A Customer Support Application Using Argumentation in Multi-Agent Systems

Jaume Jordán, Stella Heras, and Vicente Julián
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
València, Spain
Email: {jjordan,sheras,vinglada}@dsic.upv.es

Abstract—Multi-Agent Systems are suitable to provide a framework that allows to perform collaborative processes in distributed environments. In a customer support system with operators attending incidences, the problem to solve is to find out the best solution for the problems reported to the system. Each operator can have its own view about which is the best solution in each case and thus, conflicts of opinion among agents arise. Therefore, to engage in an argumentation dialogue is a suitable way for a group of agents (representing operators) to obtain an agreement about the best solution to solve an incidence. In this paper, an argumentation framework for a Multi-Agent System applied to customer support is proposed to help agents to reach an agreement and jointly solve incidences.

Keywords: Argumentation, Multi-Agent Systems, Case-Based Reasoning.

I. INTRODUCTION

Nowadays, companies have strong competitors in all markets. Thus, a good customer support can be the key to increase the satisfaction of customers and hence, the profits of a company. Also, the experience and skills of operators are decisive to obtain a quick and accurate response to the customers’ needs.

A common customer support system of a company consists of a network of operators that must solve the incidences (also known as tickets) received in a Technology Management Centre (TMC). TMCs are entities which control every process implicated in the provision of technological and customer support services to private or public organisations. In a TMC, there are a number of operators whose role is to provide the customers with technical assistance. This service is commonly offered via a call centre. The call centre operators have computers provided with a helpdesk software and phone terminals connected to a telephone switchboard that balances the calls among operators.

The experience about the problem-solving process and the final solution applied to each problem could be a good way to improve the performance of the helpdesk software that the operators of the call centre use. Case-Based Reasoning (CBR) systems have been widely applied to cope with this task. A CBR system tries to solve a problem (case) by means of reusing the solution of an old similar case [6]. This solution is previously stored in a memory of cases (case-base) and it can either be retrieved and applied directly to the current problem, or revised and adapted to fit the new problem. The suitability of CBR systems in helpdesk applications to manage call centres has been guaranteed for the success of some of these systems from the 90s to nowadays [1], [7], [9].

These approaches propose systems for human-machine interaction where the CBR functionality helps the operators to solve problems more efficiently by providing them with potential solutions via the helpdesk software. In this paper, we extend a previous work that presented a CBR module that acts as a solution recommender for customer support environments [3]. The CBR module is flexible in order to be easily integrable with any existing helpdesk software in a company. However, to maintain all the data centralised in one CBR is inefficient and a distributed approach is necessary in that case. Also, in many companies the operators that work for a specific project sign secrecy clauses and are not able to share certain knowledge with other operators working in different projects. Therefore, an approach that proposes a centralised CBR that stores information about solutions of previous incidences provided by different operators is unrealistic in this case.

Multi-Agent Systems (MAS) are suitable to provide a framework that allows to perform collaborative processes in distributed environments. Sometimes, in very large systems the information has to be distributed. Thus, a MAS approach is a good choice to have the data distributed among agents and to preserve the knowledge privacy. We use an approach that involves CBR and argumentation in MAS [5], [8]. In this paper, we present a customer support application based on a case-based argumentation framework for MAS. This application improves the performance of a simple helpdesk with CBR since the decision about the solution to apply for an incidence (ticket) is made by an agreement between agents after an argumentation dialogue. Each agent of the MAS represents an operator or an expert (depending on the adopted role) of the call centre and it has its own CBR (with knowledge about previous problems solved and the solution applied). Agents that play the operator role represent technicians of the call centre, while agents playing the expert role represent specialised technicians with more specific knowledge about the suitable solution to apply for each incidence. Hence, the operators and experts engage in an argumentation dialogue to argue about the solution proposed by each one and decide the best one to apply.

The customer support application using argumentation in
MAS proposed in this work is an intelligent system that allows information fusion in the sense that it integrates different types of information (domain specific information of several types, depending on the nature of the incidences received, and arguments) and intelligent technologies (CBR, Argumentation and MAS). Also, the information treated by the system is stored in case-bases and transformed in arguments for the agreement process.

This paper is structured as follows. This Section is an introduction to the problem and the contents of the paper. In Section II the argumentation framework used for the customer support application developed is presented. In Section III the customer support application using argumentation in MAS is explained. Section IV shows an evaluation of the performance of the developed application. Finally, Section V presents the conclusions extracted from this work.

