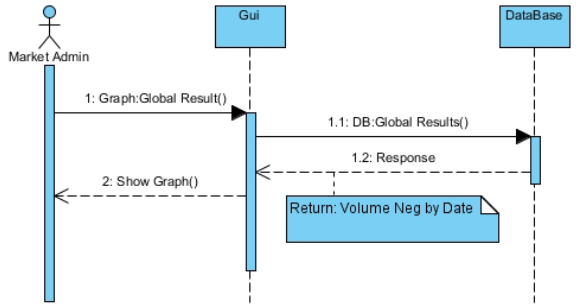


Figura 6: Diagrama de sequência: `GraphGlobalResult`

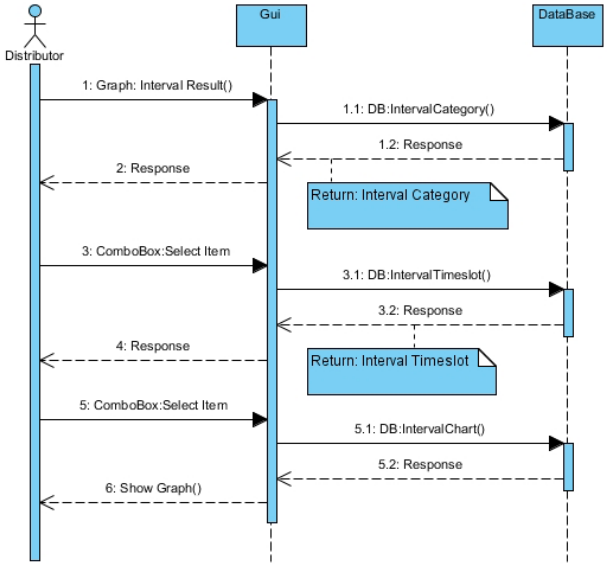


Figura 7: Diagrama de sequência: `GraphIntervalResult`

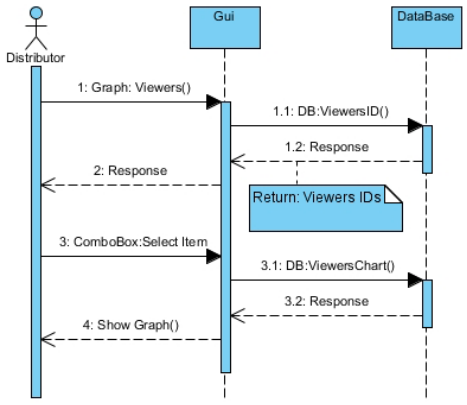


Figura 8: Diagrama de sequência: `GraphViewer`

