Case-based Argumentation Infrastructure for Agent Societies

Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital
Universitat Politècnica de València

Author:
Jaume Jordán Prunera

Supervisor:
Dr. Vicente Julián Inglada

September, 2011

Contents

List of Figures

Agraïments

A la meua família, pel seu suport incondicional, paciència, ànims i comprensió. Per haver-se esforçat sempre en donar-me l'oportunitat de poder arribar fins ací, en tots els sentits.

Als amics, amigues, companys i companyes, pels bons moments d'humor, de compartir experiències, sentiments i suport mutu.

Als companys i companyes del GTI-IA, per ajudar-me sempre que ho he necessitat i pel bon ambient de treball.

A Stella, per haver compartit els seus coneixements i les seues investigacions amb mi. Pels llargs dies de treball i discussions on he aprés molt. I per la seua inestimable ajuda en tot el meu treball. A Vicente, per confiar en mi i donar-me l'oportunitat de treballar amb ell. Per compartir la seua experiència i dirigir-me correctament. Pel seu suport, comprensió i ajuda en aquest i altres treballs.

Moltes gràcies a tots, sense vosaltres res no tindria sentit.

Chapter 1

Introduction

1.1 Motivation

Argumentation theory has produced important benefits on many AI research areas, from its first uses as an alternative to formal logic for reasoning with incomplete and uncertain information to its applications in Multi-Agent Systems (MAS) [Bench-Capon and Dunne, 2007] [Rahwan and Simari, 2009]. Currently, the study of argumentation in this area has gained a growing interest. The reason behind is that having argumentation skills increases the agents' autonomy and provides them with a more intelligent behaviour.

An autonomous agent should be able to act and reason as an individual entity on the basis of its mental state (beliefs, desires, intentions, goals, etc.). As member of a MAS, an agent interacts with other agents whose goals could come into conflict with those of the agent. In addition, agents can have a social context that imposes dependency relations between them and preference orders among a set of potential values to promote/demote. For instance, an agent representing the manager of a company could prefer to promote the value of *wealth* (to increase the economic benefits of the company) over the value of *fairness* (to preserve the salaries of his employees). Therefore, agents must have the ability of reaching agreements that harmonise their mental states and that solve their conflicts with other agents by taking into account their social context. Argumentation is a natural way of reaching agreements between several parties with opposing positions about a particular issue. The argumentation techniques, hence, can be used to facilitate the agents' autonomous reasoning and to specify interaction protocols between them [Rahwan, 2006].

Recently, the ASPIC project[1] [Amgoud et al., 2006] made an effort to consolidate the work done in argumentation languages and protocols, argument visualisation and editing tools and, generally, in argumentation frameworks for MAS. However, the argumentation infrastructure proposed in this project does not offer support for agent societies and their

[1]European Union's 6th Framework ASPIC Project (IST-002307), http://www.fri.uni-lj.si/en/laboratories/ailab/136/project.html

agents' social context. Also, it was not focused on proposing argumentation technologies for open MAS. Nevertheless, it is important to include support for agent societies and their agents' social context because it allows to simulate different models of organizations and societies to resolve problems taking into account the agents' social context and the structure of the organization.

In this work, we propose an infrastructure to develop and execute argumentative agents in an open MAS. This infrastructure offers the necessary components to develop agents with argumentation capabilities, including the communication skills and the argumentation protocol, and it offers support for agent societies and their agents' social context. The main advantage of having this infrastructure is that it is possible to create agents with argumentation capabilities to resolve a specified problem. We consider that this approach could obtain better results than other distributed approaches due to the argumentation process between agents and their reasoning skills, which allow them to reach to more efficient solutions to problems and more satisfactory agreements. In the argumentation dialogue the agents try to reach an agreement about the best solution to apply for each proposed problem.

1.2 Objectives

The main objective of this work is to create a new infrastructure that allows agents to incorporate argumentative reasoning methods in an open MAS, taking into account the agent societies where the agents are situated in and their social context. Furthermore, the infrastructure must provide the necessary services and components to work as it is expected. In order to arrive to the final goal, we propose several more specific objectives to achieve:

- Study of frameworks or infrastructures with CBR technology and argumentation. This review of the state of the art will give a vision of the current frameworks and technologies available.

- Detailed analysis of the argumentation framework chosen to develop the infrastructure. The chosen framework must be studied to understand all the tools needed to design an appropriate infrastructure. Therefore, the framework has been analysed and explained.

- Design of the infrastructure. Analyse the necessary components that the infrastructure needs to work, and how each component must work. Design each component to fit the needs of the infrastructure and its behaviour. This objective is divided in the following points:

  - To select an open MAS platform with support for agent societies and their agents' social context.

  - To design the argumentation skills of the agents. This includes the agents' logic and the argumentation protocol to follow.
  - To select and design the knowledge representation language.
  - To design the knowledge interchange mechanism with the necessary communicative locutions using the message mechanism of the selected agent platform.
  - To design a CBR architecture with an efficient case-base indexing and a suitable case retrieving.
  - To design an efficient persistence of the case-bases that allows to reuse the data in future argumentation processes.

- Development of the infrastructure. The infrastructure has to be correctly implemented using the necessary components and modules to support the designs commented before.
- Validation and evaluation of the infrastructure with an example. Different tests will be designed to validate the infrastructure and to evaluate its performance. To do this, an example with real data has been used to perform the tests.

1.3 Structure of the document

This chapter has introduced the motivation and the objectives of this Master thesis. The rest of the document is structured as follows.

Chapter 2 introduces the concepts of open Multi-Agent Systems, Case-Based Reasoning and Knowledge Interchange with ontologies. It also reviews the current applications of Argumentation in AI.

Chapter 3 describes the argumentation framework on which the infrastructure created in this work is based.

Chapter 4 explains the infrastructure proposed and developed on this work. All defined modules are described in detail.

Chapter 5 presents a real application developed with the infrastructure. This application is a call centre where a group of operators must solve incidences reported by users. Here, the infrastructure is empirically validated and evaluated testing its functionalities and its performance.

Chapter 6 summarises the main contributions of this work and proposes future work on this area. Finally, the bibliographical work published during the development of this research is referenced.

Chapter 2

State of the Art

2.1 Introduction

The argumentation theory has produced important benefits on many AI research areas, from its first uses as an alternative to formal logic for reasoning with incomplete and uncertain information to its more recent applications in Multi-Agent Systems (MAS) [Bench-Capon and Dunne, 2007, Rahwan and Simari, 2009]. Currently, the study of argumentation in this area has gained a growing interest. The reason behind is that having argumentation skills increases the agents' autonomy and provides them with a more intelligent behaviour.

An autonomous agent should be able to act and reason as an individual entity on the basis of its mental state (beliefs, desires, intentions, goals, etc.). As member of a MAS, an agent interacts with other agents whose goals could come into conflict with those of the agent. Moreover, if a dynamic and open MAS is considered, the knowledge that an agent has about the environment, its neighbours and its mental state can change in the course of time. In addition, agents can have a social context that imposes dependency relations between them and preference orders among a set of potential values to promote/demote. Therefore, agents must have the ability of reaching agreements that harmonise their mental states and that solve their conflicts with other agents by taking into account their social context and values. Argumentation is a natural way of reaching agreements between several parties with opposing positions about a particular issue. The argumentation techniques, hence, can be used to facilitate the agents' autonomous reasoning and to specify interaction protocols between them [Rahwan, 2006].

Case-Based Reasoning (CBR) [Aamodt and Plaza, 1994] is another research area where the argumentation theory has produced a wide history of successful applications. According to the CBR methodology, a new problem can be solved by searching in a case-base for similar precedents and adapting their solutions to fit the current problem.This reasoning methodology has a high resemblance with the way by which people argue about their positions, trying

to justify them on the basis of past experiences. The argumentation theory concepts and techniques have been successfully applied in a great number of CBR systems, specially in those that work in legal domains, where a plaintiff and a defendant argue over their opposing positions in court.

In this chapter, the most important research areas, where this work is involved, are revised. Firstly, in Section 2.2 open MAS are described, including the THOMAS framework and the Magentix2 platform, used in the proposed infrastructure. Section 2.4 describes the CBR methodology due to its applications in argumentation and hence, to the argumentation framework that uses the created infrastructure. In Section 2.3 we introduce the importance of the knowledge interchange between agents by using ontologies. Therefore, all the main concepts about ontologies and used tools are described. Finally, Section 2.5 makes a review of the current applications of argumentation in AI that include the CBR methodology. With the introduction of concepts and the review of applications of this Chapter, we pretend to give the reader a general vision about case-based argumentation in open MAS.

2.2 Open Multi-Agent Systems

The concept of Multi-Agent Systems (MAS) evolved from Distributed Artificial Intelligence [O'Hare and Jennings, 1996]. MAS are distributed systems but are not as traditional systems. The main difference is that in a MAS, each agent pursues its own objectives. However, in a classic distributed system all nodes are intended to achieve a common goal. Moreover, in many cases the MAS are considered a part of Artificial Intelligence. The main reason of that is because the AI wants to emulate human behaviour and reasoning, and MAS are the platform to simulate societies by means of "intelligent" software agents.

Therefore, MAS can be viewed as a set of autonomous agents working together to solve problems. These agents have some information and the ability to solve the problem at hand. Therefore, the resolution must be done cooperatively by some form of communication. The data tend to be decentralized and the computation is asynchronous. In addition, agents can decide the tasks to be done and who should perform them.

It is necessary to clarify the concept of software agent, as it is the basis of any MAS. However, there is no concise definition and widely accepted by the scientific community. One of the first definitions is: "an agent is defined as an entity whose state is viewed as a set of mental components such as beliefs, capabilities, choices and agreements" [Shoham, 1993]. Another of the most cited and more common is: "an agent is a computer system located in an environment that is able to perform actions independently to achieve their design goals" [Wooldridge, 2002]. These definitions are an example of the discussion about the concept of agent. However, the following definition along with the above helps to understand the meaning of a software agent, "The concept of software agent characterizes an entity with

a robust architecture adaptable and it can work in different environments or computing platforms and it is capable of reach different objectives in an intelligent and autonomous way exchanging information with the environment, other human or computer agents" [Garijo, 2002]. A more detailed discussion about the concept of agent can be found in [Mas, 2005].

Despite the short history of MAS, currently there are multitude of applications in various fields. Their use has been extensive in the industry, so, there are applications for control and manufacturing planning, product design, air traffic control, power management, location of containers, control of production lines, etc [Jennings and Wooldridge, 1998]. In addition, there are also applications in other fields such as medicine, information retrieval, e-commerce and telecommunications. This demonstrates the widespread interest in the use of MAS given the advantages that they offer.

Currently, a new paradigm in the field of MAS has become more important. In this paradigm, agents can enter or leave the system, interact and dynamically form groups (e.g. agents' coalitions or organisations) to solve problems. This kind of MAS are called open MAS. Thus, open systems allow the entry of new components during the execution of the system that may not have been considered in the design phase [Gonzalez-Palacios and Luck, 2007]. Therefore, agents that participate in a system of this type can use different protocols or be developed with different languages or architectures. This provides great flexibility and diversity to the system. However, regardless of the design or development that has had a component, it will join the system by acquiring a certain role for which a set of rules are established to control its behaviour.

Open MAS are appropriate in dynamic environments where agents can enter or leave the system continuously [Zambonelli et al., 2003]. Thus, in the design phase can the number of agents that will be present in the system can be unknown. Therefore, this dynamism and the heterogeneity of different agents that can enter the organization must be considered in the design of an open MAS. As a consequence, it is necessary to establish control and security agents as they can get unreliable or conflicting objectives of the organization. Another very complex part of open MAS is the development of open communications, mainly due to the heterogeneity of the components that may enter the system.

Some open MAS application examples are e-commerce systems and information agent systems [Dastani et al., 2003]. In these cases, agents adopt roles of buyer and seller temporarily. There are also some works that address the problems of open systems using internal agents representing the external agents that apply to participate in the organization [Esteva et al., 2001]. Hence, the external agents are not involved directly in the organization avoiding many problems of security and control.

The proposed infrastructure in this work needs an agent platform to run the agents. In addition, the infrastructure has to manage the agent societies in an open MAS environment.

These needs determine a concrete type of platform to use as the base of our infrastructure. We have decided to use Magentix2 platform because it has all the characteristics that our infrastructure needs. On the one hand, it is an open source agent platform to execute agents and to perform communications. On the other hand, it integrates the THOMAS framework to manage virtual organizations, agent societies, and services in open MAS. Following, we explain the THOMAS framework and the Magentix2 platform.

2.2.1 THOMAS

THOMAS[1] [Giret et al., 2009] [Carrascosa et al., 2009] [del Val et al., 2009] (Me**TH**ods, Techniques and Tools for **O**pen **M**ulti-**A**gent **S**ystems) is an open multi-agent systems framework based on web services and organizations. It has been developed by *Grupo de Tecnología Informática - Inteligencia Artificial* of the *Departamento de Sistemas Informáticos y Computación*, from the Universitat Politècnica de València.

The THOMAS architecture basically consists of a set of modular services. Although based on the FIPA architecture, THOMAS expands the capabilities of this architecture to manage organizations. Therefore, a new module has been included to achieve this objective, the redefinition of FIPA ***D**irectory **F**acilitator*. Thus, it can handle the services by following the guidelines for ***S**ervice **O**riented **A**rchitectures* (SOA). The services are the most important thing in THOMAS framework. Agents have a set of services in different modules or components for access to the infrastructure. The main components of the THOMAS framework are:

- ***S**ervice **F**acilitator* (SF). It offers simple and complex services for active agents and organizations. Its basic functionality is to provide a yellow page search service and another green page service for service descriptions.

- ***O**rganization **M**anagement **S**ystem* (OMS). It is responsible for the management of organizations and their entities. Therefore, it allows to create and manage any organization.

- ***P**latform **K**ernel* (PK). Responsible for the basic management of an agent platform.

2.2.2 Magentix2

Magentix2[2] is an agent platform for open multi-agent systems. It has been developed by *Grupo de Tecnología Informática - Inteligencia Artificial* of the *Departamento de Sistemas Informáticos y Computación*, from the Universitat Politècnica de València. Its main objective

[1] http://www.gti-ia.upv.es/sma/tools/Thomas/index.php
[2] http://www.gti-ia.upv.es/sma/tools/magentix2/index.php

is to bring the agent technology to real domains: business, industry, logistics, e-commerce, health-care, etc.

The final goal of Magentix2 is to provide new services and tools that allow for the secure and optimized management of open MAS. Magentix2 provides support at three levels:

- Organization level: technologies and techniques related to agent societies and virtual organizations. The THOMAS framework allows to manage virtual organizations and agent societies by means of different services and control mechanisms.
- Interaction level: technologies and techniques related to communications between agents.
- Agent level: technologies and techniques related to individual agents (such as reasoning and learning).

In order to offer these support levels, Magentix2 is formed by different building blocks, and provides technologies to the development and execution of MAS.

The communication in Magentix2 is performed using AMQP[3]. This industry-grade open standard is designed to support reliable, high-performance messaging over the Internet. It facilitates the interoperability between heterogeneous entities. Magentix2 allows heterogeneous agents to interact with each other via FIPA-ACL[4] messages, which are exchanged over the AMQP standard.

An interesting feature of the Magentix2 platform is the Tracing Service Support. This support allows to share information between the agents of the MAS in an indirect way by means of trace events. It is based on the publish/subscribe software pattern, which allows subscribers to filter events attending to some attributes (content-based filtering), so that agents only receive the information they are interested in and only requested information is transmitted. Magentix2 incorporates a Trace Manager (TM), which is in charge of coordinating the process of event tracing, allowing agents to publish/unpublish, to subscribe/unsubscribe, to trace information, or to look up available trace information at run time.

The security is another important part that is taken into account in Magentix2. The platform incorporates a security module which provides key features regarding security, privacy, openness and interoperability not offered by other current Agent Platforms. This module is based on open standards (AMQP, SSL,SASL, X.509 certificates, WS-Security, FIPA-ACL) and open source technologies (Qpid, NSS, Axis2, Rampart). The security module allows the development of not only secure but also open MAS in which previously unknown agents can enter and leave the MAS at any moment. The main component of the security

[3]http://www.amqp.org/

[4]http://www.fipa.org/specs/fipa00061/SC00061G.html

module is the Magentix2 Management Service (MMS), which is a WSSecurity compliant secure web service that controls the creation of agent certificates for platform users.

An interesting feature of Magentix2 are the conversational agents (CAgents). CAgents allow the automatic creation of simultaneous conversations based on interaction protocols. CAgents can use pre-defined interaction protocols, define their own interaction protocols and also dynamically change interaction protocols at runtime. CAgents are composed of two main components: Conversation Factories (CFactories) and Conversation Processors (CProcessors). A CFactory defines an interaction protocol as finite state machines by means of nodes and arcs between them. CFactories are in charge of creating CProcessors that will execute the defined interaction protocol. CFactories manage automatically incoming messages, deciding if a message belongs to an ongoing CProcessor or if a new one has to be created. Moreover, CFactories allow agents to maintain several conversations following simultaneously the same interaction protocol and also manage concurrence aspects.

