Computational Intelligence, Volume xxx, Number 000, 2012

An Infrastructure for Argumentative Agents

JAUME JORDÁN, STELLA HERAS, SOLEDAD VALERO, AND VICENTE JULIÁN

Departamento de Sistemas Informáticos y Computación,
Universitat Politècnica de València, València (Spain)

Multi-Agent Systems are suitable to provide a framework that allows to perform collaborative processes in a social context. Furthermore, argumentation is a natural way of reaching agreements between several parties. However, it is difficult to find infrastructures of argumentation offering support for agent societies and their social context. Offering support for agent societies allows to represent more realistic environments to have argumentation dialogues. We propose an infrastructure to develop and execute argumentative agents in an open MAS. It offers tools to develop agents with argumentation capabilities. It also offers support for agent societies and their social context. The infrastructure is publicly available. Also, it has been implemented in an application scenario where argumentative agents try to reach an agreement about the best solution to solve a problem reported to the system.

Key words: Argumentation; Case-Based Reasoning; Multi-Agent Systems.

1. INTRODUCTION

Argumentation theory has produced important benefits on many Artificial Intelligence (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. The argumentation techniques, hence, can be used to facilitate the agents' autonomous reasoning and to specify interaction protocols between them (Heras et al., 2012).

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. Moreover, ASBO (Muñoz and Botía, 2008, 2009) allows MAS to attack to arguments and making explicit the argumentation process structure through an OWL-based ontology. Nevertheless, we do not know about

Address correspondence to Jaume Jordán, Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Camino de Vera s/n. 46022 València, Spain; e-mail: jjordan@dsic.upv.es

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

any infrastructure of argumentation in open MAS offering support for agent societies and their social context (dependencies and values) to generate, select and evaluate arguments.

In this work, we propose an infrastructure to develop and execute argumentative agents in a MAS. This work is an extension of the one presented in (Jordán et al., 2011). We use a case-based argumentation framework as the base to build an infrastructure to deal with different kinds of problems (Heras et al., 2013). 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 social context. The main advantage of having this infrastructure is that it is possible to create agents with argumentation capabilities to solve a specific problem. Also, the infrastructure allows agents to store in the form of cases the information about the argumentation dialogues that they hold. Therefore, agents can argue for agreement with a final justified decision, taking full advantage of the knowledge of the agents. This approach can 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. Our approach is a hybrid system (Corchado et al., 2010; Abraham et al., 2009) that integrates Case-Based Reasoning (CBR) methodology (Kolodner, 1993), argumentation and MAS.

This paper is structured as follows. Section 2 presents current applications of argumentation in AI. Section 3 presents the argumentation framework implemented in the infrastructure for argumentative agents. Section 4 explains the complete infrastructure developed in this work. Section 5 shows an example of the use of the presented infrastructure in a customer support application, and also an evaluation of the performance of the developed application. Finally, Section 6 presents the conclusions extracted from this work.

2. RELATED WORK

Nowadays, the argumentation research in AI is experiencing a new reactivation, mainly motivated by recent and interesting contributions developed in MAS. On the 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 was 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. Rahwan et al. (Rahwan et al., 2003) analyses this and other argumentation-based negotiation frameworks. A wide review of the situation of the argumentation research in AI was also 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) was 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 (for example generation, selection and evaluation of the arguments' components, and the management of the dialogue itself) is a key issue to deal with argumentation-based dialogues in MAS. Argumentation in AI has historically followed two different approaches based on rule-based or case-based systems. The dynamics of open MAS makes difficult to specify in advance the set of rules that govern the behaviour of agents. Thus, a case-based approach to track arguments exchanged among agents and learn from the experience is a most suitable option to perform argumentation in agent societies. Case-Based Reasoning (CBR) (Kolodner, 1993) is a suitable methodology to manage argumentation processes in two-party disagreement situations. 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 (Heras et al., 2009):

- 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, 1989, 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. How-

ever, 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,b, 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 case 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., 2006, 2012). 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, 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 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 were implemented and tested in the medical domain (Karacapilidis and Papadias, 2001), (Tolchinsky et al., 2006). 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.

The literature review shows that a formal and context-independent framework that defines an argumentation theory for agents with learning capabilities that are situated in a society and have values to promote/demote does not already exist. This motivated the development of our case-based argumentation framework. The examples about systems that successfully apply CBR to manage argumentation in MAS demonstrate the suitability of this reasoning methodology to provide agents with the ability to argue. However, the current approaches are domain dependent and some assume the interaction of humans with the system. Therefore, they are not suitable to be applied in a general domain where agents argue in the context of a society in a MAS. Also, none of the systems reviewed has the ability of managing agents' values and dependency relations in the argumentation process. This ability is crucial in agent societies, where these are important concepts of the social context of agents.

3. ARGUMENTATION FRAMEWORK FOR AGENT SOCIETIES

In this Section, we briefly introduce the case-based argumentation framework (Heras et al., 2013) that forms the basis of the infrastructure proposed in this paper. This framework supports argumentation in MAS in which the participating software agents are able to manage and exchange arguments between themselves, taking into account the agents' social context.

In this Section we explain the main features of the argumentation framework that we implement in the infrastructure. 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 and the argument-cases. The domain-cases represent previous problems and their solutions. The argument-cases represent past argumentation experiences and their final outcome. 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.

In the framework, an *agent society* (Heras et al., 2012) 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*.

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 case-base (from now on domain case-base), 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 has an argument-cases case-base (from now on argument case-base). The argument-cases represent past argumentation experiences and their final outcome.

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

Definition 1 (Argument):

$$Arg = \langle \phi, v, S \rangle \tag{1}$$

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*).

Definition 2 (Support Set):

$$\begin{aligned} S = \{&\{premises\}, \{domainCases\}, \{argumentCases\}, \\ &\{distinguishingPremises\}, \{counterExamples\}\} \end{aligned} \tag{2}$$

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 uses 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 proposes a solution different from the argument being attacked. They consist of features of the problem description that were 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 that has as support set 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 justify 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. If 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.

The argumentation framework presented in this Section has all the necessary components for our proposed infrastructure and specifies the knowledge resources of the agents. Furthermore, we consider that this framework is computationally tractable. In the next Section, we explain the infrastructure that supports this argumentation framework.

4. INFRASTRUCTURE

In this Section, the infrastructure to support the argumentation framework explained in Section 3 is described. This infrastructure offers the necessary tools to develop agents with argumentation capabilities, including the communication skills and the argumentation protocol in an open MAS, and it offers support for agent societies and their agents' social context. The main advantage of having this infrastructure is that it supports the development of agents with argumentation capabilities to solve a specified problem. 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 a given problem. The problem to solve has to be of a previously specified domain. Therefore, the training data

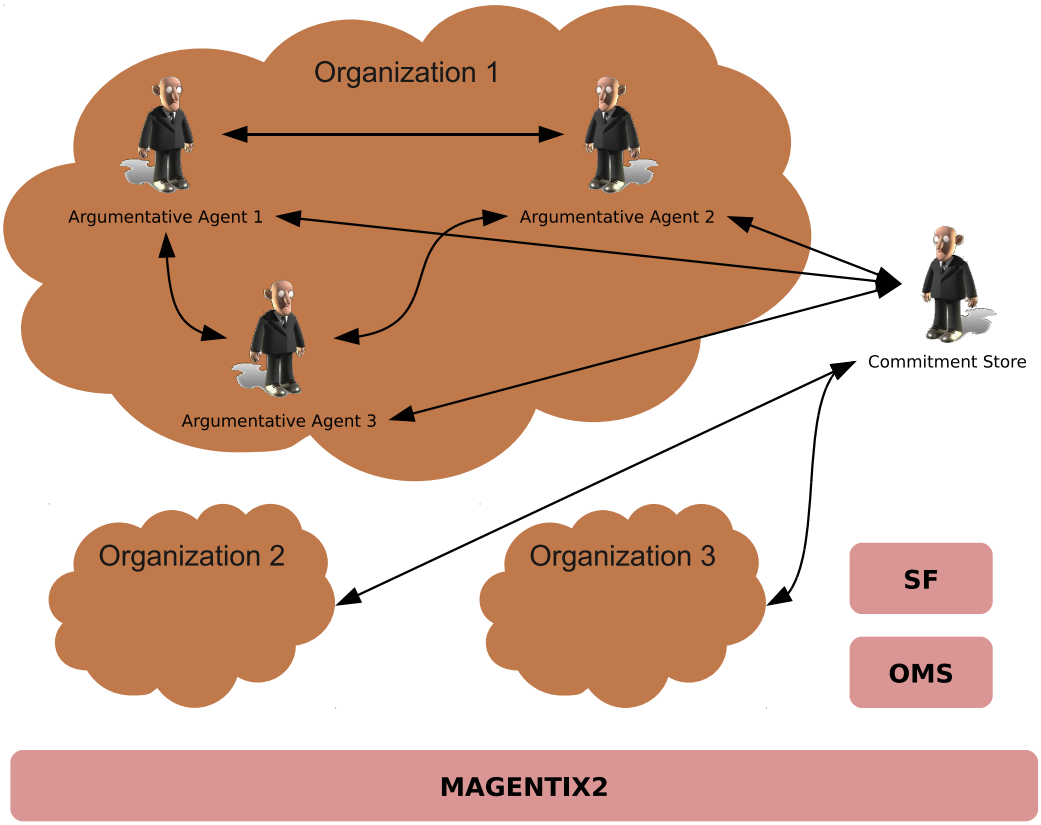

FIGURE 1: Implemented infrastructure using the agent platform Magentix2

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 also have the argument-cases, that represent past argumentation experiences and their final outcome. The argumentative agents will use the domain-cases and the argument-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 1. The main components of the infrastructure are the argumentative agents, the Commitment Store, and the knowledge interchange mechanism. As can be seen in Figure 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.