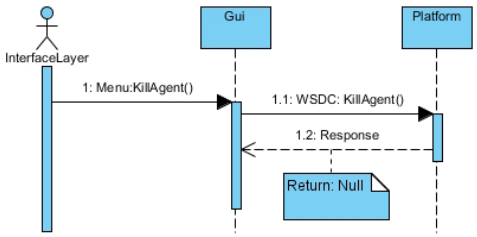


Figura 9: Diagrama de sequência: `KillAgent`

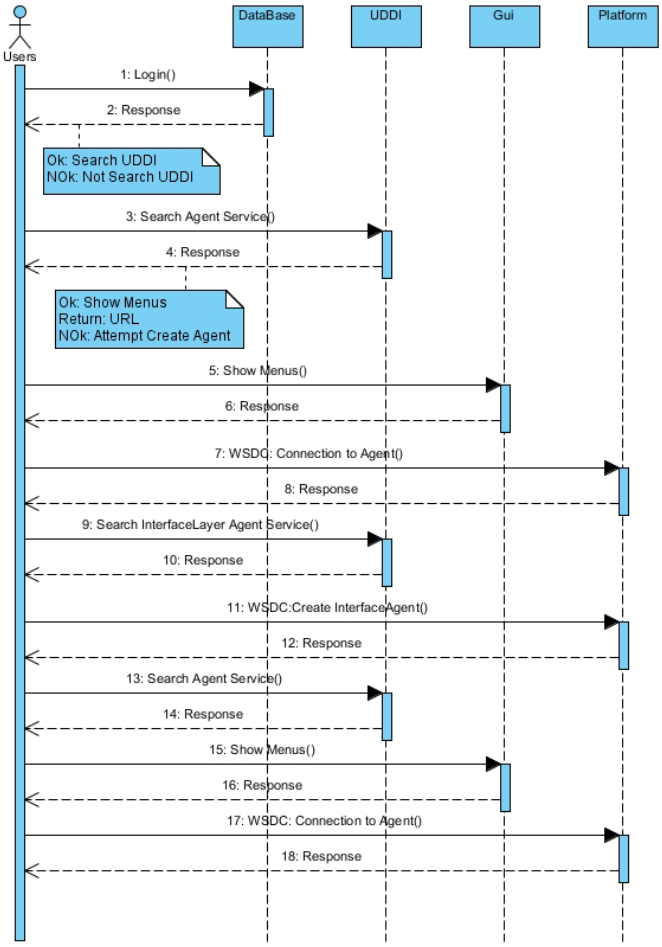


Figura 10: Diagrama de sequência: `Login`

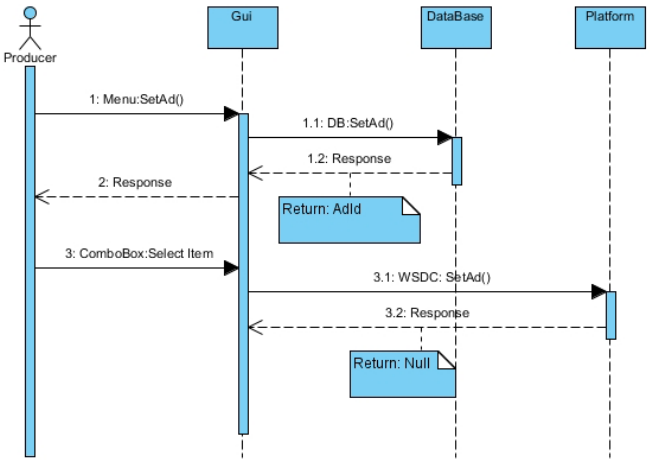


Figura 11: Diagrama de sequência: `SetAd`

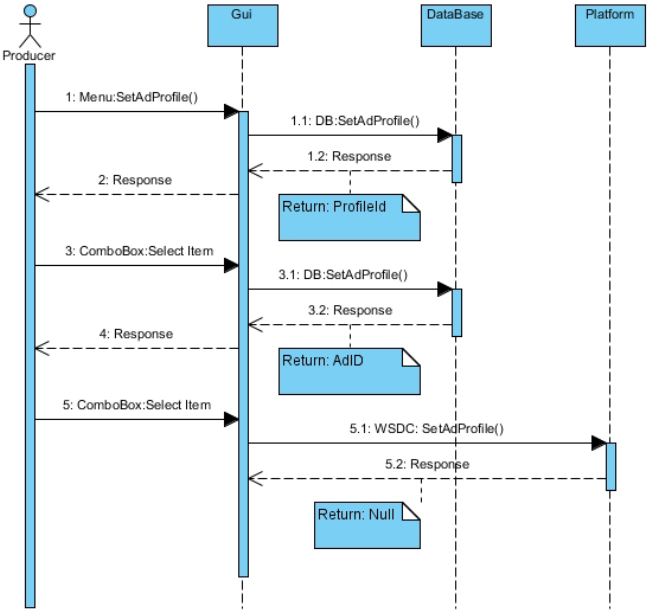