The THOMAS framework explained previously is also included in the Magentix2 platform. This framework together with the Magentix2 platform permits to manage agent societies in MAS. With THOMAS and Magentix2, a MAS can be built with support for virtual organizations, agent societies, service execution and development of any kind of agent.

2.3 Knowledge Interchange

In an open MAS it is important to have a well established knowledge interchange mechanism. Nowadays, the most applied way to perform the communication between heterogeneous agents is to use ontologies to facilitate the understanding between agents.

An ontology represents knowledge as a set of concepts of a concrete domain taking into account the relationship between those concepts and their properties. It is used as a knowledge representation about the world or some part of it.

OWL 2[5] (**O**ntology **W**eb **L**anguage) is an ontology language based on XML[6]. It is developed as a follow-on from RDF[7] and RDFS[8]. All its elements (classes, properties and individuals) are defined as RDF resources, and identified by URIs. The main objective of OWL 2 is to be an standard used over the World Wide Web.

The **OWL API**[9] is a Java API and reference implementation for creating, manipulating and serialising OWL Ontologies. In the infrastructure proposed in this work, the OWL API has been used to create parsers to manage the data specified in OWL ontologies. With this parsers, agents can manage and understand the shared data specified in OWL.

[5] http://www.w3.org/TR/owl2-overview/
[6] http://www.w3.org/XML/
[7] http://www.w3.org/RDF/
[8] http://www.w3.org/TR/rdf-schema/
[9] http://owlapi.sourceforge.net/

Another challenge that must be dealt with is how to communicate arguments between the agents of a particular MAS or between the agents of different systems. When working with open MAS, where the system dynamicity and the heterogeneity between agents is assumed by default, this functionality is particularly challenging. The research in this area has already been started by the ASPIC community[10] [Amgoud et al., 2006], which is developing its standardisation proposal for an argument interchange format (**AIF**) [Chesñevar et al., 2006]. The format introduces an abstract formalism for representing concepts about arguments, argument networks, communication and argumentation context in MAS capable of argumentation-based reasoning. Since the AIF is being agreed upon the argumentation and MAS expert research communities, it is likely to be adopted by many researchers as an standard for argument communication. The AIF is used in our infrastructure to interchange arguments between the agents involved in argumentation dialogues.

2.4 Case-based Reasoning

Case-Based Reasoning (CBR) [Kolodner, 1993, Aamodt and Plaza, 1994] methodology is the process of solving a new problem using similar solutions applied in the past. In this methodology, solved problems of past situations represented as cases are stored in a database called case-base. In computer reasoning, the CBR methodology is formalized as a four-step process:

1. **Retrieve**: In this step one or more cases stored in the case-base are retrieved. These cases are similar to the current problem to solve. To retrieve the cases different types of algorithms can be used to calculate the similarity between the problem to solve and the cases stored in the case-base. Typically, a case is composed, at least, by the problem that represents and its solution.

2. **Reuse**: Adapt the solution from the retrieved case to fit the current problem.

3. **Revise**: Test the solution in the real world (by simulation) and, if necessary, revise the adaptation.

4. **Retain**: Store the last applied solution in form of a new case in the case-base.

The work done in the eighties about legal CBR fostered the argumentation research in the AI community [Rissland et al., 2006]. From then on, the good results of CBR systems in argumentation domains suggest that this type of reasoning is suitable to manage argumentation processes. Nowadays, MAS research community is endeavouring to broaden the applications of the paradigm to more real environments, where heterogeneous

[10] European Union's 6th Framework ASPIC Project (IST-002307), http://www.fri.uni-lj.si/en/laboratories/ailab/136/project.html

agents could enter in (or leave) the system, form societies and interact with other agents [Ossowski et al., 2007]. Also, MAS have been proposed as a suitable technology to implement the new paradigm of computing as interaction [Luck and McBurney, 2008], where large systems can be viewed or designed in terms of the services they offer and the entities that interact to provide or consume these services. The high dynamism of these open MAS gives rise to a greater need for a way of reaching and managing agreements that harmonise conflicts.

Moreover, this type of systems also poses other potential problems to overcome. Common assumptions about the agents of most MAS, such as honesty, cooperativeness and trustworthiness cannot be longer taken as valid hypothesis in open MAS. Therefore, there is an obvious need for providing the agents of an open MAS with individual reasoning and learning capabilities that make them more intelligent and autonomous and prevent them from the potential attacks of interested agents.

2.5 Current Applications of Argumentation in AI

Nowadays, the argumentation research in AI is experiencing a new reactivation, mainly motivated by recent and interesting contributions developed in MAS. On one hand, the argumentation theory has been studied in MAS to manage the agent's practical reasoning. Practical reasoning is a well-known area in philosophy, but which historically has received less attention in AI than the theoretical reasoning. This type of reasoning analyses which specific action should be performed in a particular situation, instead of the theoretical reasoning objective of deciding the truthfulness of beliefs. However, the theoretical reasoning about the state of the world and the effects of the potential actions to perform is also essential. Therefore, both types of reasoning must be considered in MAS. In [Rahwan and Amgoud, 2006], an argumentation-based approach for practical reasoning has been proposed. In this work, Dung's abstract argumentation framework [Dung, 1995] is instantiated to generate consistent desires and plans to achieve them. The works developed by Atkinson in her thesis and hers subsequent research are also other important contributions to the modelling of argumentation processes that allow the agents to reason about what is the best action to execute [Atkinson, 2005].

The argumentation techniques have been successfully used to reach agreements that ensure the coherence of the agents' mental state and to structure their interaction in disagreement situations. Parsons et al. [Parsons et al., 1998] proposed a seminal theoretical framework that unifies argumentation-based reasoning and communication for negotiation in MAS. More recently, Rahwan et al. [Rahwan et al., 2003] analyses this and other argumentation-based negotiation frameworks. A wide review of the current situation of the argumentation research in AI has also been published in the special issue on argumentation of the journal *Artificial Intelligence* [Bench-Capon and Dunne, 2007] and

in the book [Rahwan and Simari, 2009]. As it has been commented before, an effort to consolidate the work done in argumentation languages and protocols, argument visualisation and editing tools and, generally, in argumentation frameworks for MAS, was performed by the ASPIC project [Amgoud et al., 2006]. As a result, the *Argument Interchange Format* (AIF) has been proposed to serve as a convergence point for theoretical and practical work in this area [Willmott et al., 2006]. All these advances show how the study of argumentation in AI, and more concretely in MAS, is currently a research area that has a high activity and a growing interest.

Argument management (generation, selection, evaluation etc. of the components of arguments and the management of the dialogue itself) is a key issue to deal with in argumentation-based dialogues in MAS. CBR is a suitable methodology to manage argumentation processes in two-party disagreement situations as it has been reported in the previous section. To date, few research has cope with the use of CBR methodology to facilitate the argumentation between the agents of MAS. The current approaches are focused on managing two types of dialogues between agents: argumentation-based negotiation and collaborative deliberation. Following, some relevant approaches are described in an attempt to show the promising advantages of using CBR to aid argumentation in open MAS:

- The PERSUADER system: This system acts as a mediator in the implementation domain of labour management disputes between a company and its trade union [Sycara, 1987, Sycara, 1989, Sycara, 1990]. This was a seminal framework that integrated for the first time concepts of argumentation theory and CBR to create a negotiation model in a MAS. PERSUADER uses a mediator agent that manages the negotiations between two agents representing the company and the trade union. The mediator dialogues with the parts trying to reach an agreement, which is a contract that is accepted by both agents. A contract consists of a set of attributes (e.g. salaries, pensions and holidays) whose value must be decided. PERSUADER studied the argumentation in a non-cooperative domain, where each agent has its own objectives and tries to derive its maximum own benefit from the negotiation. The main objective of the mediator and hence, the objective of the dialogue in this framework, is to negotiate with both agents and persuade them to collaborate. One of the CBR objectives is to infer the model of beliefs and preferences of an unknown agent. In this way, the mediator retrieves the information about past negotiations with similar agents that was stored in precedent cases and adapts it to the current context. Another CBR objective in PERSUADER is to retrieve past cases that act as arguments for persuading an agent to accept a specific contract.

- CBR for Argumentation with Multiple Points of View: Nikos Karacapilidis et al. developed a model that integrates CBR and argumentation for supporting decision

making in discussion processes. This model was implemented in the Argument Builder Tool (ABT) of the multi-agent framework for collaborative deliberation HERMES [Karacapilidis and Papadias, 2001], [Karacapilidis et al., 1997]. This is an Argumentation-based Decision Support System (ADSS) that helps a group of users (human agents) to build sound arguments to defend their positions in favour or against other alternative positions in a discussion. HERMES maps the argument process into a discussion graph with tree structure and shows graphically the possible discourse acts that the agents could instance. The system uses CBR to make the appropriate queries to the (internal or external) databases that store information that support the positions of the agents that participate in the argument and, thus, to generate discourse acts that successfully show their interests and intentions. However, as it is only a support system, afterwards the agents are free to adopt or not the ABT's proposals. In this framework is the system itself who manages the interaction between the agents, being the CBR engine a reasoning component integrated in it. Therefore, the case-base is common for all agents and belongs to the system. The cases are flexible entities that store a set of argumentation elements that can be interpreted depending on the state of the discourse and each agent's point of view. Therefore, the main objective of the CBR methodology in the system is to examine the current discussion and to suggest the participants the best discourse acts to fire, according with their points of view and preferences. Thus, the contents of the HERMES case-base represent past argumentation processes.

- Case-based Negotiation Model for Reflective Agents: Leen-Kiat Soh and Costas Tsatsoulis designed a case-based negotiation model for reflective agents (agents aware of their temporal and situational context). This model uses CBR to plan/re-plan the negotiation strategy that allows the most effective negotiation on the basis of past negotiations [Soh and Tsatsoulis, 2001a, Soh and Tsatsoulis, 2001b, Soh and Tsatsoulis, 2005]. In this framework, a set of situated agents that control certain sensors try to track several mobile targets. The aim of the agents is to coordinate their activities and collaborate to track the path to as many targets as possible. The agents' sensors have limited power and coverage and each agent only controls a subset of sensors. Although the cooperativeness is assumed, each agent has individual tasks to fulfil. Therefore, when an agent has not enough coverage or power capabilities to track a target, it needs to negotiate and persuade other agents and achieve that they leave their tasks and help it to track the target. The agents of this model are autonomous entities that own two separated and private case-bases. Each agent has a *CBR manager* that allows it to learn to negotiate more effectively by using the knowledge of past negotiations. The cases contents store descriptions that characterise the agents' context in a previous negotiation. The argumentation style of this framework views persuasion as a negotiation protocol of information interchange between two agents that try to

reach an agreement by using an argumentation process. An important contribution of this framework was the introduction of learning capabilities for the agents by using the CBR methodology.

- Argument-based selection Model (ProCLAIM): Pancho Tolchinsky et al. extended the architecture of the decision support MAS for the organ donation process CARREL+ [Vázquez-Salceda et al., 2003] with ProCLAIM, a new selection model based on argumentation [Tolchinsky et al., 2006a, Tolchinsky et al., 2006c, Tolchinsky et al., 2006b]. In CARREL+, a *donor agent (DA)* and a set of *recipient agents (RAs)* argue about the viability of the organ transplant to some recipient. If an agreement is not reached, the organ is discarded. ProCLAIM includes a *mediator agent (MA)* that controls the collaborative deliberation dialogue and uses a *CBR engine* to evaluate the arguments about organ viability that the agents submit. The final decision must fulfil several guidelines that, in ProCLAIM case, are the human organs acceptability criteria that CARREL stores in the *Acceptability Criteria Knowledge Base (ACKB)*. The mediator agent uses a case-base to store all relevant information about past donation processes.

- Argumentation-based Multi-Agent Learning (AMAL): Santiago Ontañón and Enric Plaza developed the Argumentation Based Multi-Agent Learning (AMAL) framework [Ontañón and Plaza, 2006, Ontañón and Plaza, 2007]. The agents of this framework are autonomous entities able to independently solve classification problems and to learn by experience, storing the knowledge acquired during the solving process in their private case-bases. The set of possible classification classes is predefined in the framework. The aim of the interaction between the agents is to increase the solution quality by aggregating the knowledge of a group of expert agents. Therefore, they engage in a collaborative deliberation dialogue. The AMAL framework is a newly contribution to the study of argumentation-based learning models for MAS whose agents have individual learning capabilities. This model also differs from many other argumentation frameworks on its dynamic computation of the relation preference between arguments. In addition, the argumentation style is completely case-based.

The implementation domain differs almost on each framework, being HERMES and ProCLAIM the ones that somehow share a common purpose: to provide decision support for a group decision-making. In addition, among other applications, both have been implemented and tested in the medical domain [Karacapilidis and Papadias, 2001], [Tolchinsky et al., 2006c]. In this aspect, the main difference between them is that HERMES helps agents to select the best argument to instantiate in a particular context and hence, to win the discussion, while in ProCLAIM the system assists the mediator agent (and not the donor agents) to decide which agent has posed the best argument and should be the winner of the discussion. Therefore, although working in a similar domain, these systems are

aimed at solving different subproblems inside the more general problem of supporting group decision-making.

Similarly, although HERMES, ProCLAIM and also the AMAL framework share the same dialogue type (deliberation), the final objective of the interaction between the agents of these systems is quite different: HERMES is mainly centred on the argument diagramming and its graphical representation, helping agents to follow the discussion and supporting them with tools to pose better arguments; ProCLAIM deals with the internal deliberation of the mediator agent, supporting only this agent to make the best decision among the set of potential winners and finally; in the AMAL framework all agents have the common objective of deciding the best classification tag for a specific object and act as a group of experts that cooperate by aggregating their knowledge in the deliberation process. In the same way, PERSUADER and Soh's frameworks also share the dialogue type (negotiation), but from a different perspective. Thus, while in PERSUADER the mediator agent completely centralises the negotiation process and the company and the trade union do not keep a direct interaction, in Soh's framework all agents are autonomous and able to play an initiator role that starts and manages a direct dialogue with other agents.

With respect to the CBR objective, in all frameworks the CBR methodology has been mostly used to generate, select or evaluate arguments on the face of previous similar experiences. Consequently, as in any CBR system, the contents of the case-base in each framework consist of a set of elements that describe these previous experiences.

Some approaches using CBR and argumentation to persuade, negotiate or to reach agreements have been described in this section. The intention of this study and the whole Chapter is to give to the reader a view of the current state of the art before explaining the Argumentation framework of the next chapter. This framework is used to create the case-based argumentation infrastructure for agent societies of this work.

Chapter 3

Argumentation Framework

In this chapter we explain a computational framework, proposed in [Heras et al., 2011c], for supporting argumentation MAS in which the participating software agents are able to manage and exchange arguments between themselves, taking into account the agents' social context. First, we explain a formal definition for an agent society. After that, we introduce the knowledge resources that agents can use to generate, select and propose their positions (solution proposals) and arguments to support them. The knowledge resources used are the domain-cases of a database. These cases represent previous problems and their solutions. Furthermore, we present the argument types of the framework (support and attack arguments) and their support set, that is a set of elements that supports the argument. Finally, the argumentation protocol that agents follow to engage in argumentation processes is shown. This protocol is the mechanism to manage arguments and define the argumentation dialogue that agents follow.

3.1 Agent society

In the framework, an *agent society* is defined in terms of a set of *agents* that play a set of *roles*, observe a set of *norms* and a set of *dependency relations* between roles and use a *communication language* to collaborate and reach the global objectives of the *group*. This definition, based on the approach of [Dignum, 2003] and [Artikis et al., 2009], can be adapted to any open MAS where there are norms that regulate the behaviour of agents, roles that agents play, a common language that allow agents to interact defining a set of permitted locutions and a formal semantics for each of these elements.

