Navarro Llácer, M.; Heras Barberá, SM.; Botti Navarro, VJ.; Julian Inglada, VJ. (2013). Towards real-time agreements. Expert Systems with Applications. 40(10):3906-3917. doi:10.1016/j.eswa.2012.12.087.

Towards Real-Time Agreements

Martí Navarro and Stella Heras and Vicente Botti and Vicente Julián

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n 46022 Valencia (Spain)

Abstract

In this paper, we deal with the problem of real-time coordination with the more general approach of reaching *real-time agreements* in MAS. Concretely, this work proposes a *real-time argumentation framework* in an attempt to provide agents with the ability of engaging in argumentative dialogues and come with a solution for their underlying agreement process within a bounded period of time. The framework has been implemented and evaluated in the domain of a customer support application. Concretely, we consider a society of agents that act on behalf of a group of technicians that must solve problems in a Technology Management Centre (TMC) within a bounded time. This centre controls every process implicated in the provision of technological and customer support services to private or public organisations by means of a call centre. The contract signed between the TCM and the customer establishes penalties if the specified time is exceeded.

Keywords: Agreement Technologies, Real-Time, Multi-Agent Systems

1. Motivation

Many Multi-Agent Systems (MAS) operate in resource and time-constrained environments. In such domains, scarce resources must be shared between different agents taking into account that these agents must perform their tasks before a deadline is met. In addition, agents in open MAS are autonomous entities that can have their own knowledge resources, can play different roles and can have different objectives and preferences over values that they want to promote with their actions (e.g. an interested agent could want

Email address: mnavarro@dsic.upv.es, sheras@dsic.upv.es (Martí Navarro and Stella Heras and Vicente Botti and Vicente Julián)

to promote its own *wealth* in spite of the *fairness* in a negotiation to allocate a scarce resource). Also, they can be linked by different types of dependency relations in what we call an *agent society* (Heras et al., 2012). The high dynamism of the application domains of open MAS requires agents to have a way of harmonising the conflicts that come out when they have to collaborate or coordinate their activities. On top of the simpler ability to interact, agents need a mechanism to argue (persuade other agents to accept their points of view, negotiating the terms of a contract, etc.) and reach agreements (Sierra et al., 2011) to collaborate and coordinate their activities. Furthermore, if these agents operate in real-time environments, the appropriate agreement mechanism that they should use will have the added difficulty of allowing agents to reach agreements within a specified time. Therefore, in these environments there is a need for coordinating the agents that make use of the available resources and reach agreements in a *real-time* fashion.

Methods for the development of real-time MAS have been proposed in the literature (V. Julián and V. Botti, 2004). Also, real-time coordination in MAS has been studied from different perspectives, as robotics (O. Kahtib, 1986), traffic management (Choy et al., 2003) or route planning (Navarro et al., 2012). When the final objective of the coordination process is to reach an agreement, to coordinate agents by engaging in argumentation dialogues with their opponents in a discussion is a common approach (Kraus et al., 1998)(Parsons et al., 1998)(Amgoud and Prade, 2004). However, as the social context of agents determines the way in which agents can argue and reach agreements, this context should have a decisive influence in the computational representation of arguments, in the argument management process and in the way agents develop strategies to argue with other agents. To deal with this challenge, we have recently proposed a case-based argumentation framework that allows agents to argue in agent societies, taking into account their roles, preferences over values and dependency relations. This work also proposes a reasoning process by which agents can automatically generate, select and evaluate arguments in an agent society and learn from argumentation experiences to be able to develop dialogue strategies that help them to reach their objectives more efficiently (Heras et al., 2013b). Nevertheless, our original reasoning process didn't take real-time considerations into account to bound the time that agents can spend to reach agreements

and hence, cannot be applied in real-time scenarios.

In this paper, we deal with the problem of real-time coordination with the more general approach of reaching *real-time agreements* in MAS. Therefore, our notion of a real-time agreement may refer to the fair allocation of a scarce resource, but also to the outcome of a negotiation process to make a deal in a sale operation, to the final label assigned in a classification problem, to the final state of the beliefs database of an agent in a beliefs revision problem, and many more. However, any of these processes must be temporal bounded. This is an important aspect to consider in any real-time argumentative process, where a community of agents are able to reach agreements through argumentation before a deadline is met. Any agreement reached after this deadline will be considered of a low quality, or even not valid if we are in a hard real-time environment. Therefore, it is necessary to control every step of the argumentative process in order to predict its duration in time (i.e. we must know the temporal cost of executing each step of the process). Taking into account this prediction, we can guarantee that the agents engaged in the argumentation dialogue will reach an agreement about the outcome of the agreement process (e.g. they will decide the final allocation for a resource, the label of an individual in a classification problem, etc.) before the specified deadline is met. The response provided by the agents does not need to be optimal, since in some cases the best response may require more processing time, but at least it will be provided on time.

This work proposes a *real-time argumentation framework* in an attempt to provide agents with the ability of engaging in argumentative dialogues and come with a solution for their underlying agreement process within a bounded period of time. The structure of this paper is as follows: section 2 presents the methodology used in this work; section 3 shows the reasoning process that agents can follow to engage in in real-time argumentation processes; section 4 shows an application example of our proposal; related works are shown in section 5; finally, section 6 shows the conclusions of this work.

2. Background

In open multi-agent argumentation systems the arguments that an agent generates to support its position can conflict with arguments of other agents and these conflicts are solved by means of argumentation dialogues between them. In (Heras et al., 2013a) we

presented a computational case-based argumentation framework that agents can use to reason about argumentation processes. Also, in (Navarro et al., 2011) new case-based reasoning (CBR) techniques, called Temporal Bounded CBR (TB-CBR), to adapt the CBR methodology for its use in real-time MAS were proposed. The combination of these works forms the background of the real-time argumentation framework proposed in this paper. Thus, this section briefly reviews these proposals.

2.1. Case-based Argumentation Framework

As proposed in (Heras et al., 2013a), this section illustrates the main components of our original case-based argumentation framework by following a running example based on the evaluation scenario of section 4. This framework will be adapted in Section 3 to be used in real-time scenarios. Thus, let us suppose that we have a MAS that manages a call centre that provides customer support, where a set of agents representing technicians must reach agreements about the best solution to apply to different types of software or hardware problems reported to the centre. In addition, the company that runs the call centre has signed a *Service Level Agreement (SLA)* with the customer that has contracted the support service. Among other terms, this SLA specifies a maximum acceptable time within the customer must receive a solution for the problem that has reported to the call centre. Therefore, to solve each problem, a group of technicians engage in an argumentation dialogue proposing their individual solutions and justifying the reasons that they have to propose them as the best solution for the problem within a specific time. From our point of view, a problem can be characterised by a set of features that describe it. In our framework, agents can use two types of *knowledge resources* to generate, select and evaluate arguments:

A case-base with domain-cases: that represent previous problems and their solutions. Agents can use this knowledge resource to generate their positions and arguments. The *position* of an agent represents the *solution* that this agent proposes. Also, agents increase their domain knowledge at the end of each real-time argumentation dialogue by adding new cases to their domain-cases case-base.

A case-base with argument-cases: that store previous argumentation experiences

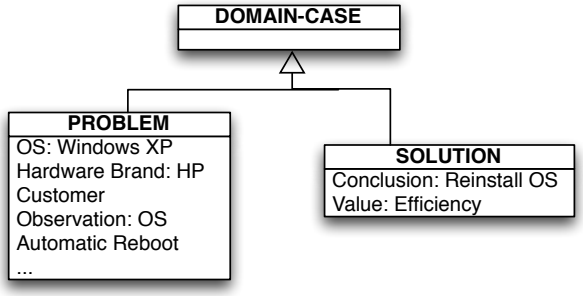

Figure 1: Structure of a Domain-Case in the Call Centre Scenario

and their final outcome. Agents use this resource to select the best position and argument to put forward in a specific situation in view of how suitable a similar position or argument was in a similar real-time argumentation dialogue. Also, agents store the new argumentation knowledge gained in each real-time agreement process, improving the agents' argumentation skills.