Figura 12: Diagrama de sequência: `SetAdProfile`

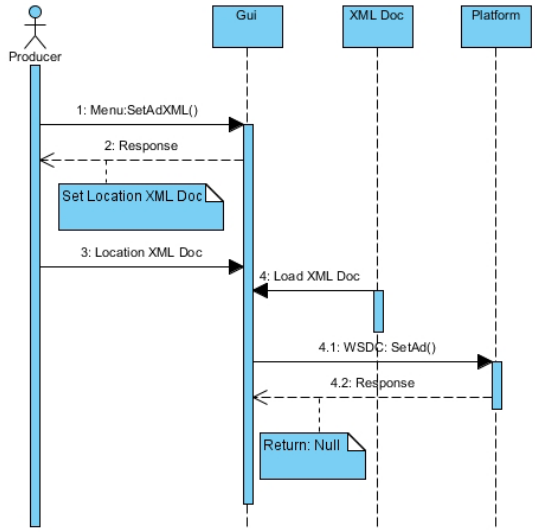


Figura 13: Diagrama de sequência: `SetAdXml`

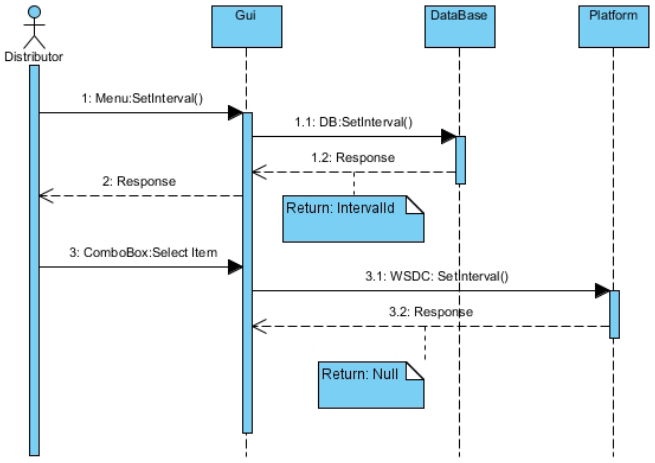


Figura 14: Diagrama de sequência: `SetInterval`

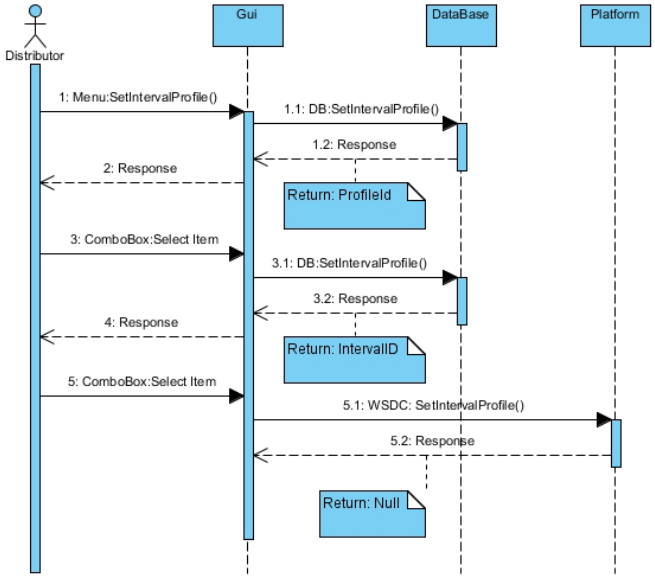