However, the values that individual agents or groups want to promote or demote and preference orders over them have also a crucial importance in the definition of an argumentation framework for agent societies. These values could explain the reasons that an agent has to give preference to certain beliefs, objectives, actions, etc. Thus, they represent the motivation of agents to act in a specific way. For instance, an agent representing the

manager of a company could prefer to promote the value of *wealth* (to increase the economic benefits of the company) over the value of *fairness* (to preserve the salaries of his employees). Also, dependency relations between roles could imply that an agent must change or violate its value preference order. For instance, a manager could impose their values to an expert or a base operator could have to adopt a certain preference order over values to be accepted in a group. Therefore, the framework uses the view of [Bench-Capon and Atkinson, 2009], who stress the importance of the audience in determining whether an argument is persuasive or not for accepting or rejecting someone else's objectives. Thus, it has been included in the definition of agent society the notion of values and preference orders among them. Definition 1 provides a formal specification for the model of society:

Definition 1 (Agent Society)**.** *An Agent society in a certain time t is defined as a tuple* $S_t = < Ag, Rl, D, G, N, V, Roles, Dependency, Group, val, Valpref_Q >$ *where:*

- $Ag = \{ag_1, ag_2, ..., ag_I\}$ *is a finite set of* I *agents members of* S_t *in a certain time* t*.*
- $Rl = \{rl_1, rl_2, ..., rl_J\}$ *is a finite set of* J *roles that have been defined in* S_t*.*
- $D = \{d_1, d_2, ..., d_K\}$ *is a finite set of* K *possible dependency relations among roles defined in* S_t*.*
- $G = \{g_1, g_2, ..., g_L\}$ *is a finite set of groups that the agents of* S_t *form, where each* $g_i, 1 \leq i \leq L, g_i \in G$ *consist of a set of agents* $a_i \in Ag$ *of* S_t*.*
- N *is a finite set of norms that affect the roles that the agents play in* S_t*.*
- $V = \{v_1, v_2, ..., v_P\}$ *is a finite set of* P *values predefined in* S_t*.*
- $Roles : Ag \rightarrow 2^{Rl}$ *is a function that assigns an agent its roles in* S_t*.*
- $Dependency_{S_t} : \forall d \in D, <_D^{S_t} \subseteq Rl \times Rl$ *defines a reflexive, symmetric and transitive partial order relation over roles.*
- $Group : Ag \rightarrow 2^G$ *is a function that assigns an agent its groups in* S_t*.*
- $val : Ag \rightarrow V$ *is a function that assigns an agent the set of values that it has.*
- $Valpref_Q : \forall q \in Ag \bigcup G, <_q^{S_t} \subseteq V \times V$ *defines a reflexive, symmetric and transitive partial order relation over the values of an agent or a group.*

3.2 Knowledge Resources, Argument Types and Support Set

In open multi-agent argumentation systems the arguments that an agent generates to support its position can conflict with arguments of other agents and these conflicts are solved by means

of argumentation dialogues between them. In the framework there is a domain-cases database, with cases that represent previous problems and their solutions. The domain-cases are used to generate positions (solutions) to defend and arguments to support them or attack other positions. The structure of these cases is domain-dependent and consist of a set of features that describe the problem to solve and the solution applied. The framework also have an argument-cases case-base. The argument-cases represent past argumentation experiences and their final outcome. The argument-cases are used in the framework to select the best position to propose in view of the social context and current problem to solve taking into account past argumentation experiences.

Arguments that agents interchange are defined as tuples of the form:

Argument

$$Arg = \{\phi, v, < S >\} \tag{3.1}$$

where ϕ is the conclusion of the argument, v is the value (e.g. economy, quality, solving speed) that the agent wants to promote with it and $< S >$ is a set of elements that support the argument (*support set*).

Support Set

$$\begin{aligned} S =< \{premises\}, \{domainCases\}, \{argumentCases\}, \\ \{distinguishingPremises\}, \{counterExamples\} > \end{aligned} \tag{3.2}$$

A support set is formed by the following elements:

- Premises: which are features that match with some features of the problem description. These are the features that characterise the problem and that the agent has used to retrieve similar domain-cases from its case-base. Note that the premises used might be all features of the problem description or a sub-set.
- Domain cases: which are cases that represent previous problems and their solutions whose features match with some features of the problem description.
- Argument cases: which are cases that represent past argumentation experiences with their final outcome. These cases are used to select the best position to propose in view of the current context of the problem and the argumentation experience of the agent.
- Distinguishing premises: which are premises that can invalidate the application of a knowledge resource to generate a valid conclusion for an argument. These premises are extracted from a domain-case that propose a different solution to the argument to attack. They consist of features of the problem description that where not considered to draw the conclusion of the argument to attack.

- Counter-examples: which are cases that are similar to a case (their descriptions match with some or all features of the problem description) but have different conclusions.

Agents generate arguments when they are asked to provide evidence to support a position (*support arguments*) or when they want to attack others' positions or arguments (*attack arguments*).

The first case happens because, by default, agents are not committed to show evidences to justify their positions. Therefore, an opponent has to ask a proponent for an argument that justifies its position before attacking it. Then, if the proponent is willing to offer support evidences, it can generate a support argument which support set is the set of features (premises) that describe the problem and match the knowledge resources (domain-cases) that it has used to generate and select its position. Note that the set of premises could be a subset of the features that describe the problem to solve (e.g. when a position has been generated from a domain-case that has a subset of features of the problem in addition to other different features).

The second case happens when the proponent of a position generates an argument to justify it and an opponent wants to attack the position or more generally, when an opponent wants to attack the argument of a proponent. Arguments in the framework can be attacked by putting forward distinguishing premises and counter-examples. The attack arguments that the opponent can generate depend on the elements of the support set that justifies the conclusion of the argument of the proponent:

- If the justification for the conclusion of the argument is a set of premises, the opponent can generate an attack argument with a distinguishing premise that it knows. It can do it, for instance, if it is in a privileged situation and knows extra information about the problem or if it is implicit in a case that it used to generate its own position, which matches the problem specification. In the latter, the opponent could generate an attack argument with this case as counter-example.

- If the justification is a domain-case or an argument-case, then the opponent can check its case-base of domain-cases and try to find counter-examples to generate an attack argument with them. Alternatively, it can also try to generate an attack argument with a distinguishing premise from its own known premises and cases that invalidates the proponent's justification.

3.3 Position generation and selection

To generate a position that defends the application of a concrete solution to the current problem to solve, the agent retrieves from its domain case-base those cases that match with the specification of the current problem. With the solutions that were applied in these

cases, the agent generates a potential solution for the problem at hand, which represents its position with respect to the problem. Note that the set of retrieved cases could provide different solutions for the same problem. The matching of the extracted domain-cases with the current problem is specified by a similarity degree that is computed by different distance algorithms.

With the generated positions or solutions to apply to the current problem to solve, the agents must select the most suitable position in view of its similarity to the problem and the acceptance that had in the past in other argumentation dialogues (stored as argument-cases). On the one hand, the similarity degree between the problem and the set of available positions is computed in the position generation. On the other hand, the acceptance that had in the past a similar position is calculated by using a *Suitability Factor (SF)* from the argumentation point of view. To compute this SF, the following elements are considered.

Actually, what the agent does is to decide which argument-case (and thus, which position) is most suitable in view of its past experience. The parameters shown in the following formulas are considered as criteria for making such decision. In the formulas, *argC* is the number of argument-cases in *arg* with the same conclusion than the current argument-case, *argAccC* are those in *argC* that were deemed acceptable, *argAccCAtt* are those in *argAccC* that were attacked, *minAtt* and *maxAtt* are the minimum and maximum number of attacks received by any position generated, *minS* and *maxS* are the minimum and maximum number of steps from any retrieved argument-case to the last node of its dialogue graph and *minKr* and *maxKr* are the minimum and maximum number of knowledge resources used to generate any position.

- **Persuasiveness Degree** (PD): is a value that represents the expected persuasive power of a position by checking how persuasive an argument-case with the same problem description and conclusion that the position associated argument-case was in the past. To compute this degree, the number *argAccC* of argument-cases that were deemed acceptable out of the total number of argument-cases *argC* with the same problem description and conclusion retrieved is calculated:

$$PD = \begin{cases} 0, & \text{if } argC = \emptyset \\ \frac{argAccC}{argC}, & \text{otherwise} \end{cases} \tag{3.3}$$

  with $argAccC, argC \in N$ and $PD \in [0, 1]$, from less to more persuasive power. Note that the persuasiveness degree of a position is not decreased if positions with the same problem description and different conclusions are found in the argument-base, since this difference does not necessarily implies that the current position is wrong or less persuasive. In fact, there are many possible reasons for having different conclusions for the same problem description (e.g. different background knowledge, different reasoning

algorithms to generate positions or even a domain admitting several solutions for the same problem).

- **Support Degree** (SD): is a value that provides an estimation of the probability that the conclusion of the current argument-case was acceptable at the end of the dialogue. It is based on the number of argument cases *argAccC* with the same problem description and conclusion that where deemed acceptable out of the total number of argument-cases *arg* retrieved.

$$SD = \begin{cases} 0, & \text{if } arg = \emptyset \\ \frac{argAccC}{arg}, & \text{otherwise} \end{cases} \tag{3.4}$$

with $argAccC, arg \in N$ and $SD \in [0, 1]$ from less to more support degree.

- **Risk Degree** (RD): is a value that estimates the risk for a position to be attacked in view of the attacks received for a position(s) with the same problem description and conclusion in the past. It is based on the number of argument cases *argAccCAtt* that were attacked out of the total number of *argAccC* argument cases with the same problem description and conclusion retrieved that were deemed acceptable.

$$RD = \begin{cases} 0, & \text{if } argAccC = \emptyset \\ \frac{argAccCAtt}{argAccC}, & \text{otherwise} \end{cases} \tag{3.5}$$

with $argAccCAtt, argC \in N$ and $RD \in [0, 1]$, from less to more risk of attack.

- **Attack Degree** (AD): is a value that provides an estimation of the number of attacks *att* received by a similar position(s) in the past. To compute this degree, the set of arguments with the same problem description that were deemed acceptable is retrieved. Then, this set is separated in several subsets, one for each different conclusion. The sets whose conclusion match with the conclusions of the positions to assess are considered, while the other sets are discarded. Thus, we have a set of argument-cases for each different position we want to evaluate. For each argument-case in each set, the number of attacks received is computed (the number of distinguishing premises and counter-examples received). Then, for each set of argument-cases, the average number of attacks received is computed. The attack degree of each position is calculated by a linear transformation:

$$AD = \begin{cases} 0, & \text{if } maxAtt{=}minAtt \\ \frac{att - minAtt}{maxAtt - minAtt}, & \text{otherwise} \end{cases} \tag{3.6}$$

with $minAtt, maxAtt, att \in N$ and $AD \in [0, 1]$ from less to more degree of attack.

- **Efficiency Degree** (ED): is a value that provides an estimation of the number of steps that took to reach an agreement posing a similar position(s) in the past. It is based on the depth n from the node representing the argument-case of the similar

position to the node representing the conclusion in the dialogue graphs associated to the similar argument-cases retrieved. To compute this degree, the same process to create the subsets of argument-cases than in the above degree is performed. Then, for each argument-case in each subset, the number of dialogue steps from the node that represents this argument-case to the end of dialogue is computed. Also, the average number of steps per subset is calculated. Finally, the efficiency degree of each position is calculated by a linear transformation:

$$ED = \begin{cases} 0, & \text{if } maxS{=}minS \\ 1 - \frac{n-minS}{maxS-minS}, & \text{otherwise} \end{cases} \tag{3.7}$$

with $minS, maxS, n \in N$ and $ED \in [0, 1]$ from less to more efficiency.

- **Explanatory Power** (EP): is a value that represents the number of pieces of information each position covers. It is based on the number kr of knowledge resources were used to generate each position. To compute this number, the same process to create the subsets of argument-cases than in the above degrees is performed. Then, for each argument-case in each set, the number of knowledge resources in the justification part is computed (the number of domain-cases and argument-cases). Then, for each set of argument-cases, the average number of knowledge resources used is computed. The explanatory power of each position is calculated by a linear transformation:

$$EP = \begin{cases} 0, & \text{if } maxKr{=}minKr \\ \frac{kr-minKr}{maxKr-minKr}, & \text{otherwise} \end{cases} \tag{3.8}$$

with $minKr, maxKr, kr \in N$ and $EP \in [0, 1]$ from less to more explanatory power.

The Suitability Factor in terms of the acceptance that had in the past a similar position in the argumentation dialogues is computed by the following formula:

$$SF = w_{PD} * PD + w_{SD} * SD + w_{RD} * (1 - RD) + w_{AD} * (1 - AD) + w_{ED} * ED + w_{EP} * EP \tag{3.9}$$

Finally, with the similarity degree of the positions and the different degrees that represent the suitability factor in terms of argumentation dialogues of the positions, the next formula compute the final suitability of the positions generated:

$$Suitability = w_{simD} * simD + w_{SF} * SF \tag{3.10}$$

3.4 Argumentation Protocol

The agents of the framework need a mechanism to manage the arguments and perform the argumentation dialogue. Therefore, an argumentation protocol has been defined

[Heras et al., 2011a]. This protocol is represented by a set of locutions that the agents use to communicate each other depending on their needs, and an state machine that defines the behaviour of an agent in the argumentation dialogue.

The set of allowed locutions of our argumentation protocol are the following:

- *open_dialogue*(a_s, ϕ), where ϕ is a problem q to solve in the system application domain. With this locution an agent a_s opens the argumentation dialogue, asking other agents to collaborate or negotiate to solve a problem that it has been presented with.

- *enter_dialogue*(a_s, ϕ), where ϕ is a problem q to solve in the system application domain. With this locution, an agent a_s engages in the argumentation dialogue to solve the problem.

- *withdraw_dialogue*(a_s, ϕ), where ϕ is a problem q to solve in the system application domain. With this locution, an agent a_s leaves the argumentation dialogue to solve the problem.

- *propose*(a_s, ϕ), where ϕ is a position p. With this locution, an agent a_s puts forward the position p as its proposed solution to solve the problem under discussion in the argumentation dialogue.

- *why*(a_s, a_r, ϕ), where ϕ can be a position p or an argument arg. With this locution, an agent a_s challenges the position p or the argument arg of an agent a_r , asking it for a support argument.

- *no_commit*(a_s, ϕ), where ϕ is a position p. With this locution, an agent a_s withdraws its position p as a solution for the problem under discussion in the argumentation dialogue.

- *assert*(a_s, a_r, ϕ), where ϕ is an argument arg that supports a position. With this locution, an agent a_s sends to an agent a_r an argument that supports its position.

- *accept*(a_s, a_r, ϕ), where ϕ can be an argument arg or a position p to solve a problem. With this locution, an agent a_s accepts the argument arg or the position p of an agent a_r.

- *attack*(a_s, a_r, ϕ), where ϕ is an argument arg. With this locution, an agent a_s challenges the argument arg of an agent a_r.

- *retract*(a_s, a_r, ϕ), where ϕ is an argument arg. With this locution, an agent a_s informs an agent a_r that it withdraws the argument arg that it put forward in a previous step of the argumentation dialogue.

Figure 3.1 shows the state machine that defines the behaviour of an agent in an argumentation dialogue and the process that follows to propose positions, defend them and

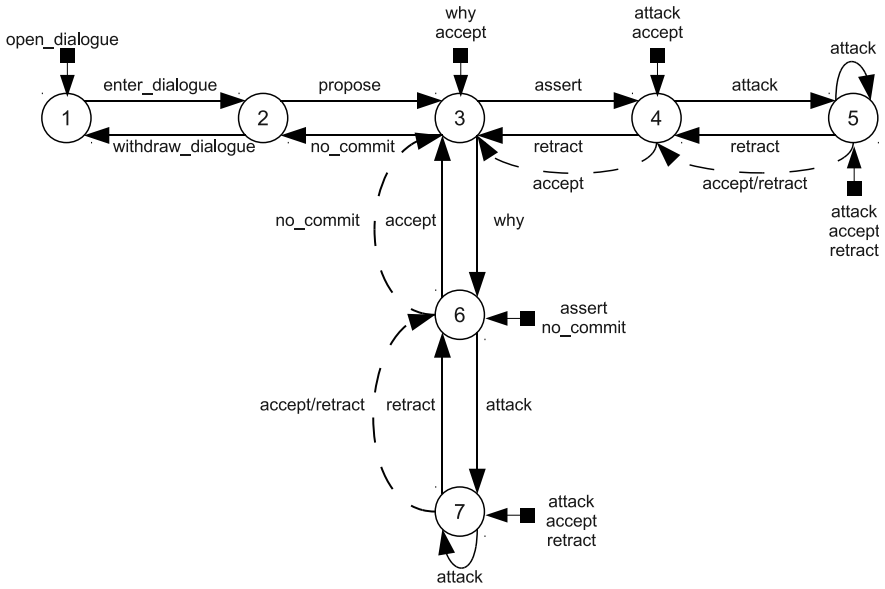


Figure 3.1: Argumentation state machine of the agents

attack others' positions [Jordán et al., 2011a]. The transitions between states depend on the locutions that the agent could use in each situation. The states of the argumentation state machine are described as follows:

1. The first state is the initial state. When the agent is initialised it remains in this state waiting for an *open_dialogue* locution. Also, the agent will come back to this state when the initiator agent communicates that the dialogue has finished. The *open_dialogue* locution inform the agent that a new dialogue to solve a problem (ticket) has started. The agent will retrieve such cases of its case-base which features match the given ticket with a similarity degree greater than a given threshold. Finally, if the agent has been able to retrieve similar domain-cases and use their solutions to propose a solution for the current problem, the agent will engage in the dialogue with the locution *enter_dialogue* and will go to the state 2. The agent only engages in the dialogue if it has solutions to propose.