Figure 1 shows the structure of a domain-case in a call centre domain. In this example a technician of the call centre, say *operator 1 (O1)*, has to solve the problem of a HP computer with Windows XP that reboots at random. Here, *O1* has found the domain-case *DC1* that matches de description of the problem to solve (also including the extra feature *"Model"*). With this case, *O1* can build its position *"Reinstall OS"* and promote the value *"efficiency"*.

The argument-case of Figure 2 represents an argument that *O1* generated to justify its position when arguing with another operator *O2*. Argument-cases have three possible types of components: the *problem* description that characterises the state of the world when the argument was stored, with a *domain context* that consists of a set of *premises* and a *social context* that includes information about the *proponent* and the *opponent* of the argument and their *group*. Moreover, we also store the preferences (*ValPref*) of each agent or group over the set of *values* pre-defined in the system and the *dependency relation* between the proponent and the opponent. We consider two types of dependency relations (Dignum and Weigand, 1995): *Power*, when an agent has to accept a request from other agent because of some pre-defined domination relationship between them; and *Charity*, when an agent is willing to answer a request from other agent without being obliged to

do so. In the argument-case of Figure 2 proponent and opponent play the operator role and have a charity dependency relation between them. Also, both operators belong to the same group, which provides *software support* and does not impose any specific value preference order over its members. In the *solution* part, the *conclusion* of the case, the *value* promoted, and the *acceptability status* of the argument at the end of the dialogue are stored. This status shows if the argument was deemed *acceptable*, *unacceptable* or *undecided*. Thus, the conclusion of the argument-case in Figure 2 shows the solution proposed by *O1* when it was presented with the automatic reboot problem. The argument-case promotes the same value than the solution generated from *DC1* (*efficiency*). In addition, the conclusion part includes information about the *attacks* that the argument received during the real-time argumentation process. This information is used in our framework to emphasize the persuasive power of an argument that was finally accepted when it received attacks and its proponent was able to rebut them. Finally, the *justification* part of an argument-case stores the information about the knowledge resources that were used to generate the argument represented by the argument-case. Thus, the justification part of the argument-case of Figure 2 includes the domain-case *DC1*. In addition, the justification of each argument-case has a *dialogue-graph* (or several) associated, which represents the dialogue where the argument was put forward (*DG1* in Figure 2). In this way, the sequence of arguments that were put forward in a dialogue is represented, storing the complete conversation as a directed graph that links argument-cases. This graph is used in our framework to improve the efficiency of an argumentation dialogue, for instance, ending early a current dialogue that is very similar to a previous one that ended up in disagreement.

In our proposal, arguments that agents interchange are tuples of the form: $Arg = \{\phi, v, < S >\}$, where ϕ is the conclusion of the argument, v is the value that the agent wants to promote and $< S >$ is a set of elements that justify the argument (the support set). This set consists of different elements, depending on the argument purpose. On the one hand, if the argument provides a justification for a proposed solution, the support set includes the set of *premises* that represent the context of the domain where the argument has been put forward (those premises that match the problem to solve and

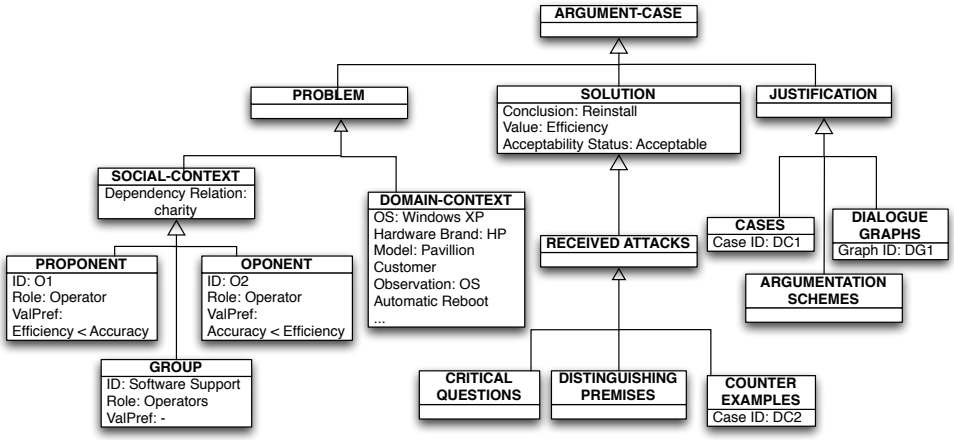

Figure 2: Structure of an Argument-Case in the Call Centre Scenario

other extra premises that were assumed) and optionally, any knowledge resource used by the proponent to generate the argument (*domain-cases*) and select it (*argument-cases*). This type of argument is called a *support argument*. On the other hand, if the argument attacks the argument of an opponent, it is called an *attack argument* and its support set can include any of the allowed attacks of our framework. These are *distinguishing premises* or *counter-examples*, as proposed in (Bench-Capon and Sartor, 2003).

A distinguishing premise is a premise that does not appear in the description of the problem to solve and has different values for two cases, or a premise that appears in the problem description and does not appear in one of the cases. A counter-example for a case is another case which problem description matches the current problem to solve and also subsumes the problem description of former case, but proposing a different solution. Therefore, the premise *"Model"* would be a distinguishing premise between the domain-case of Figure 1 and another domain-case, say *DC2*, that has exactly the same problem description than *DC1*, but that stores a different model for the HP computer. Also, assuming that *DC2* proposes an alternative solution (e.g *"Check for Memory Errors"*) that promotes the value *accuracy*, it could be used to generate a counter-example attack to *DC1* and vice-versa. In our running example, if *O1* generates the support argument $SA1 = \{$"$Reinstall$", $Efficiency$, $< \{Windows\ XP, HP, Pavillion, OW\ Automatic\ Reboot\}$, $\{DC1\}$, -, -, -, -, - $>\}$ to justify its position in view of another different position generated by *O2*, the latter could attack this argument with an attack argument $AA2 = \{$*"Check for*

Memory Errors", *Accuracy*, $< \{Windows\ XP, HP, Pavillion, OW\ Automatic\ Reboot\}$, -, -, -, -, -, $\{DC2\} >\}$, which presents the counter-example *DC2*. However, as shown in Figure 2, *O1*'s support argument remained acceptable at the end of the dialogue (when the argument-case was retained), which means that *O1* was able to rebut the attack or that *O2* withdrawn it.

2.2. Temporal Bounded Case-based Reasoning

Case-based Reasoning systems (CBR) provide agent-based systems with the ability to reason and learn from the experience of agents. To do it, these systems reuse or adapt past solutions to solve current similar problems. However, in real-time environments agents have a bounded time to reason and generate answers to the requests that they receive. Thus, CBR techniques must take into account temporal restrictions to observe real-time constraints. CBR systems are highly dependent on their application domain, and therefore, designing a general CBR model that might be suitable for any type of real-time domain is unattainable. In (Navarro et al., 2011) a new CBR methodology, called Temporal Bounded CBR (TB-CBR), is presented to provide some guidelines to adapt CBR for its use in real-time systems. Mainly, two factors have been controlled to temporal bound the reasoning process of the system that implements this approach: the case-base data structure and the temporal execution of the reasoning phases of the TB-CBR cycle (retrieve, reuse, revise and retain, as is common for CBR systems).

The design decision about the data structure of the case-base and the different algorithms that implement each CBR phase are important factors for determining the execution time of the CBR cycle. The number of cases in the case-base is another parameter that affects the temporal cost of the retrieval and retain phases. Thus, a maximum number of cases in the case-base must be pre-defined by the designer. Note that, usually, the temporal cost of the algorithms that implement these phases depends on this number. In any case, the retrieval and retention time can be reduced by using an indexing algorithm. These algorithms organize the case-base by selecting a specific feature (or set of features) from the cases, grouping together those cases that share the same values for these features. This reduces the cost of the search for similar cases (for retrieval or previous to the introduction of new cases in the case-base) to a specific set of cases with the same index

as the current case (Patterson et al., 2005) (Quinlin, 1993) (Wess et al., 1993).

Moreover, the time dedicated to execute the different phases must be controlled. To do this, the worst case execution time (WCET) of each function executed in each phase must be known. The sum of all function times represents the cost of running a complete cycle of TB-CBR. By using the WCET of each function we are ensuring that the time to execute the TB-CBR cycle does not exceed this calculated time.