Figura 15: Diagrama de sequência: `SetIntervalProfile`

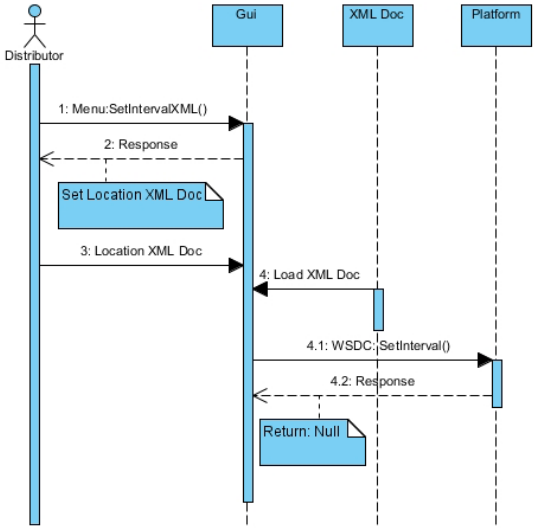


Figura 16: Diagrama de sequência: `SetIntervalXml`

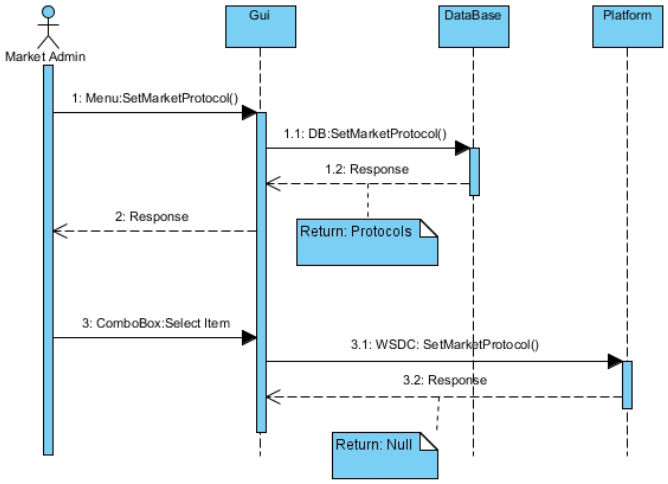


Figura 17: Diagrama de sequência: `SetMarketProtocol`

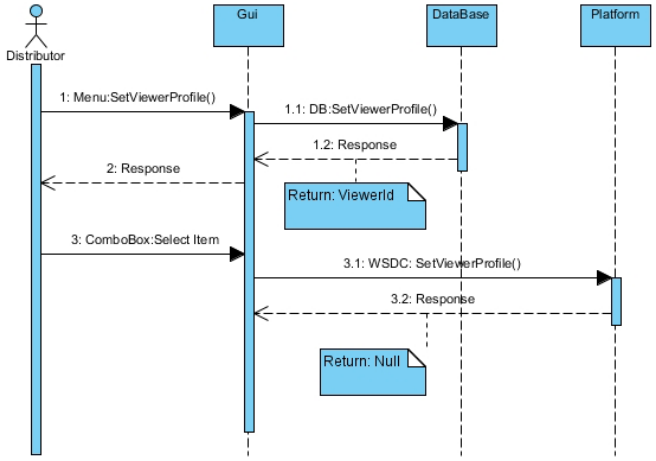