2. This is the proposing state. When the agent is in this state it has retrieved a list of similar domain-cases to the current problem to propose a solution (position to defend). If there are several solutions to propose, it will select the most similar to the problem and go to state 3. Otherwise, the agent will use the *withdraw_dialogue* locution and will go to state 1.

3. This is a central state because the agent can try to attack other positions or defend its position from the attacks of other agents. First, the agent checks if there is any

why petition from other agent. This locution is used to ask other agents to justify its position. The agent that received the *why* petition will *assert* a support argument to the opponent if it can. This implies going to state 4. If the agent is not able to provide a support argument to defend its position it will go to state 2 and try to propose another position. If the agent has not received any *why* petition, it will ask an agent (chosen randomly) that has a different position to justify it, using the *why* locution. This implies going to state 6.

4. In this state, the agent that has put forward a support argument for its position waits for an *attack* or an *accept* locution. After certain time has passed and nothing is received, the agent will return to state 3. In the case that an attack is received, the agent will try to reply with another attack. If it is not able to reply, it will retract its support argument and go to state 3. Otherwise, if it replies with an attack it will go to state 5.

5. This state represents the situation where the agent is engaged in an attack phase defending its position. When possible, every attack received will be replied with another attack and the agent will remain in this state. When the agent cannot reply an attack with other attack, it will retract its last attack and go to the state 4. In the case of receiving an *accept* locution, it means that the attacking agent accepts the last given attack. That implies to go to state 4 where the attacking agent must accept the support argument and hence, the position of the proponent agent.

6. When the agent enters to this state it is waiting for an *assert* or a *no_commit* locution. When some waiting time has passed and nothing is received, the agent will return to state 3. If the agent receives an *assert* locution and it is not able to attack the support argument received with this locution, it will accept the other agent's position and go to state 3. However, if an attack argument can be generated, it will be send to the other agent and change to state 7. In the case that a *no_commit* locution is received it means that the other agent retracts its position. Then, the agent will go to state 3.

7. This state represents when the agent is engaged in an attack phase attacking other agent's position. The agent will try to reply to any attack received for its attacking arguments and remain in this state while it can reply. If an *accept* locution is received the last attack argument has been accepted by the other agent, thus its position is defeated and the agent will go to state 6 to wait another support argument or a *no_commit* locution. Nevertheless, if an attack of the other agent cannot be replied, the agent has to accept the other agent's attack argument, retracts its attack argument and go to state 6. Then, it must go to state 3 after accepting the other agent's position.

3.5 Remarks

In this Chapter, we have studied the argumentation framework proposed in [Heras et al., 2011c]. We have defined the concept of agent society and we have explained the different knowledge resources and argument types. In addition, we have described the position generation and selection. Finally, the argumentation protocol that the argumentative agents follow has been explained.

After performing the review, we consider that this framework has all the necessary components for our proposed infrastructure and specifies the knowledge resources of the agents and the argumentation protocol to follow. Furthermore, we consider that this framework is computationally tractable.

In the next Chapter, we propose an infrastructure that supports this argumentation framework. Our infrastructure permits to develop argumentative agents taking into account their social context, as it is specified in the argumentation framework. The infrastructure also follows the reasoning and persistence mechanisms specified in the framework.

Chapter 4

Infrastructure

4.1 Introduction

In this Chapter, the proposed infrastructure to support the argumentation framework of Chapter 3 is described. We have developed an infrastructure to develop argumentative agents in an open MAS. This infrastructure offers the necessary tools to develop agents with argumentation capabilities, including the communication skills and the argumentation protocol, and it offers support for agent societies and their agents' social context. The main advantage of having this infrastructure is that it is possible to create agents with argumentation capabilities to resolve a specified problem. We consider that this approach could obtain better results than other distributed approaches due to the argumentation process between agents and their reasoning skills. In the argumentation dialogue the agents try to reach an agreement about the best solution to apply for each proposed problem.

This infrastructure allows the users to create groups of argumentative agents to perform argumentation dialogues to reach an agreement about a solution to the given problem. The problem to solve has to be of a specified domain. Therefore, the training data has to be prepared to store it as domain-cases. A domain-case is formed basically by a list of attributes or premises with their features that specify the problem and the solution applied to solve it. The argumentative agents will use these domain-cases to learn about the problem and solve new problems by engaging in argumentation dialogues.

The components of the infrastructure and the interactions between them are represented in Figure 4.1. The main components of our infrastructure explained in this Chapter are the argumentative agents, the Commitment Store and the knowledge interchange mechanism. As we can see in Figure 4.1, there are different organizations or groups composed by some argumentative agents. Also, the Commitment Store interacts with all the argumentative agents to store the positions and the arguments generated in the argumentation dialogue. The knowledge interchange is made with concepts of a defined ontology that is used as a language representation of the cases.

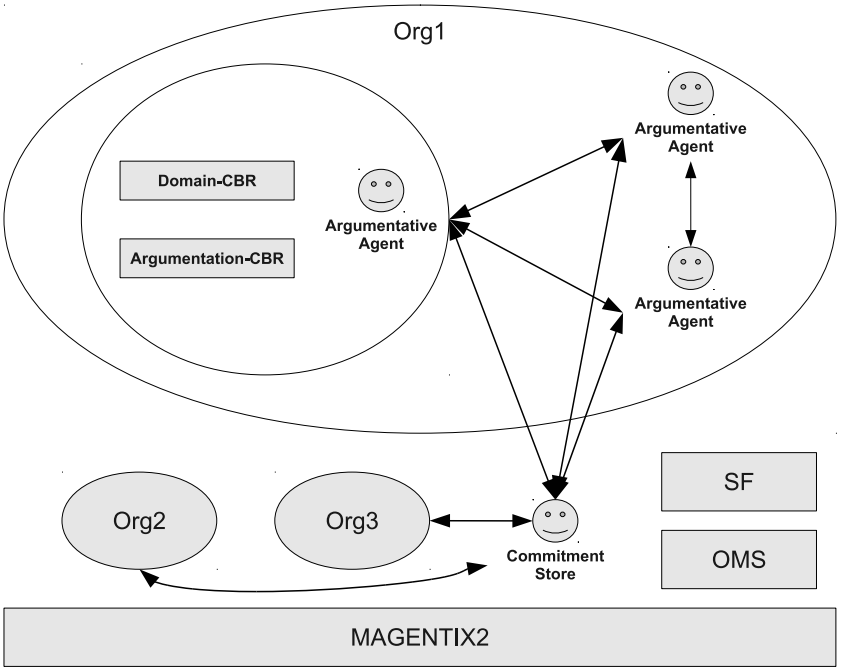


Figure 4.1: Infrastructure

The agent platform used in the implemented infrastructure is Magentix2[1]. As it has been explained in Section 2.2, this is a platform that provides new services and tools that allow for the secure and optimised management of open MAS. There are two modules of the platform, concretely from THOMAS framework, represented in Figure 4.1, the *Organization Manager Service* (OMS) and the *Service Facilitator* (SF). The OMS is in charge of managing the organizations, groups, roles and norms of the system. The SF registers the different services that can offer the agents and acts as yellow pages to find services. In our system, this platform is used for the communication between the agents. A more detailed description of the OMS and the SF can be found in [Argente et al., 2011]. The argumentative agents and the Commitment Store agent are extensions of the Magentix2 BaseAgent[2].

In this Chapter, we explain the following components of the infrastructure:

- Argumentative agents: which are the agents with argumentation capabilities to engage in an argumentation dialogue to reach an agreement about the best solution to apply to a problem. The main components of the argumentative agents are:

  - Domain CBR: which is a CBR that stores domain knowledge of previous solved problems. It is used by the argumentative agent to generate and select the position (solution) to defend in an argumentation dialogue.

[1] http://users.dsic.upv.es/grupos/ia/sma/tools/magentix2/index.php

[2] http://users.dsic.upv.es/grupos/ia/sma/tools/magentix2/archivos/javadoc/es/upv/dsic/gti_ia/core/BaseAgent.html

  - Argumentation CBR: which is a CBR that stores past argumentation experiences. It is used to select the best position to defend in the view of the acceptance that had a similar position in the past.
  - Argument management process: this includes how the positions, support arguments and attack arguments are generated by the agents using their knowledge resources and their domain and argumentation CBRs. It is also defined the argumentation mechanism, which is the argumentation protocol followed by the argumentative agents.

- Commitment Store: which is a resource of the argumentation framework developed as an agent of the platform. This resource stores all the relevant information of the argumentation dialogues. The argumentative agents communicate with the Commitment Store to give or request information about the argumentation dialogue.

- Knowledge interchange mechanism: which is the mechanism that the argumentative agents and the Commitment Store use to interchange knowledge. The agents of the infrastructure interchange their knowledge using ontologies that allow the common understanding between heterogeneous agents. They also interchange the knowledge by their communication mechanism in form of FIPA-ACL[3] messages.

The structure of packages and classes of the developed infrastructure is shown in Table 4.1 to see how the different components of the infrastructure are organized in the implementation.

4.2 Argumentative agents

The most important elements of the infrastructure are the argumentative agents. These agents have all the components needed to engage in an argumentation dialogue and reach an agreement with other agents about the best solution to apply for a problem. The solution applied to solve a problem in the past and the information about the problem-solving process can be reused to propose a solution to other similar problem. This solution is previously stored in a memory of cases (case-base) and it can either be retrieved and applied directly to the current problem, or revised and adapted to fit the new problem. Case-Based Reasoning (CBR) systems have been widely applied to perform this task [Acorn and Walden, 1992, Watson, 1997, Roth-Berghofer, 2004].

The implementation of the argumentative agents is an extension of the Magentix2 BaseAgent. The argumentative agents have two CBR based modules: Domain CBR and Argumentation CBR. These modules are shown in Figure 4.1 and are described following.

[3] http://www.fipa.org/specs/fipa00061/SC00061G.html

Package	**Classes**
algorithms	Metrics SimilarityAlgorithms
argCBR	ArgCBR OWLArgCBRParser
configuration	Configuration
database	Access
dialogueGame	ArgAgent RandAgent VotAgent
domainCBR	DomainCBR OWLDomainParser
knowledgeResources	[See Table 4.2]
test	ArgCBRTests DomainCBRTests

Table 4.1: Structure of packages and classes in the infrastructure

4.2.1 Domain CBR

The argumentative agents have their own domain CBR. This CBR is prepared to store cases of previous solved problems.

A domain-case is composed basically by a set of features or *premises* that describe the problem that the case solved and a list of solutions applied with their promoted values as shown in Figure 4.2. The cases in the case-base are organised by their features. A case is equal to another if it has the same features with the same values in each feature. Thus, to retain cases in the case-base the features are used as indexes to organise the cases. Also the features are used to extract similar cases from the case-base in the retrieve phase. A summary of the knowledge resources defined in the ontology of the infrastructure[4] to interchange knowledge is shown in Table 4.2. The knowledge resources of this Table also corresponds to the classes of the package *knowledgeResources* of the infrastructure. The most important knowledge resources are explained in this Chapter.

Different algorithms based on the *normalized Euclidean distance*, *normalized Tversky distance* and *weighted Euclidean distance* can be used to measure the similarity degree between different cases of the domain case-base. These algorithms are easy to implement and they work well with different domains. Other algorithms could be used, but there is not objective of this work to evaluate the different alternatives. To retrieve domain-cases of

[4]http://users.dsic.upv.es/∼vinglada/docs/

the case-base, a set of features that describes the problem is given to calculate and obtain the most similar domain-cases to the given problem description. Also, the list of similar domain-cases returned is limited by a threshold of similarity degree, which can be specified depending on the application domain.

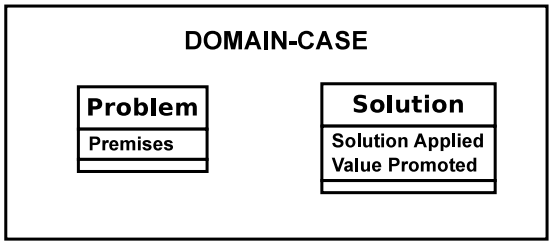


Figure 4.2: Structure of the Domain-case

Case-base indexing

We use a hash table to store the domain-cases in the case-base. This data representation allows to improve the efficiency of case extraction. However, the solution which has been chosen is not completely optimal because it could create long lists that have to be used. In any case, this strategy will always improve the simplest implementation of the base case, that would be an unordered list.

As the domain CBR is a general implementation prepared to store cases of different domains, the hash function of the hash table is not established a priori. The user can indicate if there is one of the attributes or premises that can be used as an appropriate index to build the hash function. When the data is loaded, if an index is not specified it will be used the minor premise identifier of each case as an index. This design allows to have different lists in the hash table. It is not the optimal way, but it is a good way to deal with a generic domain CBR without a specified index.

Therefore, to make the extractions of domain-cases similar to the problem to solve, the minor premise identifier of the entire premises is selected to extract the corresponding list of domain-cases. If an index was specified, the extraction would be made using it. These extraction will have constant cost O(1) and obtains a list of domain-cases that have the premise. As expected, the list could be potentially large, depending on the times that the premise appears in the domain-cases. However, a lot of cases without that premise can be discarded. Then, the domain-cases of the list that match with the problem to solve will be selected. This has a lineal cost O(n).

Retrieve

As it has been commented previously, to retrieve a domain-case of the case-base it is necessary to use the specified index. If an index has not been specified, the extraction is made taking

the minor premise identifier of the desired premises to extract the domain-cases that match those premises.

There are two different methods in the domain CBR to retrieve domain-cases:

- *retrieveAndRetain(DomainCase, threshold)*: This method returns a list of similar domain-cases to the domain-case specified as parameter, with a similarity degree greater than the specified threshold. This similarity degree is obtained with the *normalized Euclidean distance*, *normalized Tversky distance* or *weighted Euclidean distance* between the premises, depending on the specified in the configuration file of the infrastructure. In addition, the domain-case specified as parameter is stored in the case-base by using the method *addCase*.

- *retrieve(premises, threshold)*: This method follows the same procedure that the last one, but without retaining the domain-case. Furthermore, the first parameter is not a complete domain-case, but a *HashMap*[5] of premises.

Retention

It is very important to perform a retention phase in a CBR because it contributes to learn new cases and adapt the system to new conditions.

In the domain CBR, two cases are considered equal only if they have the same premises or attributes with the same values. If the case does not exist in the case-base it is stored. If it exist, the associated solutions will be added to enrich the domain-case. This functionality is implemented in the method *addCase*.

The retention of cases is normally made when the list of similar domain-cases are retrieved. This is implemented in the *retrieveAndRetain* method explained before, that includes a call to the method *addCase*. Furthermore, when the agent calls to the method *doCache*, it will store all the case-base in a file that is specified.

Case-base persistence

The load and persistence of the domain case-base has been created by using ontologies. Concretely, an ontology has been defined using the OWL 2[6] language, as commented before. In this way, heterogeneous agents can use it as common language to interchange solutions and arguments generated from the case-bases of the argumentation framework.

There is a method that loads the domain case-base of a file that is specified making a call to the OWL 2 parser. Then, each domain-case read is added to the domain CBR. In addition, there is another method that stores the domain case-base in a file that is specified.

[5] http://download.oracle.com/javase/1.4.2/docs/api/java/util/HashMap.html
[6] http://www.w3.org/TR/owl2-overview/

This function makes a call to the OWL 2 parser and converts the domain-cases to the ontology classes and stores the ontology in the file.

4.2.2 Argumentation CBR

The Argumentation CBR consists of a CBR module with argumentation data. This CBR stores as argument-cases arguments previously used in argumentation dialogues as argument-cases. The structure of the Argument-case is shown in Figure 4.3.

The argumentative agents have their own argumentation CBR. This module is used to generate better arguments in the argumentation dialogues taking into account similar previous argumentation experiences where similar solutions were proposed. Thus, the best argument to propose in the current problem to solve will be selected in view of the acceptance that had a similar argument in the past. Therefore, argument-cases store information related to the domain and the social context where previous arguments (and their associated solution) were used. The information about the domain consists of a set of features to compare cases and information about the social context where the solution was applied (the agents that participated in the dialogue, their roles or their value preferences). The latter information can determine if certain arguments are more persuasive than others for particular social contexts (their acceptability status was set to *accepted* at the end of the dialogue where the argument was put forward) and hence, agents can select the best solution to propose and an argument to support it.