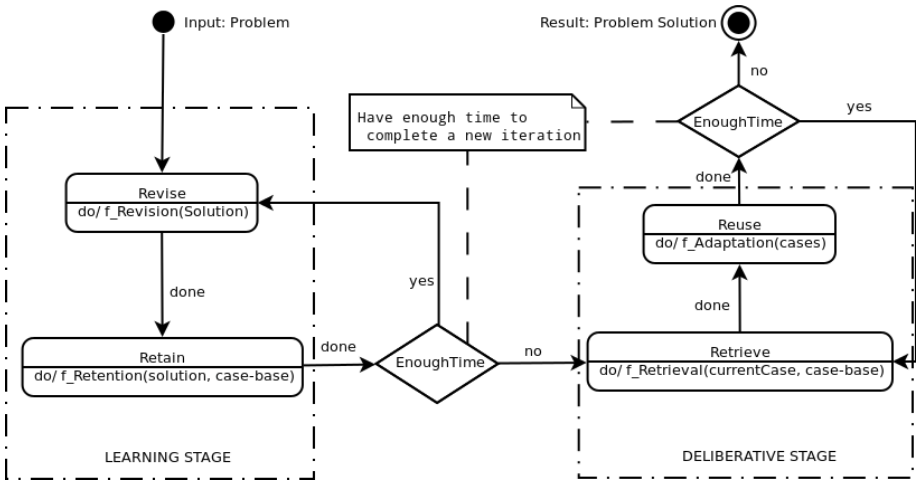

Figure 3: Temporal Bounded CBR cycle.

In (Navarro et al., 2011), we propose a modification of the classic CBR cycle in order to adapt it to be applied in real-time domains. Figure 3 shows a graphical representation of this approach. Firstly, we group the four reasoning phases that implement the *cognitive task* of the real-time agent into two stages defined as: the *learning stage*, which consists of the revise and retain phases and the *deliberative stage*, which includes the retrieve and reuse phases. Both phases will have their own execution time scheduled. Therefore, the designer can choose to either assign more time to the deliberative stage or keep more time for the learning stage (and thus, design agents that are more sensitive to updates). The $timeManager(t_{max})$ function is in charge of completing this task. Using this function the designer must specify how the real-time agent acts in the environment. Regardless of the decision taken by the designer, the $timeManager$ function should allow sufficient time for the deliberative stage to ensure a minimal answer. These new CBR stages must be

designed as an anytime algorithm (Dean and Boddy, 1988), where the process is iterative and each iteration is time-bounded and may improve the final response. The anytime behavior of the TB-CBR is achieved through the use of two loop control sequences. The loop condition is built using the *enoughTime* function, which determines if a new iteration is possible according to the total time that the TB-CBR has to complete each stage.

In accordance with this, the operation of our time bounded CBR cycle is shown in Algorithm 1. Firstly, the main difference that can be observed between the classic CBR cycle and the TB-CBR cycle is the starting phase. Our real-time application domain and the restricted size of the case-base (as explained in the following sections) gives rise to the need to keep the case-base as up to date as possible. Commonly, recent changes in the case-base will affect the potential solution that the CBR cycle is able to provide for a current problem. Therefore, the TB-CBR cycle starts at the learning stage, checking if there are previous cases waiting to be revised and possibly stored in the case-base. In our model, the solutions provided at the end of the deliberative stage will be stored in a solution list (*solutionQueue*) while a feedback about their utility is received. When each new CBR cycle begins, this list is accessed and while there is enough time, the learning stage of those cases whose solution feedback has been recently received is executed. If the list is empty, this process is omitted.

After this, the deliberative stage is executed. The deliberative stage is only launched if the real-time agent has a problem to solve. These problems are stored in a list (*problemQueue*) as they arise. Thus, the retrieval algorithm is used to search the case-base and retrieve a case that is similar to the current case (i.e. the one that characterizes the problem to be solved). The solutions are stored just after the end of the deliberative stage. Each time a similar case is found, it is sent to the reuse phase where it is transformed into a suitable solution for the current problem by using a *reuse* algorithm. Therefore, at the end of each iteration of the deliberative stage, the TB-CBR method is able to provide a solution for the problem at hand, although this solution can be improved in following iterations if the deliberative stage has enough time to perform them.

According to the temporal analysis of each phase of the CBR cycle, the anytime behavior of the TB-CBR is achieved through the use of two loop control sequences. The

loop condition is built using the *enoughTime* function, which determines if a new iteration is possible according to the total time that the TB-CBR has to complete each stage.

Algorithm 1 TB-CBR algorithm

Require: t_{max}, case-base

1: ($t_{learning}$,$t_{deliberative}$) $\longleftarrow$ timeManager(t_{max})
2: **if** SolutionQueue $\neq \emptyset$ **then**
3: **while** enoughTime(t_{now},t_{revise},t_{retain},$t_{learning}$) and SolutionQueue $\neq \emptyset$ **do**
4: r $\longleftarrow$ pop(SolutionQueue)
5: {adequate $\longleftarrow$ $f\_Revision(r)$}$^{\leq t_{revise}}$
6: **if** adequate **then**
7: $\{f\_Retention(r, case - base)\}^{\leq t_{retain}}$
8: **if** ProblemQueue $\neq \emptyset$ **then**
9: problem $\longleftarrow$ pop(ProblemQueue)
10: **repeat**
11: {cases $\longleftarrow$ Push($f\_Retrieval(problem, case - base)$)}$^{\leq t_{retrieve}}$
12: {solution $\longleftarrow$ $f\_Adaptation(cases)$}$^{\leq t_{reuse}}$
13: bestSolution $\longleftarrow$ bestSolution(solution,bestSolution)
14: **until** $\neg$enoughTime(t_{now},$t_{retrieve}$,t_{reuse},$t_{deliberative}$)
15: solutionQueue $\longleftarrow$ push(bestSolution)
16: **Return** bestSolution

3. Real-Time Argumentation

This section presents the adaption to real-time environments of the framework introduced in Section 2.1, and thus the original model becomes a real-time case-based argumentation framework. The original process allows agents to use the knowledge resources presented before to generate, select and evaluate their positions and arguments in real-time and automatically learn from the experience. Now, to temporal bound the original reasoning process, we have used the TB-CBR approach (see Figure 4). Thus, the process can be divided into three phases: generation and selection of positions, where the agent generates its potential positions and select the best one to propose; evaluation of positions, where the agents evaluates the positions generated by other agents; and finally, argument management, where agents can either defend their positions if they are attacked or else, attack other different positions proposed by their partners. To defend

their positions, agents have to evaluate the attack arguments received and generate and select the best support argument to propose. To attack different positions, agents have to ask the agent that they want to attack for providing them with a support argument for the position to attack. Then, agents can generate and select the best argument to attack the support argument provided. In this section we focus on the adaption of the original algorithms of the agents' reasoning process to real-time scenarios. For a complete documentation of the pseudocode of each algorithm we refer the reader to the work published in (Heras et al., 2013a).

The following sections explain these phases from the perspective of their temporal dimension. Along them, we assume that a set of agents with different positions are arguing to reach a real-time agreement to solve a complex problem that could be described with a set of features.

3.1. Position Generation and Selection

In the first step of our real-time reasoning process, an agent can generate its individual position to solve the problem at hand. To perform this process, the agent retrieves from the domain case-base those cases that match with the specification of the current problem and generates its solution (or a list of potential solutions) by reusing the solution(s) applied to the retrieved cases. Thus, the set of retrieved cases could provide different solutions for the same problem. Then, the agent can use its case-base of argument-cases to select the best position to propose.

The first step for this selection is to order the positions in subsets, taking into account the value promoted by each position. Thus, the agent will assign to each subset a *Suitability Level (SL)*. Positions that promote the agent's most preferred value will be labelled with suitability level 1, positions that promote the second most preferred value will be labelled with level 2 and so on. Then, positions will be ordered within each level by its *Similarity Degree (SimD)* with the problem to solve, computed by using a domain-dependent similarity measure.