Figura 18: Diagrama de sequência: `SetViewerProfile`

Anexo B Diagramas de Casos de Uso

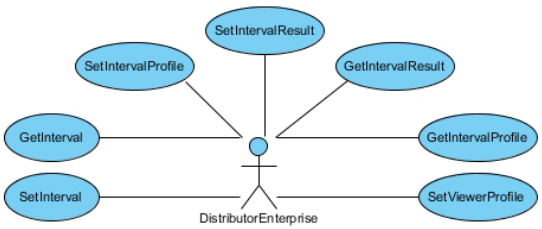


Figura 19: Diagrama de caso de uso: `DistributorEnterprise`

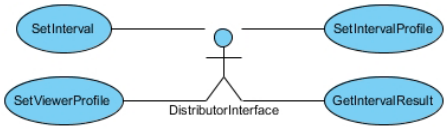


Figura 20: Diagrama de caso de uso: `DistributorInterface`

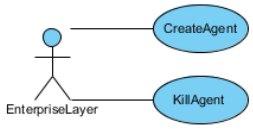


Figura 21: Diagrama de caso de uso: `EnterpriseLayer`

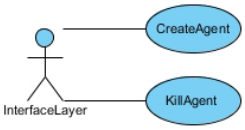


Figura 22: Diagrama de caso de uso: `InterfaceLayer`

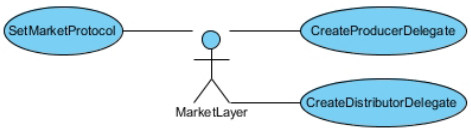


Figura 23: Diagrama de caso de uso: `MarketLayer`

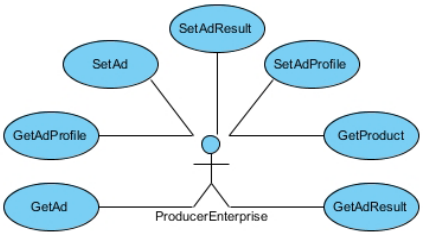


