Information Systems Frontiers manuscript No.
(will be inserted by the editor)

An Ontological-based Knowledge-Representation Formalism for Case-Based Argumentation

Stella Heras · Vicente Botti · Vicente Julián

Received: date / Accepted: date

Abstract In open multi-agent systems, agents can enter or leave the system, interact, form societies, and have dependency relations with each other. In these systems, when agents have to collaborate or coordinate their activities to achieve their objectives, their different interests and preferences can come into conflict. Argumentation is a powerful technique to harmonise these conflicts. However, in many situations the social context of agents determines the way in which agents can argue to reach agreements. In this paper, we advance research in the computational representation of argumentation frameworks by proposing a new ontological-based, knowledge-representation formalism for the design of open MAS in which the participating software agents are able to manage and exchange arguments with each other taking into account the agents' social context. This formalism is the core of a case-based argumentation framework for agent societies. In addition, we present an example of the performance of the formalism in a real domain that manages the requests received by the technicians of a call centre.

Keywords Agreement Technologies · Argumentation · Multi-Agent Systems

1 Introduction

Open Multi-Agent Systems (MAS), where agents can enter or leave the system, interact, and dynamically form agent organisations to solve problems, seem to be a suitable technology for implementing large systems in terms of the services they offer and the entities that interact to provide or consume these services [22]. However, the high dynamism of the domains where open MAS commonly operate requires agents to have a way of reaching agreements that harmonise the conflicts that arise when they have to collaborate or coordinate their activities. In addition, agents in open MAS can form societies that link each other via dependency relations. These relations can emerge from agents' interactions or they can

Extended version of the paper presented by auhors at Agreement Technologies Conference 2013

S. Heras
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n 46022 Valencia (Spain)
E-mail: sheras@dsic.upv.es

be predefined by the system. Nevertheless, the dependencies between agents are part of their *social context*, which has an important influence on the way agents can reach agreements with other agents. To illustrate this idea, let us introduce an example scenario where a group of technicians is arguing to solve complex problems that have been reported to a call centre that is implemented as a MAS. In call centres, technicians provide technical assistance to individuals and/or organizations that have contracted the support service with the call centre company (e.g., users of hardware products, users of a specific brand of devices, or public administrations that have subcontracted the technical support or their computer systems). The call centre technicians usually have different responsibilities and authority. When a problem is reported to the centre, basic technicians (e.g., *operators*) try to solve the problem individually. However, if this first level of support fails, these complex problems are redirected to be solved by an *expert* or by a group of technicians that must collaborate in order to reach an agreement about the best solution to provide. Also, to make this decision different considerations can be taken into account, such as quality of response, speed (to fulfill the company Service Level Agreements (SLAs)), economic issues, etc. Therefore, in our scenario, we have a group of software agents that represent technicians that engage in agreement processes to solve problems. These agents have different social contexts, that include dependency relations with each other and different preferences over the *value* that is promoted by the final solution decided. We understand values to be the underlying reason that an agent has to prefer one solution over another (e.g., quality, speed, savings). For instance, operators may not be as willing to accept opinions from other operators as they are to accept opinions from an expert. Also, technicians do not necessarily behave in the same way as project managers; the technicians may belong to a group of experts in charge of providing high-quality solutions, while if they act as project *managers*, they are more worried about providing quick responses to fulfill SLAs and to avoid possible financial penalties.

Among the wide range of agreement technologies proposed in the last few years [30], argumentation provides a natural means of dealing with conflicts and knowledge inconsistencies that greatly resembles the way in which humans reach agreements [26]. However, little work has been done to study the effect of the social context of agents in the way that they argue. Commonly, the term *agent society* is used in the Argumentation and Artificial Intelligence (AI) literature as a synonym for an *agent organisation* [11] or a *group of agents* that play specific roles, follow certain interaction patterns, and collaborate to reach global objectives [23]. Our notion of agents' social context in an argumentation process includes information related to the proponent and the opponent of an argument, the group that these agents belong to, the dependency relation that they have, and the values that they want to promote [14]. Therefore, we endorse the view of value-based argumentation frameworks [6][5], which stress the importance of the audience in determining whether or not an argument is persuasive. Value-based argumentation frameworks extend abstract argumentation frameworks by addressing issues about the rational justification of choices. Thus, our approach to an agent society differs from the notion of agent *coalitions* used in [1] and [7], which are temporary associations between agents in order to carry out joint tasks, without any consideration for the social links and values that characterise those agents.

Starting from the idea that the social context of agents determines the way in which agents can argue and reach agreements, this context might have a decisive influence on the computational representation of arguments. In this paper, we advance research in the computational representation of argumentation frameworks by proposing a new ontological-based, knowledge-representation formalism for the design of open MAS in which the participating software agents are able to manage and exchange arguments with each other taking into account the agents' social context. This formalism is the core of a case-based argumentation framework for agent societies. To allow heterogeneous agents to interact in the framework, we need a common language to represent arguments and argumentation processes. Therefore, in the current work we present a new argumentation ontology called *ArgCBRonto* to represent case-based arguments and argumentation concepts. The structure of the paper is the following: Section 2 introduces the underlying argumentation framework; Section 3 shows the ontology-based knowledge representation formalism proposed; Section 4 shows an example scenario in the call centre domain; Section 5 reviews related work; Section 6 discusses some assumptions made in this paper; and Section 7 provides conclusions about the contributions of this research.

2 Case-Based Argumentation Framework

This section presents the formal specification of the case-based argumentation framework that motivated the design of the knowledge representation formalism presented in this paper[1]. Our framework has been implemented as an argumentation API in the *Magentix2*[2] agent platform, which provides new services and tools that allow for the secure and optimised management of open MAS.

We focus on argumentation processes that are performed among a set of agents that belong to an agent society and that must reach an agreement to solve a problem taking into account their social dependencies. Each agent builds its individual position in view of the problem (a solution for it). At this level of abstraction, we assume that this could be a generic problem of any type (e.g., a resource allocation problem, an agreed upon classification, a joint prediction, etc.) that could be characterised with a set of features. The following sections summarise the elements that characterise our argumentation framework.

2.1 The Logical Language

The logical language represents argumentation concepts and possible relations among them. In our framework, these concepts are represented in the form of cases and argumentation schemes. Therefore, the logical language of the argumentation framework is defined in terms of the vocabulary to represent these resources. The vocabulary of cases and schemes is defined by using the *ArgCBROnto* ontology presented in section 3. We have selected the Ontology Web Language OWL-DL[3] as the logical language to represent the vocabulary of cases. This variant is based on Description Logics (DL) and guarantees computational completeness and decidability. Note that OWL-DL does not assume *closed world reasoning* with *negation as failure*. By contrast, OWL-DL uses *open world reasoning* with *negation as unsatisfiability*. Therefore, something is false only if it can be proved to contradict other information in the ontology. This implies that two *classes* (i.e. *concepts* in description logics terminology) are *contradictory* if and only if they are specifically declared as such with the owl property *complementOf* and similarly, two *instances* (i.e. *individuals* in description logics terminology) are contradictory if and only if they are specifically declared as such with the owl property *differentFrom*.

[1] This is an extended and revised version of the formal specification of the framework presented in [14]

[2] Publicly available at `http://www.gti-ia.upv.es/sma/tools/magentix2/`

[3] *http://www.w3.org/TR/owl-guide/*

Definition 1 (Logical Language) *Let $\mathscr{L}$ be an OWL-DL language, C and D two owl classes, $c \in C$ and $d \in D$ two owl instances of the classes C and D respectively, and complementOf and differentFrom two contrariness functions from $\mathscr{L}$ to $2^{\mathscr{L}}$. Then, if C complementOf D (correspondingly c differentFrom d) and D complementOf C (d differentFrom c), C and D (c and d) are called contradictory. However, if C complementOf D (correspondingly c differentFrom d) and D $\neg$complementOf C (d $\neg$differentFrom c), C (c) is a contrary of D (d).*

To illustrate the difference between negation as failure and negation as unsatisfiability let us propose the following example in the call centre domain. Suppose that a manager is arguing with an operator to decide if, based on a case-base of previous experiences, the solution proposed by the operator should be selected as the best solution to provide to a customer that has reported the problem $P_{printer}$ to the call centre. If the manager cannot find a previous a case that in a similar situation applied the solution proposed by this operator, a closed world reasoner would infer that the solution is not appropriate (since the suitability of the solution cannot be proved). By contrast, an OWL-DL reasoner that follows the open world assumption would infer that, in principle, there is no reason to approve or deny the solution proposed by the operator. Thus, if there are no other reasons that prevent the manager to approve the solution, it could finally select it. This reflects the usual way of reasoning of the case-based reasoning methodology, which does not infer the negation of a conclusion by the mere fact that there is not a case in the case-base that supports this conclusion. OWL-DL allows automatic description logic reasoning over argument-cases and domain-cases. In addition, it facilitates the interoperability with other systems.

2.2 The Case-Based Notion of Argument

Following our case-based computational approach for the representation of arguments, we define a formal Argumentation Framework for an Agent Society (*AFAS*) as an instantiation of Dung's argumentation framework [10]. Thus any semantics of Dung's-like abstract argumentaion frameworks can be also applied in our framework [15]:

Definition 2 (Argumentation Framework for an Agent Society) *An argumentation framework for an agent society is a tuple AFAS = $<A, R, S_t>$ where: A is a set of arguments; R is an irreflexive binary attack relation on A; and S_t is a society of agents as defined in [15].*

The main advantages that our framework has over other existing argumentation frameworks are: 1) the ability to represent social information in arguments; 2) the possibility of automatically managing arguments in agent societies; 3) the improvement of the agents' argumentation skills; and 4) the easy interoperability with other frameworks that follow the argument and data interchange web standards. We have adopted the Argument Interchange Format (AIF) view of arguments [8] as a set of interlinked *premiss-illative-conclusion* sequences. The notion of argument is determined by our KI case-based framework for representing arguments. In our framework, agents can generate arguments by instantiating the premises that represent the context of the domain where the argument is put forward in an argumentation scheme or by retrieving similar previous cases and reusing their solutions. Therefore, agents construct arguments by using their individual knowledge bases, which contain these types of knowledge resources.