After that, the agent will use the argumentation knowledge stored in its argument-cases case-base. For each position generated, the agent creates an argument-case that represents this position. This is a necessary translation to add the current social context

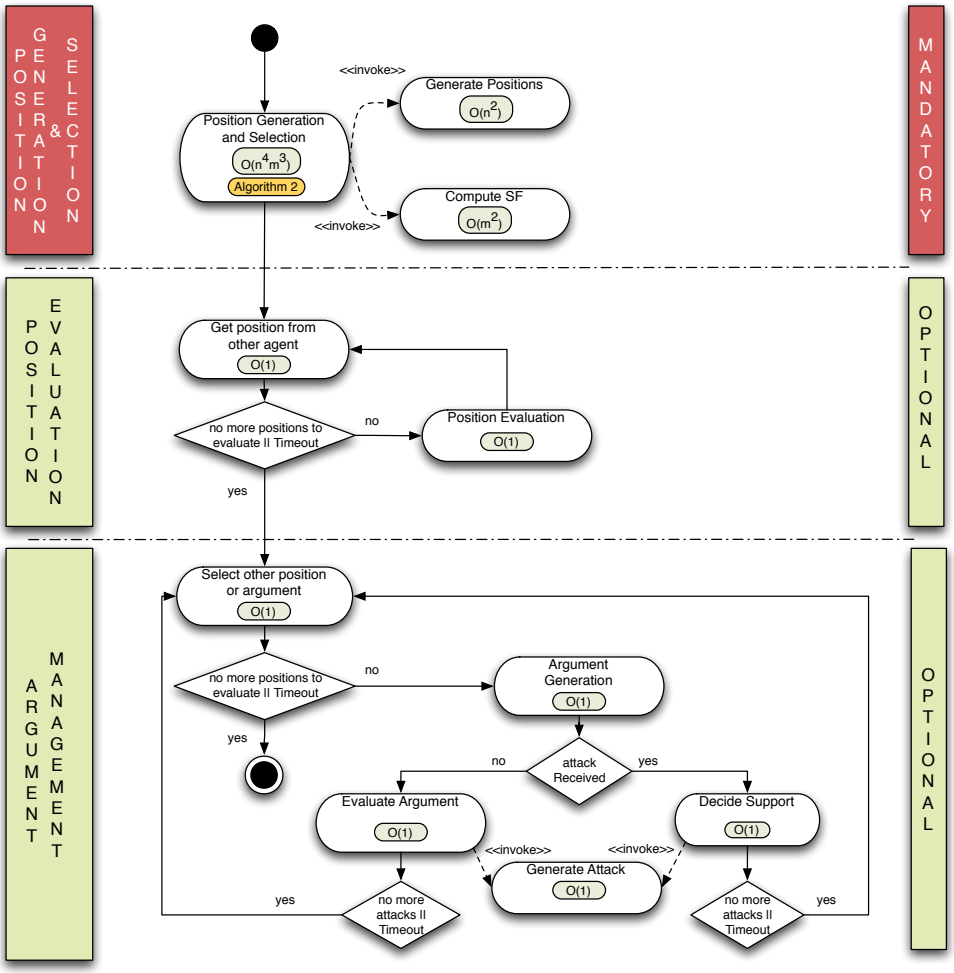

Figure 4: Real-Time Argumentation Process

to the position and be able to make queries to the argument-cases case-base and retrieve argument-cases that represent similar arguments that justify or attack the position. Then, the agent compares the argument-case created for each position with its case-base of argument-cases and retrieves the sets of argument-cases that match it. In this way, the agent can assign to each position a *Support Factor (SF)* from the argumentation point of view and decide which argument-case (and thus, which position) is most suitable to propose in view of its past argumentation experience and its current social context. As criteria for making such decision, we consider the following parameters:

- **Persuasiveness Degree (PD):** is a value that represents the expected persuasive power of an argument by checking how persuasive an argument-case with the same

problem description and conclusion was in the past. To compute this degree, the number of argument-cases that were deemed acceptable out of the total number of retrieved argument-cases with the same problem description and conclusion as the current argument is calculated.

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

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

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

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

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

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

Finally, the suitability factor of a new argument-case and its associated position is computed by the formula:

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

where $w_i \in [0, 1], \sum w_i = 1$ are weight values that allow the agent to give more or less importance to each decision criteria. Finally, positions are ordered from more to less suitability by following the equation:

$$Suitability = w_{SimD} * SimD + w_{SF} * SF \tag{2}$$

where $w_i \in [0, 1], \sum w_i = 1$ are weight values that allow the agent to give more or less importance to the similarity degree or the support factor. The most suitable position of suitability level 1 is selected as the one that the proponent agent is going to propose and defend first. However, each agent keeps the rest of positions to make alternative proposals if its original position is rebutted.

Algorithm 2 shows the pseudocode of the algorithm that implements the generation of positions, the generation of the associated argument-cases and the selection of positions. This pseudocode modifies the version presented in (Heras et al., 2013a) by adding the estimated temporal cost for each function. The temporal cost of executing this phase has been bounded to get the worst case execution time. This allows the agent to know how long it takes to extract a valid position. Thus, if the temporal restrictions of the application domain require the agent to propose a position in less time than it needs to generate al least one suitable position, the agent can reject the request and do not engage in the real-time agreement process. In this way, the agent does not waste computation time on tasks that it knows that it will not be able to complete on time.

In the algorithm, the function *generatePositions* generates the k first positions by using the algorithm *generatePositions*. In the worst case, the function has to check the whole case-base of domain-cases, incurring a temporal cost $O(n)$ (where n is the number the cases stored in the domain-cases case-base) for finding and extracting similar domain-cases. Then, for each domain-case extracted, the function reuses its solution to generate a position. Again, in the worst case (in the temporal sense), we can generate n positions. Therefore, the *generatePositions* function has an asymptotic temporal cost of $O(n^2)$.

The *generateArgumentCase* is a function that generates for each position its associated argument-case. This generation only changes the way in which we represent positions and we assume for it a constant temporal cost that is negligible in terms of the total temporal cost of the algorithm 2.

The *retrieveSimilarityDegree* is a function that retrieves the similarity degree of each position with regard to the problem to solve. This degree is stored with each position and we assume that the temporal cost for checking this information is constant and negligible in terms of the total temporal cost of the algorithm 2.

The *computeSF* is a function that computes the support factor for each position by means of its associated argument-case. In the worst case, this function has to check the whole case-base of argument-cases twice: one to retrieve the target set of argument-cases for the persuasiveness degree, support degree and risk degree and another to retrieve the average number of attacks, steps and knowledge resources of each subset retrieved

to compute the attack degree, efficiency degree and explanatory power. Therefore, this process has an asymptotic temporal cost of $O(m^2)$, where m is the number the cases stored in the argument-cases case-base.

The *selectPosition* is a domain-dependent function that sorts the set of positions from more to less suitable with respect to some domain-dependent criteria. Most comparison algorithms have a worst case temporal cost of $O(q^2)$, being q the number of positions generated by the algorithm. In our case, if we generate one position from each domain-case that is stored in the case-base, we have that $q = n$, where n is the size of the domain-cases case-base. Thus, we estimate this cost for selecting the best position to put forward as $O(n^2)$.

Finally, the *mostSuitable* is a domain-dependent function that returns the most suitable position to solve the problem (e.g. the first position from the sorted list computed by the *selectPosition*) function. Again, if we assume that the temporal cost for checking this information is constant, it can be negligible in terms of the total temporal cost of the algorithm 2.