II. ARGUMENTATION FRAMEWORK

In this section we propose a simplification of a computational framework, proposed in [4], for design and implementation of MAS in which the participating software agents are able to manage and exchange arguments between themselves. Here, we introduce the knowledge resources that agents can use to generate, select and propose their positions (solutions proposals) and arguments to support them. The knowledge resources used are the domain-cases of a database. These cases represent previous problems and their solutions. Furthermore, we present the argument types of the framework (support and attack arguments) and their support set, that is a set of elements that support the argument. Finally, the argumentation protocol that agents follow is shown. This protocol is the mechanism to manage arguments and define the argumentation dialogue that agents follow.

A. Knowledge Resources, Argument Types and Support Set

In open multi-agent argumentation systems the arguments that an agent generates to support its position can conflict with arguments of other agents and these conflicts are solved by means of argumentation dialogues between them. In our framework we have a domain-cases database, with cases that represent previous problems and their solutions. The domain-cases are used to generate positions (solutions) to defend and arguments to support them or attack other positions. The structure of these cases is domain-dependent and consist of a set of features that describe the problem to solve and the solution applied.

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

\( a) \text{Argument: } \text{Arg} = \{\phi, v, < S >\} \), where \( \phi \) is the conclusion of the argument, \( v \) is the value (e.g. economy, quality, solving speed) that the agent wants to promote with it and \( < S > \) is a set of elements that support the argument (support set).

\( b) \text{Support Set: } S = \{\text{premises}, \{\text{domainCases}\}, \{\text{distinguishingPremises}\}, \{\text{counterExamples}\}\} \)

A support set is formed by the following elements:

- Premises: which are features that match with some features of the problem description. These are the features that characterise the problem and that the agent has used to retrieve similar domain-cases from its case-base. Note that the premises used might be all features of the problem description or a sub-set.
- Domain cases: which are cases that represent previous problems and their solutions whose features match with some features of the problem description.
- Distinguishing premises: which are premises that can invalidate the application of a knowledge resource to generate a valid conclusion for an argument. This premises are extracted from a domain-case that propose a different solution to the argument to attack. They consist of features of the problem description that where not considered to draw the conclusion of the argument to attack.
- Counter-examples: which are cases that are similar to a case (their descriptions matches with some or all features of the problem description) but have different conclusions.

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

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

The second case happens when the proponent of a position generates an argument to justify it and an opponent wants to attack the position or more generally, when an opponent wants to attack the argument of a proponent. Arguments in our framework can be attacked by putting forward distinguishing premises and counter-examples.

The attack arguments that the opponent can generate depend on the elements of the support set of the argument of the proponent:

- If the justification for the conclusion of the argument is a set of premises, the opponent can generate an attack argument with a distinguishing premise that it knows. It can do it, for instance, if it is in a privileged situation and knows extra information about the problem or if it is implicit in a case that it used to generate its own position, which matches the problem specification. In the latter, the opponent could generate an attack argument with this case as counter-example.
• If the justification is a domain-case, then the opponent can check its case-base of domain-cases and try to find counter-examples to generate an attack argument with them. Alternatively, it can also try to generate an attack argument with a distinguishing premise from its own known premises and cases that invalidates the proponent’s justification.

B. Argumentation Protocol

The agents of the framework need a mechanism to manage the arguments and perform the argumentation dialogue. Therefore, an argumentation protocol has been defined. This protocol is represented by a set of locutions that the agents use to communicate each other depending on their needs, and an state machine that defines the behaviour of an agent in the argumentation dialogue.

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

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

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

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

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

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

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

• assert\((a_s, a_r, \phi)\), where \(\phi\) is an argument \(arg\) that supports a position. With this locution, an agent \(a_s\) sends to an agent \(a_r\) an argument that supports its position.

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

• attack\((a_s, a_r, \phi)\), where \(\phi\) is an argument \(arg\). With this locution, an agent \(a_s\) challenges the argument \(arg\) of an agent \(a_r\).