The infrastructure has been implemented using the agent platform Magentix2[2]. Inside the Magentix2 main package, the infrastructure is publicly available including an application scenario described in Section 5. Magentix2 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 represented in Figure 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

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

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 Conversational Agent (CAgent)[3]. This kind of Magentix2 agent allows the automatic creation of simultaneous conversations based on interaction protocols. CAgents can use predefined interaction protocols, define their own interaction protocols and also dynamically change interaction protocols at runtime. In the infrastructure, an argumentation protocol has been defined for the argumentative agents. The main advantage of this is that the actions of the argumentation protocol can be easily modified to allow the argumentative agents to take different decisions in the dialogues.

Following, we explain the main components of the infrastructure:

- Argumentative agents: these 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 argumentative agents are:
  - Domain CBR module: this is a CBR that stores cases that contain domain knowledge about previous solved problems.
  - Argumentation CBR module: this is a CBR that stores cases that contain past argumentation experiences.
  - 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 CBR modules.
  - Argumentation protocol: this is the argumentation protocol followed by the argumentative agents to perform the argumentation dialogues.
- Commitment Store: this is a resource of the argumentation framework that stores all the information about the agents participating in the problem-solving process, the argumentation dialogues between them, their positions and arguments Hamblin (1970). By making queries to this resource, every agent can read the information of the dialogues that it is involved in. In the infrastructure, it has been implemented as an agent to allow a good communication with the other agents. Concretely, it is an extension of the Magentix2 CAgent, as the argumentative agents are. Figure 2 shows an example of messages interchange between an argumentative agent and the Commitment Store. 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.
- Knowledge interchange mechanism: this is the mechanism that the argumentative agents and the Commitment Store use to interchange knowledge. The case-bases of the domain CBR module and the argumentation CBR module are stored as OWL 2[4] data of an ontology[5] that we have designed to act as language representation of the cases. 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 agents can easily understand them. They also interchange the knowledge by FIPA-ACL[6] messages.

[3] http://users.dsic.upv.es/grupos/ia/sma/tools/magentix2/archivos/javadoc/es/upv/dsic/gti_ia/cAgents/CAgent.html
[4] http://www.w3.org/TR/owl2-overview/
[5] http://gti-ia.dsic.upv.es/vinglada/docs/
[6] http://www.fipa.org/specs/fipa00061/SC00061G.html

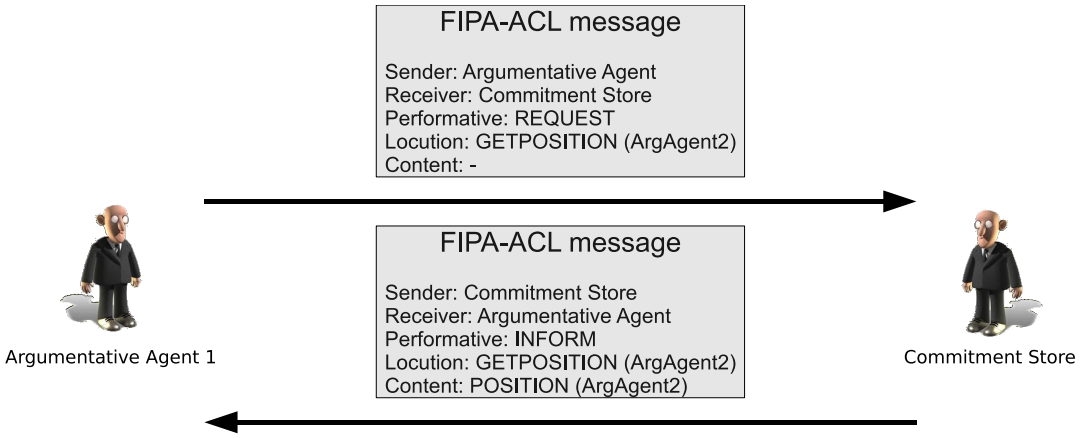

FIGURE 2: Messages interchange between an argumentative agent and the Commitment Store

The most important elements of the infrastructure are the argumentative agents. In this Section we explain all their features. 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 case-base and it can either be retrieved and applied directly to the current problem, or revised and adapted to fit the new problem. CBR systems have been widely applied to perform this task (Acorn and Walden, 1992; Watson, 1997; Roth-Berghofer, 2004). As commented before, the argumentative agents have two CBR based modules: Domain CBR module and Argumentation CBR module. These modules, the argument management process, and the argumentation protocol are described in detail in the following subsections.

4.1. Domain CBR module

The argumentative agents have their own domain CBR module. This CBR module 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 3. 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. The features are also used to retrieve similar cases from the case-base in the retrieve phase.

Different algorithms based on several similarity measures as *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 it is not an objective of this work to evaluate the different alternatives. To retrieve domain-cases of 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.

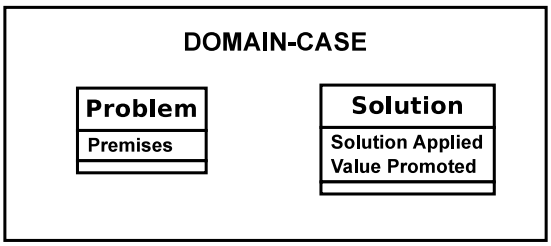

FIGURE 3: Structure of the Domain-case

4.1.1. *Case-base indexing and retrieving.* We use a hash table to store the domain-cases in the case-base. This data representation allows to improve the efficiency of case retrieval. 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 module 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 premise with the lower 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 module without a specified index.

Therefore, to retrieve domain-cases similar to the problem to solve, each premise identifier of the entire premises is selected to retrieve the list of domain-cases. These retrievals will have constant cost O(1) and obtain several lists of domain-cases that have, at least, one of the premises. Then, all lists are joined in one, deleting the possible duplicated cases. As expected, this list could be potentially large. However, a lot of cases without any of the premises of the similar domain-case to solve can be discarded. Then, the domain-cases of the list that match with the problem to solve will be selected. This has a linear cost O(n). Otherwise, if an index was specified, the retrieval of the candidate cases (retrieved cases that have similar premises to the problem to solve) would be made using it with constant cost.

4.1.2. *Retention.* It is very important to perform a retention phase in a CBR module because it contributes to learn new cases and adapt the system to new conditions. In the domain CBR module, 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 exists, the associated solutions will be added to enrich the domain-case.

4.1.3. *Case-base persistence.* The persistence of the domain case-base has been implemented by using ontologies. Concretely, an ontology has been defined using the OWL 2 language. In this way, heterogeneous agents can use it as a common language to interchange solutions and arguments generated from the case-bases of the argumentation framework. We store domain cases as instances of concepts of the defined ontology.

4.2. Argumentation CBR module

The Argumentation CBR module consists of a CBR module with argumentation data. This CBR stores as argument-cases arguments previously used in argumentation dialogues. The argumentative agents have their own argumentation CBR module. This knowledge 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.

As can be seen in Figure 4, an argument-case stores information related to the domain and the social context where previous arguments (and their associated solution) were proposed. The problem description of an argument-case includes information about the domain context where the argument was put forward and information about the social context where the solution was applied (the agents that participated in the dialogue, their roles, their value preferences and the dependency relation between them). 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.