Definition 3 (Knowledge Base) *A Knowledge-Base in a case-based argumentation framework for agent societies consists of a finite set of $\mathscr{K} \subseteq \mathscr{L}$ elements, where $\mathscr{L}$ is a logical language to represent these elements and $\mathscr{K} = \mathscr{K}_p \cup \mathscr{K}_{dc} \cup \mathscr{K}_{ac} \cup \mathscr{K}_{as}$, where each of these* *subsets are disjoint.* $\mathscr{K}_p$ *is a finite set of premises;* $\mathscr{K}_{dc}$ *is a finite set of domain-cases;* $\mathscr{K}_{ac}$ *is a finite set of argument-cases; and* $\mathscr{K}_{as}$ *is a finite set of argumentation schemes.*

Agents use a *domain-cases* case-base to generate positions and arguments by reusing previous similar experiences. The structure of domain-cases is domain-dependent and we assume that we have an ontology to represent them in the specific application domain where the case-based argumentation framework is implemented. For instance, in our running example this ontology would include concepts (attributes) to characterise the software or hardware errors reported by the call centre customers. Let us assume that by means of an argumentation dialogue, the operator *Stella* effectively solved the problem $P_{printer}$ by proposing the solution *S1 = "Open the lid 1, remove the jammed paper and reset"* that *Stella*'s group agreed to as being the best solution proposed (here we assume that *Stella* did not have any domain-case in its domain-cases case-base representing this problem-solving situation, but *Stella* received the solution from another operator). Therefore, a new case:

$DCstella_1$ = {{*Brand = br1, Model = myPrint, Year = 2012, Network Printer = No, Paper Jam = Yes, Error Message = No*}, *S1*}

will be added, improving *Stella*'s problem solving skills.

By using their case-base of *argument-cases*, agents can select the best positions and arguments to put forward in a specific step of the argumentation dialogue in view of how persuasive similar positions and arguments were in the past. Thus, this knowledge resource improves the agents' argumentation skills by learning from argumentation processes. The structure of argument-cases is generic for all application domains and will be presented in Section 3 by using the *ArgCBROnto* ontology.

An argumentation scheme consists of a set of premises and a conclusion that is presumed to follow from them. Also, each argumentation scheme has an associated set of *critical questions* that represent potential attacks to the conclusion supported by the scheme. This knowledge resource can be used to generate positions and arguments. The concrete argumentation schemes that agents of our argumentation framework use depend on the application domain. For instance, in the call centre example we could have an argumentation scheme $AS1$ that changes the value preference order of a group of operators if the SLA contracted by the customer in the support service that they are providing is about to be breached (inspired by Waltons's *argument from an exceptional case* [32]):

Major Premise: *if the case of x is an exception, then the value preference order of the group can be waived and changed by* $quality < speed$ *in the case of x.*
Minor Premise: *the case of approaching the deadline contracted in a SLA is an exception.*
Conclusion: *therefore, the value preference order of the group can be waived and changed by* $quality < speed$ *in the case of approaching the deadline contracted in a SLA.*

With this scheme, operators of this group must provide quick solutions for the problems that they receive, even if their individual value preferences give priority to high quality solutions.

Note that by proposing these knowledge resources for our case-based argumentation framework, we do not assume that all agents participating in the agreement process must have this specific architecture. In fact, in addition to knowing the rules of the protocol that manages the argumentation dialogue to reach the agreement, all agents can understand and interchange arguments if they share the *ArgCBROnto* ontology.

Therefore, arguments that agents interchange are defined as tuples of the form:

Definition 4 (Argument) *$Arg = \langle \phi, v, \{S\} \rangle$, where ϕ is the conclusion of the argument, v is the value that the agent wants to promote with this argument, and S is a set of elements that support the argument (support set).*

This support set can consist of different elements, depending on the purpose of the argument. On one hand, if the argument provides a justification for a proposed position, it is called a *support argument*. Its support set is the set of features (*premises*) that represent the context of the domain where the argument has been put forward (those premises that match the problem to solve and other extra premises that do not appear in the description of this problem but that have also been considered in order to draw the conclusion of the argument). Optionally, any knowledge resource used by its proponent to generate the argument (*domain-cases*, *argument-cases*, or *argumentation schemes*). On the other hand, if the argument attacks the argument of an opponent, it is called an *attack argument* and its support set can also include any of the attacks that are allowed in our framework. These are: *critical questions*, *distinguishing premises*, or *counter-examples*.

When critical questions are instantiated by an opponent agent, the conclusion of the argument that is drawn by using their associated argumentation scheme is temporally rebutted (until new information demonstrates its validity, if possible). For instance, in the call centre example, the $AS1$, which is an adaption of *Walton*'s original scheme, could include the critical question *"Is the case of SLA a recognized type of exception?"*. Thus, if any operator of the group affected by the application of this scheme could provide evidence that demonstrates that this is not the case, the value preference order of the group would not be changed.

Distinguishing premises are premises that can invalidate the application of a case to generate a valid conclusion for an argument (i.e., is a premise that does not appear in the problem description and has different values for two cases or a premise that appears in the problem description that does not appear in one of the cases). For instance, let us assume that another operator of the call centre that is engaged in the argumentation dialogue to solve the problem $P_{printer}$ (e.g., *Vicente*) has a domain-case:

$DCvicente_1$ = {{*Brand = br1, Model = myPrint, Year = 2012, Network Printer = No, Paper Jam = Yes, Type of paper = recycled*}, *S2*}

in its domain-cases case-base. This domain-case proposes the alternative solution *S2 = "Do not use recycled paper with this printer model"* that states that this model of printer does not work well with recycled paper. In that case, *Stella* could cite the distinguishing premise *Error Message* to attack the solution proposed by *Vicente*. Also, note that the premise *Type of paper* cannot be cited as a distinguishing premise since it does not appear in the $P_{printer}$ problem description.

A counter-examples is a case that is similar to another case (their descriptions match), but they have different conclusions. In other words, a counter-example is a previous domain-case or an argument-case that was deemed acceptable, where the problem description of the counter-example matches the current problem to solve and also subsumes the problem description of the case but proposes a different solution. In the above examples, if the operator agent *Vicente* had another domain-case:

$DCvicente_2$ = {{*Brand = br1, Model = myPrint, Year = 2012, Network Printer = No, Paper Jam = Yes, Error Message = no, Type of paper = recycled*}, *S2*}

that proposed the same solution $S2$, it could use $DCvicente_2$ as counter-example for *Stella*'s domain-case $DCstella_1$, since the $DCvicente_2$ description subsumes $DCstella_1$ and proposes a different solution.

Thus, the support set of an argument can consist of the following tuple of support elements, depending on the supporting or attacking purpose of the argument:

Definition 5 (Support Set) *S* = { {*premises*}, {*domainCases*}, {*argumentCases*}, {*argumentationSchemes*}, {*criticalQuestions*}, {*distinguishingPremises*}, {*counterExamples*} }

Therefore, arguments can be constructed by aggregating different support and attack elements, which are structures that support intermediate conclusions that lead to the conclusion of the argument itself.

2.3 The Concept of Conflict between arguments

The concept of conflict between arguments defines in which way arguments can attack each other. There are two typical attacks studied in argumentation: *rebut* and *undercut*. In an abstract definition, rebuttals occur when two arguments have contradictory conclusions. Similarly, an argument undercuts another argument if its conclusion is inconsistent with one of the elements of the support set of the latter argument or its associated conclusion. This section shows how our framework instantiates these two attacks. Intuitively, arguments cannot be attacked on the support set premises that match the description of the problem to solve; they can only be attacked on those extra premises that represent the context of the domain where the argument was put forward and that do not appear in the description of the problem. Alternatively, arguments can be attacked on those premises that appear in the description of the problem to solve but that have not been considered to draw the conclusion of the argument (they do not appear in the support set of the argument). Taking into account the possible elements of the support set, rebut and undercut attacks can be formally defined as follows.

Let $Arg_1 = \langle \phi_1, value_1, \{S_1\} \rangle$ and $Arg_2 = \langle \phi_2, value_2, \{S_2\} \rangle$ be two different arguments, where S_1 = $\{\{Premises\}_1$, ..., $\{CounterExamples\}_1\}$, S_2 = $\{\{Premises\}_2$, ..., $\{CounterExamples\}_2\}$, $\sim$ stands for the logical negation, $\Rightarrow$ stands for the logical implication and $conc(x)$ is a function that returns the conclusion of a formula x. Then:

Definition 6 (Rebut) Arg_1 *rebuts* Arg_2 *iff* $\phi_1 = \sim\phi_2$ *and* $\{Premises\}_1 \supseteq \{Premises\}_2$

That is, if Arg_1 supports a different conclusion for a problem description that includes the problem description of Arg_2. To illustrate this concept, let us assume that the operator agent *Stella* has generated the support argument:

> $SAstella_1$ = ⟨ *S1, speed,* {{*Brand = br1, Model = myPrint, Year = 2012, Network Printer = No, Paper Jam = Yes, Error Message = No*}, $DCstella_1$, ∅, ∅, ∅, ∅, ∅ ⟩

to justify its proposed solution *S*1 (promoting, for instance the value of *speed*) and that the operator agent *Vicente* has generated an attack argument:

> $AAvicente_1$ = ⟨ *S2, quality,* {{*Brand = br1, Model = myPrint, Year = 2012, Network Printer = No, Paper Jam = Yes, Error Message = No, Type of paper = recycled*}, ∅, ∅, ∅, ∅, ∅, $DCvicente_2$ ⟩

with a counter-example that proposes the alternative solution *S*2 for the problem. In this situation, $AAvicente_1$ *rebuts* $SAstella_1$.

Definition 7 (Undercut) Arg_1 undercuts Arg_2 if
$1)\phi_1 = \sim conc(as_k) \mid \exists cq \in \{CriticalQuestions\}_1 \wedge \exists as_k \in \{ArgumentationSchemes\}_2 \wedge$
$cq \Rightarrow \sim conc(as_k)$, *or*
$2)\phi_1 = dp \mid (\exists dp \in \{DistinguishingPremises\}_1 \wedge \exists pre_k \in \{Premises\}_2 \wedge dp = \sim pre_k) \vee$
$(dp \notin \{Premises\}_2)$, *or*
$3)\phi_1 = ce \mid (\exists ce \in \{CounterExamples\}_1 \wedge \exists dc_k \in \{DomainCases\}_2 \wedge conc(ce) = \sim conc(dc_k)) \vee$
$(\exists ce \in \{CounterExamples\}_1 \wedge \exists ac_k \in \{ArgumentCases\}_2 \wedge conc(ce) = \sim conc(ac_k))$

That is, if the conclusion drawn from Arg_1 makes one of the elements of the support set of Arg_2 or its conclusion non-applicable in the current context of the argumentation dialogue. In case 1 Arg_1 undercuts Arg_2 by posing a critical question that attacks the conclusion of Arg_2, inferred by using an argumentation scheme. In case 2, Arg_1 undercuts Arg_2 by showing a new premise which value conflicts with one of the premises of Arg_2 or else, does not appear in the problem description of Arg_2. Finally, in case 3 Arg_1 undercuts Arg_2 by putting forward a counter-example for a domain-case or an argument-case that was used to generate the conclusion of Arg_2. Coming back to our running example on the call centre application domain, if the operator agent *Vicente* generates a support argument for its proposed solution $S2$ promoting for instance the value *quality*, but in this case based on its domain-case $DCvicente_1$ as:

> *SAvicente*$_1$ = ⟨ *S2, quality,* {{*Brand = br1, Model = myPrint, Year = 2012, Network Printer = No, Paper Jam = Yes, Type of paper = Recycled*}, *DCvicente*$_1$, ∅, ∅, ∅, ∅, ∅ ⟩

the operator agent *Stella* could *undercut* this argument with the attack argument:

> *AAstella*$_1$ = ⟨ *S1, speed,* {{*Brand = br1, Model = myPrint, Year = 2012, Network Printer = No, Paper Jam = Yes, Error Message = No*}, ∅, ∅, ∅, ∅, {*Error Message = No*}, ∅} ⟩

by invalidating the applicability of $DCvicente_1$, since it does not consider the distinguishing premise *"Error Message"* that the problem description includes.