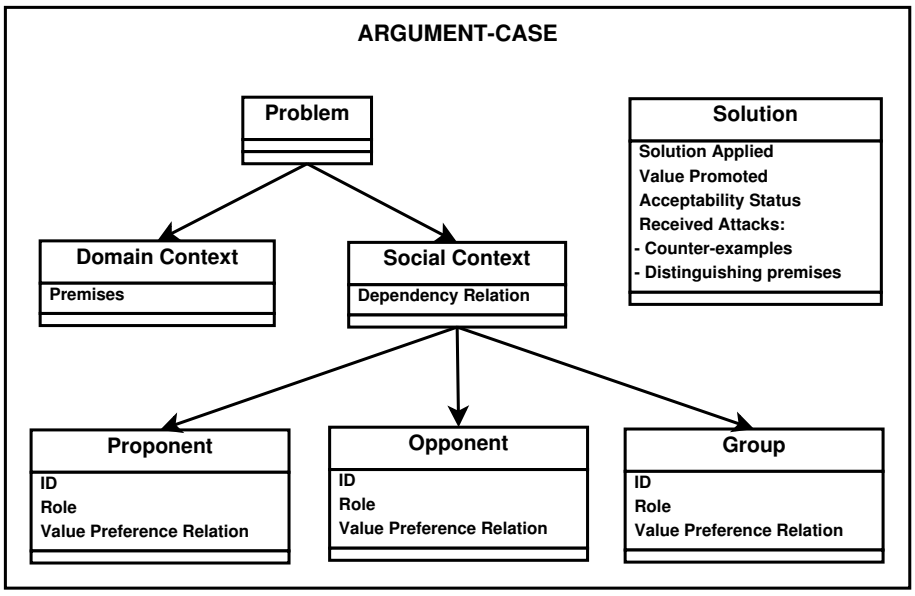


Figure 4.3: Structure of the Argument-case

Case-base indexing

To store argument-cases in a case-base, we use a hash table to improve the efficiency of extraction. The solution which has been chosen is the same as that used in the Domain CBR, but always without any index. As explained before, this implementation is not optimal but it improves the simplest approach.

The hash function of the hash table is based on the premises (or attributes) of the domain context of the argument-case. Specifically, they are sorted based on the minor premise identifier of the entire premises of the domain context. This design is appropriate because the queries of the cases are based on the domain context. That is, extract the argument-case that dealt with the specific domain context in the past.

Therefore, to extract an argument-case or a set of argument-cases, the minor premise identifier of the entire premises is selected to extract the corresponding list of argument-cases. This extraction will have constant cost O(1) and get a list of argument-cases that have the premise. As expected, the list could be potentially large, depending on the times that the premise appears in the argument-cases. However, a lot of cases without that premise can be discarded. Then, the argument-cases of the list that match with the desired will be selected. This has a lineal cost O(n).

The argument-cases of the case-base are composed by: *ArgumentProblem*, *ArgumentSolution*, *ArgumentJustification* and *timesUsed*. The first three attributes are instances of classes that represent the problem, solution, and the justification of the case. Also, the attribute *timesUsed* is the number of times that the case has been used. A summary of the knowledge resources defined in the ontology of the infrastructure to interchange knowledge is shown in Table 4.2. Following, we explain the three classes of the argument-case:

ArgumentProblem: It consists of the domain context and the social context of the argument-case.

- *DomainContext*: Domain context of the argument-case. It is composed by:
  - *Premises*: HashMap with the premises of a domain-case. Implemented on a HashMap to improve the extraction having a constant cost O(1) and to facilitate comparisons. Each element of the HashMap is formed by a pair <integer, premise> where the integer is the identifier of the premise.
- *SocialContext*: Social context of the argument-case. It is composed by:
  - *Proponent*: Proponent of the argument. It is an instance of *SocialEntity* class.
  - *Opponent*: Opponent of the argument. It is an instance of *SocialEntity* class.
  - *Group*: The proponent and the opponent belong to the Group. It is an instance of *Group* class.

  - *DependencyRelation*: Dependency relation between the proponent and the opponent. This dependency relation represents different types of hierarchies or contracts between the roles of the agents. Can be one of the following three types:
    * Power: when an agent has to accept a request from another agent because of some pre-defined domination relationship between them.
    * Authorisation: when an agent has committed itself to another agent for a certain service and a request from the latter leads to an obligation when the conditions are met.
    * Charity: when an agent is willing to answer a request from another agent without being obliged to do so.

ArgumentSolution: It contains different premises and cases associated to the solution of the argument-case, as well as their type and acceptability status.

- *ArgumentType*: Argument type. It can be:
  - *Inductive*: which uses the premises that describe the problem to solve.
  - *Presumptive*: which uses similar argument-cases and/or domain-cases stored in the case-bases of the system.
  - *Mixed*: which uses premises and cases.
- *AcceptabilityStatus*: Acceptability status of the argument. It can be *acceptable*, when the argument has been accepted; *unacceptable*, when the argument has been rejected; and *undecided*, when the argument has not been accepted or rejected yet.
- *Distinguising premises*: List of distinguishing premises. These premises can invalidate the application of a knowledge resource to generate a valid conclusion for an argument.
- *Counter-examples*: List of cases, domain-cases or argument-cases that represent counter-examples. Counter-examples are cases similar to a case (their descriptions match with some or all features of the problem description) but have different conclusions.

ArgumentJustification: It is the justification of the argument-case:

- *Cases*: List of cases associated to the argument-case to justify its validity. They can be domain-cases or argument-cases.

Retrieve

In the argumentation CBR the argument-cases can be retrieved in different ways depending on the interests of the agent that use it. This means that the agent can get cases that have been attacked, accepted, and various combinations, depending on its interests. Nevertheless, there is a common method used in all extraction recoveries. This extraction (method *getMostSimilarArgCases(ArgumentCase)*) refers to obtaining the argument-cases that have the same domain context and a similar social context. The similarity of the social context is computed by some parameters and adjusted with some weights specified in the configuration file of the infrastructure.

The following methods for obtaining lists of argument-cases that meet certain conditions are used to calculate the suitability factor of the Equation 3.9:

- getSameProblemAcceptedArgCases(ArgumentCase): Get similar argument cases to the given one with the same problem description and accepted.
- getSameProblemConclusionArgCases(ArgumentCase): Get similar argument cases to the given one with the same problem description and conclusion.
- getSameProblemConclusionAcceptedArgCases(ArgumentCase): Get similar argument cases to the given one with the same problem description, conclusion and accepted.
- getSameProblemConclusionAcceptedAttackedArgCases(ArgumentCase): Get similar argument cases to the given one with the same problem description, conclusion, accepted and with attacks.
- getAttackDegree(ArrayList<ArgumentCase>, ArrayList<Position>): Returns the attack degree of each given position.
- getEfficiencyDegree(ArrayList<ArgumentCase>, ArrayList<Position>): Returns the efficiency degree of each initial position.
- getExplanatoryPower(ArrayList<ArgumentCase>, ArrayList<Position>): Returns the explanatory power of the given positions.
- getDegrees(ArgumentProblem, Solution, ArrayList<Position>, index): Returns a list with the degrees (attack, efficiency, explanatory power, persuasiveness, support and risk) of an argument problem.

The four first methods explained above are used to calculate the different degrees. The next three methods obtain the commented degrees as their names indicate. And finally, the last method is the one that uses the others to compute all the degrees: Persuasiveness Degree (Equation 3.3), Support Degree (Equation 3.4), Risk Degree (Equation 3.5), Attack Degree

(Equation 3.6), Efficiency Degree (Equation 3.7) and Explanatory Power (Equation 3.8). This method is to be used by the agent to obtain the degrees and calculate the suitability factor of the position, and then, its final suitability (Equation 3.10).

Retention

Two cases are considered equal only if they have the same domain context, social context, conclusion and acceptability status. If a new case generated during the argumentation process does not exist in the case-base it is stored. If it exist, the associated domain-cases and attacks received will be added to enrich the argument-case. This functionality is implemented in the method *addCase*.

The argument-case retention is made once the argumentation process has finished. Each generated argument in the process is taken to call the method *addCase*. Therefore, the case is added to the CBR, that is, in the case-base in memory. After, when the agent calls to the method *doCache*, it will store all the case-base in a specified file.

Case-base persistence

As with by the domain CBR, the load and persistence of the argumentation case-base has been created using ontologies. Concretely, an ontology has been defined using the OWL 2 language as commented before. In this way, heterogeneous agents can use it as common language to interchange solutions and arguments generated from the case-bases of the argumentation framework.

To store the argument-cases case-base in a file that is specified we use the same approach than with the domain CBR. A method loads the argument-cases making a call to the OWL 2 parser and taking the data from a file that is specified. Furthermore, another method converts all the argument-cases to data in OWL 2 using the parser. Then, the OWL 2 data is stored in a file that is specified.

4.2.3 Argument Management Process

The argument management process that the argumentative agents perform is described below. The argument management process includes: position generation, support arguments generation, attack arguments generation and argumentation mechanism.

Position generation

A position is a solution that defends an agent as the correct one to apply to the problem to solve. The position generation is made in two steps. First, the agent searches in its domain CBR the most similar domain-cases to the current problem to solve. With them, the agent is able to propose its position. Then, the agent evaluates the suitability of each position

using the argumentation CBR. To do that, such argument-cases that its features match with the domain-cases extracted are retrieved. Then, each position is evaluated in function of the chances of being well defended. As this evaluation is based on argument-cases, the best position to propose in the current case to solve will be selected in view of the acceptance that had a similar argument in the past.

To perform this evaluation, the argumentative agents use the similarity degree that the domain CBR returned from the selected domain-case and the different degrees calculated by the Argumentation CBR. Formula 3.9 obtains the Suitability Factor from the argumentation point of view of positions with the different degrees explained in Section 3.3. Finally, Formula 3.10 computes the final suitability of a position, taking into account all the degrees and its similarity to the problem to solve. In the framework, the weights of the formulas can be established when the agent is created.

Algorithm 1 generatePositions

Require: ProblemDescription {*The description of the problem to solve*}

```
1: matchCases = ∅
2: solutions = ∅
3: positions = ∅
4: similarity = 0
5: SD = ∅
6: i = 0; j = 0; k = 0
7: for all c ∈ DomainCasesCB do
8:    similarity = computeSimilarity(ProblemDescription, c)
9:    if similarity > δ then
10:      matchCases[i] = c {If the similarity exceeds certain threshold, the domain-case is
         selected to generate the position}
11:      SD[i] = similarity {The similarity degree of this domain-case is stored}
12:      i++
13:   end if
14: end for
15: solutions = generateSolutions(matchCases)
16: if lenght(solutions) ≥ 1 then
17:   for [s = 0;s < lenght(solutions);s + +] do
18:      positions = addPosition(ProblemDescription, solutions[s], SD[i])
19:   end for
20: end if
21: Return positions
```

The method that generates the positions is called *generatePositions*. It receives a problem

description as parameter and return a list of positions to defend, ordered from more to less suitable. The pseudocode of this method is shown in Algorithm 1.

Support arguments generation

A support argument is an argument that justifies a position. The support set of this kind of argument can be formed by argument-cases, domain-cases and premises. These cases and premises justify the solution defended by the position since the features of the problem match and they promote the same value and solution. To generate a support argument for a position, the argumentative agents search for similar argument-cases that can justify the current position. Also, they include the domain-case that was used to create the position in the support set. Finally, a list of possible support arguments is generated with different combinations of the available support elements in the support set. This list is ordered by a suitability degree [Heras et al., 2011b] in terms of the acceptance that had the similar argument-case in the past taking into account the domain and social contexts.

The argumentative agents use the method *generateSupportArguments* to create support arguments for a position. This method receives as parameters the current position and the opponent agent identifier to take into account the social context and the possible support arguments. It returns a list of support arguments, ordered from more to less suitable.

Attack arguments generation

An attack argument is an argument that attacks a support argument or another attack argument. The attack argument has a different solution to the argument attacked. To generate the attack argument, the premises that has the argument to attack and the social context (the relation with the other agent) are taken into account. With this information, the argumentative agents extract the argument-cases that match with the current position and have a similar social context. Then, if the dependency relation permits to attack the other agent, the attack will be based on the most suitable argument-case extracted in terms of the acceptance that had the argument-case in the past. The argumentative agents always try to generate a counter-example attack in the first time, but a distinguishing premise attack will be generated when all possible counter-example attacks had been used without success.

The argumentative agents have a method called *generateAttackArgument* that generates and attack argument if it is possible. The pseudocode of this method is shown in Algorithm 2. It receives as parameters the argument to attack and the opponent agent identifier. The method *generateAttackArgument* uses two different methods to generate the two types of attacks:

- *generateCEAttack*(ArrayList<SimilarArgumentCase>, HashMap<Integer, Premise>, DependencyRelation, opponentAgentID): this method generates a counter-example

attack argument. The parameters are the list of argument-cases that match with the domain and social context of the defended position, the premises of the opponent's position, the dependency relation between the agents, and the opponent agent identifier.

- *generateDPAttack*(ArrayList<SimilarArgumentCase>, HashMap<Integer, Premise>, DependencyRelation, opponentAgentID): this method generates a distinguishing premises attack argument. The parameters are the same as the *generateCEAttack* method.

Algorithm 2 generateAttackArgument

Require: incArgument, DomainCasesCB, ArgumentCasesCB {*The argument to attack and the case-bases of domain-cases and argument-cases*}

```
1: support = checkSupportSet(incArgument)
2: supportElement = selectElementToAttack(support)
3: if (supportElement = Domain-Case) or (supportElement = Argument-Case) then
4:    CE=generateCounterExample(incArgument)
5:    if CE ≠ ∅ then
6:       generateCEAttack(incArgument)
7:    else
8:       generateDPAttack(incArgument)
9:    end if
10: end if
11: if supportElement = Premises then
12:    generateDPAttack(incArgument)
13: end if
14: if support = ∅ then
15:    generateCEAttack(incArgument)
16: end if
```

Argumentation mechanism

The agents of the infrastructure need a mechanism to manage the arguments and perform the argumentation dialogue. Therefore, an argumentation protocol (Figure 3.1) has been defined in [Jordán et al., 2011a], as explained in Section 3.4. This protocol is represented by a set of locutions that the agents use to communicate each other depending on their needs, and a state machine that defines the behaviour of an agent in the argumentation dialogue.

The state machine has been implemented inside a loop which has one case for each state. Also, in each state the different locutions that can be received and generated are taken into account to act in consequence following the argumentation protocol. Inside each state of

the protocol, the corresponding actions are performed using the necessary calls to different functions of the agent.

There are some states where the agent has to wait some time to receive messages. In these states, the wait time is performed by sleeping the agent. The agents have a method called *listenAndReviseQueue* that waits some specified time and then it revises the messages queue to search if there is a message with the locution that the agent is waiting. Normally, the strategy of waiting for a type of message consists on executing several times the method *listenAndReviseQueue* with a short time to avoid checks if the message arrives soon.

4.3 Commitment Store

The Commitment Store is a resource of the argumentation framework that stores all the information about the agents participating in the problem-solving process, argumentation dialogues between them, positions and arguments. By making queries to this resource, every agent can read the information of the dialogues that it is involved in. In the infrastructure, the Commitment Store has been implemented as an agent to allow a good communication with the other agents. Concretely, it is an extension of the Magentix2 SingleAgent[7] that it is also an extension of the Magentix2 BaseAgent, as the argumentative agents.

This agent has a method that makes a passive wait for messages. This means that the agent is waiting for messages without consuming resources of the system. Therefore, when the Commitment Store receives a message it obtains the locution of it and treat the data as corresponds. It could be a message to store something about the dialogue (e.g. positions, arguments, etc), or to give some information to an argumentative agent participating in a dialogue. If it is the second case, the Commitment Store sends a message with the requested information and the corresponding locution.

In Figure 4.4, an example of messages interchange between an argumentative agent and the Commitment Store is shown. The argumentative agent of the example makes a request to the Commitment Store to get the position of another argumentative agent (ArgAgent2). Then, the Commitment Store responds to the request with a message that has attached (an object of the defined ontology) the position requested by the argumentative agent.

4.4 Knowledge interchange mechanism

The case-bases of the domain CBR and the argumentation CBR are stored as OWL 2[8] data of an ontology[9] that we have designed to act as language representation of the cases.

[7] http://users.dsic.upv.es/grupos/ia/sma/tools/magentix2/archivos/javadoc/es/upv/dsic/gti_ia/core/SingleAgent.html

[8] http://www.w3.org/TR/owl2-overview/

[9] http://users.dsic.upv.es/~vinglada/docs/

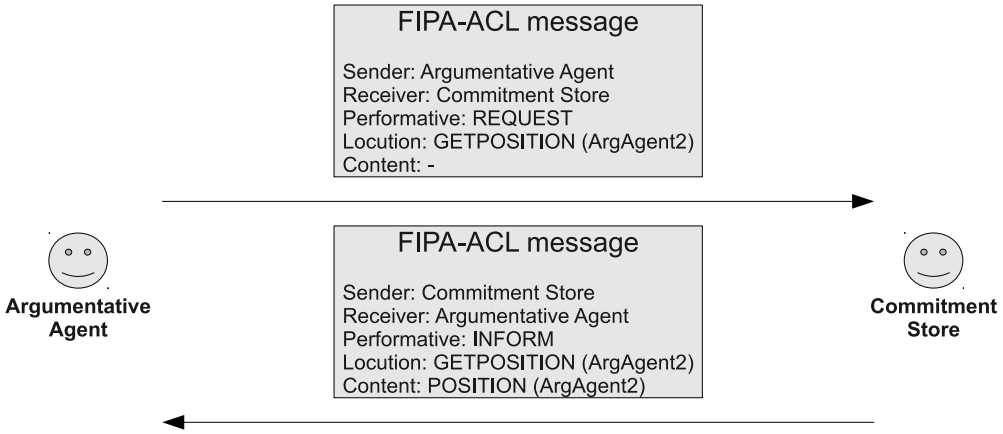


Figure 4.4: Messages interchange between an argumentative agent and the Commitment Store

In this way, heterogeneous agents can use it as common language to interchange solutions and arguments generated from the case-bases of the argumentation framework. The main advantage of using ontologies is that the structures and features of the cases are well specified and heterogeneous agents can easily understand them. The ontology parsers developed, by using the OWL API[10], provide an API to read and write data in the case-bases.