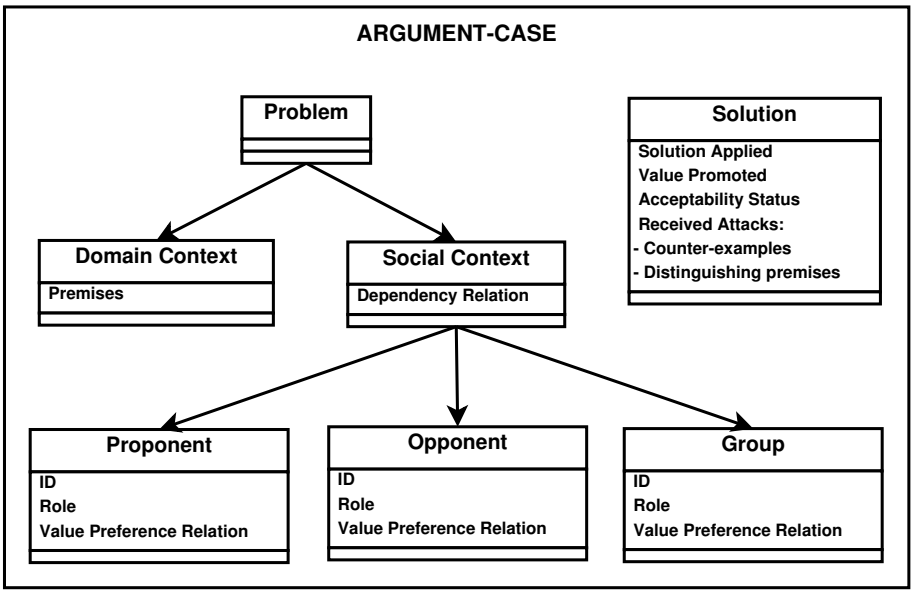

FIGURE 4: Structure of the argument-case

4.2.1. *Case-base indexing and retrieving.* To store argument-cases in a case-base, we use a hash table to improve the efficiency of retrieval. The solution which has been chosen is the same as that used in the Domain CBR module, 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 premise with the lower 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, to retrieve the argument-case that dealt with the specific domain context in the past.

Therefore, to retrieve argument-cases similar to the problem to solve, each premise identifier of the entire premises is selected to retrieve the list of argument-cases. These retrievals will have constant cost O(1) and obtain several lists of argument-cases that have, at least, one of the premises. Then, all lists are joint in one, deleting the possible duplicate cases. As expected, this list could be potentially large. However, a lot of cases without any of the premises of the problem to solve can be discarded. Then, the argument-cases of the list that match with the problem to solve will be selected. This has a linear cost O(n). Otherwise,

if an index was specified, the retrieval of the candidate cases would be made using it with constant cost.

Furthermore, having the first list of candidate argument-cases that fit the domain context, 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. The similarity of the social context is computed by some parameters and adjusted with some weights specified when the argumentative agent is created.

4.2.2. *Retention.* One of the most important phases of CBR is the retention of new cases generated, contributing to learning and adaptability of the system. 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. The argument-case retention is made once the argumentation process has finished.

4.2.3. *Case-base persistence.* As with by the domain CBR module, the persistence of the argument case-base has been implemented by 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 a common language to interchange solutions and arguments generated from the case-bases of the argumentation framework. We store argument cases as instances of concepts of the defined ontology.

4.3. Argument Management Process

The argument management process that the argumentative agents perform is the reasoning process that they follow to generate positions and arguments. In the following subsections we describe these parts of the argument management process: position generation, support arguments generation and attack arguments generation.

4.3.1. *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 retrieves from its domain CBR module 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 module. 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 similar arguments in the past.

4.3.2. *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., 2013) that provides an estimation of the actual expected acceptance of an argument by taking into account how persuasive this argument was in a similar domain and social context in the past.

4.3.3. *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 have the argument to attack and the social context (the relation with the other agent) are taken into account. With this information, the argumentative agents retrieve 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 level in the hierarchy of the agent to attack is equal or lower to the level of the attacker agent), the attack will be based on the most suitable argument-case retrieved 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.

4.4. Argumentation protocol

The argumentative agents need a mechanism to manage the arguments and perform the argumentation dialogue. To deal with this functionality, an argumentation protocol has been defined. This protocol is represented by a set of utterances that the agents use to communicate with other agents, a formal axiomatic semantics, and the dialogue rules that define the behaviour of an agent in the argumentation dialogue (McBurney et al., 2003; McBurney and Parsons, 2009).

The syntax of the utterances (locutions) is as proposed in (McBurney and Parsons, 2004):

$locution(a_s, \phi)$ or $locution(a_s, a_r, \phi)$

where $Agent(a_s)$ (the sender) and $Agent(a_r)$ (the receiver) are individuals of the $Agent$ concept and ϕ is the content of the utterance. The former locution is addressed to all participants in the dialogue, whereas the latter is specifically sent to $Agent(a_r)$. We denote the set of well-formed formulae in $\mathcal{SHOIN}(D)$ as $\mathcal{D}$. Then, $\phi \in \mathcal{D}$ can represent statements about problems to solve, facts about the world or different types of arguments. Also, we denote the set of individuals members of the concept $Argument$ as $\mathcal{A}$ such that $\forall arg \in \mathcal{A}, Argument(arg)$. Therefore, Φ is said to be an argument in support of ϕ if $\Phi \in \mathcal{A}/\Phi \vdash^{+} \phi$. Correspondingly, Φ is said to be an argument against ϕ if $\Phi \in \mathcal{A}/\Phi \vdash^{-} \phi$.

In this work, we follow the standard notation of *modal logics* of knowledge and belief described in (Shoham and Leyton-Brown, 2009, chapter 13). Thus, we use the modal operators

$K_i\phi$: "Agent a_i knows ϕ"
$B_i\phi$: "Agent a_i believes that ϕ is true"
$C_g\phi$: "ϕ is common knowledge for any agent in the group g if any agent of the group knows it and knows that it is common knowledge"

and the modal connective

$\Diamond\phi$ is satisfied now if ϕ is satisfied either now or at some future moment.

Note that here we make a distinction between what agents *know* (which is considered to be true) and what agents *believe* (which forms part of the mental state of an agent and can be true or not).

In addition, as proposed in McBurney and Parsons (2004), we use the following simplified elements of *FIPA*'s communicative act library specification[7]:

Done[$locution(a_s, \phi)$*, preconditions]*

[7] http://www.fipa.org/specs/fipa00037/SC00037J.html

which indicates that $locution(a_s, \phi)$ (or correspondingly $locution(a_s, a_r, \phi)$) has been put forward by agent a_s (addressed to agent(s) a_r) with content ϕ and the specified *preconditions* hold before this utterance and

Feasible[condition, $locution(a_s, \phi)$*]*

which means that if *condition* can take place, $locution(a_s, \phi)$ (or correspondingly $locution(a_s, a_r, \phi)$) will be put forward by agent a_s (addressed to agent(s) a_r) with content ϕ.

Further notation that we use throughout this section is the next:

a_s: the $Agent(a_s)$ sender of the locution.
a_r: the $Agent(a_r)$ receiver of the locution.
arg_i: an $Argument(arg_i)$ of an $Agent(a_i)$.
SS_i: the $SupportSet(SS_i)$ of the $Argument(arg_i)$ that has put forward an $Agent(a_i)$.
CS_i: the commitment store of an $Agent(a_i)$.
q: the $Problem(q)$ under discussion.
p_i: the $Solution(p_i)$ (or position) proposed by an $Agent(a_i)$ to solve the $Problem(q)$.

The axiomatic semantics that specify the meaning and consequences of each utterance in our protocol are the following:

$\{$***pre***$\}\ OPENDIALOGUE(a_s, \phi)\ \{$***post***$\}$

- *pre*: $Agent(a_s)$ wants to inform other agents about the proposition ϕ, which is a $Problem(q)$ to solve. Note that until agents enter the dialogue, their commitment stores CS are not created.
  $(K_{a_s} q) \wedge (\nexists CS_{a_s})$
- *post*: All agents in the society S_t know that $Agent(a_s)$ wants to solve $Problem(q)$.
  $(\Diamond C_{S_t} K_{a_s} q)$

$\{$***pre***$\}\ ENTERDIALOGUE(a_s, \phi)\ \{$***post***$\}$

- *pre*: $Agent(a_s)$ knows ϕ (the $Problem(q)$ reported by $Agent(a_i)$) and informs other participants that are engaged in the dialogue that it is willing to enter the dialogue to solve $Problem(q)$.
  $(Done[OPENDIALOGUE(a_i, q), ...]) \wedge (K_{a_s} q)$
- *post*: Other participants of the $Group(g)$ are informed that $Agent(a_s)$ is willing to engage in a dialogue to solve $Problem(q)$. Also, the commitment store CS_{a_s} is created and $Agent(a_s)$ starts to belong to the $Group(g)$ of agents engaged in the dialogue.
  $(\Diamond C_g(a_s \in g)) \wedge (\exists CS_{a_s})$

$\{$***pre***$\}\ WITHDRAWDIALOGUE(a_s, \phi)\ \{$***post***$\}$

- *pre*: $Agent(a_s)$ that has engaged in the argumentation dialogue to solve ϕ (the $Problem(q)$) wants to leave from the dialogue and report it to the other agents of the $Group(g)$ that are engaged in the dialogue. Note that agents cannot withdraw the dialogue before withdrawing any position $Solution(p)$ that they have proposed.
  $(Done[ENTERDIALOGUE(a_s, q), K_{a_s} q]) \wedge (\nexists p \in CS_{a_s})$
- *post*: Other participating agents of the $Group(g)$ know that $Agent(a_s)$ no longer participates in the dialogue to solve $Problem(q)$. Also, the commitment store CS_{a_s} of $Agent(a_s)$ is deleted.
  $(\Diamond C_g(a_s \notin g)) \wedge (\nexists CS_{a_s})$