Figura 24: Diagrama de caso de uso: `ProducerEnterprise`

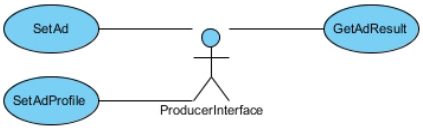


Figura 25: Diagrama de caso de uso: `ProducerInterface`

Anexo C Modelo EER da Base de Dados

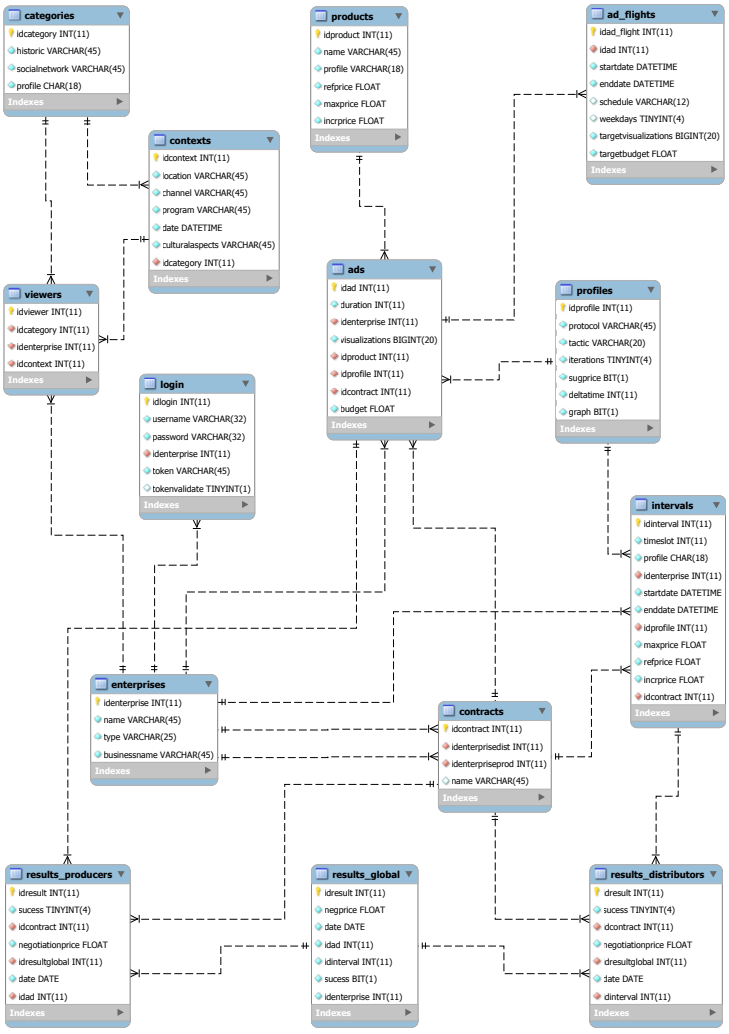


Figura 26: Base de dados - Modelo EER