The agents of the infrastructure are extensions of the Magentix2 BaseAgent. Then, the communication is performed with the mechanisms that the Magentix2 platform offers. The Magentix2 BaseAgent has a method called *onMessage* that is executed when the agent receives a FIPA-ACL message. Therefore, the agents of the infrastructure have a queue of messages to store the messages that arrive at any time. When the *onMessage* method is executed, the message that arrives is stored in the messages queue. Only when it is a special message that indicates the end of the dialogue or the agent has to be killed a different behaviour is performed and the message is not stored. In addition, there is another method called *sendMessage* in the agents of the infrastructure that is used to send messages to other agents. This method receives as parameters the name of the receiver agent, the locution of the message, the dialogue identifier and the object to send (if necessary). The *sendMessage* method creates the message to send and include all the corresponding information. Then, it uses the *send* method of the Magentix2 BaseAgent to send the message to the receiver. Note that the included object (if it is necessary) is an instance of a class that represents a concept of the defined ontology. In this way, heterogeneous agents can understand each other.

[10]http://owlapi.sourceforge.net/

Class	**Description**
Argument	Argument of the argumentation process
ArgumentationScheme	Argumentation scheme
ArgumentCase	Argument case stored in the case-base
ArgumentJustification	Justification of an argument-case
ArgumentNode	Node of an argument in a dialogue graph
ArgumentProblem	Problem that treat an argument-case
ArgumentSolution	Solution of an argument-case
Author	Author of an argument
Case	Case. Specialized in domain-case or argument-case
CaseComponent	Case Component. Empty class specialized in Justification, Problem and Solution
Conclusion	Conclusion or solution of a problem
Context	Empty class specialized in DomainContext and SocialContext
DialogueGraph	Dialogue graph
DomainCase	Domain case
DomainContext	Domain context
Group	Group of agents. It has its preference value. It extends SocialEntity
Justification	Justification of an argument-case. Counter-examples, premises...
Norm	Norm of a SocialEntity
Premise	Premise that belongs to a DomainCase or an ArgumentCase
Problem	Problem represented in a DomainCase or an ArgumentCase
SimilarArgumentCase	Argument case with a numeric attribute of similarity
SocialContext	Social context where it is defined the relation between two SocialEntity
SocialEntity	Social entity that has the data that represents an agent
Solution	Solution of the problem that represents a DomainCase or an ArgumentCase
SupportSet	Support set of an argument
ValPref	Preference value and ordered list of the preference values
Value	Value promoted by a Solution, SocialEntity or Group
ValueNode	Value node

Table 4.2: Classes of the package *knowledgeResources*

Chapter 5

Call Centre Example

In this chapter, we give an example of a call centre application that uses the implemented infrastructure. This example is intended to validate the proposed infrastructure.

5.1 Customer support application

Nowadays, companies have to offer a good customer support to take an advantage over their competitors. A good customer support depends, in many cases, on the experience and skills of its operators. A quick and accurate response to the customers problems ensures their satisfaction and a good reputation for the company and, therefore, it can increase its profits.

A common customer support system settled in a company consists of a network of operators that must solve the incidences (also known as *tickets*) received in a Technology Management Centre (TMC). TMCs are entities which control every process implicated in the provision of technological and customer support services to private or public organizations. In a TMC, there are a number of operators whose role is to provide the customers with technical assistance. This help is commonly offered via a call centre. The call centre operators have computers provided with a helpdesk software and phone terminals connected to a telephone switchboard that balances the calls among operators. Commonly, the staff of a call centre is divided into three levels: 1) Base operators, who receive customer queries and answer those ones from which they have background training; 2) Expert operators, who are the technicians in charge of solving new problems; 3) Managers, who are in charge of organising working groups, of assigning problems to specific operators and of creating generic solutions.

The solution applied to each problem and the information about the problem-solving process could be a suitable way to improve the customer support offered by the company. The suitability of CBR systems in helpdesk applications to manage call centres has been guaranteed for the success of some of these systems from the 90s to nowadays [Acorn and Walden, 1992, Watson, 1997, Roth-Berghofer, 2004].

These approaches propose systems for human-machine interaction where the CBR

functionality helps the operators to solve problems more efficiently by providing them with potential solutions via the helpdesk software. We have applied the case-based argumentation infrastructure for agent societies presented in this work to extend a previous work that presented a CBR module that acts as a solution recommender for customer support environments [Heras et al., 2009]. The CBR module is flexible and multi-domain. However, to integrate the knowledge of all experts in a unique CBR module can be complex and costly in terms of data mining (due to extra large case-bases with possible out-of-date cases). Moreover, to have a unique but distributed CBR could be a solution, but to assume that all operators are willing to share unselfishly their knowledge with other operators is not realistic. In this case, the modelling of the system as a MAS will be adequate. Finally, several experts could provide different solutions and hence, they need a mechanism to negotiate and reach an agreement about the best solution to apply.

Now, we propose to automate the system by representing the operators by means of software agents that can engage in an argumentation process to decide the best solution to apply for each new incidence. The case-based argumentation infrastructure for agent societies presented in this work has been applied to a prototype that provides support to the operators and experts of a call centre via a helpdesk application.

In our prototype, the operators and experts of a call centre are represented by agents that use an automated helpdesk and argue to solve an incidence. Every agent has individual CBR resources and preferences over values (e.g. economy, quality, solving speed). A solution to a problem promotes one value. Thus, each agent has its own preferences to choose a solution to propose. Furthermore, agents can play two different roles: *operator* and *expert*. The main difference between an operator and an expert is that the second one has more specific domain knowledge to solve certain types of problems. Also, dependency relations between roles could imply that an agent must change or violate its value preference order. For instance, an expert could impose their values to an operator and the last could have to adopt a certain preference order over values. Therefore, we endorse the view of [Bench-Capon and Atkinson, 2009], who stress the importance of the audience in determining whether an argument is persuasive or not for accepting or rejecting someone else's proposals. The data-flow for the argumentation process of the helpdesk application presented can be seen in [Jordán et al., 2011b].

In order to show how the prototype works, the data-flow for the problem-solving process to solve each ticket is shown in Figure 5.1 and described below:

1. Some argumentative agents run in the platform and represent the operators of the call centre. The manager of the group acts as the *initiator* of the dialogue. This agent has a special behaviour to receive tickets to solve and create a new dialogue with the agents of his group. The process begins when a ticket that represents an incidence to solve is received by the initiator agent. Then, this agent sends the ticket to the agents of their group.

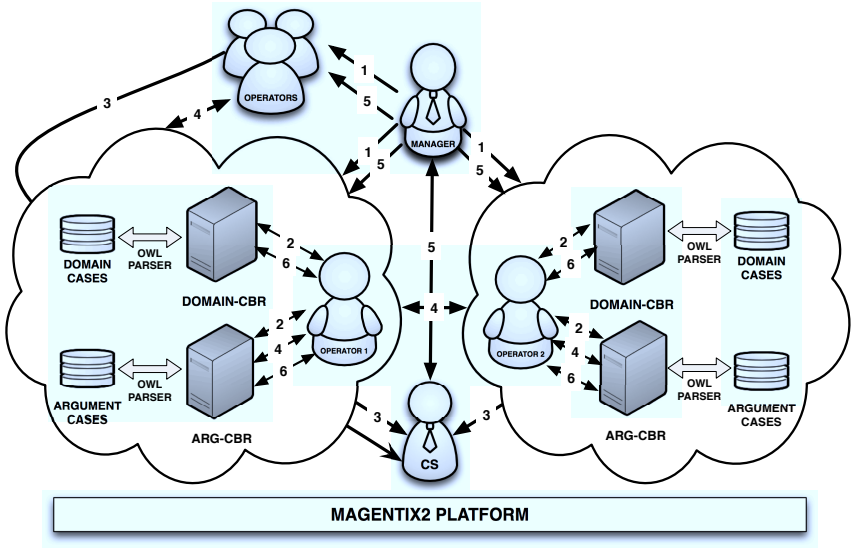


Figure 5.1: Data-flow for the argumentation process of the helpdesk application

2. Each agent evaluates if it can engage in the dialogue offering a solution. To do that, the agent makes a query to its domain CBR to obtain potential solutions to the ticket based on solutions applied to similar tickets. If one or more valid solutions are retrieved, the agent will be able to defend a position in the dialogue. A valid solution is any solution derived from a domain-case of the domain CBR with one or more solutions and with a similarity degree greater than a given threshold. Moreover, the agent makes a query to its argumentation CBR for each possible position to defend. With these queries a *suitability degree* of the positions is obtained. This degree represents if a position will be easy to defend based on past similar argumentation experiences. Then, all positions to defend are ordered and proposed from more to less suitability degree [Heras et al., 2011b].

3. When agents have a position to defend (a proposed solution), these positions are stored by the commitment store agent. Thus, other agents can check the positions of all dialogue participants. Every agent tries to attack the positions that are different from its position.

4. The argumentation process consists on a series of steps by which agents try to defend their positions by generating counter-examples for the positions and arguments of other agents. A counter-example for a case is generated by retrieving from the domain case-base other case that matches the features of the former, but has a different conclusion. If different counter-examples can be generated, agents select the best attack to rebut the position of other agent by making a query to their arguments case-base, extending each case with the current social context. In this way, agents can gain knowledge about how each potential counter-example worked to attack the position of an agent in a past

argumentation experience with a similar social context.

5. The dialogue finishes when certain time has passed without new positions or arguments proposed. The initiator agent makes queries to the commitment store agent to determine if the dialogue must finish. Then, this agent retrieves the active positions of the participating agents and the most frequent will be the solution (or a random choice in case of draw). The initiator agent communicates the solution to the participating agents.

6. Finally, each agent updates its argumentation CBR with the new arguments produced in dialogue and its domain CBR with the final solution applied.

5.2 Evaluation

In this Section, we make different evaluations of our prototype to validate the infrastructure. Firstly, we explain the unitary tests performed to validate the correct operation of the different modules of the infrastructure. Then, we evaluate the percentage of solved problems in respect to the domain-cases case-bases of each agent. Similarly, we analyze the prediction error with different combinations of operators, experts and policies. Also, we show the learning of the CBRs with the increasing of their cases with the iterations. Finally, we analyze the generated locutions in the argumentation dialogues to see how the agents argue in different conditions.

In the following tests shown, the domain-cases case-bases of each agent are populated randomly by using some of the 48 cases of a case-base of computer problems, increasing the number of cases from 5 to 45 cases in each round. Each problem is described by a set of features (e.g. the type of problem, the log provided by the system, etc.) and the description of the solution applied. To diminish the influence of random noise, all results report the average of 48 simulation runs per round. In each round, an agent is selected randomly as initiator of the process. This agent has access to the whole case-base of computer problems and in each run takes the corresponding case to solve and sends it to the other agents. In this way, the initiator knows which was the real solution applied to the problem and can compare this value to the solution decided by the agreement process. To make the evaluation the tests have been performed with the following decision policies:

- CBR-Random (CBR-R): which consists on choose randomly a solution of those proposed by the agents by using its individual case-base. Each agent proposes a solution if it is possible, but without any argumentation process.

- CBR-Majority (CBR-M): which consists on selecting the solution most frequently proposed by the agents, again using a CBR methodology, and also without any argumentation process.

- CBR-Argumentation (CBR-A): where agents are provided with the proposed case-based argumentation functionalities and perform an argumentation dialogue to select the best solution of those proposed by the group.

In all decision policies, agents propose solutions using its own CBR. So, an agent will be able to propose a solution if in its CBR there is a case that match with the ticket to solve.

In the performed tests, we use two different configurations: groups with different combinations of operators; and groups with operators and experts. The main difference between these two configurations is that in the second configuration, some agents have been allowed to play the role of an *expert*, while the rest of agents play the role of *operators*. An expert is an agent that has specific knowledge to solve certain types (categories) of problems and has its case-base of domain-cases populated with cases that solve them. Thus, the expert domain-cases case-base has as much knowledge as possible about the solution of past problems of the same type. That is, if the expert is configured to have 5 domain-cases in its domain-cases case-base, and there are enough suitable information in the original tickets case-base, these cases represent instances of the same type of problems. In the case that the tickets case-base has less than 5 cases representing such category of problems, 3 for instance, the remaining two cases are of the same category (if possible).

In our case, the expert agent has an *authorisation* dependency relation over other technicians. This dependency relation means that when an agent has committed itself to other agent for a certain service, a request from the latter leads to an obligation when the conditions are met. Therefore, if the expert agent is able to propose a solution for the ticket requested, it can generate arguments that support its position and that will defeat other operators' arguments, due to the defeat relation among arguments. This relation assigns more importance to the arguments of an agent that has an authorisation dependency relation over other agents. However, in the CBR-Random and the CBR-Majority policies there is not argumentation dialogue, so this dependency relation is not taking effect, but the proposals of the expert have the same influence than other operators proposals in the final solution selected.

5.2.1 Unitary Tests

Following, we explain the unitary tests performed to validate the correct operation of the modules of the infrastructure.

Domain and Argumentation CBRs

Here we describe the unitary tests developed to verify the Domain and Argumentation Case-Based Reasoning (CBR) modules of this project. The objective of these verification tests is to check that the modules behave as expected and return the correct values. With this aim

the non-existence of the following elements is verified:

- Inaccuracies: cases that used as queries do not match themselves.
- Duplications: duplicated cases in the case-base.
- Inconsistencies: identical queries that give rise to contradictory conclusions.

The verification tests are based on those proposed in [Althoff et al., 1995, Watson, 1997]. To implement repeatable tests, the *JUnit*[1] framework has been used. First, we use a class for each CBR (Domain and Argumentation) that runs a suite with all domain verification tests adapted to each CBR. In these classes, a function called `setUp` creates an instance of the `DomainCBR` class or the `ArgCBR` class, which initialize the Domain CBR or the Argumentation CBR and loads all default cases.

The tests are executed for each of the retrieval algorithms used (*normalized Euclidean similarity*, *normalized Tversky similarity* and *weighted Euclidean similarity*).

Retrieval Accuracy: If you are using nearest-neighbour or inductive retrieval, a case should exactly match itself. To perform this test the case-bases of the CBRs are loaded with all default cases. Then, each case is used to query the CBR. As every case was loaded in the case-base, these queries should retrieve a case for each query with 100% of similarity. If a case does not retrieve itself or does not match exactly, there is something wrong with the algorithms used in the retrieval step of the CBR cycle.