$\{$***pre***$\}\ PROPOSE(a_s, \phi)\ \{$***post***$\}$

- *pre*: An $Agent(a_s)$ that has engaged in a dialogue to solve ϕ (the $Problem(q)$) wants to propose its position $Solution(p)$ as a solution for the problem and reports it to the other agents of the $Group(g)$. An agent cannot propose a new position without withdrawing a previous $Solution(r)$ from its commitment store, if any.
$(Done[ENTERDIALOGUE(a_s, q), ...]) \wedge (B_{a_s} p) \wedge (\forall r \neq p)(\nexists r \in CS_{a_s})$
- *post*: Other participating agents of the $Group(g)$ know that $Agent(a_s)$ has proposed position $Solution(p)$ as solution for $Problem(q)$ and it is inserted in the commitment store CS_{a_s} of $Agent(a_s)$.
$(\Diamond C_g B_{a_s} p) \wedge (p \in CS_{a_s})$

$\{$***pre***$\}\ WHY(a_s, a_r, \phi)\ \{$***post***$\}$

- *pre*: $Agent(a_s)$ wants to challenge $Agent(a_r)$ to provide a justification for the position $Solution(p)$.
$(Done[PROPOSE(a_r, p), B_{a_r} p]) \wedge (K_{a_s} B_{a_r} p) \wedge (p \notin CS_{a_s})$

- *post*: $Agent(a_r)$ knows that $Agent(a_s)$ does not believe $Solution(p)$ and has the dialogical commitment of justifying it with an *Argument(arg)* $\vdash^+$ *Solution(p)* or else of withdrawing it.
$(\Diamond K_{a_r} \neg B_{a_s} p) \wedge ((Feasible[\exists arg / arg \vdash^+ p, ASSERT(a_r, a_s, arg)]) \vee$
$(Feasible[\nexists arg / arg \vdash^+ p, NOCOMMIT(a_r, p)]))$

$\{$***pre***$\}\ NOCOMMIT(a_s, \phi)\ \{$***post***$\}$

- *pre*: $Agent(a_s)$ that has put forward ϕ (the position $Solution(p)$) wants to withdraw it and report this change to the other participating agents of the $Group(g)$ engaged in the dialogue.
$(p \in CS_{a_s}) \wedge (C_g B_{a_s} p)$
- *post*: Other participating agents of the $Group(g)$ know that $Agent(a_s)$ no longer proposes $Solution(p)$ as its position to solve the problem at hand. Also, $Solution(p)$ is deleted from the commitment store CS_{a_s} of $Agent(a_s)$.
$(\Diamond C_g \neg B_{a_s} p) \wedge (p \notin CS_{a_s})$

$\{$***pre***$\}\ ASSERT(a_s, a_r, \phi)\ \{$***post***$\}$

- *pre*: An $Agent(a_s)$ wants to provide a justification for its position $Solution(p)$ and reports it to an agent $Agent(a_r)$ that has challenged it.
$(Done[WHY(a_r, a_s, p), ...]) \wedge (\exists arg)\ (arg \vdash^+ p) \wedge$
$(\nexists arg_{a_s} \in CS_{a_s})(arg \vdash^- arg_{a_s})$
- *post*: $Agent(a_r)$ knows that $Agent(a_s)$ has provided $Argument(arg)$ as a justification for its position and it is inserted in the commitment store CS_{a_s} of $Agent(a_s)$.
$(\Diamond K_{a_r} B_{a_s} arg) \wedge (arg \in CS_{a_s})$

$\{$***pre***$\}\ ACCEPT(a_s, a_r, \phi)\ \{$***post***$\}$ This utterance has different semantics depending on the content of ϕ. On one hand, ϕ can be a position $Solution(p)$ proposed by $Agent(a_r)$.

- *pre*: $Agent(a_s)$ wants to accept a position $Solution(p)$ proposed by $Agent(a_r)$.
$(K_{a_s} B_{a_r} p) \wedge (B_{a_s} p)$
- *post*: $Agent(a_r)$ knows that $Agent(a_s)$ has accepted its position. Also, this position is inserted into the commitment store CS_{a_s} of $Agent(a_s)$ (and replaces a previous position if any).
$(\Diamond K_{a_r} B_{a_s} p) \wedge (p \in CS_{a_s}) \wedge (\nexists p_s \in CS_{a_s})(p_{a_s} \neq p)$

On the other hand, ϕ can be an argument $Argument(arg_r)$ proposed by $Agent(a_r)$.

Utterance	Lliteral meaning
$FINISHDIALOGUE(a_s)$	Performative to inform an agent a_s that it must perform the necessary actions (if any) before withdrawing from the dialogue.
$DIE(a_s)$	Performative to inform an agent a_s that it must *shutdown* its execution.
$GETALLPOSITIONS(a_s, cs)$	Performative to request the Commitment Store cs the list of available positions at a certain step of the dialogue. The Commitment Store uses the same performative to answer this request.

Table 1: Life Cycle Performatives of the argumentation protocol

- *pre*: $Agent(a_s)$ wants to accept the $Argument(arg_r)$ proposed by $Agent(a_r)$ and there is not any inconsistent argument in the commitment store CS_{a_s} of $Agent(a_s)$.
  $(K_{a_s} B_{a_r} arg_r) \wedge (B_{a_s} arg_r) \wedge (\nexists arg_{a_s} \in CS_{a_s})(arg_r \vdash^{-} arg_s)$
- *post*: $Agent(a_r)$ knows that $Agent(a_s)$ has accepted its argument. Also, this argument is inserted into the commitment store CS_{a_s} of $Agent(a_s)$.
  $(\Diamond K_{a_r} B_{a_s} arg_r) \wedge (arg_r \in CS_{a_s})$

{***pre***} $ATTACK(a_s, a_r, \phi)$ {***post***} This utterance has different semantics depending on its content ϕ, which represents different types of arguments. In any case an argument cannot be inserted in the commitment store of an agent without deleting first any inconsistent argument. With the $ATTACK$ locution, the $Agent(a_s)$ puts forward an $Argument(arg_s)$ to attack the $Argument(arg_r)$ of an $Agent(a_r)$, such that $Argument(arg_s) \vdash^{-} Argument(arg_r)$. $Argument(arg_s)$ can be of different types.

On one hand, if *Argument*(arg_r) is a support argument with one or more premises in its support set, such that *hasSupportSet*$(arg_r, SS_r) \wedge$ *Premise*$(pr_r) \wedge$ *hasPremise*(SS_r, pr_r), *Argument*(arg_s) can be a *distinguishing-premise* attack.

- *pre*: $Agent(a_s)$ wants to attack the support $Argument(arg_r)$ of an $Agent(a_r)$ with an $Argument(arg_s)$, such that *hasSupportSet(arg,* SS_s*)* $\wedge$ *Premise(DP)* $\wedge$ *hasDistinguishingPremise*$(SS_s$*, DP)*.
  $(Done[ASSERT(a_r, a_s, arg_r), ...]) \wedge (\exists arg_s)(arg_s \vdash^{-} arg_r) \wedge (\nexists arg_{s'} \in CS_{a_s})$ $(arg_s \vdash^{-} arg_{s'})$
- *post*: $Agent(a_r)$ knows that $Agent(a_s)$ does not believe its support $Argument(arg_r)$ and $Argument(arg_s)$ is inserted into the commitment store CS_{a_s} of $Agent(a_s)$.
  $(\Diamond K_{a_r} \neg B_{a_s} arg_r) \wedge (K_{a_r} B_{a_s} arg_s) \wedge (arg_s \in CS_{a_s})$

On the other hand, if $Argument(arg_r)$ is a support argument with one or more *argument-cases* or *domain-cases* in its support set, such that *hasSupportSet*$(arg_r, SS_r) \wedge Case(c_r) \wedge (hasDomainCase(SS_r, c_r) \vee hasArgumentCase(SS_r, c_r))$, then $Argument(arg_s)$ can be a *counter-example* attack. In that case, the axiomatic semantics coincide with the previous case.

In each step of the dialogue, the different utterances that can be received and generated are specified and the corresponding actions of the argument management process are performed. These actions can be easily modified to change the behaviour of the argumentative agents. Also, the performatives $FINISHDIALOGUE$,DIE and $GETALLPOSITIONS$ (see Table 1) have been added to manage the life cycle of argumentative agents and get information from the Commitment Store.