2.4 The Notion of Defeat between arguments

Once possible attacks between arguments have been established, whether or not these attacks result in defeats depends on the defeat relation defined between a pair of arguments.

Let $Arg_1 = \langle \phi_1, value_1, \{S_1\} \rangle$ posed by agent ag_1 and $Arg_2 = \langle \phi_2, value_2, \{S_2\} \rangle$ posed by agent ag_2 be two conflicting arguments; let $Valpref_{ag_i} \subseteq VxV$, be an irreflexive, antisymmetric and transitive relation $<^{S_t}_{ag_i}$ over the agent's ag_i values in the society S_t; and let us define the possible dependency relations $Dependency_{S_t}$ between roles in our framework as:

- Pow_{S_t} (*Power*): when an agent has to accept a request from another agent because a predefined domination relationship between them (e.g. in a society S_t that operates in the call centre example, $Operator <^{S_t}_{Pow} Manager$, since basic operators must comply with the commands made by the manager of the centre.
- $Auth_{S_t}$ (*Authorisation*): when an agent has committed itself to another agent for a certain service and a request from the second agent leads to an obligation when the conditions are met (e.g. in the society S_t that is providing a support service to a specific customer of the call centre, $Operator <^{S_t}_{Auth} Expert$ since experts in this support service are believed to have specialised knowledge to solve problems of this type.

- $Char_{S_t}$ (*Charity*): when an agent is willing to answer a request from another agent without being obliged to do so (e.g. in the society S_t, by default $Operator <^{S_t}_{Char} Operator$).

  Then:

Definition 8 (Defeat) *Arg_1 defeats Arg_2 if*
((rebuts(Arg_1,Arg_2) $\wedge \sim$ undercut(Arg_2,Arg_1)) $\vee$ undercuts(Arg_1,Arg_2)) $\wedge$ $value_1 <^{S_t}_{ag_1} value_2$ $\notin Valpref_{ag_1}$ $\wedge$ Role(ag_1) $<^{S_t}_{Pow}$ Role(ag_2) $\notin Dependency_{S_t}$ $\wedge$ Role(ag_1) $<^{S_t}_{Auth}$ Role(ag_2) $\notin$ $Dependency_{S_t}$

Therefore, we express that the argument Arg_1 $defeats_{ag_1}$ from the ag_1 point of view the argument Arg_2 as $defeats_{ag_1}(Arg_1,Arg_2)$ if Arg_1 rebuts Arg_2 and Arg_2 does not undercut Arg_1, or else Arg_1 undercuts Arg_2, and ag_1 does not prefer the value promoted by Arg_2 to the value promoted by Arg_1 and ag_2 does not have a power or authority relation over ag_1. The first type of defeat poses a stronger attack on an argument, directly attacking its conclusion. In addition, an argument can strictly defeat another argument if the first defeats the second and the second does not defeat the first.

Definition 9 (Strict Defeat) *Arg_1 strictly defeats Arg_2 if Arg_1 defeats Arg_2 and Arg_2 does not defeat Arg_1*

For instance, in the call centre example, the argument of an expert will always strictly defeat the argument of a basic operator, assuming that the expert has an *authorisation* dependency relation over operators. Also, whether the argument of an operator defeats or not the argument of another operator strictly depends on the value preference order of the latter. For instance, if we consider that *Vicente* prefers to promote the value *quality* over *speed*, the undercut attack that *Stella* has posed with the argument $AAstella_1$ in the example of the previous section would not succeed from the *Vicente*'s point of view.

3 ArgCBROnto Ontology

As introduced above, we have designed the *ArgCBRonto* ontology to define the representation language of the case-based knowledge resources of our argumentation framework. Ontologies provide a common vocabulary to understand the structure of information among different software agents. In addition, ontologies allow assumptions about the domain to be made explicit, which facilitates changing these assumptions as new knowledge about the domain is acquired. The high dynamism of the domains where open MAS commonly operate gives rise to many changes in the domain knowledge that agents have available and they must be able to handle the consequences of these changes. Thus, the vocabulary of domain-cases, argument-cases and argumentation schemes is defined by using the *ArgCBROnto* ontology. The *ArgCBROnto* ontology follows the approach of the case-based KI systems proposed in [9]. KI-CBR enables automatic reasoning with semantic knowledge in addition to the syntactic properties of cases. This allows one to make semantic inferences with the elements of cases and to use more complex measures to compute the similarity between them. Therefore, this ontology provides a common language to represent resources and is computationally tractable. It is rich enough to represent different types of domain-specific and general knowledge, generic enough to represent different types of arguments, and compliant with the technological standards of data and argument interchange on the Web. Following, we provide a general view of the *ArgCBRonto* ontology for the argumentation framework proposed in this paper. Focussing on the concepts that define the knowledge resources presented in Section 2. The complete specification of the ontology is publicly available at `users.dsic.upv.es/∼vinglada/docs`.

In the top level of abstraction, the terminological part of the ontology distinguishes among several disjoint concepts. These include the following concepts: *Case*, which is the basic structure used to store the argumentation knowledge of agents; *CaseComponent*, which represents the usual parts that cases have in CBR systems; and *ArgumentationScheme*, which represents the argumentation schemes that the framework has.

$Case \sqsubseteq Thing \quad Case \sqsubseteq \neg CaseComponent$
$CaseComponent \sqsubseteq Thing \quad CaseComponent \sqsubseteq \neg ArgumentationScheme$
$ArgumentationScheme \sqsubseteq Thing \quad ArgumentationScheme \sqsubseteq \neg Case$

As pointed out above, there are two disjoint types of cases, *domain-cases* and *argument-cases*.

$ArgumentCase \sqsubseteq Case \quad DomainCase \sqsubseteq Case\ ArgumentCase \sqsubseteq \neg DomainCase$

Cases have the same three possible types of components that usual cases of CBR systems have: the description of the state of the world when the case was stored (*Problem*); the conclusion of the case (*Solution*); and the explanation of the process that gave rise to this conclusion (*Justification*). These concepts are disjoint.

$Problem \sqsubseteq CaseComponent \quad Solution \sqsubseteq CaseComponent \quad Justification \sqsubseteq CaseComponent$
$Problem \sqsubseteq \neg Solution \quad Solution \sqsubseteq \neg Justification \quad Problem \sqsubseteq \neg Justification$

Domain-cases have the usual problem, solution, and justification parts.

However, argument-cases have a more specialised description for these components (*ArgumentProblem*, *ArgumentSolution*, and *ArgumentJustification*), which includes an extended set of properties.

$ArgumentProblem \sqsubseteq Problem\ ArgumentSolution \sqsubseteq Solution\ ArgumentJustification \sqsubseteq Justification$
$ArgumentCase \sqsubseteq \forall hasArgumentProblem.ArgumentProblem$
$ArgumentCase \sqsubseteq \forall hasArgumentSolution.ArgumentSolution$
$ArgumentCase \sqsubseteq \forall hasArgumentJustification.ArgumentJustification$

Cases also have a unique identifier *ID* and a *creation date* as properties, with their corresponding range and domain.

$\top \sqsubseteq \forall hasID.Integer$ $\top \sqsubseteq \forall$ $hasID^{-}$.(Case $\sqcup$ SocialEntity $\sqcup$ Value $\sqcup$ Argument $\sqcup$ ArgumentationScheme $\sqcup$ Premise)[4] $\top \sqsubseteq \forall$ hasCreationDate.Date $\top \sqsubseteq \forall hasCreationDate^{-}$.(Case $\sqcup$ ArgumentationScheme)[3]

As indicated previously, argumentation schemes represent stereotyped patterns of common reasoning in the application domain where the framework is implemented. Each argumentation scheme consists of a set of *premises*, a *conclusion* drawn from these premises, and a set of *critical questions* that represent potential attacks to the conclusion supported by the scheme. These critical questions can be classified as *presumptions* that the proponent of the argumentation scheme has made or *exceptions* to the general inference rule that the scheme represents [28]. In the case of pressumptions, the proponent has the burden of proof if the critical question is asked, whereas in the case of exceptions the burden of proof falls on the opponent that has questioned the conclusion of the scheme.

[4] Note that this property has several concepts of the *ArgCBRonto* ontology as domain, some of which will be introduced later in this article.

$ArgumentationScheme \sqsubseteq Thing$
$ArgumentationScheme \sqsubseteq \forall hasPremise.Premise$ $ArgumentationScheme \sqsubseteq \forall hasConclusion.Conclusion$
$ArgumentationScheme \sqsubseteq \forall hasPresumption.Premise$ $ArgumentationScheme \sqsubseteq \forall hasException.Premise$

In addition, for each argumentation scheme, the *ArgCBROnto* ontology stores information about its unique *ID*, its *title*, its *creation date*, and its *author*.

$\top \sqsubseteq \forall argTitle.String$ $\top \sqsubseteq \forall argTitle^{-}.ArgumentationScheme$
$\top \sqsubseteq \forall creationDate.Date$ $\top \sqsubseteq \forall creationDate^{-}.ArgumentationScheme$
$ArgumentationScheme \sqsubseteq \forall hasAuthor.Author$

As explained in Section 2, arguments in our framework can be generated by using domain-cases, argument-cases, and argumentation schemes. In the *ArgCBRonto* ontology, arguments have a conclusion, a promoted value, and a support set. They also have a unique identifier *ID*:

$Argument \sqsubseteq Thing$ $SupportSet \sqsubseteq Thing$
$Argument \sqsubseteq \forall hasConclusion.Conclusion$ $Argument \sqsubseteq \forall promotesValue.Value$
$Argument \sqsubseteq \forall hasSupportSet.SupportSet$

In the *ArgCBROnto* ontology, the elements of the support set are represented with the following properties:

$SupportSet \sqsubseteq \forall hasPremise.Premise$
$SupportSet \sqsubseteq \forall hasDomainCase.DomainCase$ $SupportSet \sqsubseteq \forall hasArgumentCase.ArgumentCase$
$SupportSet \sqsubseteq \forall hasArgumentationScheme.ArgumentationScheme$
$SupportSet \sqsubseteq \forall hasPresumption.Premise$ $SupportSet \sqsubseteq \forall hasException.Premise$
$SupportSet \sqsubseteq \forall hasDistinguishingPremise.Premise$ $SupportSet \sqsubseteq \forall hasCounterExample.Case$

The argument-cases are the main structure that we use to implement our framework and to computationally represent arguments in agent societies. Also, their structure is generic and domain-independent. However, as pointed out above, the structure of domain-cases is completely domain-dependent, and we assume that we have a specific ontology to represent them in each application domain. Therefore, we focus the remainder of this section on the ontological description of argument-cases.