Algorithm 2 Position Generation and Selection

Require: ProblemDescription, k, w_{SimD}, w_{PD}, w_{SD}, w_{RD}, w_{AD}, w_{ED}, w_{EP} *//The description of the problem to solve, the maximum number of positions to generate, and the weights for each element of the similarity degree and the support factor*

1: positions = $\emptyset$
2: argumentCases = $\emptyset$
3: SimD = $\emptyset$
4: SF = $\emptyset$
5: selectedPositions = $\emptyset$
6: positions = generatePositions(ProblemDescription, k) *//*$O(n^2)$
7: **for** $[i = 1; i \leq lenght(positions); i++]$ **do**
8: argumentCases[i] = generateArgumentCase(ProblemDescription, positions[i]) *//*$O(1)$
9: SimD[i] = retrieveSimilarityDegree(positions[i]) *//*$O(1)$
10: **for** $[i = 1; i \leq lenght(argumentCases); i++]$ **do**
11: SF[i] = computeSF(ProblemDescription, argumentCases[i], argumentCases, w_{PD}, w_{SD}, w_{RD}, w_{AD}, w_{ED}, w_{EP}) *//*$O(m^2)$
12: selectedPositions = selectPosition(positions, argumentCases, SD, SF) *//*$O(n^2)$
13: **Return** mostSuitable(selectedPositions) *//*$O(1)$

Summarising, the asymptotic cost to execute the process to generate and select positions in the worst case is $O(n^4 * m^3)$ where n is the number the cases stored in the domain-cases case-base and m is the number of cases stored in argument-cases case-base. Therefore, to be able to make a prediction and bound the temporal cost of the algorithm 2, the maximum number of cases in the domain-cases and the argument-cases case-base must be known and fixed in advance. In this way, if the contents of the agent's case-bases contain the necessary information to generate and select an appropriate position, the algorithm will be able to generate at least one position on time. In this sense, as shown in Figure 4 the position generation and selection phase is mandatory and the other two phases (i.e. position evaluation and argument management) are optional and executed while the agent has still time to perform its temporal bounded reasoning process.

3.2. Position Evaluation

Once the process to generate and select positions has finished, agents of our framework can either receive attacks to their positions or decide to challenge the positions of other agents. Thus, agents are able to evaluate its position with regard to other positions. The first step to evaluate an agent's position is to check if it is consistent with the positions of other agents. For the sake of simplicity, here we assume that a position is consistent with other position if they totally match (they are the same). Agents do not attack consistent positions, but they can still generate support arguments if their own positions are challenged. Another possibility arises when a proponent agent has been able to generate the position of another agent, but it has selected another position as more suitable to propose. In that case, the proponent would accept the opponent's position if the latter has a power relation over the proponent and would try to attack the opponent's position otherwise. Finally, if the opponent's position is not in the set of positions generated by the proponent and the opponent does not have a power relation over the proponent, the proponent can try to generate an argument to attack the opponent's position and otherwise, accept the opponent's position.

In our real-time argumentation scenario the number of agents engaged in the dialogue cannot be determined a priori. Therefore, the temporal cost of evaluating the positions of other agents cannot be bounded. Thus, this phase has been implemented as an anytime

process. The agent has a maximum time (*deadline*) to evaluate all positions generated by other agents. Once this deadline is exceeded, the agent stops the evaluation and proceeds to the next phase of its reasoning process. Next sections explain the type of arguments that agents can generate and how these arguments are selected and evaluated in a real-time agreement process.

3.3. Argument Management

Agents generate arguments when they are asked to justify their positions (*support arguments*) or when they attack others' positions or arguments (*attack arguments*). A support argument has a support set that consists of the set premises that describe the problem and of any of the knowledge resources used to generate the position to justify (domain-cases and argument-cases). Attack arguments are generated when the proponent of a position provides an argument to justify it and an opponent wants to challenge the position or else, when an opponent wants to attack the argument of a proponent. If the agent receives an attack, it must evaluate its current argument in view of the incoming argument that poses the attack.

The attack arguments that an agent can generate depend on the elements of the support set of the attacked argument. On one hand, if the support set includes a set of premises, the agent can generate an attack argument with a distinguishing premise. On the other hand, if the support set includes a domain-case or an argument-case, the agent can check its case-bases to find counter-examples to generate the attack.

As for the case of generating positions, agents can generate several attack arguments. Then, to select the best attack argument to put forward in a specific step of the real-time agreement process, the agent uses the information of its argument-cases case-base and selects such one that is expected to have higher persuasive power in view of the agent's previous experiences.

The process to generate and select arguments is very similar than the process to generate and select positions. In the worst case, this process has to check the whole domain-cases and argument-cases case-base, incurring the same temporal cost of algorithm 2, which is $O(n^4 * m^3)$ where n is the number the cases stored in the domain-cases case-base and m is the number of cases stored in argument-cases case-base. Therefore, to be

able to make a prediction and bound the temporal cost of the argument generation and selection process, the maximum number of cases in the domain-cases and the argument-cases case-base must be also known and fixed.

Finally, agents must evaluate their arguments in view of the arguments proposed by their opponents. Then, a proponent agent can decide if an opponent's argument conflicts with its argument and hence, its argument is deemed acceptable, non-acceptable or remains undecided (it cannot make a decision over it). This evaluation is performed by using the *defeat relation* between arguments defined in the original case-based argumentation framework (Heras et al., 2012), which specifies which attacks over arguments succeed. If the proponent argument defeats the opponent's, the latter acceptability status will change to non-acceptable. Then, the opponent can try to generate a new argument to counter-attack. On the contrary, if the proponent's argument cannot defeat the opponent's, the proponent has to withdraw its last argument and, if there are any, send to the opponent an alternative argument or propose an alternative position from its list of generated positions. In this case, the proponent's argument acceptability status would preliminary change to undecided. An argument that remained undefeated at the end of the real-time agreement process is deemed acceptable. The final acceptability status of arguments is decided by following a *dialogue-game* protocol (Jordán et al., 2011).

At this external level, the number of agents participating in the argumentation process cannot be known in advance and hence, we cannot estimate the number of arguments that the agent has to evaluate and the underlying attacks that the agent can receive or generate. As in the case of the evaluation of positions, we use here an anytime process by which the agent has been assigned a deadline to perform all interactions with the rest of agents. When this deadline is finished, the argument management phase ends and the agent must provide its final solution. Therefore, in the worst case where the agent has not enough time to argue, it does nor participate in the argumentation dialogue. Otherwise, if the allowed time is large enough, it can make an intensive use of its domain and argumentation knowledge and engage in the argumentation process proposing and defending all positions that it is able to generate.

4. Integrating Real-time Agreement Agents in a real application

In this section, we perform tests to evaluate the performance of the proposed real-time case-based argumentation framework. With this objective, the framework has been implemented in the domain of a customer support application. Concretely, we consider a society of agents that act in behalf of a group of technicians that must solve problems in a Technology Management Centre (TMC) (Heras et al., 2009b)(Heras et al., 2013a), which control every process implicated in the provision of technological and customer support services to private or public organisations by means of a call centre. Therefore, we set a society composed by call-centre technicians playing the role of *operator*. Operators form groups that must solve the problems that the call centre receives, commonly known as *tickets* in the call-centres jargon. The dependency relations in this society establish that technicians with the same role have a *charity* relation among them. In addition, the company that runs the call centre has signed a *Service Level Agreement (SLA)* with the customer that has contracted the support service. Among other terms, this SLA specifies a maximum acceptable time within the customer must receive a solution for the problem that has reported to the call centre. If the solution exceeds this time, the centre can receive an economic penalization in its contract with the customer, which is greater as time passes.

In this application domain we assume that each technician has a helpdesk application to manage the big amount of information that processes the call centre. In addition, this helpdesk would implement an argumentation module to solve each ticket as proposed in our framework. Hence, we assume the complex case where a ticket must be solved by a group of agents representing technicians that argue to reach an agreement over the best solution to apply. Each agent has its own knowledge resources (acceded via his helpdesk) to generate a solution for the ticket. The argumentation module of each agent includes a Domain-CBR engine that makes queries to its domain-cases case-base and an Argumentation-CBR engine that makes queries to its argument-cases case-base. The system has been implemented by using the Real-Time Multi-Agent Platform *jART* (Navarro et al., 2004). This platform is implemented in RT-Java and allows to execute agents in a time-controlled way, ensuring the fulfillment of the agents tasks within a

specific time.

The data-flow for the argumentation dialogue to solve each ticket is the following:

1. The system presents a group of technicians with a new ticket to solve.
2. An agent, called the initiator agent, opens a new argumentation dialogue.
3. Technicians enter in the dialogue.
4. If possible, each technician generates his own position by using the argumentation module and propose it. This module supports the argumentation framework proposed in this paper. In this sense, technicians propose their solutions before the maximum time specified in the SLA contracted by the customer is met. All technicians that are willing to participate in the argumentation process are aware of the positions proposed in each moment.
5. The technicians argue to reach an agreement over the most suitable solution by following a persuasion dialogue controlled by a dialogue game protocol (Jordán et al., 2011). The dialogue proceeds as a set of parallel dialogues between technicians. Therefore, one technician can only argue with another at the same time. In each parallel dialogue, two technicians take turns to challenge the positions of the other technician, to assert arguments to justify their positions or to attack the other technician's positions and arguments. If a position is attacked but the opponent agent cannot defeat it, the position receives one vote. Otherwise, the proponent of the position must withdraw it from the dialogue.
6. The dialogue ends when an agreement is reached (i.e. only one position remains undefeated in the argumentation dialogue) or a deadline specified in the SLA is met. The best solution is proposed to the user and feedback is provided and registered by each technician helpdesk. In this way, agents update their case-bases with the information of the actual solution applied and the arguments generated during the argumentation dialogue. If there is no agreement reached, the best solution can be selected by using different policies, such as to select the most voted position or the most frequent.