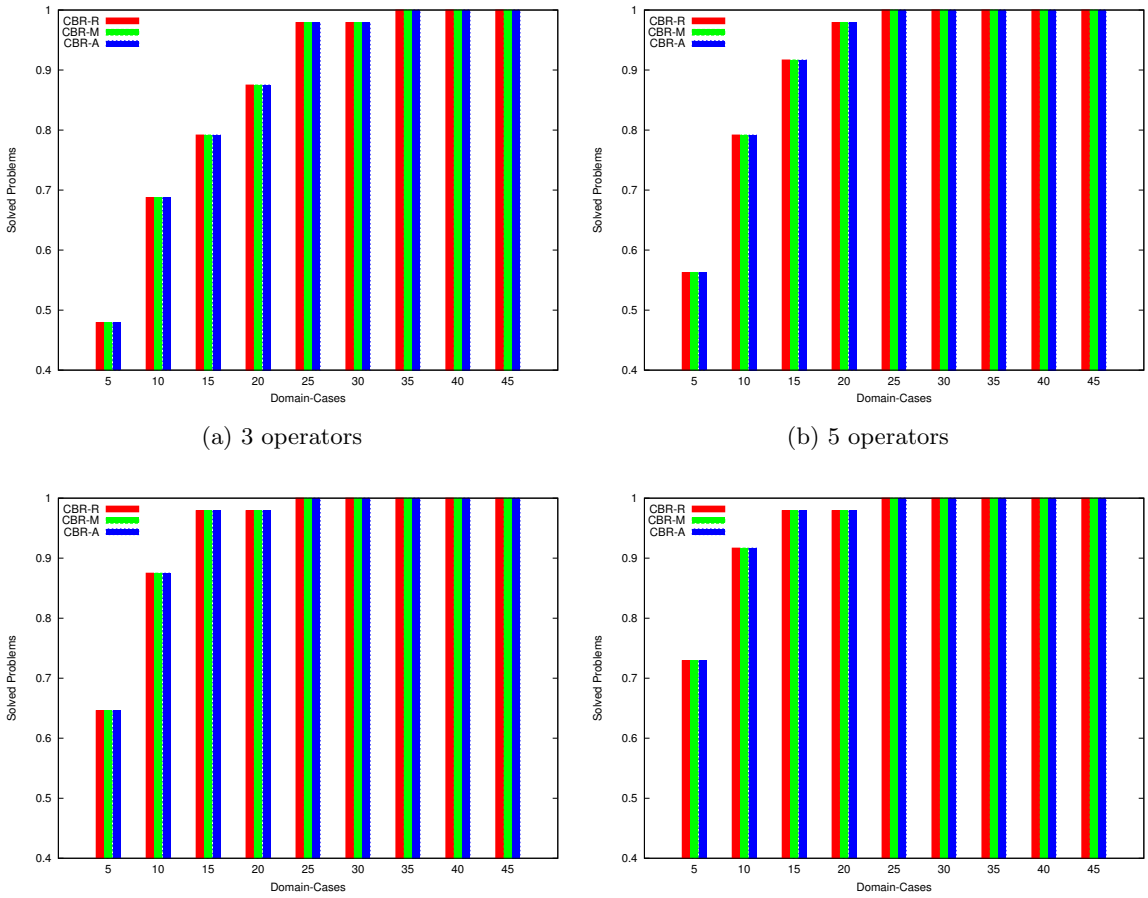
Retrieval Consistency: Using nearest-neighbour or inductive retrieval, if you perform the same search twice it should retrieve the same cases from the case-base and with the same degree of similarity. To perform this test the case-bases of the CBRs are loaded with all default cases. Then, each case is used to query the CBR. For each query, the CBR should retrieve the same cases with the same percentage of similarity. Otherwise, something is wrong with the retrieval algorithms.

Case Duplication: A case should match itself, but it should not be identical to other case in the case-base. Duplications do not make the system to provide incorrect answers, but affect its performance and increases its response time. To perform this test the case-bases of the CBRs are loaded with all default cases. Then, each case is used to query the CBR. Finally, the list of similar cases is checked to ensure that there are no duplicates in the list.

The three explained tests have been executed with the Domain CBR and the Argumentation CBR obtaining the expected results. Therefore, we can say that these modules are implemented correctly to perform their functionalities.

[1] http://www.junit.org/

5.2.2 Percentage of problems that were solved with respect to the knowledge of the agents



(a) 3 operators

(b) 5 operators

(c) 7 operators

(d) 9 operators

Figure 5.2: Solved problems

In the tests shown in Figure 5.2, we make an evaluation of the percentage of solved problems (agents provide a solution if possible, regardless of its level of suitability) with respect to the size of the case-bases of domain-cases of the agents. In all cases, all policies achieve the same results. As expected, the percentage of problems solved by the system increases with the number of domain-cases. Also, as the number of operators grows, the percentage of solved problems is higher. This is because with more operators there is more knowledge, and it is more probable that one or more operators know the solution.

Figure 5.3 shows the percentage of solved problems of a group with 6 operators and an expert. The main difference between these results and the previous is that the percentage of solved problems is higher than with the other combinations of operators without an expert. The reason of these results is that the expert contributes with more knowledge and hence, it is more probable to give a solution to solve the problem.

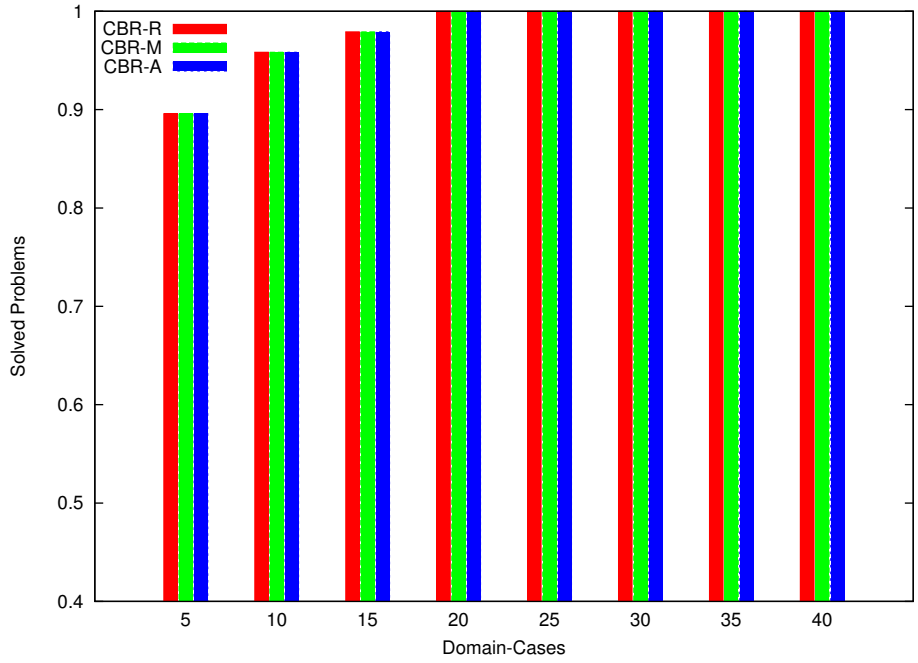


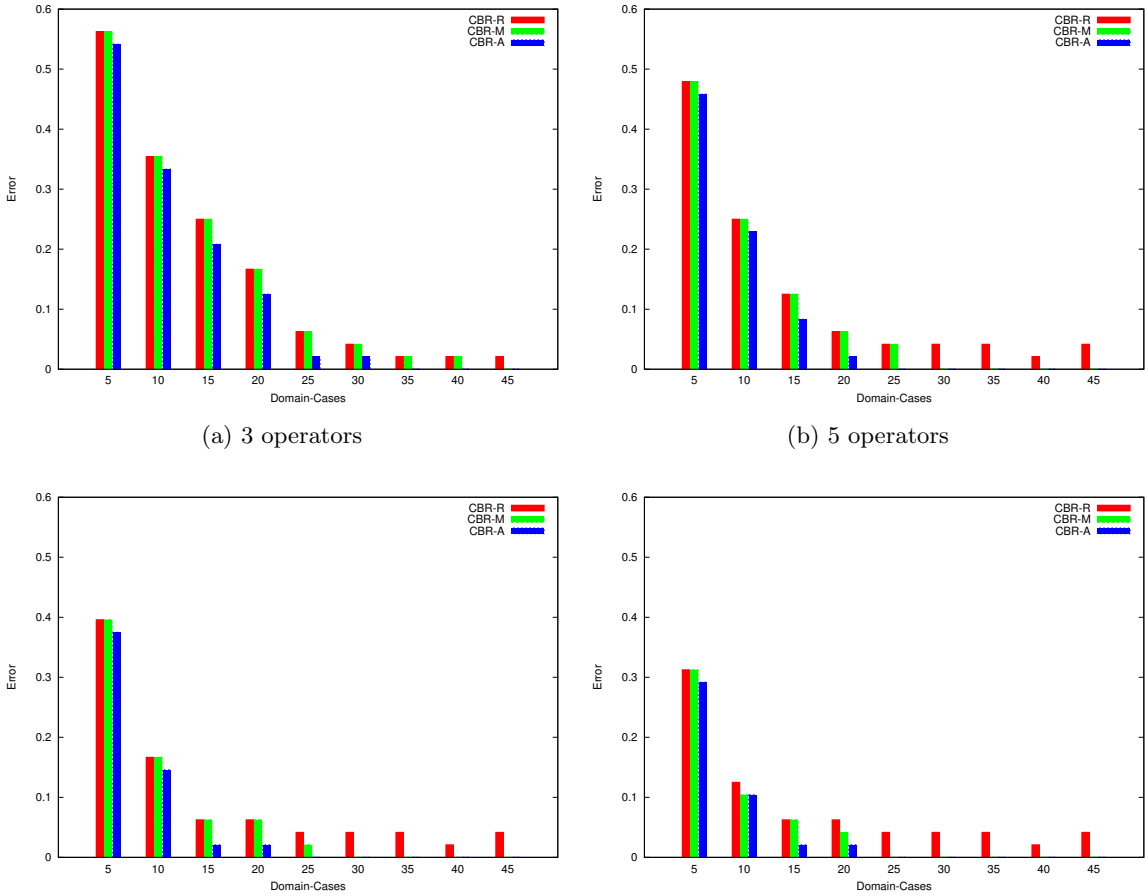
Figure 5.3: Solved problems with 6 operators and 1 expert

5.2.3 Prediction error with respect to the knowledge of the agents

For the tests shown in Figure 5.4, we evaluate the average error in the prediction of the best solution to apply with regard to the size of the case-bases of domain-cases of the agents. As we can expect, the prediction error of the system decreases as the number of domain-cases grows. In addition, the prediction error of the CBR-Argumentation policy is always lower or equal than the other policies. Argumentation allows agents to argue and hence, the solution with more justification elements prevails. Thus, the best solution has more probability of being proposed and the error decreases.

Obviously, if more agents participate in the problem solving process, the probability that one or more of them have a suitable domain-case that can be used to provide a solution for the current problem increases. The same happens if the number of domain-cases of the agents case-base increases. This applies also in the case of the CBR-Random policy, although this policy never achieves the 100% of correct solution predictions. Also, the results achieved by the CBR-Argumentation policy improve those achieved by the other policies, even when the domain-cases case-bases are populated with a small number of cases. These results demonstrate that if agents have the ability of arguing, the agents whose solutions are more supported by evidence have more possibilities of wining the argumentation dialogue and hence, the quality of the final solution selected among all potential solution proposed by the agents increases.

In the tests shown in Figure 5.5, an agent has been allowed to play the role of an *expert*, while the rest of agents play the role of *operators*. As explained before, an expert is an



(a) 3 operators

(b) 5 operators

(c) 7 operators

(d) 9 operators

Figure 5.4: Prediction error

agent that has specific knowledge to solve certain types (categories) of problems and has its case-base of domain-cases populated with cases that solve them. Thus, the expert domain-cases case-base has as much knowledge as possible about the solution of past problems of the same type. In our case, the expert agent has an *authorisation* dependency relation over other technicians. In this tests, the general prediction error is lower than in the other group without any expert. The reason of that improvement in the results is that the expert is providing the best solution that it knows. In the CBR-R and CBR-M policies, is providing more specialized knowledge. In the CBR-Argumentation policy, it is imposing its opinion about which is the best solution to apply. As the expert has more specialized knowledge, the probability of selecting the best solution increases.

5.2.4 Learning of the Domain CBR and the Argumentation CBR

In the next test, we evaluate the number of domain-cases and argument-cases that the agents learn with respect to time. To perform this test, all agents follow the CBR-Argumentation

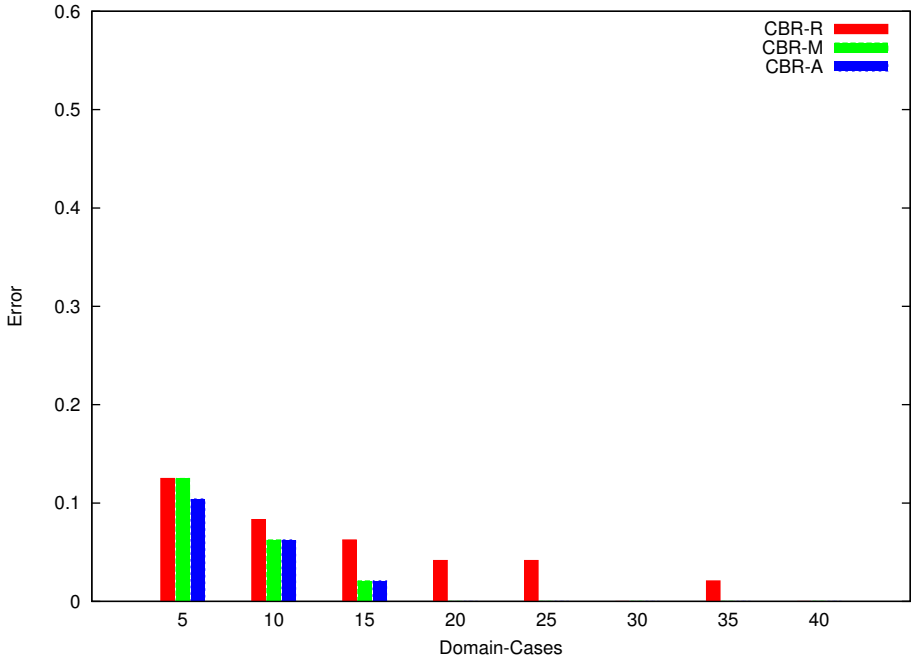
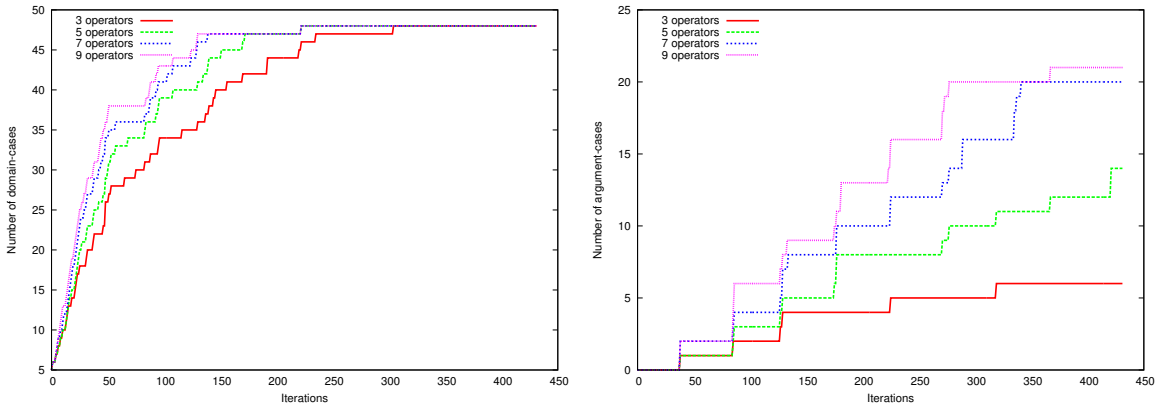


Figure 5.5: Prediction error with 6 operators and 1 expert

policy, with an initial number of 5 domain-cases in their domain-cases case-bases. The argument-cases case-base of all agents are initially empty. In each iteration, the agents use their CBR modules to propose and select positions and arguments and after this process, each agent updates its case-bases with the knowledge acquired.

The knowledge acquired about past problem solving processes should increase with the time until some threshold, where the learning process should stabilize (because the cases in the case-bases of the agents cover most possible problems and arguments in the domain). To perform this test, we have executed several rounds to simulate the use of the system over certain period of time. For each repetition, we compute the average number of domain-cases and argument-cases in the case-bases of the agents. Figure 5.6 shows the results obtained in this test. The experiment has been repeated for 3, 5, 7 and 9 agents and the average number of domain-cases (DC) and argument-cases (AC) that all agents learn in each iteration has been computed. As expected, in all cases, the agents are able to learn the 48 domain-cases of the tickets case-base. However, if more agents participate in the dialogue, the quantity of domain knowledge that agents have available and interchange among them increases and the domain-cases case-bases are more quickly populated. Also, the quantity of argument-cases that agents are able to learn increases with the number of agents, since more potential positions and arguments give rise to more complex argumentation dialogues. As shown in the figure, the learning curve for the argument-cases is less soft than for the domain-cases, presenting peaks at some points. This is due to the fact that at some points of the dialogue, the agents can learn a specific domain-case that change its opinion about the best solution

to apply for a specific category of problem. Therefore, the outcome of subsequent dialogues differ from the outcome that could be expected taking into account past similar dialogues and the argument-cases learning rate of the agents in those situations notably increases.