Argument-cases have two main objectives: 1) they can be used by agents as knowledge resources to generate new arguments and to select the best position to put forward in view of past argumentation experiences; and 2) they can be used to store new argumentation knowledge that agents gain in each dialogue, improving the agents' argumentation skills.

Section 4 shows the structure of an argument-case in our example application domain. The argument-cases have three main parts: the description of the *problem* that the case represents, the *solution* applied to this problem, and the *justification* of why this particular solution was applied. An argument-case stores the information about a previous argument that an agent posed in a certain step of a dialogue with other agents.

Argument-case Problem:

The problem description has a *domain context* that consists of the *premises* of the argument and that represents the context of the domain where the argument was put forward. Each premise has a unique identifier *ID*, a *name*, and a *content*, which can be of several types depending on the application domain.

$Context \sqsubseteq Thing$ $DomainContext \sqsubseteq Context$ $Problem \sqsubseteq \forall hasDomainContext.DomainContext$
$Premise \sqsubseteq Thing$
$\top \sqsubseteq \forall hasName.String$ $\top \sqsubseteq \forall hasName^{-}.Premise$
$\top \sqsubseteq \forall hasContent.Type$ $\top \sqsubseteq \forall hasContent^{-}.Premise$

In addition, if we want to store an argument and use it to generate a persuasive argument in the future, the features that characterise the audience of the previous argument (the *social context*) must also be kept. Therefore, we have two disjoint types of contexts in our ontology, which are the usual domain context and the social context.

$SocialContext \sqsubseteq Context$
$DomainContext \sqsubseteq \neg SocialContext$ $ArgumentProblem \sqsubseteq \forall hasSocialContext.SocialContext$

For the definition of the social context of arguments, we store the social information about each *social entity* related to the argument in the argument-case. This social entity can be an *agent* (either the proponent of the argument or the opponent to which the argument is addressed) or the *group* to which the agent belongs.

$SocialEntity \sqsubseteq Thing$
$Agent \sqsubseteq SocialEntity$ $Group \sqsubseteq SocialEntity$ $Agent \sqsubseteq \neg Group$

For the sake of simplicity, in this paper, we assume that in each step of the dialogue, one proponent agent generates an argument and sends it to one opponent agent that belongs to its same group. However, either the proponent or the opponent's features could represent information about agents that act as representatives of a group and any agent can belong to different groups at the same time. Therefore, the social context of argument-cases includes information about the *proponent* and the *opponent* of the argument (which can be an agent or a group) and information about their *group*. Also, groups are formed by at least two agents.

$SocialContext \sqsubseteq \forall hasProponent.(Agent \sqcup Group)$ $SocialContext \sqsubseteq \forall hasOpponent.(Agent \sqcup Group)$
$SocialContext \sqsubseteq \forall hasGroup.Group$
$Group \sqsubseteq \forall hasMember.Agent$ $Group \sqsubseteq \geq 2 hasMember$

Specifically, each social entity of the argument-case has a unique *ID* that identifies it in the system and the *role* that the agent or the group was playing when it sent or received the argument (e.g., trade unionist, business manager, etc.). These rose should not be confused with the role of proponent and opponent from the argumentation perspective.

$\top \sqsubseteq \forall hasRole.String$ $\top \sqsubseteq \forall hasRole^{-}.SocialEntity$

Moreover, if known, we also store the preferences of each agent or group over the pre-defined set of general *values* in the system (e.g., security, solidarity, savings, etc.). These preferences (*ValPref*) affect the persuasive power of the proponent's argument over the opponent's behaviour. In the case of the group, we use this feature to store the values that the group can impose on its members if the conditions of the domain require it.

$Value \sqsubseteq Thing$ $ValPref \sqsubseteq Thing$
$ValPref \sqsubseteq \forall hasPreferred.Value$ $SocialEntity \sqsubseteq \forall hasValPref.ValPref$

Finally, the *dependency relation* between the proponent's role and the opponent's role is also stored in the social context of the argument-cases. Therefore, in our *ArgCBRonto* ontology, we have three types of dependency relations:

$\top \sqsubseteq \forall hasDependencyRelation.(Power \sqcup Authorisation \sqcup Charity)$
$\top \sqsubseteq \forall hasDependencyRelation^{-}.SocialContext$

Solution:

The *conclusion* of the case (for both domain-cases and argument-cases), and the *value* promoted in this specific situation are stored in the solution part.

$Conclusion \sqsubseteq Thing$
$Solution \sqsubseteq \forall hasConclusion.Conclusion \quad Solution \sqsubseteq \forall promotesValue.Value$

For argument-cases we have a more specialised description for the solution part (*ArgumentSolution*). This includes the *argument type* that defines the method by which the conclusion of the argument was drawn is stored. By default, we do not assume that agents have a pre-defined set of rules to infer deductive arguments from premises, which is difficult to maintain in open MAS. In our framework, agents have the following ways of generating new arguments: *Presumptive arguments* can be generated by using the premises that describe the problem to solve and an argumentation scheme whose premises match them; and *Inductive arguments* can be generated by using similar argument-cases and/or domain-cases that are stored in their case-bases.

$ArgumentSolution \sqsubseteq Solution$
$\top \sqsubseteq \forall hasArgumentType.(Inductive \sqcup Presumptive) \quad \top \sqsubseteq \forall hasArgumentType^{-}.ArgumentSolution$

Moreover, the solution part of the argument-cases stores the information about the *acceptability status* of the argument at the end of the dialogue. This feature shows if the argument was deemed *acceptable*, *unacceptable* or *undecided* in view of the other arguments that were put forward during the dialogue.

$\top \sqsubseteq \forall hasAcceptabilityStatus.(Acceptable \sqcup Unacceptable \sqcup Undecided)$
$\top \sqsubseteq \forall hasAcceptabilityStatus^{-}.ArgumentSolution$

Regardless of the final acceptability status of the argument, the information about the possible *attacks* that the argument received is also stored in the solution part of the argument-case. These attacks could represent the justification for an argument to be deemed unacceptable or else reinforce the persuasive power of an argument that, despite being attacked, was finally accepted. Argument-cases can store different types of attacks, depending on the type of argument that they represent: for presumptive arguments, critical questions (*presumptions* or *exceptions*) associated with the scheme [32] are stored; and for inductive arguments, as proposed in [6], either *distinguishing premises*) or *counter-examples* are stored. Therefore, the *ArgCBROnto* ontology represents the different types of attacks that arguments can receive as follows:

$ArgumentSolution \sqsubseteq \forall hasPresumption.Premise \quad ArgumentSolution \sqsubseteq \forall hasException.Premise$
$ArgumentSolution \sqsubseteq \forall hasDistinguishingPremise.Premise$
$ArgumentSolution \sqsubseteq \forall hasCounterExample.Case$

Justification:

In the *ArgCBROnto* ontology, the justification part of a case stores a *description* that can explain why this particular solution was applied to solve the case and the final results achieved.

$\top \sqsubseteq \forall hasDescription.String \quad \top \sqsubseteq \forall hasDescription^{-}.Justification$

In the special case of argument-cases, the justification specialises in an *ArgumentJustification*, which stores the information about the knowledge resources that were used to generate the argument represented by the argument-case (e.g., the set of *argumentation schemes* in presumptive arguments, the set of *cases* in inductive arguments, and both sets in mixed arguments).

$ArgumentJustification \sqsubseteq Justification$
$ArgumentJustification \sqsubseteq \forall\ hasArgumentationScheme.ArgumentationScheme$
$ArgumentJustification \sqsubseteq \forall\ hasCase.Case$

In addition, the justification of each argument-case has an associated *dialogue-graph* that represents the dialogue where the argument was put forward. The same dialogue graph can be associated with several argument-cases, and an argument-case can be associated to several graphs. Each dialogue graph has a *root* and a set of nodes, which we call *argument nodes*. An argument node has an *argument-case*, a *parent* argument node, and a *child* argument node. Thus, the *ArgCBROnto* ontology represents the sequence of arguments that were put forward in a dialogue, storing the complete conversation as a directed graph that links argument-cases. This graph can be used later to improve the efficiency of argumentation dialogues, for instance, by selecting those arguments that in the past led to quicker agreement processes in similar situations. Alternatively, opponent moves in a dialogue (the arguments that the opponent is going to present) could be inferred by looking at a similar previous dialogue with the same opponent.

$DialogueGraph \sqsubseteq Thing\ \ ArgumentNode \sqsubseteq Thing\ \ ArgumentNode \sqsubseteq \forall hasArgumentC.ArgumentCase$
$ArgumentNode \sqsubseteq \forall hasParent.ArgumentNode\ \ ArgumentNode \sqsubseteq \forall hasChild.ArgumentNode$
$DialogueGraph \sqsubseteq \forall hasRoot.ArgumentNode\ \ DialogueGraph \sqsubseteq \forall hasNode.ArgumentNode$
$ArgumentJustification \sqsubseteq \forall hasDialogueGraph.DialogueGraph$

Following a CBR methodology, the proposed knowledge resources allow agents to automatically generate, select, and evaluate arguments. However, the specification of this case-based reasoning process is out of the scope of this paper and was presented in [16]. Here we have focused on defining the knowledge representation formalism that agents can use to represent arguments and the information related to argumentation in order to automatically manage this information efficiently. The argument-case structure presented is flexible enough to represent different types of arguments and their associated information. Also, the KI approach that is followed allows semantic reasoning with the concepts that represent the cases. Nevertheless, the value of some features in argument-cases and domain-cases might remain unspecified in some domains. For instance, in some open MAS, the preferences over values of other agents might not be known previously, although agents could try to infer the unknown features by using CBR adaptation techniques [21]. This and other open questions will be discussed in Section 6.

4 Example Scenario

To exemplify the use of our framework, let us develop the running example for the call centre domain that is used throughout this paper. Our scenario consists of a group of technicians that are represented by software agents in a MAS that must provide technical support to solve the problems reported by the customers of a call centre. We have different levels of technicians, each of which has different problem-solving knowledge and responsibilities:

basic *operator* agents that provide first-level support to any problem reported to the system; *expert* agents that have higher level of expertise to provide specialised support to solve complex problems in specific domains; and *manager* agents that carry out managerial duties among the support services that the centre provides to different customers.