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

Figure 1 shows the state machine that defines the behaviour of an agent in an argumentation dialogue and the process that follows to propose positions, defend them and attack others’ positions. The transitions between states depend on the locutions that the agent could use in each situation. The states of the argumentation state machine are described as follows:

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

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

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

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

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

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

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

III. CUSTOMER SUPPORT APPLICATION USING ARGUMENTATION IN MULTI-AGENT SYSTEMS

The argumentation framework described in section II has been applied to a customer support application domain. A prototype that provides support to the operators and experts of a call centre has been implemented in a helpdesk application.

In our prototype, the operators and experts of a call centre are represented by agents that access to an automated helpdesk and argue to solve an incidence. Every agent has individual CBR resources and preferences over values (e.g. economy, quality, solving speed). A solution to a problem promotes one value. Thus, each agent has its own preferences to choose a solution to propose. Furthermore, agents can play two different roles: operator and expert. The main difference between an operator and an expert is that the second one has more domain knowledge. Also, dependency relations between roles could imply that an agent must change or violate its value preference order. For instance, an expert could impose their values to an operator and the last could have to adopt a certain preference order over values. Therefore, we endorse the view of [2], who stress the importance of the audience in determining whether an argument is persuasive or not for accepting or rejecting someone else’s proposals.
Following, we describe the different modules of the implemented prototype:

- **Magentix2**: to develop this prototype we have used the Magentix2 agent platform\(^1\). Magentix2 is an agent platform that provides new services and tools that allow for the secure and optimised management of open MAS. In our system, this platform is used for the communication between the agents.

- **Domain CBR**: consists of a CBR module with data about previous problems solved in the call centre. This CBR is initialised with past tickets of the helpdesk application. To make a query, the user has to provide a ticket and a threshold of similarity. The domain CBR module searches the domain case-base and returns a list of similar domain-cases to the given ticket. The similarity algorithm used is based on the Euclidean distance between the attributes of the tickets. In addition, with every CBR cycle performed, the module adds, modifies or deletes one or more domain-cases of the case-base.

- **Argumentation agent**: it is an agent with a domain CBR capable to engage in an argumentation dialogue to solve an incidence. These agents learn about the domain problem adding and updating cases into the domain case-base with each CBR run. Furthermore, the agent can play any role defined before (operator or expert). In our prototype, this agent is a extension of Magentix2 Base-Agent\(^2\).

- **Commitment Store**: it is a resource of the argumentation framework that stores all the information about the agents participating in the problem-solving process, argumentation dialogues between them, positions and arguments. By making queries to this resource, every agent can read the information of the dialogues that it is involved in. It has been implemented as a Magentix2 Base-Agent to allow a good communication with the agents.

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

1) At the beginning, some argumentation agents run in the Magentix2 agent platform. One of these agents is chosen randomly to be the *initiator* of the dialogues. The initiator agent is in charge of receiving the tickets or incidences to solve and create a new dialogue with the agents in the platform. The dialogue begins when the initiator agent sends the ticket to the other agents with the *open dialogue* locution.

2) Agents receive the *open dialogue* locution with the ticket to solve and they evaluate if they can engage in the dialogue offering one or more solutions. In that case, each agent will enter in the dialogue with the locution *enter dialogue*. Then, the agent will propose a position (a solution for the problem).

3) When agents have a position to defend, the argumentation dialogue begins. The position of each agent is stored by the commitment store agent. Thus, other agents can check the positions of all dialogue participants. Each agent will try to attack every different position from its own. The agents will build support and attack arguments to defend their position and attack the other agents’ position, as it is explained in the previous section. The agents will use the following locations during the argumentation dialogue: *why, assert, attack, accept, retract* and *no_commit*.

4) The dialogue finishes when no new positions or arguments are proposed after a certain time. Then, the initiator agent retrieves the active positions of the participants and the most frequent is selected as the final solution to propose. In case of draw, the final solution will be the most accepted position by other agents during the argumentation dialogue.

### IV. Evaluation: CBR vs CBR Argumentation

In Sections II and III we have presented an argumentation framework and a prototype of a customer support application that uses case-based argumentation in MAS. In this section we make an evaluation of the prediction error between a simple CBR approach and the case-based argumentation approach explained before.

Agents have an individual argumentation system that implements the case-based argumentation framework presented before. The domain-cases case-bases of each agent are populated randomly by using some of the 48 cases of a case-base of computer problems, increasing the number of cases from 5 to 40 cases in each round. Each problem is described by a set of features (e.g. the type of problem, the log provided by the system etc.) and the description of the solution applied. To diminish the influence of random noise, all results report the average of 48 simulation runs per round. In each round, an agent is selected randomly as initiator of the process. This agent has access to the whole case-base of computer problems and in each run takes the corresponding case to solve and sends it to the other agents. In this way, the initiator knows which was the real solution applied to the problem and can compare this value to the solution decided by the agreement process.