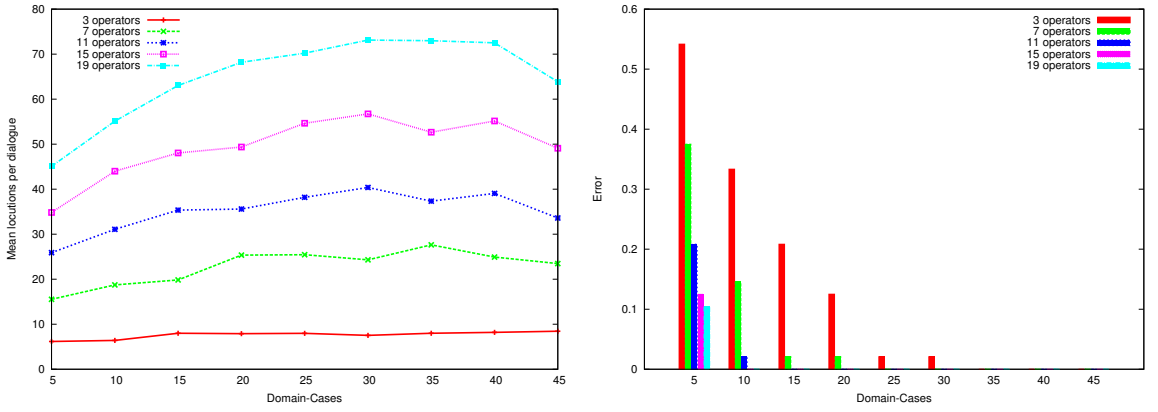


(a) Domain CBR learning (b) Argumentation CBR learning

Figure 5.6: Learning of Domain CBR and Argumentation CBR with respect to time

5.2.5 Generated locutions in the dialogues

In order to see how the agents argue, we show the generated locutions in the dialogues. As explained before, a locution is a conversational particle which an argumentative agent express something. For example, ask about someone's position, propose positions, attack positions or arguments, retract positions or arguments, and so on. Therefore, the locutions generated in the dialogues are an interesting way to measure the complexity of the dialogues.



(a) Locutions (b) Error

Figure 5.7: Mean locutions generated per dialogue and prediction error

Figure 5.7a represents the mean of generated locutions per dialogue of 48 different dialogues for each value of domain-cases in the case-bases of the argumentative agents, from 5

to 45. The policy applied is CBR-Argumentation since it performs argumentation dialogues. As expected, with more agents there are more generated locutions in the dialogues. This is because the agents must talk with each other while not defending the same position. Furthermore, with more talking and more generated locutions the dialogue is longer, so it lasts more time and this could be a problem if we care about the solving speed. However, as we can see in the Figure 5.7b, it is not necessary to involve too many agents in the same dialogue. An optimal number of agents achieves a low prediction error and it does not produce many locutions and hence, the dialogue is shorter.

With the tests performed before and the results shown in Figure 5.7, we can conclude that a number between 7 and 11 agents can be the best in this situation. With this number of agents, the domain-cases are distributed between the agents and they arrive to an agreement applying the correct solution. If there are less agents, they do not have all the necessary knowledge to achieve the best results. However, if there are more agents all the knowledge is distributed and the prediction error is 0, but the dialogues are longer. Furthermore, this number of agents depends on the initial domain-cases that can be distributed among agents. Therefore, in other domains with more or less knowledge, the recommended agents to achieve the best results will be different.

Chapter 6

Conclusions

This chapter summarises the main contributions of this research and identifies future work to extend these contributions. The Chapter also presents the list of publications related to this work.

6.1 Contributions

The main contributions of this Master thesis are the design and implementation of an infrastructure to develop and execute argumentative agents. This includes all the necessary tools and components needed in a MAS platform to support the argumentation capabilities and the representation of the agents' social context. This infrastructure allows to solve problems of a specified domain using argumentation and taking into account the social context of the argumentative agents. An argumentation process is performed to reach an agreement about the best solution to apply to a new problem using past knowledge of the domain and argumentation experiences.

This infrastructure has been validated with a real example and it has been evaluated obtaining, with argumentation strategies, better performance than other reasoning approaches without argumentation. We can conclude that the infrastructure is very useful for problem-solving and as a recommender system.

Moreover, this work has produced the following contributions:

- Review of the state of the art of argumentation frameworks and infrastructures with CBR technology and argumentation.
- Design of the infrastructure and all the components needed.
  - The agents' logic and their argumentation protocol have been designed to achieve the expected behaviour. This behaviour allows the argumentative agents to engage in argumentation dialogues to reach an agreement about the best solution to apply to a problem.

  - We have defined an ontology using OWL 2 to represent all the needed knowledge in the infrastructure to facilitate the understanding between heterogeneous agents.
  - A knowledge interchange mechanism has been designed using FIPA-ACL messages to communicate the agents. The ontology defined for the knowledge representation is also used as common language to interchange knowledge between agents.
  - We have designed two different CBR architectures for the domain CBR and the argumentation CBR. These designs include the case-bases indexing and the case retrieving algorithms in an efficient approach that works well in the performed tests.

- Development of the infrastructure including all the designed tools and components to run argumentative agents and perform argumentation dialogues.

- Validation and evaluation of the infrastructure. The validation has shown the correct operation of the proposed infrastructure. Furthermore, the evaluation shows the good performance of the infrastructure and its application to real problems.

Therefore, we can say that all the objectives proposed in the beginning of this Master thesis (Chapter 1) have been achieved.

6.2 Future work

The work done in this Master thesis leaves some open issues that can extend and improve the proposed infrastructure. Therefore, we propose the following as future work:

- The presented example of application of this infrastructure is going to be applied to real company due to the good performance obtained. This application will demonstrate the utility of this infrastructure and its applicability in the real world.

- Design and development of an argumentation framework with adaptation capabilities in agent societies. This framework should help to build consensus between different parties, having the ability to adapt argumentation strategies depending on the conditions of the agent society where the argumentation process is performed. This work is complex and can be planned as the beginning of a possible PhD work. The objectives to accomplish in this research project will be:

  - Study of agent oriented adaptive systems for agent organizations and argumentation.
  - Study of known models of organization, rules, adaptation and argumentation.
  - Design of an argumentation framework with adaptation capabilities in agent societies with the established model.

– Implementation and development of the designed argumentation framework with adaptation capabilities in agent societies.

6.3 Related publications

- J. Jordán, S. Heras, and V. Julián, "A customer support application using argumentation in multi-agent systems". *14th International Conference on Information Fusion*, pages 772–778. IEEE. 2011. **CORE C**

- J. Jordán, S. Heras, S. Valero and V. Julián. "An Argumentation Framework for Supporting Agreements in Agent Societies Applied to Customer Support". *6th International Conference on Hybrid Artificial Intelligence Systems (HAIS-11)*, volume LNAI 6678, pages 396–403. Springer. 2011. **CORE C**

Bibliography

[Aamodt and Plaza, 1994] Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations and system approaches. *AI Communications*, 7, no. 1:39–59.

[Acorn and Walden, 1992] Acorn, T. and Walden, S. (1992). Smart: Support management automated reasoning technology for compaq customer service. volume 4, pages 3–18.

[Althoff et al., 1995] Althoff, K.-D., Auriol, E., Barletta, R., and Manago, M. (1995). *A Review of Industrial Case-Based Reasoning Tools*.

[Amgoud et al., 2006] Amgoud, L., Bodensta, L., Caminada, M., McBurney, P., Parsons, S., Prakken, H., van Veenen, J., and Vreeswijk, G. (2006). Project N 002307 ASPIC, Argumentation Service Platform with Integrated Components. Deliverable D2.6. Technical report, ASPIC Consortium.

[Argente et al., 2011] Argente, E., Botti, V., Carrascosa, C., Giret, A., Julián, V., and Rebollo, M. (2011). An Abstract Architecture for Virtual Organizations: The THOMAS approach. *Knowledge and Information Systems*, pages 1–35.

[Artikis et al., 2009] Artikis, A., Sergot, M., and Pitt, J. (2009). Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 10(1).

[Atkinson, 2005] Atkinson, K. (2005). A dialogue game protocol for multi-agent argument over proposals for action. *Autonomous Agents and Multi-Agent Systems. Special issue on Argumentation in Multi-Agent Systems*, 11(2):153–171.

[Bench-Capon and Atkinson, 2009] Bench-Capon, T. and Atkinson, K. (2009). *Argumentation in AI*, chapter Abstract argumentation and values, pages 45–64.

[Bench-Capon and Dunne, 2007] Bench-Capon, T. and Dunne, P. (2007). Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10-15):619–938.

[Carrascosa et al., 2009] Carrascosa, C., Giret, A., Julián, V., Rebollo, M., Argente, E., and Botti, V. (2009). Service oriented multi-agent systems: An open architecture. In *Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1–2.

[Chesñevar et al., 2006] Chesñevar, C., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., Simari, G., South, M., Vreeswijk, G., and Willmott, S. (2006). Towards an argument interchange format. *The Knowledge Engineering Review*, 21(4):293–316.

[Dastani et al., 2003] Dastani, M., Dignum, V., and Dignum, F. (2003). Role-assignment in open agent societies. In *2nd Int. Joint Conference on Autonomous Agents and Multi-agent Systems*, pages 489–496.

[del Val et al., 2009] del Val, E., Criado, N., Rebollo, M., Argente, E., and Julián, V. (2009). Service-oriented framework for virtual organizations. In *International Conference on Artificial Intelligence (ICAI)*, volume 1, pages 108–114. CSREA Press.

[Dignum, 2003] Dignum, V. (2003). *PhD Dissertation: A model for organizational interaction: based on agents, founded in logic*. PhD thesis.

[Dung, 1995] Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n -person games. *Artificial Intelligence*, 77:321–357.

[Esteva et al., 2001] Esteva, M., Rodriguez, J., Sierra, C., Garcia, P., and Arcos, J. (2001). On the formal specification of electronic institutions. In *Agent Mediated Electronic Commerce*, volume 1991, pages 126–147.

[Garijo, 2002] Garijo, F. (2002). Tecnología de agentes: Experiencias y perspectivas para el desarrollo de nuevos servicios y aplicaciones. In *Bole.tic*, volume 24, pages 1–9.

[Giret et al., 2009] Giret, A., Julián, V., Rebollo, M., Argente, E., Carrascosa, C., and Botti, V. (2009). An open architecture for service-oriented virtual organizations. In *Seventh international Workshop on Programming Multi-Agent Systems. PROMAS 2009*, pages 23–33.

[Gonzalez-Palacios and Luck, 2007] Gonzalez-Palacios, J. and Luck, M. (2007). Towards compliance of agents in open multi-agent systems. In *Software Engineering for Multi-Agent Systems V, Lecture Notes in Computer Science*. Springer.

[Heras et al., 2011a] Heras, S., Botti, V., and Julián, V. (2011a). Case-Based Argumentation Framework. Dialogue Protocol. Technical report, Universitat Politècnica de València, http://hdl.handle.net/10251/11096.

[Heras et al., 2011b] Heras, S., Botti, V., and Julián, V. (2011b). Case-Based Argumentation Framework. Reasoning Process. Technical report, Universitat Politècnica de València, http://hdl.handle.net/10251/11094.

[Heras et al., 2011c] Heras, S., Botti, V., and Julián, V. (2011c). A Computational Argumentation Framework for Agent Societies. Technical report, Universitat Politècnica de València.

[Heras et al., 2009] Heras, S., García-Pardo, J. A., Ramos-Garijo, R., Palomares, A., Botti, V., Rebollo, M., and Julián, V. (2009). Multi-domain case-based module for customer support. *Expert Systems with Applications*, 36(3):6866–6873.

[Jennings and Wooldridge, 1998] Jennings, N. R. and Wooldridge, M. (1998). *Applications of intelligent agents*, pages 3–28. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Jordán et al., 2011a] Jordán, J., Heras, S., and Julián, V. (2011a). A customer support application using argumentation in multi-agent systems. In *14th International Conference on Information Fusion*, pages 772–778. IEEE.

[Jordán et al., 2011b] Jordán, J., Heras, S., Valero, S., and Julián, V. (2011b). An argumentation framework for supporting agreements in agent societies applied to customer support. In *6th International Conference on Hybrid Artificial Intelligence Systems (HAIS-11)*, volume LNAI 6678, pages 396–403. Springer.

[Karacapilidis and Papadias, 2001] Karacapilidis, N. and Papadias, D. (2001). Computer supported argumentation and collaborative decision-making: the hermes system. *Information Systems*, 26(4):259–277.

[Karacapilidis et al., 1997] Karacapilidis, N., Trousse, B., and Papadias, D. (1997). Using case-based reasoning for argumentation with multiple viewpoints. pages 541–552.

[Kolodner, 1993] Kolodner, J. (1993). *Case-based Reasoning*.

[Luck and McBurney, 2008] Luck, M. and McBurney, P. (2008). Computing as interaction: agent and agreement technologies. In *IEEE International Conference on Distributed Human-Machine Systems*.

[Mas, 2005] Mas, A. (2005). *Agentes Software y Sistemas Multi-Agente. Conceptos, Arquitecturas y Aplicaciones*. Pearson. Prentice Hall.

[O'Hare and Jennings, 1996] O'Hare, G. M. P. and Jennings, N. R. (1996). *Foundations of Distributed Artificial Intelligence*. John Wiley & Sons, Inc., New York, NY.

[Ontañón and Plaza, 2006] Ontañón, S. and Plaza, E. (2006). Arguments and counterexamples in case-based joint deliberation.

[Ontañón and Plaza, 2007] Ontañón, S. and Plaza, E. (2007). Learning and joint deliberation through argumentation in multi-agent systems.

[Ossowski et al., 2007] Ossowski, S., Julian, V., Bajo, J., Billhardt, H., Botti, V., and Corchado, J. M. (2007). Open issues in open mas: An abstract architecture proposal. volume 2, pages 151–160.

[Parsons et al., 1998] Parsons, S., Sierra, C., and Jennings, N. R. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292.

[Rahwan, 2006] Rahwan, I. (2006). Argumentation in multi-agent systems. *Autonomous Agents and Multiagent Systems, Guest Editorial*, 11(2):115–125.

[Rahwan and Amgoud, 2006] Rahwan, I. and Amgoud, L. (2006). An argumentation-based approach for practical reasoning. pages 347–354.

[Rahwan et al., 2003] Rahwan, I., Ramchurn, S. D., Jennings, N. R., McBurney, P., Parsons, S., and Sonenberg, L. (2003). Argumentation-based negotiation. *The Knowledge Engineering Review*, 18(4):343–375.

[Rahwan and Simari, 2009] Rahwan, I. and Simari, G., editors (2009). *Argumentation in Artificial Intelligence*. Springer.

[Rissland et al., 2006] Rissland, E. L., Ashley, K. D., and Branting, L. K. (2006). Case-based reasoning and law. *The Knowledge Engineering Review*, 20(3):293–298.

[Roth-Berghofer, 2004] Roth-Berghofer, T. R. (2004). Learning from homer, a case-based help-desk support system. pages 88–97.

[Shoham, 1993] Shoham, Y. (1993). Agent-oriented programming. In *Artificial Intelligence*, volume 60(1), pages 51–92.

[Soh and Tsatsoulis, 2001a] Soh, L.-K. and Tsatsoulis, C. (2001a). Agent-based argumentative negotiations with case-based reasoning. pages 16–25.

[Soh and Tsatsoulis, 2001b] Soh, L.-K. and Tsatsoulis, C. (2001b). Reflective negotiating agents for real-time multisensor target tracking. pages 1121–27.

[Soh and Tsatsoulis, 2005] Soh, L.-K. and Tsatsoulis, C. (2005). A real-time negotiation model and a multi-agent sensor network implementation. *Autonomous Agents and Multi-Agent Systems*, 11(3):215–271.

[Sycara, 1987] Sycara, K. (1987). *Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods*. PhD thesis.

[Sycara, 1989] Sycara, K. (1989). Argumentation: Planning other agents' plans. volume 1, pages 517–523.

[Sycara, 1990] Sycara, K. (1990). Persuasive argumentation in negotiation. *Theory and Decision*, 28:203–242.

[Tolchinsky et al., 2006a] Tolchinsky, P., Cortés, U., Modgil, S., Caballero, F., and López-Navidad, A. (2006a). Increasing human-organ transplant availability: Argumentation-based agent deliberation. *IEEE Intelligent Systems*, 21(6):30–37.

[Tolchinsky et al., 2006b] Tolchinsky, P., Modgil, S., and Cortés, U. (2006b). Argument schemes and critical questions for heterogeneous agents to argue over the viability of a human organ.

[Tolchinsky et al., 2006c] Tolchinsky, P., Modgil, S., Cortés, U., and Sànchez-Marrè, M. (2006c). Cbr and argument schemes for collaborative decision making. volume 144, pages 71–82.

[Vázquez-Salceda et al., 2003] Vázquez-Salceda, J., Cortés, U., Padget, J., López-Navidad, A., and Caballero, F. (2003). The organ allocation process: A natural extension of the carrel agent-mediated electronic institution. *AI Communications*, 16(3):153–165.

[Watson, 1997] Watson, I. (1997). *Applying case-based reasoning. Techniques for enterprise systems.* Morgan Kaufmann Pusblishers, Inc.

[Willmott et al., 2006] Willmott, S., Vreeswijk, G., Chesñevar, C., South, M., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., and Simari, G. (2006). Towards an argument interchange format for Multi-Agent Systems. In *3rd International Workshop on Argumentation in Multi-Agent Systems, ArgMAS-06*, pages 17–34.

[Wooldridge, 2002] Wooldridge, M. (2002). *An introduction to Multiagent Systems.* John Wiley and Sons Ltd.

[Zambonelli et al., 2003] Zambonelli, F., Jennings, N., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. In *ACM Transactions on Software Engineering and Methodology*, volume 12, pages 317–370.