Figure 5 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 attack

others' positions. In the figure, dotted states represent *wait states* where the argumentative agent waits for messages from other agents or the Commitment Store. Also, dotted lines represent transitions between states when these incoming messages, with their associated performative, are received. Therefore, the transitions between states depend on a set of *dialogue rules* that specify what utterances can be made under what circumstances in the dialogue. The rules of the argumentation dialogue are described as follows:

(1) **R1:** when the agent is initialized it remains in the *Open* state waiting for an *OPEN-DIALOGUE* performative. The agent will move back to this state when the initiator agent communicates that the dialogue has finished. The *OPENDIALOGUE* performative informs the agent that a new dialogue to solve a problem has started. Also, when an agent received the *DIE* performative in this state, it must shutdown its execution.
(2) **R2:** in the *Enter* state, the agent will retrieve such cases of its domain case-base whose features match the given problem with a similarity degree greater than a given threshold. 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 performative *ENTERDIALOGUE* and will go to the state "Propose". The agent only engages in the dialogue if it has solutions to propose. Otherwise, the agent can refuse to engage in the dialogue with the performative *WITHDRAWDIALOGUE*.
(3) **R3:** when the agent is in the *Propose* 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 suitable and go to the state "Central". Otherwise, the agent will leave the dialogue with the performative *WITHDRAWDIALOGUE*.
(4) **R4:** in the *Central* state, 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* request from other agent. This performative is used to ask an argumentative agent to justify its position. In that case, the agent will go to the state "Assert" to try to generate a support argument for its position. If the agent has not received any *WHY* request before a specified *timeout*, it will go to the state "Query Positions" to challenge the positions of other agents. Also, an agent can be reported by other agent that the latter *ACCEPT*s its position. Alternatively, the agent can receive a *FINISHDIALOGUE* performative in this state, and it will go to the state "Send Position" to start the actions to leave the dialogue when it has proposed yet a position.
(5) **R5:** in the *Assert* state, the agent that received the *WHY* request will *ASSERT* a support argument to the opponent if it can. This implies going to the state "Wait Attack" and wait for incoming attack arguments. If the agent is not able to provide a support argument to defend its position, it must move back to the state "Propose" with the performative *NOCOMMIT* to withdraw its position from the dialogue and if possible, propose another generated position. Also, argumentative agents do not respond to the same *WHY* query from the same opponent agent twice. In this case, the argumentative agent will move back to the state "Central" and ignore the repeated *WHY* request.
(6) **R6:** in the *Wait Attack* state, the agent that has put forward a support argument for its position waits for an *ATTACK* or an *ACCEPT* performative. In the case that an *ATTACK* is received, the agent will go to the state "Defend" to try to rebut the attack. If the agent receives an *ACCEPT*, it means that the opponent agent has accepted its position and the proponent agent will move back to state "Central".
(7) **R7:** in the *Defend* state, an agent that has received an *ATTACK* from an opponent agent tries to reply with another *ATTACK*. In this case, the proponent agent will move back to the state "Wait Attack" to wait for the response of the opponent agent. However, if the proponent agent is not able to counter-attack, it must move back to the state "Propose"

with the performative *NOCOMMIT* to withdraw its position from the dialogue and if possible, propose another generated position.

(8) **R8:** in the *Query Positions*, state the agent that has decided to challenge the positions of other agents requests the Commitment Store the list of current positions proposed in the dialogue with the performative *GETALLPOSITIONS*.

(9) **R9:** in the *Get Positions* state, the agent that has requested the Commitment Store for active positions in the dialogue moves to this state to wait for an answer. Here, the agent can receive the list of positions with the same performative *GETALLPOSITIONS*. Alternatively, the agent can receive a *FINISHDIALOGUE* performative in this state, and it will go to the state "Send Position" to start the actions to leave the dialogue when it has proposed yet a position. Also, if the agent does not receive any response before a specified *timeout*, it will move back again to the state "Central".

(10) **R10:** in the *Why* state, the agent that has received a list of potential positions to challenge makes a random choice to challenge one of them with the performative *WHY* (from these positions that are different to its own). Otherwise, if there are no positions to challenge and ask for a justification, the agent moves back to the state "Central".

(11) **R11:** in the *Wait Assert* state, the agent that has challenged a position waits for a response from the challenged agent. If it receives such a response with the performative *ASSERT*, it tries to rebut the position of the challenged agent and moves to the state "Attack". Otherwise, if the challenged agent cannot provide a justification for its position, it must withdraw such position from the dialogue with the performative *NOCOMMIT* and the challenger moves back to the state "Central". Also, if the agent does not receive any response before a specified *timeout*, it will move again to the state "Central".

(12) **R12:** in the *Attack* state, if an agent has received a justification for a challenged position, it tries to generate an attack with the performative *ATTACK*. In this case, the agent moves to the state "Attack2". However, if it is not able to attack the position, it will accept it with the performative *ACCEPT* and move back to the state "Central".

(13) **R13:** in the *Attack2* state, once an agent has generated an attack for the position of a proponent agent, it waits in this state for the answer of the proponent. Thus, if the proponent is able to counter-attack and sends a performative *ATTACK*, the agent will move back to the state "Attack" to try to generate a new attack to rebut the position of the proponent. Otherwise, the proponent must withdraw its position from the dialogue with the performative *NOCOMMIT* and the attacker agent moves back to the state "Central".

(14) **R14:** an agent reaches the *Send Position* state when it has received a performative *FINISHDIALOGUE* to start the actions to leave the dialogue when it has proposed yet a position. Here, the agent sends its current position to update the Commitment Store information and avoid possible inconsistencies. After that, the agent moves to the state "Solution" to wait for the final solution applied to solve the problem.

(15) **R15:** in the *Solution* state, when the dialogue has finished, the initiator agent selects the final solution to apply and conveys this information to all dialogue participants. When agents receive this information, they update their case-bases with the data learnt from the dialogue and move back to the state "Open".

(16) **R16:** agents move to the *Die* state when they receive the performative *DIE* and shutdown their execution. This is the final state of the argumentation protocol.

In this Section we have described the main components of the infrastructure. The Commitment Store and the knowledge interchange mechanism have been explained. Furthermore, we have explained in detail the argumentative agents and their components: the Domain CBR module, the Argumentation CBR module, the argument management process, and the argumentation protocol. In the next Section we apply the infrastructure to a real problem, a call centre application. Also, several tests are performed to evaluate the infrastructure.

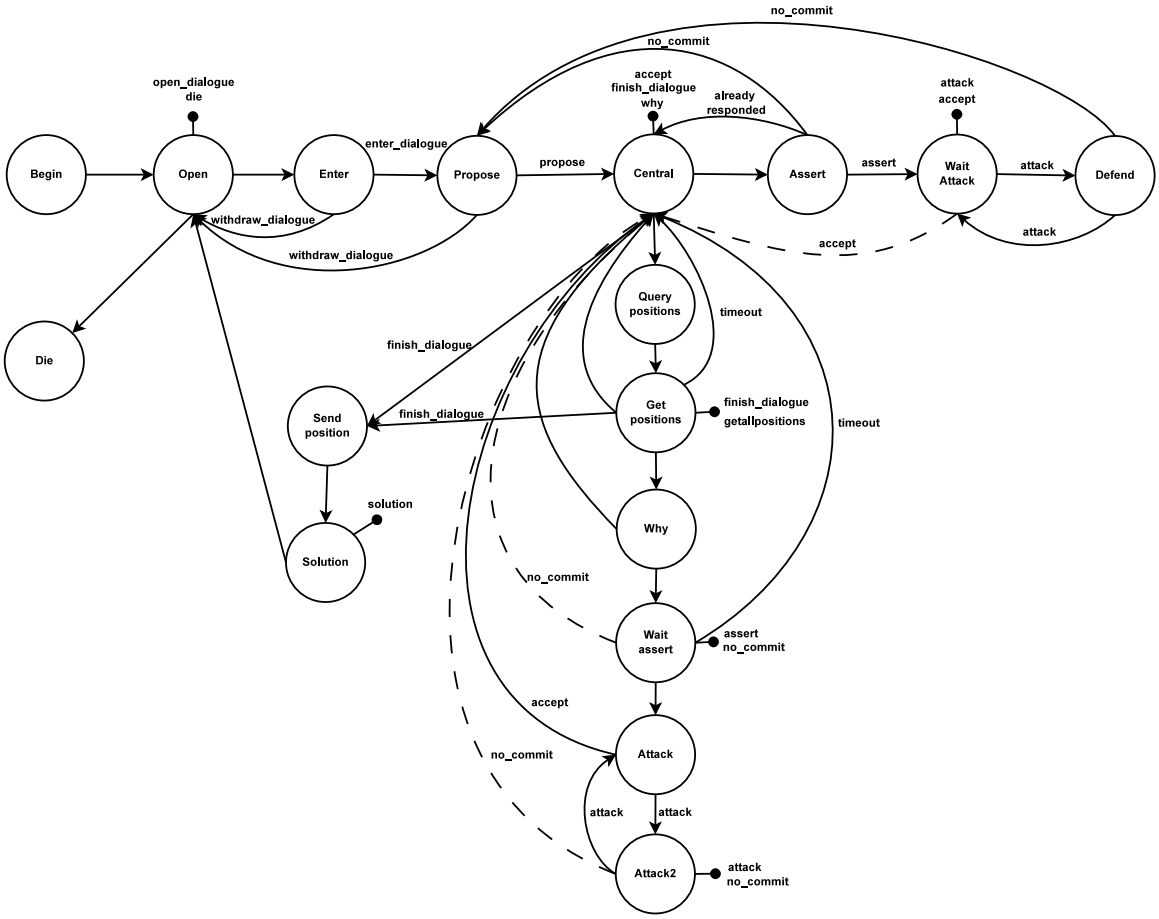

FIGURE 5: Argumentation Protocol that defines the behaviour of an agent in the protocol and the process that follows to propose positions

5. EVALUATION

In this Section, 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. Call centre 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 customer 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 call centre. In a call centre, there are a number of operators whose role is to provide the customers with technical assistance. 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 different levels, such as: Base operators, who receive customer queries and answer the ones from which they have

background training; and Expert operators, who are the technicians in charge of solving new problems.