Therefore, we consider an agent society S to be composed of call-centre technicians that have the following dependency relations with each other:

- Operator $<^S_{Char}$ Operator; Operator $<^S_{Auth}$ Expert; Operator $<^S_{Pow}$ Manager
- Expert $<^S_{Char}$ Expert; Expert $<^S_{Pow}$ Manager
- Manager $<^S_{Char}$ Manager

Each technician can have its own values (e.g., quality, speed, savings), its own preferences over them, and also belong to different groups intended to solve specific types of problems or be assigned to specific projects. These groups can also have their own social values.

To guarantee high-quality service, the company subscribes to Service Level Agreements (SLAs) with the customer, where the different characteristics of the services to be provided are specified (e.g., the maximum time to provide a response for a request). In case of breach of the agreements, the company is penalised economically. We also assume that each technician has a helpdesk application with an underlying MAS to manage the large number of customer support requests that the call centre receives. This helpdesk registers the request information, tracks the request over its resolution process, warns when the maximum time to solve the request is about to expire, and helps the technician to solve the request. To this end, each agent of the MAS includes a domain CBR module and an argumentation CBR module that perform the generation, selection, and evaluation of positions and arguments as proposed in [17]. Hence, complex problems can be solved by a group of agents that argue to reach an agreement over the best solution to apply by using their helpdesks, which are interconnected in the company intranet. Therefore, each agent represents a technician and has its own knowledge resources to generate a solution for the problem that it receives. Thus, the agreement process consists of both an individual phase by which an agent individually generates, selects, and proposes its position as the best solution to apply and a collaborative phase by which the agent engages in an argumentation dialogue with other agents and tries to persuade them to accept its position.

For purposes of clarity, in this example, all agents belong to the same group (which provides the customer support service *Hardware Support (HS)*). In this setting, suppose that two agents that play the role of operators, *Stella* and *Vicente* are arguing to decide the best solution for a problem $P_{printer}$ that reports a print failure in a *br1 myPrint* printer purchased in 2012, which is not connected to the network, jams the paper, and does not show any error message in the display. Also, the *Hardware Support (HS)* service is supervised by an *expert* agent and this expert by a *manager* agent that plays the role of the company's director.

The value preference order of the group promotes savings (SA) over quality (QU) and speed (SP) (i.e., it promotes saving money in each service over providing high-quality solutions or speed responses, SP<QU<SA) and commands a dependency relation of charity between two operators and a power relation between a manager and an operator. *Stella* prefers quick solutions over high-quality solutions and savings (SA<QU<SP), *Vicente* prefers to provide high-quality solutions over quick responses and savings (SA<SP<QU) and, by default, the *expert* and the *manager* have a value preference order for the group *HS* (SP<QU<SA). Also, all agents have their own knowledge resources (domain-cases case-base, argument-cases case-base, and argumentation schemes ontology).

The premises of the domain context would store data that characterises the type of problems that can be reported to the customer service that is contracted. These data are premises that are specified in the SLA of the service. For instance, the premises may represent the printer *brand*, the printer *model*, the *year* of purchase, whether it is a *network printer*, whether it produces a *paper jam*, whether it shows an *error message* in the display, the *type of paper* that the printer is using, the type of problem to be dealt with, etc.

In the first step of the argumentation process, both operators will search their case-bases of domain-cases (*DCstella* and *DCvicente*, respectively) to generate their positions. To query the case-bases, the problem is formatted as a target domain-case without a solution (i.e., a *ticket*), represented in the OWL specification of a *ArgCBRonto* as[5]:

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#Ticket">
    <rdf:type rdf:resource="ArgCBRonto.owl#DomainCase"/>
    <Premise rdf:resource="ArgCBRonto.owl#Brand_br1"/>
    <Premise rdf:resource="ArgCBRonto.owl#Model_myPrint"/>
    <year rdf:datatype="XMLSchema#integer">2012</year>
    <Premise rdf:resource="ArgCBRonto.owl#NetworkPrinter_no"/>
    <Premise rdf:resource="ArgCBRonto.owl#PaperJam_yes"/>
    <Premise rdf:resource="ArgCBRonto.owl#ErrorMessage_no"/>
    <ProblemType rdf:resource="ArgCBRonto.owl#PrinterErrors"/>
    <Solution rdf:resource="ArgCBRonto.owl#"/>
</owl:NamedIndividual>
```

Then, let us suppose that *Stella* has queried her domain-cases case-base by using an OWL semantic reasoner and has found a domain-case $DCstella_1$ that represents a problem of the same type that was solved by opening the lid 1, removing the jammed paper, and resetting the device:

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#DCstella1">
    <rdf:type rdf:resource="ArgCBRonto.owl#DomainCase"/>
    <Premise rdf:resource="ArgCBRonto.owl#Brand_br1"/>
    <Premise rdf:resource="ArgCBRonto.owl#Model_myPrint"/>
    <year rdf:datatype="XMLSchema#integer">2012</year>
    <Premise rdf:resource="ArgCBRonto.owl#NetworkPrinter_no"/>
    <Premise rdf:resource="ArgCBRonto.owl#PaperJam_yes"/>
    <Premise rdf:resource="ArgCBRonto.owl#ErrorMessage_no"/>
    <ProblemType rdf:resource="ArgCBRonto.owl#PrinterErrors"/>
    <Solution rdf:resource="ArgCBRonto.owl#S1"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#S1">
    <rdf:type rdf:resource="ArgCBRonto.owl#Solution"/>
    <solutionID rdf:datatype="XMLSchema#integer">1</solutionID>
    <Conclusion rdf:datatype="XMLSchema#string">Open the lid 1,
     remove the jammed paper and reset</conclusion>
    <Value rdf:datatype="XMLSchema#string">SP</Value>
</owl:NamedIndividual>
```

[5] In this example, the general *ArgCBRonto* ontology has been instantiated in our concrete call-centre domain application. For purposes of simplicity, not all posile attributes of OWL instances are shown.

Therefore, *Stella* can generate position pos_{st1} with this conclusion, which, for instance, promotes its most preferred value (speed)[6].

In addition, *Vicente* has also retrieved a similar domain-case ($DCvicente_1$), which shows how the same problem was solved by telling the customer that recycled paper should not be used with this printer model. In these cases, *Vicente* is assuming the truth of an extra premise (*Type of Paper*) that specifies that the printer is loaded with recycled paper. Therefore, *Vicente* can generate the position $pos_{vicente}$, promoting its most preferred value (quality) by re-using the solution of this domain-case:

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#DCvicente1">
    <rdf:type rdf:resource="ArgCBRonto.owl#DomainCase"/>
    <Premise rdf:resource="ArgCBRonto.owl#Brand_br1"/>
    <Premise rdf:resource="ArgCBRonto.owl#Model_myPrint"/>
    <year rdf:datatype="XMLSchema#integer">2012</year>
    <Premise rdf:resource="ArgCBRonto.owl#NetworkPrinter_no"/>
    <Premise rdf:resource="ArgCBRonto.owl#PaperJam_yes"/>
    <Premise rdf:resource="ArgCBRonto.owl#PaperType_recycled"/>
    <ProblemType rdf:resource="ArgCBRonto.owl#PrinterErrors"/>
    <Solution rdf:resource="ArgCBRonto.owl#S2"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#S2">
    <rdf:type rdf:resource="ArgCBRonto.owl#Solution"/>
    <solutionID rdf:datatype="XMLSchema#integer">2</solutionID>
    <Conclusion rdf:datatype="XMLSchema#string">Do not use recycled paper with
     this printer model</Conclusion>
    <Value rdf:datatype="XMLSchema#string">QU</Value>
</owl:NamedIndividual>
```

Furthermore, if in this specific domain context we have a logic property that specifies that the *PrinterErrors* problem type can be viewed as sub-class of the *HardwareErrors* problem type:

$PrinterErrors \sqsubseteq HardwareErrors$

an OWL semantic reasoner would infer that instances belonging to a certain class will also be inferred to belong to all its super-classes and thus, the following $DCvicente_2$ domain-case would be also retrieved to generate the position $pos_{vicente}$.

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#DCvicente2">
    <rdf:type rdf:resource="ArgCBRonto.owl#DomainCase"/>
    <Premise rdf:resource="ArgCBRonto.owl#Brand_br1"/>
    <Premise rdf:resource="ArgCBRonto.owl#Model_myPrint"/>
    <year rdf:datatype="XMLSchema#integer">2012</year>
    <Premise rdf:resource="ArgCBRonto.owl#NetworkPrinter_no"/>
    <Premise rdf:resource="ArgCBRonto.owl#PaperJam_yes"/>
    <Premise rdf:resource="ArgCBRonto.owl#ErrorMessage_no"/>
    <Premise rdf:resource="ArgCBRonto.owl#PaperType_recycled"/>
```

[6] In this example, we assume that the positions and arguments of an agent promote its most preferred value by default.

```
    <ProblemType rdf:resource="ArgCBRonto.owl#HardwareErrors"/>
    <Solution rdf:resource="ArgCBRonto.owl#S2"/>
</owl:NamedIndividual>
```

Once the agents have proposed their positions, the operator *Stella* acknowledges that *Vicente* has proposed a different solution and starts the argumentation dialogue to solve P_{print} by asking *Vicente* for a justification of its position. Therefore, *Stella* asks *Vicente* to provide an argument for supporting $pos_{vicente}$. Assuming that all operators in the call centre are willing to collaborate, *Vicente* can put forward the following support arguments by using different combinations of the knowledge resources that it has used to generate its position:

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SAvicente1">
    <rdf:type rdf:resource="ArgCBRonto.owl#Argument"/>
    <SupportSet rdf:resource="ArgCBRonto.owl#SupportSet1"/>
    <Conclusion rdf:datatype="XMLSchema#string">S2</Conclusion>
    <Value rdf:datatype="XMLSchema#string">QU</Value>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SupportSet1">
    <rdf:type rdf:resource="ArgCBRonto.owl#SupportSet"/>
    <DomainCase rdf:resource="ArgCBRonto.owl#DCvicente1"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SAvicente2">
    <rdf:type rdf:resource="ArgCBRonto.owl#Argument"/>
    <SupportSet rdf:resource="ArgCBRonto.owl#SupportSet2"/>
    <Conclusion rdf:datatype="XMLSchema#string">S2</Conclusion>
    <Value rdf:datatype="XMLSchema#string">QU</Value>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SupportSet2">
    <rdf:type rdf:resource="ArgCBRonto.owl#SupportSet"/>
    <DomainCase rdf:resource="ArgCBRonto.owl#DCvicente2"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SAvicente3">
    <rdf:type rdf:resource="ArgCBRonto.owl#Argument"/>
    <SupportSet rdf:resource="ArgCBRonto.owl#SupportSet3"/>
    <Conclusion rdf:datatype="XMLSchema#string">S2</Conclusion>
    <Value rdf:datatype="XMLSchema#string">QU</Value>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SupportSet3">
    <rdf:type rdf:resource="ArgCBRonto.owl#SupportSet"/>
    <DomainCase rdf:resource="ArgCBRonto.owl#DCvicente1"/>
    <DomainCase rdf:resource="ArgCBRonto.owl#DCvicente2"/>
</owl:NamedIndividual>
```

where the support set includes the *premises* of the $P_{printer}$ problem description (not shown in the listing for the purpose of simplicity) and the domain-case(s) used by *Vicente* to generate its position. From these arguments, *Vicente* must select the one that is going to be used to answer *Stella*'s challenge. To perform this selection, it can query its argument-cases case-base to search for information about previous argumentation dialogues that could help it to select the best support argument to put forward, in view of what happened in a similar situation in the past. Then, let us assume that *Vicente* finds an argument-case *ACvicente*1 in its case-base that represents a similar previous argumentation experience with *Stella*, where it presented $DCvicente_1$ to justify its position and *Stella* attacked the argument with a distinguishing premise attack on the premise *Error Message* since it appears in the characterisation of the original problem but not on the problem description of $DCvicente_1$:

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#ACvicente1">
    <rdf:type rdf:resource="ArgCBRonto.owl#ArgumentCase"/>
    <Premise rdf:resource="ArgCBRonto.owl#Brand_br1"/>
    <Premise rdf:resource="ArgCBRonto.owl#Model_myPrint"/>
    <year rdf:datatype="XMLSchema#integer">2012</year>
    <Premise rdf:resource="ArgCBRonto.owl#NetworkPrinter_no"/>
    <Premise rdf:resource="ArgCBRonto.owl#PaperJam_yes"/>
    <Premise rdf:resource="ArgCBRonto.owl#PaperType_recycled"/>
    <ProblemType rdf:resource="ArgCBRonto.owl#HardwareErrors"/>

    <Proponent rdf:resource="ArgCBRonto.owl#Vicente"/>
    <Opponent rdf:resource="ArgCBRonto.owl#Stella"/>
    <Group rdf:resource="ArgCBRonto.owl#HS"/>
    <DependencyRelation rdf:resource="ArgCBRonto.owl#Charity"/>
<ArgumentSolution rdf:resource="ArgCBRonto.owl#S2"/>

<AcceptabilityStatus rdf:resource="ArgCBRonto.owl#Uncacceptable"/>
<DistinguishingPremise rdf:resource="ArgCBRonto.owl#ErrorMessage_no"/>
</owl:NamedIndividual>
```

As a result of this attack, the support argument that *Vicente* presented was rebutted and deemed unacceptable. With this information, *Vicente* can infer that *Stella* could make the same type of attack if *Vicente* uses $DCvicente_1$ to justify its position and hence, *Vicente* would select $SAvicente_2$ as the best support argument to propose in view of its previous experience. In this case, *Stella* cannot attack *Vicente*'s support argument (and hence, its underlying position) by posing a counter-example for $DCvicente_2$ with $DCstella_1$, since $DCstella_1$ does not subsume $DCvicente_1$. Thus, assuming that *Stella* does not have more information in its knowledge resources to generate an attack argument, *Stella* just preliminarily accept *Vicente*'s position. Note that although this position contradicts *Stella*'s position, the fact that *Stella* accepts $pos_{vicente}$ does not necessarily mean that *Stella* has to withdraw its position, but pos_{stella} remains available until it is not rebutted by another agent. In this example, *Stella* started the argumentation by challenging *Vicente*'s position. Hence, the target of discussion is $pos_{vicente}$ and not pos_{stella}, Thus, the result of the dialogue started by *Stella* has nothing to do with *Stella*'s position.

In the next step of the argumentation process, *Vicente* could challenge *Stella*'s position by asking *Stella* for a justification. In its turn, *Stella* can generate the support argument $SAstella_1$ to justify *Stella*'s position with the domain-case $DCstella_1$:

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SAstella1">
    <rdf:type rdf:resource="ArgCBRonto.owl#Argument"/>
    <SupportSet rdf:resource="ArgCBRonto.owl#SupportSet4"/>
    <Conclusion rdf:datatype="XMLSchema#string">S1</Conclusion>
    <Value rdf:datatype="XMLSchema#string">SP</Value>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SupportSet4">
    <rdf:type rdf:resource="ArgCBRonto.owl#SupportSet"/>
    <DomainCase rdf:resource="ArgCBRonto.owl#DCstella1"/>
</owl:NamedIndividual>
```

Now, *Vicente* would realise that its domain-case $DCvicente_2$ subsumes $DCstella_1$ and proposes a different solution, so *Vicente* attacks $SAstella_1$ with this counter-example:

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#AAvicente1">
    <rdf:type rdf:resource="ArgCBRonto.owl#Argument"/>
    <SupportSet rdf:resource="ArgCBRonto.owl#SupportSet5"/>
    <Conclusion rdf:datatype="XMLSchema#string">notS1</Conclusion>
    <Value rdf:datatype="XMLSchema#string">QU</Value>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SupportSet5">
    <rdf:type rdf:resource="ArgCBRonto.owl#SupportSet"/>
    <CounterExample rdf:resource="ArgCBRonto.owl#DCvicente2"/>
</owl:NamedIndividual>
```

This attack argument promotes the quality of solutions (QU) over the resolution speed (SP), which contradicts the preference order over values of *Stella*. Therefore, provided that the *charity* dependency relation between both operators does not oblige *Stella* to accept arguments from *Vicente* by default, this attack does not succeed from the *Stella*'s point of view. If no more positions or arguments are provided by these and other agents of the call centre, both pos_{stella} and $pos_{vicente}$ could be considered to be selected as the best solution to solve $P_{printer}$.

In a normal situation, the *expert* agent in charge of the hardware support service would make the final decision. Thus, the *expert* would select $pos_{vicente}$ as the best solution to provide to the customer since it promotes the second most preferred value of the group that attends the support service (recalling, SP<QU<SA). However, let us assume that the SLA contracted by the customer in the hardware support service is about to be breached. In that case, if the *expert* has an argumentation scheme $AS1$ that changes the value preference order of a group of operators to QU<SA<SP in this exceptional situation, the operators of this group must provide quick solutions for the problems that they receive, even if their individual value preferences give priority to high-quality solutions.

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#AS1">
    <rdf:type rdf:resource="ArgCBRonto.owl#ArgumentationScheme"/>
    <Premise rdf:resource="ArgCBRonto.owl#
       exception-value-preference_quality<speed"/>
    <Premise rdf:resource="ArgCBRonto.owl#
       approaching-deadline-SLA-exception_yes"/>
    <Conclusion rdf:resource="ArgCBRonto.owl#
       value-preference_quality<speed"/>
    <Exception rdf:resource="ArgCBRonto.owl#
```

```
      Xproject-approaching-deadline-SLA-exception_no"/>
</owl:NamedIndividual>
```

Given the argumentation scheme $AS1$, the *expert* agent can also generate a support argument $SAexpert1$ for *Stella's* position, promoting the most preferred value of its group (SP):

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SAexpert1">
    <rdf:type rdf:resource="ArgCBRonto.owl#Argument"/>
    <SupportSet rdf:resource="ArgCBRonto.owl#SupportSet6"/>
    <Conclusion rdf:datatype="XMLSchema#string">S1</Conclusion>
    <Value rdf:datatype="XMLSchema#string">SP</Value>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SupportSet6">
    <rdf:type rdf:resource="ArgCBRonto.owl#SupportSet"/>
    <ArgumentationScheme rdf:resource="ArgCBRonto.owl#AS1"/>
</owl:NamedIndividual>
```

In this case, *Vicente*'s agent cannot attack this support argument, since the expert has an *authorisation* dependency relation over it and hence, the *expert* would select pos_{stella} as the final solution to propose to the customer. However, note that the argumetation scheme $AS1$ has a critical question of the type exception that launches an issue against its conclusion. This exception captures the fact that for certain projects, approaching the deadline of a SLA is not considered as an exceptional case, so the conclusion of $AS1$ does not apply. Now, let us assume that the *manager* of the company knows that the *Hardware Support (HS)* service is within these special case of projects where reaching to deadlines should not change the value preference order of the group that is working on it. Thus, the manager can attack the support argument $SAexpert1$ with an atack argument $AAmanager1$ that raises the exception to the argumentation scheme $AS1$ and promotes the original value of the group (SA):

```
<owl:NamedIndividual rdf:about="ArgCBRonto.owl#AAmanager1">
    <rdf:type rdf:resource="ArgCBRonto.owl#Argument"/>
    <SupportSet rdf:resource="ArgCBRonto.owl#SupportSet7"/>
    <Conclusion rdf:datatype="XMLSchema#string">notS1</Conclusion>
    <Value rdf:datatype="XMLSchema#string">SA</Value>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#SupportSet7">
    <rdf:type rdf:resource="ArgCBRonto.owl#SupportSet"/>
    <CriticalQuestion rdf:resource="ArgCBRonto.owl#Exception1"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="ArgCBRonto.owl#Exception1">
    <rdf:type rdf:resource="ArgCBRonto.owl#Exception"/>
    <Premise rdf:resource="ArgCBRonto.owl#
      HSproject-approaching-deadline-SLA-exception_no"/>
</owl:NamedIndividual>
```

Then, as the *power* dependency relation of the *manager* over the *expert* and the operators make this attack to succeed, the value preference order of the group *HS* would remain unchanged. Therefore, if no new positions or arguments are presented, the *expert* agent would select $pos_{vicente}$ as the best solution to provide to the customer, since it promotes the second most preferred value of the group that attends the support service (QU).

This example shows the way in which agents can automatically use the knowledge resources of the framework to generate, select, and evaluate positions and arguments. Also, it takes into account the social context of agents to perform these activities. By sharing the *ArgCBROnto* ontology, heterogeneous agents engaged in an argumentation dialogue like this could convey and understand the information of the positions and arguments generated. Our argumentation framework has also been applied to manage the water resources of a river basin [15].