In this domain, we assume that the most efficient technicians are acknowledged and rewarded by the company. Therefore, each technician follows a *persuasion* dialogue with

their partners, trying to convince them to accept its solution as the best way to solve the ticket received, while observing the common objective of providing the best solution for a ticket on time. Note that with this approach, we are assuming that the TB-CBR process is working properly and as the systems evolves, agents learn correct solutions in their domain-cases case base and useful arguments in their argument-cases case-base, hence providing more accurate solutions.

To evaluate the system, we have run several tests populating the call centre with two different types of agents: agents that implement our original case-based argumentation framework, where no real-time considerations were taken into account in the agents' reasoning process to manage positions and arguments ($noRT$ agents); and agents that implement the real-time case-based argumentation framework proposed in this work, which follow a temporal bounded TB-CBR cycle to manage their positions and arguments (RT agents). Thus, the latter type of agents are able to provide answers within the specific time that the SLA requires. With these tests, we evaluate the efficiency of the system that implements the real-time case-based argumentation framework by comparing its performance with the one achieved by the original framework.

For the tests, a real database of 200 tickets solved in the past is used as domain knowledge. Translating these tickets to domain-cases, we have obtained a tickets case-base with 48 cases. Despite the small size of this case-base, we have rather preferred to use actual data instead of a larger case-base with simulated data. The argument-cases case-bases of each agent are initially empty and populated with cases as the agents acquire argumentation experience in execution of the system. To diminish the influence of random noise, for each round in each test, all results report the average and confidence interval of 48 simulation runs at a confidence level of 95%, thus using a different ticket of the tickets case-base as the problem to solve in each run. The results report the mean of the sampling distribution.

The experiments have been performed with a population of 7 agents representing operators (in (Heras, 2011, Chapter 6) we discuss that populations greater than this number are not appropriate to elicit useful results with the small size of our case-base). Each test has been repeated for different assignments of the domain and argumentation

knowledge that agents have. Concretely, the domain-cases of the case-bases of the agents were randomly populated and increased by 5 from 5 to 45 cases in each experimental round. Also, the argument-cases case-base of each agent were randomly populated with argument-cases and increased by 2 from 2 to 18 cases in each experimental round. These cases were obtained from a set of cases created by training the system performing several problem-solving executions. (Heras, 2011, Chapter 6) demonstrates that this is reasonable amount of argument-cases that are elicited from the possible argumentation experiences of 7 agents arguing with a maximum knowledge of 45 domain-cases in our experimental domain. Therefore, each experiment is repeated for each type of agents (noRT and RT agents) during 9 rounds. In the first round, agents have 5 domain-cases (dc) and 2 argument-cases (ac), in the second round agents have 10 domain-cases and 4 argument-cases and so on up to 45 domain-cases and 20 argument-cases.

In each simulation, an agent is selected randomly as initiator of the discussion. This agent has the additional function of collecting data for analysis and controlling the deadline specified in the SLA. However, from the argumentation perspective, its behaviour is exactly the same as the rest of agents and its positions and arguments do not have any preference over others (unless there is a dependency relation that states it). The initiator agent receives one problem to solve per run. Then, it contacts its partners (the agents of its group) to report them the problem to solve. If the agents do not reach an agreement after a maximum time[1], the initiator chooses the most supported (the most voted) solution as the final decision (or the most frequent in case of draw). If the draw persists, the initiator makes a random choice among the most frequent solutions. To check the solution accuracy (the average error in the solutions that the system provides), the solution agreed by the agents for each ticket requested is compared with its original solution, stored in the tickets case-base.

In the first tests, the call centre is first managed by a group of RT agents and after that by a group of noRT agents and there is no time limitation to provide answers to the customers requests (*no deadline* specified in the SLA). Then, the percentage of problems

[1]If the SLA does not define a maximum time, the initiator allows agents to argue until they do not have more positions and arguments to interchange.

that the system is able to solve (provide a solution, regardless of its level of suitability), and the average solution error have been analysed. One can expect that having more domain and argumentation knowledge (more domain and argument-cases in the agents' case-bases) will prevent less problems from being undecided and will increase the number of problems that were correctly solved (the mean error in the solution predicted decreases). As presented in Table 1, for both types of agents the system achieves the same performance results, which is the expected result since the core argumentation skills of noRT and RT agents are exactly the same (they only differ in the temporal management of their reasoning process). The percentage of problems solved by the system increases and the mean error decreases with the number of domain and argument-cases that agents have, up to the 100% of solved problems with no errors from 25 domain-cases and 10 argument-cases onwards.

Cases / noRT & RT	5dc/2ac	10dc/4ac	15dc/6ac	20dc/8ac	25dc/10ac	30dc/12ac	35dc/14ac	40dc/16ac	45dc/18ac
Solved Problems %	64.58%	87.50%	97.92%	97.87%	100%	100%	100%	100%	100%
Mean Error %	39.58%	16.67%	6.25%	4.26%	0.00%	0.00%	0.00%	0.00%	0.00%

Table 1: Percentage of problems that the system is able to solve and error average (with no deadline in the SLA)

Also, Figure 5 shows that the average time that noRT and RT agents take to solve problems are very similar. At first, the mean resolution time increases as the contents of the case-bases do, since agents have more justification elements and argumentation skills to generate more positions to propose and to defend their positions from attacks, which gives rise to longer agreement processes. However, up to 25 domain-cases and 10 argument-cases onwards the case-bases of agents begin to include many redundant data and agents reach agreements more quickly (since they propose the same or very similar positions). Therefore, the average time of the agreement process decreases and tends to the minimum exchange of locutions to communicate data among agents.

As shown in Figure 5, both noRT and RT agents need a minimum of 1200 milliseconds to provide a solution with the fewer possible amount of information in their case-bases (regardless of its quality) for the requests received by the system. Now, to test the performance of the system with a tight deadline specified in the SLA, we have repeated the above tests allowing agents having only 1100 milliseconds to provide solutions. After

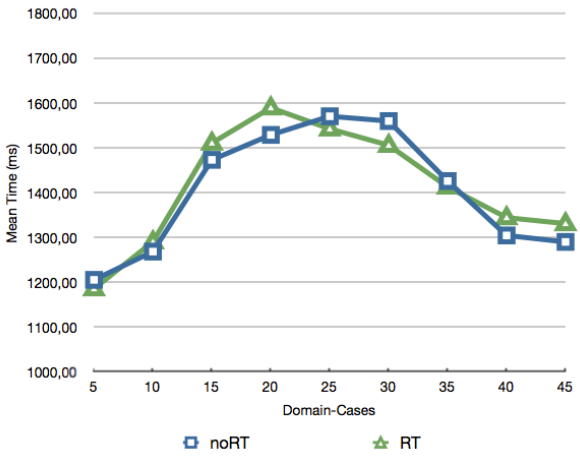

Figure 5: Average Time of the Agreement Process in milliseconds (with no deadline in the SLA)

this deadline, we consider that the problem has not been solved and thus, the agreement process has failed (counting as a new error).