To reuse previous solutions 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. Therefore, many CBR applications have been proposed in this domain. 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 which presented a CBR module (without argumentation capabilities) that acts as a solution recommender for customer support environments (Heras et al., 2009). That 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 module 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. Thus, several experts could provide different solutions and hence, they need a mechanism to negotiate and reach an agreement about the best solution to apply.

In this work, we propose an argumentation infrastructure for agent societies to automate the system. The infrastructure 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 argumentative agents that 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. In order to show how the prototype works, the data-flow for the problem-solving process to solve each ticket is shown in Figure 6 and described below:

(1) Some argumentative agents run in the platform and represent the operators and experts of the call centre. The argumentation process begins when a ticket that represents an incidence to be solved is received by an agent (i.e. the initiator of the argumentation process). Then, this agent sends the ticket to the agents of its group.
(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 module 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 position is any solution derived from a domain-case of the domain CBR module 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 module for each possible position to defend. With these queries a *suitability degree* of the positions is obtained. This degree represents if a

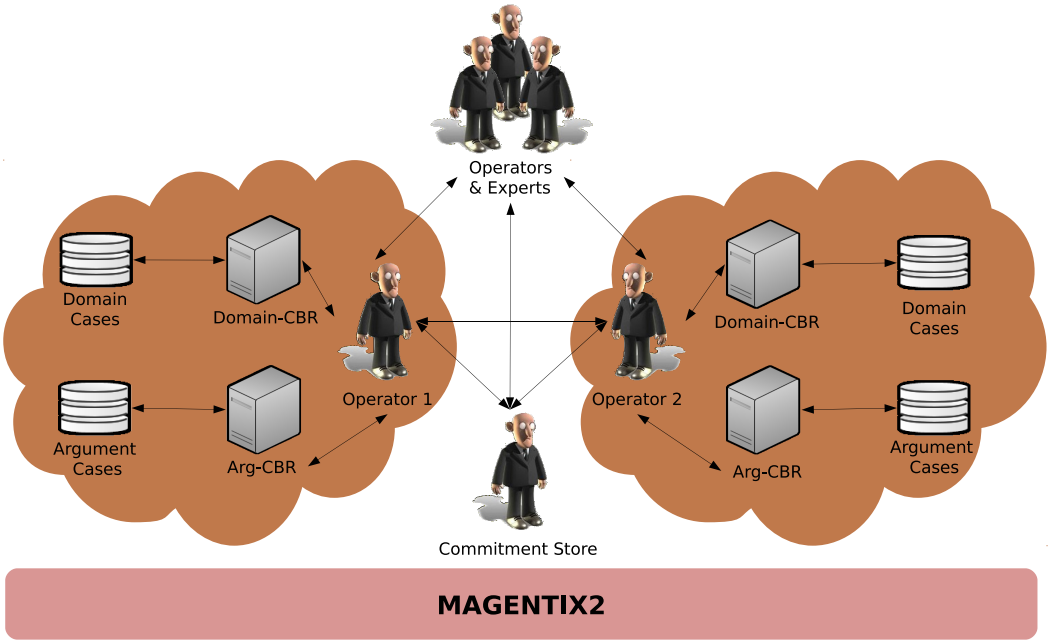

FIGURE 6: Data-flow for the argumentation process of the call centre application

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.

(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 another 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. 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. If there is more than one position available (no total agreement is reached) the most frequent position is selected as the solution (or a random choice is made in case of draw). The initiator agent communicates the solution to the participating agents.

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

5.2. Test design

In this Section, three aspects are evaluated in order to validate the infrastructure. Firstly, we evaluate the average prediction error with respect to the knowledge of the argumentative

agents. Then, we analyze the generated performatives in the argumentation dialogues to evaluate the performance of dialogues (in terms of its duration) in different conditions. Finally, the times that the operators and the experts win in the dialogues with their proposed position is analyzed to observe the behaviour of different kinds of argumentative agents in the dialogues.

In these tests, the domain case-bases of each argumentative agent are populated randomly by using some of the 48 tickets (stored as 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. An initiator 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. This agent is not an argumentative agent and it is not involved in the dialogue.

In the performed tests, we use two different roles of agents: operators and experts. On the one hand, an operator is an agent that has general knowledge to solve different types (categories) of problems. On the other hand, an expert is an agent that has specific knowledge to solve certain types (categories) of problems and has its case-base of domain-cases is populated with cases that solve them. Thus, the expert domain 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 case-base, and there is 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) from those available in the full case-base.

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 this dependency relation among them.

5.3. Prediction error with respect to the knowledge of the agents

For the test shown in Figure 7, 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. The prediction error of the system decreases as the number of domain-cases grows. Different configurations of agents (operators and experts) have been tested. On the one hand, operators with more knowledge have more domain-cases of all types of problems, so they can give solutions to more types problems. On the other hand, the experts increase their domain case-base with more than one speciality. So an expert with more domain-cases have several solutions to more than one type of problem.

Having a hybrid group of agents with a similar number of operators and experts is not optimal in terms of prediction error. In Figure 7, the groups with only operators and only experts obtain lower error rate than hybrid groups. The reason behind that is the behaviour of the experts. As commented before, an expert has an *authorisation* dependency relation over operators. In addition, an expert has specialized knowledge of certain type or types of problems, depending on the amount of domain-cases of its case-base.

Therefore, having from 1 to 5 experts in a group of 7 argumentative agents implies more error rate than other configurations when the amount of domain-cases is low. The

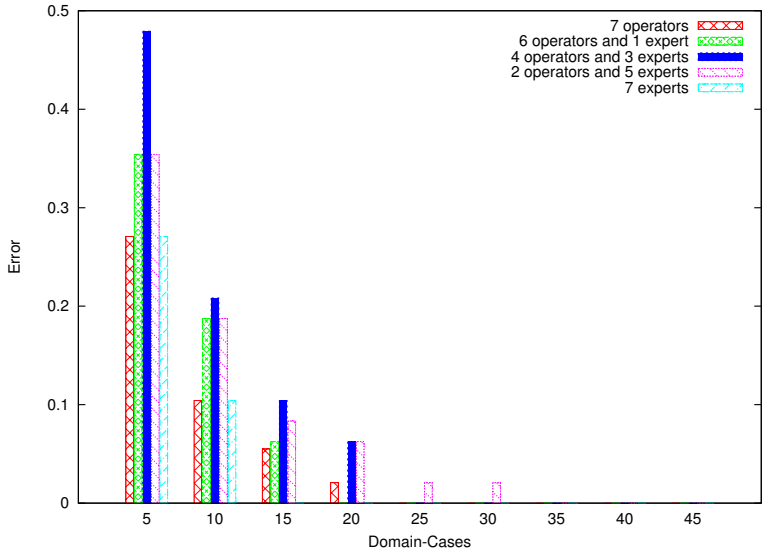

FIGURE 7: Average error in the prediction of the best solution

experts have a higher dependency relation over the technicians and they are imposing their opinion, but in most cases their solution is not the correct one. This is because they have lots of knowledge about certain types of problems, but not for all types. So for problems that the experts do not have the correct knowledge, they propose similar solutions that are incorrect. Nevertheless, a group of only experts have more knowledge together, and as their dependency relation between each other is the same, the expert or experts with the correct solution can persuade the others to accept their solution (because it is better supported than other solutions). Having the opposite configuration of agents, that is, a group formed only by operators, the results are almost equal than with a group of only experts (note that a group of only experts has lower prediction error than a group of only operators when they have enough knowledge). The reason is similar to the previous case. No one is imposing its opinion, and the knowledge is distributed to all agents. Then, if one or more operators have the correct solution, they will persuade the others to accept it.

In conclusion, a hybrid group of operators and experts obtain worse results than having groups of only operators or experts when the amount of domain-cases is low. So an expert provides better solutions if it has full knowledge of several types of problems, or if it works with some other experts that complement its knowledge (and hence the overall prediction error of the system decreases). On the other side, a group of operators with or without an expert can obtain good results because the expert (if it exists in the group) not always has an opinion to impose (note that agents do not participate in the dialogue if they are not able to generate positions) and the operators can reason about the best solution to apply. The observed behaviour in this test can be compared to a real company with operators and experts. If an expert does not have enough knowledge about all types of problems, imposes its wrong opinion to the operators and the final solution applied is not accurate. However, if this expert has full knowledge or if it is working with more experts, the solutions applied are better.