5 Related Work

Modeling how agents can reach agreements is a hot topic in MAS, and several argumentation-based approaches have been developed [20][2][26][12][37]. A main difference between these works and our proposal is that we follow a case-based reasoning approach instead of deductive or abductive reasoning based on rules. 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 potentially solved by means of argumentation dialogues between them (since a dialogue may still end in disagreement). Most argumentation frameworks and systems produce arguments by applying a set of inference rules. For instance, rule-based systems require eliciting an explicit model of the domain. In open MAS, the domain is highly dynamic and the set of rules that model it is difficult to specify in advance, even if these rules are *domain-specific inference rules* that are intended to represent domain knowledge. However, tracking the arguments that agents put forward in argumentation processes can be relatively simple. Therefore, these arguments can be stored as cases that are codified in a specific case representation language that different agents are able to understand. This approach makes possible to develop case-bases reducing the knowledge-acquisition bottleneck. Reasoning with cases is especially suitable when there is a weak or even unknown domain theory, and acquiring examples encountered in practice is easy. With case-bases, agents are able to perform *lazy learning* processes on argumentation information. For complex and highly dynamic systems, this is easier than using a rule-based system.

Among rule-based argumentation frameworks, our approach is close to the one followed by the ASPIC framework [27] and the subsequent ASPIC+ framework, specifically its extension to legal case-based reasoning [29]. In our framework, arguments can be constructed by aggregating different support and attack elements, which are structures that support intermediate conclusions that lead to the conclusion of the argument itself. These elements can be viewed as the case-based version of the sub-arguments of the ASPIC framework. Inferences in our framework are based on previous cases, on argumentation schemes, and on premises (the ones that are part of the cases and argumentation schemes that match the problem to solve) instead of being based on strict or defeasible rules. These elements can be gathered and added to the agents' knowledge bases during the development and after the deployment of the system. Then, to make inferences, a case-based reasoning algorithm determines in execution time which of these knowledge resources can be reused in the current situation to generate an argument to support or attack a specific position [16]. However, ASPIC needs to create an explicit model of the domain before making any inference, with specific strict rules, defeasible rules and axioms in its knowledge-base to be able to generate each argument. For instance, in our running example, each premise of the domain-case

$DCstella_1$ could be viewed as a *factor* in ASPIC+ and we would have to determine in advance whether they support the solution provided by *Stella* or not. Then, we would need to have a set of rules or axioms in the knowledge base that specifically link each premise of $DCstella_1$ as support element for the conclusion $S1$. Furthermore, if we need to add a new rule to ASPIC+, the system has to be checked for conflicting rules and redundant rules. In our framework, an addition or deletion of a case from the case-base does not need any further checking or debugging, although we acknowledge that while it does not affect the system's operation, it may have an impact on the outcome of the system.

Also, an important difference between the definition of argument of our case-based approach and the ASPIC rule-based approach lies in the defeasible nature of all elements of our support set, except from certain premises, as explained below. By contrast, ASPIC arguments can also be constructed by chaining strict inference rules that give rise to indefeasible arguments. Intuitively, our arguments cannot be attacked on the support set premises that match the description of the problem to solve, but only on those extra premises that represent the context of the domain where the argument was put forward and that do not appear in the description of the problem. Therefore, the premises that describe the problem to solve can be considered as *axiom premises* as defined in the ASPIC framework [27, Definition 3.5]. Similarly, the extra premises can be considered as equivalent to the *ordinary premises* of the ASPIC framework, and whether or not the attacks on them result in defeats, depends on the defeat relation specified in the argumentation framework. For instance, in $DCvicente_1$, *Paper Type* would be considered as an ordinary premise and the rest of premises as axioms. Alternatively, our arguments can be attacked based on those premises that appear in the description of the problem to solve but have not been considered to draw the conclusion of the argument (they do not appear in the support set of the argument). Again, compared with the ASPIC framework, these premises can be considered as *assumptions*, and attacks on them always succeed. For instance, the premise *Error Message* would be an assumption for $DCvicente_1$. In addition to the attacks on premises by means of distinguishing premises, our framework allows attacks on cases and argumentation schemes, which can be performed by means of counter-examples and critical questions, respectively.

Another important difference between our proposal and the ASPIC framework is that we use the OWL-DL restricted view of the contrariness relation between concepts and instances, since it follows the way of reasoning that case-based reasoning systems use. As pointed out in Section 2, OWL-DL does not assume *closed-world reasoning* with *negation as failure*; it uses *open-world reasoning* with *negation as unsatisfiability*. Therefore, something is false only if it can be proved to contradict other information in the ontology. This implies that two *concepts* are *contradictory* if and only if they are specifically declared as such with the owl property *complementOf*. Similarly, two *individuals* are contradictory if and only if they are specifically declared as such with the owl property *differentFrom*. Therefore, in our running example *S1* and *S2* need to be specifically declared as such to be considered as contradictory. This differs from the contrariness relation declared in the ASPIC framework, which also captures negation as failure.

Also, as pointed out in Section 2, we consider two types of attacks over arguments, *rebuts* and *undercuts*. Alternatively, the ASPIC framework considers a third kind of attack, the *undermining attack*. In our framework, this attack is represented by the undercutting attack of type 2, raised by putting forward a distinguishing premise (in the running example, this was the type of attack represented in the argument-case $ACvicente_1$).

Finally, the dependency relation over roles $Dependency_{S_t}$ and the agent's preference relation over values $Valpref_{ag_i}$ defined in our framework establish an *argument ordering* that is used to determine which attacks result in defeats. Thus, the argument ordering of our cased-based framework for agent societies is based on pre-defined relations over roles and on agents' preferences over values instead of on strict and defeasible rules, as proposed in [27, Definition 3.10].

Another related work that first proposed the representation of argumentation information using ontologies was [36]. This work develops a format for argument interchange (AIF) that can be used between argumentation tools and/or MAS. The *ArgCBRonto* ontology presented in this paper follows the AIF approach and extends the AIF ontology by defining a specific language for case representation that facilitates case-based reasoning processes over domain and argumentation information. Therefore, an argumentation system based on our framework can interact with other systems that comply with the AIF standard. Since elements of cases are specified by using an ontological case representation language, the *ArgCBROnto* ontology, heterogeneous agents can *understand* the arguments interchanged in the system by simply sharing our ontology.

In our case-based argumentation framework for agent societies, we endorse the view of value-based argumentation frameworks [5], which stress the importance of the audience and the values promoted by an argument in determining whether or not it is persuasive. A related work on abstract argumentation scheme frameworks [3] combines argumentation frameworks with argumentation schemes and makes use of the structure provided by the schemes to guide dialogues and provide contextual elements of argument evaluation. However, these and most works on computational models of argument take a narrow view on the argument structure. In fact, they are abstract frameworks that are aimed at studying the properties of arguments, which enable evaluation with respect to the logical relations between arguments. In contrast to our framework, the actual structure of arguments and their knowledge representation formalism are obviated. In addition, previous argumentation experiences are not used to guide current argumentation processes such as that propose.

Other works use domain-dependent structures for the computational representation of arguments. Most approaches for case-based argumentation in MAS (which use cases as previous knowledge to manage arguments) have this domain dependency or centralise the case-based argumentation abilities in a mediator agent (e.g., [31],[24],[4]). One research work that is close to the approaches on case-based argumentation is the work on experience-based argumentation using association rules, presented as the *PADUA* protocol in [33]. This work pools the opinions of several agents that have access to different datasets to predict the classification of a new example. In subsequent research, the PADUA protocol has been extended to allow multi-agent dialogues by proposing the *PISA* protocol [34][35]. In their work, the authors tackle issues like the dynamic creation of a group, the selection of a group leader and intra-group consultation to suggest moves. As in our approach, in this research agents take advantage of previous experiences to solve a new problem, but the knowledge gained from the argumentation process is not stored or used to improve the agents' argumentation skills. In addition, PISA and PADUA have been designed to solve classification problems and have been not designed to solve any type of problem that can be characterised by a set of features, which is the target of our research. Nevertheless, the extension of our framework to allow agents to dynamically create groups and argue about the best move to make as a group in each step of the dialogue still remains open for future work.

In addition, other works are devoted to the study of argumentation in social networks, with a focus on the network topology (or the structure of the group) rather than on the actual social dependencies between software agents or human users. An example of this type is the work presented in [25], which investigates how argumentation processes among a group of agents may affect the outcome of group judgments in prediction markets. Also, an example on how argumentation can enhance dialogues in social networks can be found in

[13]. However, the influence of the agent group and the social dependencies between agents in the way agents can argue must be further investigated. For instance, an agent playing the role of employee could accept arguments from an agent playing the role of project manager that it would never accept in a different situation, such as playing the role of general manager. Similarly, an agent representing a group of employees (playing the role of trade unionist) is not expected to behave in the same way when arguing with an agent playing the role of the employees' representative than it does when arguing as an individual employee.

The research work presented in [18][19] shows a novel, argumentation-based negotiation framework that allows agents to detect, manage, and resolve conflicts that are related to their social influences in a distributed manner within a structured agent society. This work proposes a new argumentation scheme that captures how agents reason about influences within a structured society. It provides a mechanism to use this scheme to identify a suitable set of social arguments, a language and a protocol to exchange these arguments, and a decision-making functionality to generate these dialogues. However, it defines the social context of agents with a set of roles that agents can play, a set of generic relationships over them, and a set of weighted social commitments for each of the active relationships, with no mention of values or preferences over them. A major difference between that proposal and our argumentation framework is the main objective pursued. In their work, authors focus on solving conflicts regarding conflicting social commitments between agents, while our framework enables argumentation to solve a generic problem by using previous experiences, not only taking into account the social dependencies between agents but also taking into account their preferences over a set of values. In addition, the authors do not specify the types of dependency relations that agents can have, leaving this concept as a generic relation. In our framework, argumentation experience is stored and reused to support agents in making decisions about the best argument to put forward in a specific situation. Agents belong to a society that imposes dependency relations on them, so they are related via power, authorisation, or charity dependencies. Thus, the specific dependency relation between a pair of agents plays an important role in deciding whether or not an argument posed in a past argumentation dialogue can still be persuasive in a current situation (where, possibly, agents hold a different dependency relation). For the time being, we do not deal with conflicts on dependency relations between agents, but this is an interesting extension that opens a pathway for future work.

6 Discussion

For purposes of simplicity, in the example proposed in this paper we have assumed that a proponent agent addresses its arguments to an opponent of its same group, having complete knowledge of the social context. However, either the proponent's features or the opponent's features could represent information about agents that act as representatives of a group and any agent can belong to different groups at the same time. In that case, another issue that this research leaves open is the problem of solving conflicts between the values inherited from the group (or from the different groups of the agent) and the agent's individual values. The decision about which values are preferred would depend on the application domain. For instance, if the argumentation framework is implemented in a collaborative application domain where agents pursue reaching the best agreement for the whole group, the group values would be given priority over individual values.

Also for reasons of simplicity, the example does not show how agents can use the dialogue graphs that are associated to argument-cases in order to take strategic decisions about which arguments are more suitable in a specific situation or about whether continuing with a current argumentation dialogue is worthwhile. For instance, to improve efficiency in a negotiation, an argumentation dialogue could be terminated if it is similar to a previous one that did not reach an agreement. Alternatively, opponent moves in a dialogue could be inferred by looking at a similar previous dialogue with the same opponent.