Table 2 compares the percentage of problems that the system is able to solve with noRT and RT agents and a strict deadline of 1100 milliseconds. In addition, Table 3 presents the error average achieved in these tests. The results obtained for both types of agents in the case that there is not deadline is also shown for comparative purposes in both tables. Results from Tables 2 show that RT agents are able to bound their reasoning cycle and to provide much more solutions on time than noRT agents, thus achieving much lower error percentages (see Table 3). However, although the percentage of solved problems increases and the average error in the solutions provided decreases with the amount of knowledge that agents have in their case-bases, they do not have enough time to solve all problems with no error, as they are with the appropriate amount of knowledge (up to 25 domain-cases and 10 argument-cases) if no deadline is specified. In fact, as presented in Table 3, the quality of the solutions provided by RT agents is worst than the one that they achieve operating without deadline, but still it is much better than the one achieved by noRT agents. Nevertheless, in many real-time application domains is preferable to have a good enough solution (even if it is not the optimum) than leave the problem unresolved. Therefore, RT agents prevent many problems to remain unresolved and the real-time process to end in disagreement.

Finally, Figure 6 shows the average time that noRT and RT agents take to solve

Solved Problems % \ Cases	5dc/2ac	10dc/4ac	15dc/6ac	20dc/8ac	25dc/10ac	30dc/12ac	35dc/14ac	40dc/16ac	45dc/18ac
noDeadline	64.58%	87.50%	97.92%	97.87%	100%	100%	100%	100%	100%
noRT (1100 ms)	10.42%	20.83%	37.50%	37.50%	38.91%	45.83%	47.92%	52.08%	52.08%
RT (1100 ms)	50.00%	70.83%	75.00%	76.60%	76.95%	77.08%	77.08%	81.25%	87.50%

Table 2: Percentage of problems that the system is able to solve (deadline 1100 ms)

Solved Problems % \ Cases	5dc/2ac	10dc/4ac	15dc/6ac	20dc/8ac	25dc/10ac	30dc/12ac	35dc/14ac	40dc/16ac	45dc/18ac
noDeadline	39.58%	16.67%	6.25%	4.26%	0.00%	0.00%	0.00%	0.00%	0.00%
noRT (1100 ms)	91.67%	79.17%	68.09%	68.09%	62.50%	54.17%	52.06%	47.92%	47.92%
RT (1100 ms)	50.00%	31.25%	25.00%	23.40%	23.13%	22.92%	22.92%	18.75%	12.50%

Table 3: Error average in the solutions provided by the system (deadline 1100 ms)

problems with a deadline of 1100 milliseconds specified in the SLA. Opposite to RT agents, noRT agents are not able to bound their reasoning cycle to propose positions and generate attacks and counter-attacks within this deadline. Thus, the mean time of their agreement processes is almost the same than the one presented in Figure 5 (variations are due to different temporal costs incurred in communicating positions and arguments via the network). This does not meet the requirements of the SLA and gives rise to the bad results presented in Tables 2 and 3 for the percentage of solved problems and mean error. On the contrary, the mean time in the agreement processes of RT agents slightly variates around the specified deadline of 1100 milliseconds. Again, variations over this time are due to different temporal costs incurred in communicating positions and arguments via the network. Therefore, RT agents succeed in adjusting their reasoning cycles to the tight deadline they are allowed, and with a medium amount of knowledge (up to 25 domain-cases and 10 argument-cases) are able to solve more than the 77% of problems with less than the 23% of error.

5. Related Work

Over the last years, the research on technologies to reach agreements in computing systems is experiencing an exponential growth. The term *agreement technologies* already appears as a topic in the main AI and MAS conferences (e.g. the International Conference of Autonomous Agents and Multi-agent Systems (`AAMAS`) and the International Joint Conference on Artificial Intelligence (`IJCAI`) and there are new conference tracks and workshops specialised on this area (e.g. Workshop on Agreement Technologies (`WAT`)).

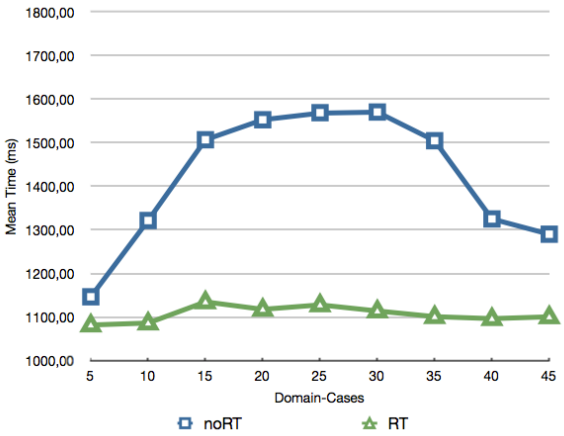

Figure 6: Average Time of the Agreement Process in milliseconds (deadline 1100 ms)

However, the transfer of these technologies to the industry is still a challenge, and the few commercial organisations which adopt them would be classified as early adopters (Luck and McBurney, 2008).

In addition, if the adoption of agreement technologies is still at its early stages, the application of these technologies to real-time scenarios is even more novel. Focusing on argumentation as an agreement technology, the literature reports few contributions on real-time applications of argumentation frameworks. In fact, these contributions are not specifically focused on bounding the reasoning process that agents follow to reach real-time agreements, but consist of negotiation systems where agents stop the negotiation process before a specific time. This is the case of the the real-time negotiation model for reflective agents proposed in (Soh and Tsatsoulis, 2005), applied to resource allocation problems (concretely, to multi-sensor target tracking). The model is an case-based negotiation model that integrates a real-time BDI architecture for the agents with a temporal logic model. These features allow to establish when certain states of the negotiation protocol have to be true and for how long, which makes real-time implementation feasible. However, in this framework cases are situated and need domain-specific adaption rules to devise strategies that are applicable to the current negotiation context from them. Opposite to our proposal, in this work persuasion is viewed as a negotiation protocol for information exchange between two agents. These agents try to reach a deal in which the initiator agent provides arguments to support a request to convince the responding agent

to share its sensing resources with it. Thus, this is a two-party agreement process where different positions and attacks among them are not considered. Also, the model assumes certain characteristics that can pose several drawbacks to its application to open MAS. On the one hand, the list of neighbours and their sensors must be known in advance. On the other hand, despite concurrency being admitted, the agents can only negotiate about one issue at the same time. Finally, the framework has strong assumptions about the honesty, cooperativeness and rationality of the agents that do not fit the reality of many real-time application domains.

Opposite to Soh's and our framework, most argumentation systems produce arguments by applying a set of inference rules. Rule-based systems require to elicit a explicit model of the domain. In open MAS the usual application domains are highly dynamic and the set of rules that model them is difficult to specify in advance. However, tracking the arguments that agents put forward in argumentation processes could be relatively simple. Therefore, these arguments can be stored as cases codified in a specific case representation language that different agents are able to understand. This is easier than creating an explicit domain model, as it is possible to develop case-bases avoiding the knowledge-acquisition bottleneck. With these case-bases, agents are able to perform *lazy learning* processes over argumentation information. Another important problem with rule-based systems arises when the knowledge-base must be updated (e.g. adding new knowledge that can invalidate the validity of a rule). Updates imply to check the knowledge-base for conflicting or redundant rules. Case-based systems are easier to maintain than rule-based systems since, in the worst case, the addition of new cases can give rise to updates in some previous cases, but does affect the correct operation of the system, although it can have an impact in its performance. Hence, a case-based representation of the domain knowledge of the system is more suitable for being applied in dynamic open MAS. From the first uses of argumentation in AI, arguments and cases are intertwined (Skalak and Rissland, 1992). Case-based argumentation particularly reported successful applications in American common law (Bench-Capon and Dunne, 2007). However, these models assumed human-computer interaction and cases were not thought to be only acceded by software agents. In MAS, the research in case-based argumentation has just a few pro-

posals (Heras et al., 2009a). These proposals are highly domain-specific (e.g. persuasion in negotiation (Sycara, 1990), sensor networks (Soh and Tsatsoulis, 2005) and classification (Ontañón and Plaza, 2007)) or centralise the argumentation functionality either in a *mediator* agent, which manages the dialogue between the agents of the system (Tolchinsky et al., 2007), or in a specific module of the system itself (Karacapilidis and Papadias, 2001).

The notion of *real-time*, it also appears in the literature on visualisation tools for collaborative argumentation (Kirschner, 2003, Chapters 6-8), which support human cognitive and discursive processes, and provide suitable representations, services and user interfaces. However, the term *real-time* is used here as a synonym of *now* and refers to produce and visualise arguments as the argumentation dialogue proceeds. The same meaning of real-time was used in the *rIBIS* real-time group hypertext system, which extends the *IBIS* informal-logic argumentation framework (Rittel and Webber, 1973). This system allows a distributed set of users to simultaneously browse and edit multiple views of a hypertext network. Also, the work presented in (M. Capobianco et al., 2004) proposes the use of *dialectical databases* to comply with real-time issues when modeling agent interaction in a MAS. These databases are used to *speed up* the inference process in the *ODeLP* logic programming language by keeping track of all possible potential arguments and the defeat relation among them.