5.4. Generated performatives in the argumentation dialogues

In order to evaluate the performance of the system in terms of the duration of dialogues, we show the generated performatives in the argumentation dialogues. As explained before, a

performative is a conversational particle by which an argumentative agent expresses something. For example, ask about someone's position, propose positions, attack positions or arguments, retract positions or arguments, and so on. Therefore, the generated performatives in the dialogues are an interesting way to measure the complexity of the dialogues.

Figure 8 represents the mean generated performatives per dialogue of 48 different dialogues for different amounts of domain-cases in the case-bases of the argumentative agents, from 5 to 45. As the number of domain-cases grows, there are more generated performatives in the dialogues. This is because agents generate more positions and arguments to propose. So when there are less domain-cases, the agents that do not have a solution to propose do not enter into the dialogue. Then, if there are more domain-cases the agents have more solutions to propose and the dialogues are longer. This tendency happens from 5 to 40 domain-cases, but with 45 domain-cases the mean generated performatives decreases. This can be explained since agents have almost all the possible knowledge of the system, then almost all of them propose the correct solution in the beginning of the dialogue and they do not need to argue. Furthermore, with more generated performatives the dialogue is longer, so it lasts more time and this could be a problem if we care about the solving speed.

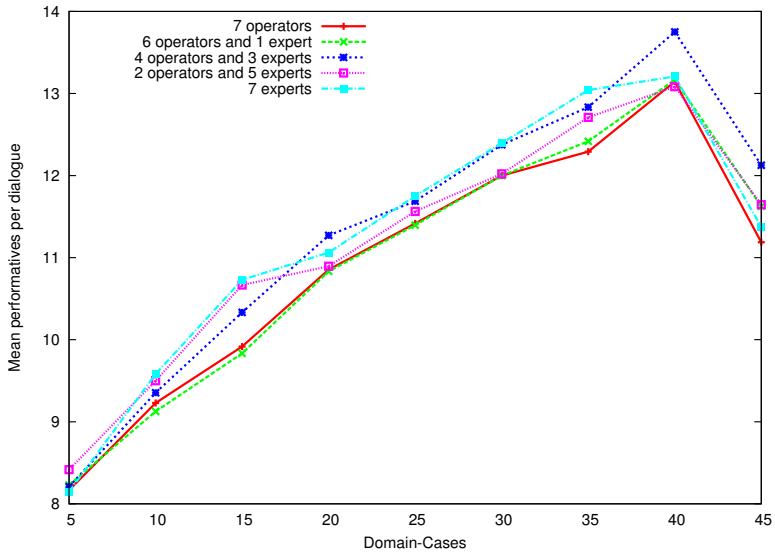

FIGURE 8: Mean generated performatives in argumentation dialogues

With the tests performed before and the results shown in Figure 8, we can conclude that the difference of generated performatives with different combinations of operators and experts is not too relevant. However, we have made other tests (not shown in this work) to find out the optimal number of agents in a dialogue. We have used a configuration of 7 argumentative agents in the tests of this work because it is the optimal number with the amount of domain-cases of this application scenario. So the absolute number of agents involved in the dialogues is really important. That is, with more agents in a dialogue there are more generated performatives, so more execution time. But, the advantage of this is that they may achieve the correct solution because they have more knowledge all together. Therefore, it is important to find out the number of agents that obtains lower prediction error but having shorter dialogues (less performatives). This number depends on the initial domain-cases that can be distributed among agents.

5.5. Percentage of times that operator or expert positions win in dialogues

Figure 9 shows the mean percentage of times that the operators or experts positions win in the argumentation dialogues. That is, if the final solution to apply has been proposed by an operator or an expert. In Figure 9, the combinations with only operators and only experts are not shown because they have 100% of chances to win. Hybrid combinations are shown to see the evolution of the percentage of times that the experts win as the number of experts grows. The data shown in the graph represents the mean percentage of times that operators or experts win of a set of tests performed with 5 to 45 domain-cases in the case-bases of the argumentative agents, as the previous tests.

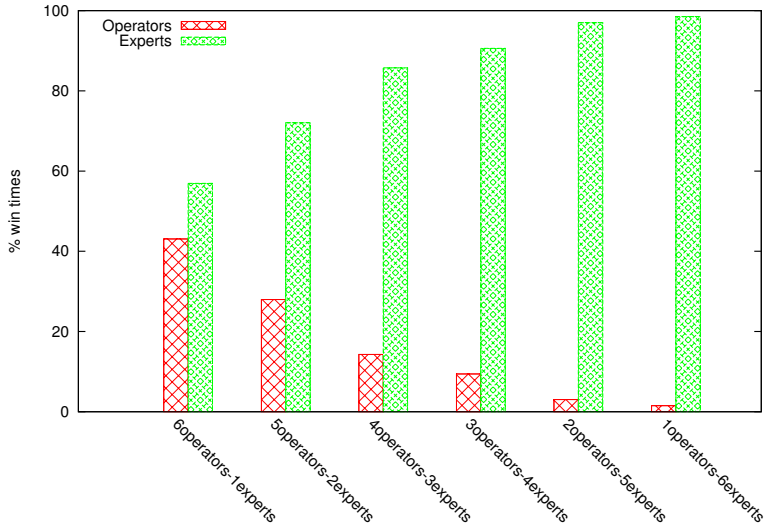

FIGURE 9: Mean percentage of win positions in argumentation dialogues

Having an expert, the percentage of times that it wins is almost of 60%. And as the number of experts grows, the percentage grows. This is again because an expert has a higher dependency relation (it is a level above in the hierarchy) over the operators and hence, it imposes its opinion about the solution to solve the problem. However, the solution given by an expert could not be the correct one, as commented before in the results shown in Figure 7. Therefore, in the situations of having from 1 to 5 experts and specially when these experts have a low number of domain-cases in their case-bases, the high percentage of times in which the experts win can give rise to a higher prediction error, because they do not have enough knowledge to propose good solutions but impose their opinion. Nevertheless, if there are more experts or they have more knowledge, they impose their opinion but the solution that they propose is the correct one.

In conclusion, in the argumentation dialogues the experts impose their opinion over the operators' opinion, and hence the experts win significantly more times than the operators. However, this is positive in terms of prediction error only when the number of experts is higher or their knowledge is more complete.

6. CONCLUSIONS

In this work, we have implemented an infrastructure to develop and execute 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. Also, it offers support for agent societies and takes into account the agents' social context. The infrastructure makes an effort to combine CBR methodology and argumentation in an open Multi-Agent System.

Furthermore, the infrastructure has been validated with an example in a customer support application. The data used to perform this evaluation is real data extracted from real problems of a company. The evaluation performed has been made with different configurations of the roles of argumentative agents (operators and experts) and increasing the domain case-base of the argumentative agents. The performed tests include the prediction error, the generated performatives in the dialogues and the percentage of winning positions between operators and experts.

The behaviour of the argumentative agents applied to a customer support domain has interesting similarities with the behaviour of a human society in a real company. An expert or experts that can impose their opinion over other operators can lead the group to obtain better results only if the experts have good enough knowledge of the domain. Otherwise, they impose incorrect solutions to other operators that could have the correct one. Therefore, if the experts or leaders of a society do not have enough knowledge it is better to work together with the whole society to obtain the best results.

As a future work, the infrastructure will be used in other domains to measure the performance and the differences having a largest database of solved problems. In addition, the results obtained in this work are a start point to simulate the behaviour of different agent societies to be compared to human societies. So the infrastructure can be used to study human societies and emergent behaviours due to their social relation.

ACKNOWLEDGEMENT

This work is supported by the Spanish government grants CONSOLIDER INGENIO 2010 CSD2007-00022, TIN2012-36586-C03-01, and TIN2011-27652-C03-01.

REFERENCES

ABRAHAM, A., E. CORCHADO, and J.M. CORCHADO. 2009. Hybrid learning machines. Neurocomputing, **72**(13-15):2729–2730.

ACORN, TIMOTHY L., and SHERRY H. WALDEN. 1992. Smart: support management automated reasoning technology for compaq customer service. *In* Proceedings of the fourth conference on Innovative applications of artificial intelligence, IAAI'92, AAAI Press. ISBN 0-262-69155-8. pp. 3–18.

AMGOUD, L., L. BODENSTA, M. CAMINADA, P. MCBURNEY, S. PARSONS, H. PRAKKEN, J. VAN VEENEN, and G. VREESWIJK. 2006. Project N 002307 ASPIC, Argumentation Service Platform with Integrated Components. Deliverable D2.6. Technical report, ASPIC Consortium.