In addition, in real systems, some features of argument-cases may be unknown. For instance, the proponent of an argument obviously knows its value preferences, and probably knows the preferences of its group, but, in a real open MAS, it is unlikely to know the opponent's value preferences. However, the proponent might know the value preferences of the opponent's group or have some previous knowledge about the value preferences of similar agents that play the same role as the opponent. If agents belong to different groups, the group features might be unknown, but the proponent could use its experience with other agents of the opponent's group and infer them. In any case, by using a DL-reasoner, we could make inferences over the the ontological representation of the data stored in the case-base and cope with this lack of knowledge, although the reliability of the conclusions drawn from previous experiences would be worse.

The actual algorithms for implementing the agents' reasoning process and the interaction protocol for interchanging arguments between agents have been published in [17] and [16]. The algorithms depend on the application domain and the design of the real system that implements our argumentation framework. The interaction protocol defines the dialogue, commitment and termination rules, and the locutions that agents use to interchange arguments. These locutions depend on the agents' communication language and determine the intention of the argument (e.g., pose an attack or ask for justifications), the argument's sender and receiver, the format of the argument's content (e.g., if complete knowledge resources or parts are sent), etc. Also, the process for posing critical questions depends on the actual ontology of argumentation schemes that agents implement and the interaction protocol that agents follow.

Finally, we acknowledge that due to the dynamism of the argumentation domain applied to open MAS, cases can quickly become obsolete. In this research, we have followed the basic approach for updating cases when a new case that is similar enough to an existing case in the case-base must be added. However, this can give rise to databases that are too large with obsolete cases that can hinder the performance of the entire system. Therefore, there is an important opportunity here to investigate new methods for the maintenance of case-bases in order to improve the adaptability of the framework.

7 Conclusions

The main contribution of this work consists on the proposal of a knowledge representation formalism for a case-based argumentation framework that allows agents to argue in agent societies, taking into account their roles, preferences over values and dependency relations. The main advantages that our framework contributes over other existent frameworks are: 1) the ability to represent social information in arguments; 2) the possibility of automatically managing arguments in agent societies; 3) the improvement of the agents' argumentation skills; and 4) the easy interoperability with other frameworks that follow the argument and data interchange web standards.

After that, we present the case-based argumentation framework that agents can use to manage positions and arguments in a way that they are computable. The framework proposed is formalised by defining the notion of argument, the logical language used to represent arguments, the concept of conflict between arguments and the notion of defeat between arguments. Moreover, the knowledge resources of the framework allow agents to use previous argumentation experiences to enhance their argumentation skills. These knowledge resources and the arguments that agents can interchange are ontologically defined in OWL-DL by using an ontology called ArgCBROnto, which allow heterogeneous agents in an open MAS to understand these concepts by sharing the ontology. An example scenario that shows how our framework allows agents to automatically engage in argumentation dialogues in a society taking into account the requirements identified has been also proposed. Finally, related work has been identified and compared with our proposal.

Acknowledgements This work is supported by the Spanish government grants [CONSOLIDER-INGENIO 2010 CSD2007-00022, TIN2011-27652-C03-01, and TIN2012-36586-C03-01] and by the GVA project [PROMETEO II/2013/019].

References

1. Amgoud, L.: An argumentation-based model for reasoning about coalition structures. In: 2nd International Workshop on Argumentation in Multi-Agent Systems, Argmas-05, pp. 1–12. Springer (2005)
2. Amgoud, L., Dimopolous, Y., Moraitis, P.: A unified and general framework for argumentation-based negotiation. In: 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS-07. IFAAMAS (2007)
3. Atkinson, K., Bench-Capon, T.: Abstract argumentation scheme frameworks. In: Proceedings of the 13th International Conference on Artificial Intelligence: Methodology, Systems and Applications, AIMSA-08, *Lecture Notes in Artificial Intelligence*, vol. 5253, pp. 220–234. Springer (2008)
4. Aulinas, M., Tolchinsky, P., Turon, C., Poch, M., Cortés, U.: Argumentation-based framework for industrial wastewater discharges management. Engineering Applications of Artificial Intelligence **25**(2), 317–325 (2012)
5. Bench-Capon, T., Atkinson, K.: Argumentation in Artificial Intelligence, chap. Abstract argumentation and values, pp. 45–64. Springer (2009)
6. Bench-Capon, T., Sartor, G.: A Model of Legal Reasoning with Cases Incorporating Theories and Values. Artificial Intelligence **150**(1-2), 97–143 (2003)
7. Bulling, N., Dix, J., Chesñevar, C.I.: Modelling coalitions: ATL + argumentation. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS-08, vol. 2, pp. 681–688. ACM Press (2008)
8. Chesñevar, C., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., Simari, G., South, M., Vreeswijk, G., Willmott, S.: Towards an Argument Interchange Format. The Knowledge Engineering Review **21**(4), 293–316 (2006)
9. Diaz-Agudo, B., Gonzalez-Calero, P.A.: Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems, *Integrated Series in Information Systems*, vol. 14, chap. An Ontological Approach to Develop Knowledge Intensive CBR Systems, pp. 173–214. Springer (2007)
10. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and N -person games. Artificial Intelligence **77**, 321–357 (1995)
11. Ferber, J., Gutknecht, O., Michel, F.: From Agents to Organizations: an Organizational View of Multi-Agent Systems. In: Agent-Oriented Software Engineering VI, *LNCS*, vol. 2935, pp. 214–230. Springer-Verlag (2004)
12. Hadidi, N., Dimopolous, Y., Moraitis, P.: Argumentative Alternating Offers. In: 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS-10, pp. 441–448. IFAAMAS (2010)
13. Heras, S., Atkinson, K., Botti, V., Grasso, F., Julián, V., McBurney, P.: How argumentation can enhance dialogues in social networks. In: Proceedings of the 3rd International Conference on Computational Models of Argument, COMMA-10, *Frontiers in Artificial Intelligence and Applications*, vol. 216, pp. 267–274. IOS Press (2010)
14. Heras, S., Botti, V., Julián, V.: On a computational argumentation framework for agent societies. In: Argumentation in Multi-Agent Systems, pp. 123–140. Springer (2011)
15. Heras, S., Botti, V., Julián, V.: Argument-based Agreements in Agent Societies. Neurocomputing **75**(1), 156–162 (2012)
16. Heras, S., Jordán, J., Botti, V., Julián, V.: Argue to Agree: a Case-Based Argumentation Approach. International Journal of Approximate Reasoning **54**(1), 82–108 (2013)

17. Jordán, J., Heras, S., Julián, V.: A Customer Support Application Using Argumentation in Multi-Agent Systems. In: 14th International Conference on Information Fusion (FUSION-11), pp. 772–778 (2011)
18. Karunatillake, N.C.: Argumentation-Based Negotiation in a Social Context. Ph.D. thesis, School of Electronics and Computer Science, University of Southampton, UK (2006)
19. Karunatillake, N.C., Jennings, N.R., Rahwan, I., McBurney, P.: Dialogue Games that Agents Play within a Society. Artificial Intelligence **173**(9-10), 935–981 (2009)
20. Kraus, S., Sycara, K., Evenchik, A.: Reaching Agreements Through Argumentation: A Logical Model and Implementation. Artificial Intelligence **104**, 1–69 (1998)
21. López de Mántaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M., Forbus, K., Keane, M., Watson, I.: Retrieval, Reuse, Revision, and Retention in CBR. The Knowledge Engineering Review **20**(3), 215–240 (2006)
22. Luck, M., McBurney, P.: Computing as interaction: agent and agreement technologies. In: IEEE International Conference on Distributed Human-Machine Systems. IEEE Press (2008)
23. Oliva, E., McBurney, P., Omicini, A.: Co-Argumentation Artifact for Agent Societies. In: 5th International Workshop on Argumentation in Multi-Agent Systems, Argmas-08, pp. 31–46. Springer (2008)
24. Ontañón, S., Plaza, E.: Learning and Joint Deliberation through Argumentation in Multi-Agent Systems. In: 7th International Conference on Agents and Multi-Agent Systems, AAMAS-07. ACM Press (2007)
25. Ontañón, S., Plaza, E.: Argumentation-Based Information Exchange in Prediction Markets. In: Argumentation in Multi-Agent Systems, *LNAI*, vol. 5384, pp. 181–196. Springer (2009)
26. Parsons, S., Sierra, C., Jennings, N.R.: Agents that reason and negotiate by arguing. Journal of Logic and Computation **8**(3), 261–292 (1998)
27. Prakken, H.: An abstract framework for argumentation with structured arguments. Argument and Computation **1**, 93–124 (2010)
28. Prakken, H., Reed, C., Walton, D.: Dialogues about the burden of proof. In: Proceedings of the 10th International Conference on Artificial Intelligence and Law, ICAIL-05, pp. 115–124. ACM Press (2005)
29. Prakken, H., Wyner, A., Bench-Capon, T., Atkinson, K.: A formalization of argumentation schemes for legal case-based reasoning in ASPIC+. Journal of Logic and Computation (In Press)
30. Sierra, C., Botti, V., Ossowski, S.: Agreement Computing. KI - Künstliche Intelligenz **DOI: 10.1007/s13218-010-0070-y** (2011)
31. Soh, L.K., Tsatsoulis, C.: A Real-Time Negotiation Model and a Multi-Agent Sensor Network Implementation. Autonomous Agents and Multi-Agent Systems **11**(3), 215–271 (2005)
32. Walton, D., Reed, C., Macagno, F.: Argumentation Schemes. Cambridge University Press (2008)
33. Wardeh, M., Bench-Capon, T., Coenen, F.P.: PISA - Pooling Information from Several Agents: Multiplayer Argumentation From Experience. In: Proceedings of the 28th SGAI International Conference on Artificial Intelligence, AI-2008, pp. 133–146. Springer (2008)
34. Wardeh, M., Bench-Capon, T., Coenen, F.P.: PADUA: a protocol for argumentation dialogue using association rules. AI and Law **17**(3), 183–215 (2009)
35. Wardeh, M., Coenen, F., Bench-Capon, T.: Arguing in Groups. In: 3rd International Conference on Computational Models of Argument, COMMA-10, pp. 475–486. IOS Press (2010)
36. Willmott, S., Vreeswijk, G., Chesñevar, C., South, M., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., Simari, G.: Towards an argument interchange format for Multi-Agent Systems. In: 3rd International Workshop on Argumentation in Multi-Agent Systems, ArgMAS-06, pp. 17–34. Springer (2006)
37. Wyner, A., Schneider, J.: Arguing from a Point of View. In: Proceedings of the First International Conference on Agreement Technologies (2012)