Opposite to the interpretation of the *real-time* concept as *doing things at the current time*, in our approach argumentation is used to ensure that agents reach real-time agreements *before* a deadline is met.

6. Conclusions

This work has presented a *real-time argumentation framework* that provides agents with the ability of engaging in argumentative dialogues and come with a solution for their underlying agreement process within a bounded period of time. In open multi-agent argumentation systems the arguments that an agent generates to support its position can conflict with arguments of other agents and these conflicts are solved by means of argumentation dialogues between them. In our proposal, a computational case-based argumentation framework that agents can use to reason about argumentation processes

and new case-based reasoning techniques, called Temporal Bounded CBR (TB-CBR), that adapt the CBR methodology for its use in real-time MAS have been combined to allow agents to cope with agreement processes in real-time scenarios.

To evaluate the framework, it has been implemented in the domain of a customer support application. Concretely, we consider a society of agents that act in behalf of a group of technicians that must solve problems in a call centre, which control every process implicated in the provision of technological and customer support services to private or public organisations. Results shown that if the agents of the centre implement our real-time argumentation framework, they are able to temporal bound their reasoning processes and to provide suitable enough solutions within a specified deadline.

However, in this paper we only cope with the temporal bounding of the argument management process to reach real-time agreements in MAS. As discussed in (G. Mahdi et al., 2010) "there is a need of an integrated and comprehensive view of the real-time phenomenon when suggested for multi-agent systems". Therefore, further work will advance research in this area by taking into account real-time issues not only on the agents reasoning process, but also on the dialogue protocol that agents follow to interchange arguments and reach agreements.

Acknowledgements

This work is supported by the Spanish government grants [CONSOLIDER-INGENIO 2010 CSD2007-00022, and TIN2012-36586-C03-01] and by the GVA project [PROMETEO 2008/051].

References

Amgoud, L., Prade, H., 2004. Reaching agreement through argumentation: A possibilistic approach, in: 9th International Conference on Principles of Knowledge Representation ans Reasonning, KR-04, AAAI Press. pp. 175–182.

Bench-Capon, T., Dunne, P., 2007. Argumentation in artificial intelligence. Artificial Intelligence 171, 619–938.

Bench-Capon, T., Sartor, G., 2003. A model of legal reasoning with cases incorporating theories and values. Artificial Intelligence 150, 97–143.

Choy, M., Srinivasan, D., Cheu, R., 2003. Cooperative, hybrid agent architecture for real-time traffic signal control. IEEE Transactions on Systems Man and Cybernetics Part A Systems and Humans 33, 597–607.

Dean, T., Boddy, M., 1988. An analysis of time-dependent planning, pp. 49–54.

Dignum, F., Weigand, H., 1995. Communication and Deontic Logic, in: Wieringa, R., Feenstra, R. (Eds.), Information Systems - Correctness and Reusability. Selected papers from the IS-CORE Workshop, World Scientific Publishing Co.. pp. 242–260.

G. Mahdi, A. Gouaïch, F. Michel, 2010. Towards an integrated approach of real-time coordination for multi-agent systems, in: Proceedings of the 4th KES international conference on Agent and multi-agent systems: technologies and applications, Part I, Springer-Verlag. pp. 253–262.

Heras, S., 2011. Case-Based Argumentation Framework for Agent Societies. Ph.D. thesis. Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València. http://hdl.handle.net/10251/12497.

Heras, S., Botti, V., Julián, V., 2009a. Challenges for a CBR framework for argumentation in open MAS. Knowledge Engineering Review 24, 327–352.

Heras, S., Botti, V., Julián, V., 2012. Argument-based agreements in agent societies. Neurocomputing 75, 156–162.

Heras, S., García-Pardo, J.A., Ramos-Garijo, R., Palomares, A., Botti, V., Rebollo, M., Julián, V., 2009b. Multi-domain case-based module for customer support. Expert Systems with Applications 36, 6866–6873.

Heras, S., Jordán, J., Botti, V., Julián, V., 2013a. Argue to agree: a case-based argumentation approach. International Journal of Approximate Reasoning 54, 82–108.

Heras, S., Jordán, J., Botti, V., Julián, V., 2013b. Case-based strategies for argumentation dialogues in agent societies. Information Sciences 223, 1–30.

Jordán, J., Heras, S., Julián, V., 2011. A customer support application using argumentation in multi-agent systems, in: 14th International Conference on Information Fusion (FUSION-11), pp. 772–778.

Karacapilidis, N., Papadias, D., 2001. Computer supported argumentation and collaborative decision-making: the HERMES system. Information Systems 26, 259–277.

Kirschner, Paul A. and Shum, S. (Ed.), 2003. Visualizing argumentation: software tools for collaborative and educational sense-making. Springer-Verlag.

Kraus, S., Sycara, K., Evenchik, A., 1998. Reaching agreements through argumentation: A logical model and implementation. Artificial Intelligence 104, 1–69.

Luck, M., McBurney, P., 2008. Computing as interaction: agent and agreement technologies, in: IEEE International Conference on Distributed Human-Machine Systems, IEEE Press.

M. Capobianco, C.I. Chesñevar, G.R. Simari, 2004. An Argument-Based Framework to Model an Agent's Beliefs in a Dynamic Environment, in: 1sr International Workshop on Argumentation in Multi-Agent Systems (ArgMAS-04), pp. 163–178.

Navarro, M., Heras, S., Julián, V., Botti, V., 2011. Incorporating temporal-bounded cbr techniques in real-time agents. Expert Syst. Appl. 38, 2783–2796.

Navarro, M., Julián, V., Soler, J., Botti, V., 2004. jart: A real-time multi-agent platform with rt-java, in: 3rd IWPAAMS, pp. 73–82.

Navarro, M., de Paz, J.F., Julián, V., Rodríguez, S., Bajo, J., Corchado, J., 2012. Temporal bounded reasoning in a dynamic case-based planning agent for industrial environments. Expert Systems With Applications 29, 7887–7894.

O. Kahtib, 1986. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. The International Journal of Robotics Research 5, 90–98.

Ontañón, S., Plaza, E., 2007. Learning and joint deliberation through argumentation in multi-agent systems, in: 7th International Conference on Agents and Multi-Agent Systems, AAMAS-07, ACM Press.

Parsons, S., Sierra, C., Jennings, N.R., 1998. Agents that reason and negotiate by arguing. Journal of Logic and Computation 8, 261–292.

Patterson, D.W., Galushka, M., Rooney, N., 2005. Characterisation of a novel indexing technique for case-based reasoning. Artificial Intelligence Review 23, 359–393.

Quinlin, J.R., 1993. C4.5 Programs for Machine Learning. Morgan Kauffman.

Rittel, H., Webber, M., 1973. Dilemmas in a general theory of planning. Policy Sciences 4, 155–169.

Sierra, C., Botti, V., Ossowski, S., 2011. Agreement Computing. KI - Künstliche Intelligenz DOI: 10.1007/s13218-010-0070-y.

Skalak, D., Rissland, E., 1992. Arguments and cases: An inevitable intertwining. Artificial Intelligence and Law 1, 3–44.

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

Sycara, K., 1990. Persuasive argumentation in negotiation. Theory and Decision 28, 203–242.

Tolchinsky, P., Atkinson, K., McBurney, P., Modgil, S., Cortés, U., 2007. Agents deliberating over action proposals using the ProCLAIM model, in: 5th International Central and Eastern European Conference on Multi-Agent Systems and Applications, CEEMAS-07, Springer. pp. 32–41.

V. Julián, V. Botti, 2004. Developing real-time multi-agent systems. Integrated Computer-Aided Engineering 11, 135–149.

Wess, S., Althoff, K.D., Richter, M., 1993. Using k-d trees to improve the retrieval step in case-based reasoning, in: European Workshop, Topics in Case-based Reasoning, pp. 67–81.