ARGENTE, E., V. BOTTI, C CARRASCOSA, A. GIRET, V. JULIÁN, and M. REBOLLO. 2011. An Abstract Architecture for Virtual Organizations: The THOMAS approach. Knowledge and Information Systems, **29**(2):379–403. . http://dx.doi.org/10.1007/s10115-010-0349-1.

ATKINSON, KATIE. 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, T., and K. ATKINSON. 2009. Argumentation in Artificial Intelligence, Chapter Abstract argumentation and values, pp. 45–64. Springer.

BENCH-CAPON, TJM, and PE DUNNE. 2007. Argumentation in artificial intelligence. Artificial Intelligence, **171**(10-15):619–938.

CORCHADO, EMILIO, AJITH ABRAHAM, and ANDRÉ CARLOS CARVALHO. 2010. Hybrid intelligent algorithms and applications. Information Science, **180**(14):2633–2634.

DUNG, PHAN MINH. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic

reasoning, logic programming, and n -person games. Artificial Intelligence, **77**:321–357.

HAMBLIN, CHARLES LEONARD. 1970. Fallacies. Methuen & Co. Ltd.

HERAS, S., V. BOTTI, and V. JULIÁN. 2009. Challenges for a CBR framework for argumentation in open MAS. Knowledge Engineering Review, **24**(4):327–352.

HERAS, S., V. BOTTI, and V. JULIÁN. 2012. Argument-based agreements in agent societies. Neurocomputing, **75**(1):156–162.

HERAS, S., J. A. GARCÍA-PARDO, R. RAMOS-GARIJO, A. PALOMARES, V. BOTTI, M. REBOLLO, and V. JULIÁN. 2009. Multi-domain case-based module for customer support. Expert Systems with Applications, **36**(3):6866–6873.

HERAS, S., J. JORDÁN, V. BOTTI, and V. JULIÁN. 2013. Argue to Agree: A Case-Based Argumentation Approach. International Journal of Approximate Reasoning, **54**(1):82–108.

JORDÁN, JAUME, STELLA HERAS, SOLEDAD VALERO, and VICENTE JULIÁN. 2011. 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 6678 of *LNAI*, Springer, pp. 396–403.

KARACAPILIDIS, NIKOS, and DIMITRIS PAPADIAS. 2001. Computer supported argumentation and collaborative decision-making: the HERMES system. Information Systems, **26**(4):259–277.

KARACAPILIDIS, NIKOS, BRIGITTE TROUSSE, and DIMITRIS PAPADIAS. 1997. Using case-based reasoning for argumentation with multiple viewpoints. *In* 2nd International Conference on Case-Based Reasoning Research and Development, ICCBR-97, Springer, pp. 541–552.

KOLODNER, JL. 1993. Case-based Reasoning.

MCBURNEY, PETER, ROGIER M VAN EIJK, SIMON PARSONS, and LEILA AMGOUD. 2003. A dialogue game protocol for agent purchase negotiations. Autonomous Agents and Multi-Agent Systems, **7**(3):235–273.

MCBURNEY, P., and S. PARSONS. 2004. Locutions for argumentation in agent interaction protocols. *In* Revised Proceedings of the International Workshop on Agent Communication, AC-04, Volume 3396 of *LNAI*, Springer, pp. 209–225.

MCBURNEY, P., and S. PARSONS. 2009. Argumentation in Artificial Intelligence, Chapter Dialogue games for agent argumentation, pp. 261–280. Springer.

MUÑOZ, ANDRÉS, and JUAN A BOTÍA. 2008. Asbo: Argumentation system based on ontologies. *In* Cooperative Information Agents XII. *Edited by* M. Klusch, M. Pchouek, and A. Polleres, Volume 5180 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 191–205. ISBN 978-3-540-85833-1. . http://dx.doi.org/10.1007/978-3-540-85834-8_16.

MUÑOZ, ANDRÉS, and JUAN A BOTÍA. 2009. On the formalization of an argumentation system for software agents. *In* Hybrid Artificial Intelligence Systems. *Edited by* E. Corchado, X. Wu, E. Oja, . Herrero, and B. Baruque, Volume 5572 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 459–467. ISBN 978-3-642-02318-7. . http://dx.doi.org/10.1007/978-3-642-02319-4_55.

ONTAÑÓN, SANTIAGO, and ENRIC PLAZA. 2006. Arguments and counterexamples in case-based joint deliberation. *In* 3rd International Workshop on Argumentation in Multi-Agent Systems, ArgMAS-06, ACM Press.

ONTAÑÓN, SANTIAGO, and ENRIC PLAZA. 2007. Learning and joint deliberation through argumentation in multi-agent systems. *In* 7th International Conference on Agents and Multi-Agent Systems, AAMAS-07, ACM Press.

PARSONS, SIMON, CARLES SIERRA, and NICK R JENNINGS. 1998. Agents that reason and negotiate by arguing. Journal of Logic and Computation, **8**(3):261–292.

RAHWAN, IYAD, and LEILA AMGOUD. 2006. An argumentation-based approach for practical reasoning. *In* 5th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS-06, ACM Press, pp. 347–354.

RAHWAN, IYAD, SARVAPALI D RAMCHURN, NICHOLAS R JENNINGS, PETER MCBURNEY, SIMON PARSONS, and LIZ SONENBERG. 2003. Argumentation-based negotiation. The Knowledge Engineering Review, **18**(4):343–375.

RAHWAN, I., and G. SIMARI editors. 2009. Argumentation in Artificial Intelligence. Springer.

ROTH-BERGHOFER, THOMASR. 2004. Learning from homer, a case-based help desk support system. *In* Advances in Learning Software Organizations. *Edited by* G. Melnik and H. Holz, Volume 3096 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 88–97. ISBN 978-3-540-22192-0. . http://dx.doi.org/10.1007/978-3-540-25983-1_9.

SHOHAM, Y., and K. LEYTON-BROWN. 2009. Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations. Cambridge University Press.

SOH, LEEN-KIAT, and COSTAS TSATSOULIS. 2001a. Agent-based argumentative negotiations with case-based reasoning. *In* Working Notes of the AAAI Fall Symposium Series on Negotiation Methods for Autonomous Cooperative Systems, AAAI Press, pp. 16–25.

SOH, LEEN-KIAT, and COSTAS TSATSOULIS. 2001b. Reflective negotiating agents for real-time multisensor target tracking. *In* 17th International Joint Conference on Artificial Intelligence, IJCAI-01, Volume 2, Morgan Kaufmann Publishers Inc., pp. 1121–1127.

SOH, LEEN-KIAT, and COSTAS TSATSOULIS. 2005. A real-time negotiation model and a multi-agent sensor network implementation. Autonomous Agents and Multi-Agent Systems, **11**(3):215–271.

SYCARA, KATIA. 1987. Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods. Ph. D. thesis, Georgia Institute of Technology.

SYCARA, KATIA. 1989. Argumentation: Planning other agents' plans. *In* 11th International Joint Conference on Artificial Intelligence, IJCAI-89, Volume 1, Morgan Kaufmann Publishers Inc., pp. 517–523.

SYCARA, KATIA. 1990. Persuasive argumentation in negotiation. Theory and Decision, **28**:203–242.

TOLCHINSKY, PANCHO, ULISES CORTÉS, SANJAY MODGIL, FRANCISCO CABALLERO, and ANTONIO LÓPEZ-NAVIDAD. 2006. Increasing human-organ transplant availability: Argumentation-based agent deliberation. IEEE Intelligent Systems, **21**(6):30–37.

TOLCHINSKY, PANCHO, SANJAY MODGIL, KATIE ATKINSON, PETER MCBURNEY, and ULISES CORTÉS. 2012. Deliberation dialogues for reasoning about safety critical actions. Autonomous Agents and Multi-Agent Systems, **25**(2):209–259.

TOLCHINSKY, PANCHO, SANJAY MODGIL, ULISES CORTÉS, and MIQUEL SÀNCHEZ-MARRÈ. 2006. CBR and argument schemes for collaborative decision making. *In* 1st International Conference on Computational Models of Argument, COMMA-06, Volume 144, IOS Press, pp. 71–82.

VÁZQUEZ-SALCEDA, JAVIER, ULISES CORTÉS, JULIAN PADGET, ANTONIO LÓPEZ-NAVIDAD, and FRANCISCO CABALLERO. 2003. The organ allocation process: A natural extension of the carrel agent-mediated electronic institution. AI Communications, **16**(3):153–165.

WATSON, I. 1997. Applying case-based reasoning. Techniques for enterprise systems. Morgan Kaufmann Pusblishers, Inc.

WILLMOTT, STEVEN, GERARD VREESWIJK, CARLOS CHESÑEVAR, MATTHEW SOUTH, JARRED MCGINNIS, SANJAY MODGIL, IYAD RAHWAN, CHRIS REED, and GUILLERMO SIMARI. 2006. Towards an argument interchange format for Multi-Agent Systems. *In* 3rd International Workshop on Argumentation in Multi-Agent Systems, ArgMAS-06, ACM Press, pp. 17–34.