Anexo D Ontologia MultiMediaBrokerage

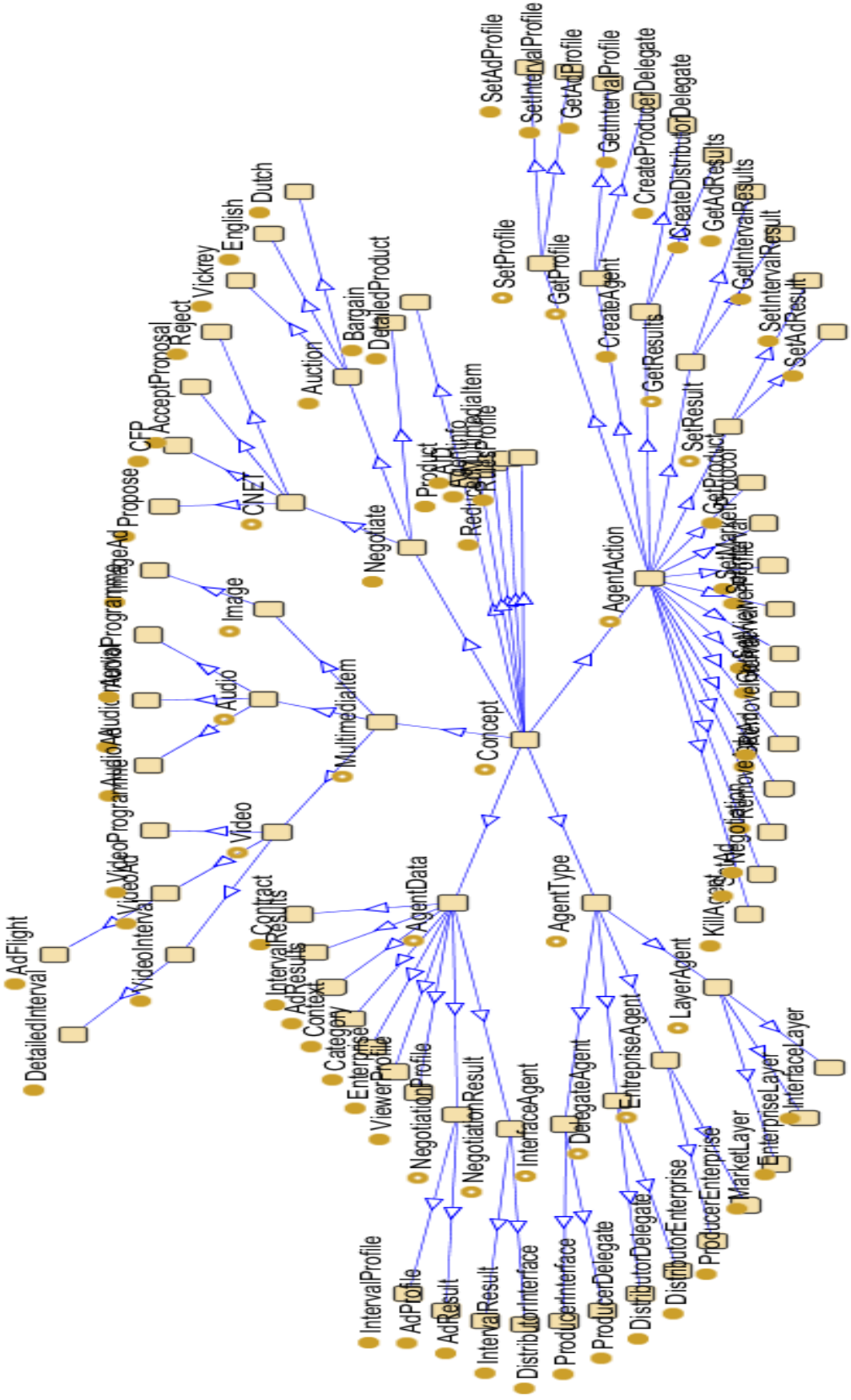


Figura 27: Ontologia MediaBrokerage

Anexo E Ontologia BBC

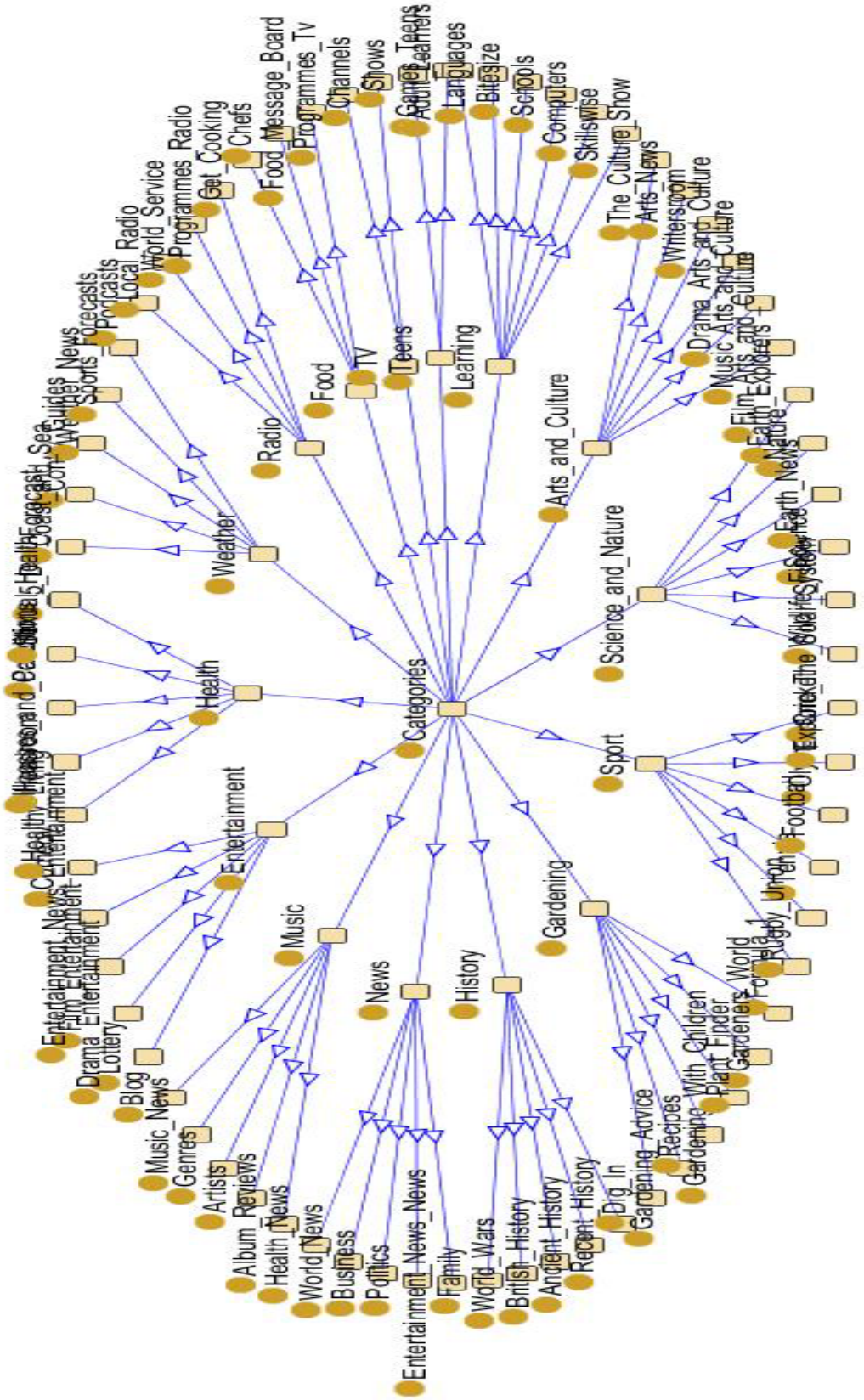


Figura 28: Ontologia BBC

Anexo F Ontologia Ads

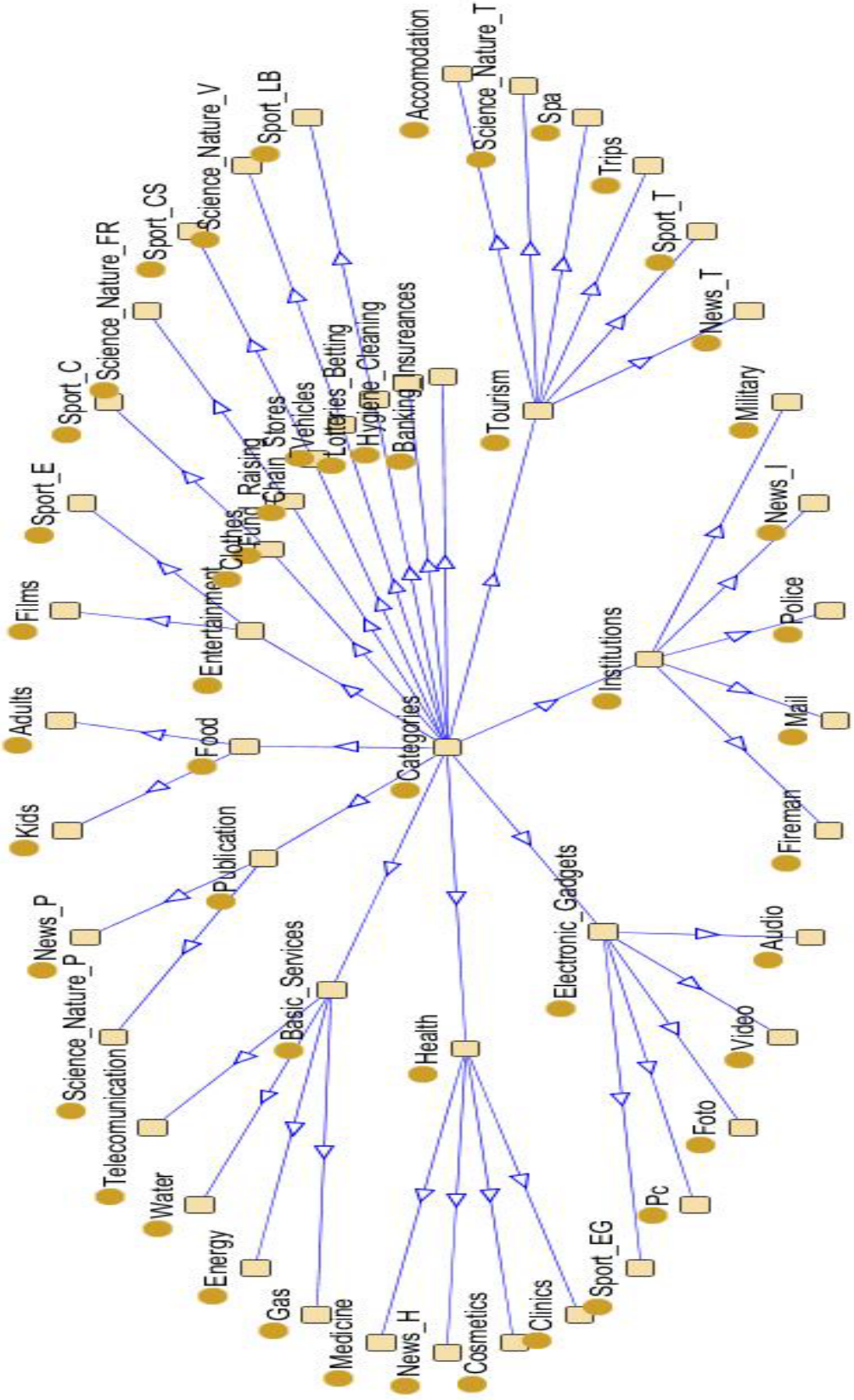


Figura 29: Ontologia Ads