To make the evaluation the tests have been performed with the following decision policies:

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

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

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

---

\(^1\)http://users.dsic.upv.es/grupos/ia/sma/tools/magentix2/index.php

\(^2\)http://users.dsic.upv.es/grupos/ia/sma/tools/magentix2/archivos/javadoc/es/upv/dsic/giu_ia/core/BaseAgent.html
In all decision policies, agents propose solutions using its own CBR. So, an agent will be able to propose a solution if in its CBR there is a case that match with the ticket to solve.

In the tests, we evaluate the average error in the prediction of the best solution to apply with regard to the size of the case-bases of domain-cases of the agents. For the first test, shown in Figure 2, we consider a group of 3 operators. As expected, all policies have better results as the number of cases grows. With 5 and 10 domain-cases, all policies obtain the same results. This is because with 3 agents and the reduced number of domain-cases, the agents are not defending different positions, and there is no argumentation dialogue. In other rounds (from 15 to 25 domain-cases), CBR-Random and CBR-Majority policies obtain the same results, but the CBR-Argumentation policy has better results, since agents can argue to decide the best solution among those proposed. Thus, the argumentation techniques improve the results of the application. Nevertheless, the results obtained by different policies are quite similar because there are only 3 agents, and normally they are defending the same positions.

For the third test we consider a group of 6 operators and 1 expert. In this test, shown in Figure 4, we evaluate again the average error in the prediction of the best solution to apply. As it has been explained in Section III, the expert can impose its arguments because it has a priority dependency relation over operators. However, in the random and the CBR-Majority policies there is no argumentation dialogue, so this dependency relation is not taking effect, but the proposals of the expert have the same influence than other operators proposals in the final solution selected. In the runs between 5 and 25 domain-cases in the agents’ case-bases, the expert has 26 domain-cases to simulate that it has more specialised knowledge about the considered. The main difference that we can observe for all policies is that the prediction error is quite lower than with a group of 3 operators. This is because with more operators there is more knowledge distributed among the agents and it is more probable to find out the correct solution for the problem at hand. As expected, all policies have better results as the number of cases grows. In this test, from 30 to 40 domain-cases there is some error with CBR-Random policy that in the first test did not appeared. The reason of this results is that the CBR-Random policy randomly chooses a solution among those proposed by the agents, and with more agents is more probable to choose an incorrect solution. Also, the CBR-Argumentation policy results are equal or better than the other policies’ results. In the results obtained with 25 domain-cases, we can see that CBR-Majority policy has some error and with 20 domain-cases has not any error. This is happening because in that case, there is more knowledge than with 20 domain-cases in the case-bases of the agents, but some agents might have domain-cases that, although matching to certain extent the problem description, report low quality solutions for that problem. As this policy chooses the most frequent solution proposed as a final solution, in that case some agents could propose an incorrect with enough frequency to be selected as the best solution to apply. Also, when the agents have 30 domain-cases they have learned the correct solutions and the prediction error decreases again.
best solution to propose for each problem. Thus, the expert agent has more probabilities of proposing the correct solution for the problems.

As we can observe in Figure 4, the general prediction error is much lower than in the first and second tests. This is because the expert agent has more knowledge (26 cases) in the runs between 5 and 25 domain-cases per agent. In addition, the prediction error of the CBR-Argumentation policy is quite lower than the others in some rounds. The reason of that improvement in the results is that the expert is providing the best solution that it knows and imposing its opinion about which is the best solution to apply and, since it has more knowledge, the probability of selecting the best solution increases. Also, the CBR-Random policy produce more error with 35 domain-cases than with 30. This is happening because the CBR-Random policy chooses randomly a solution of those proposed, and depending on the solutions proposed this policy can choose a suitable solution or not.

V. CONCLUSIONS

This paper presents a customer support application based on a case-based argumentation framework for MAS. This application improves the performance of a simple helpdesk with CBR since the decision about the solution to apply for an incidence (ticket) is made by an agreement between agents after an argumentation dialogue. Apart of the advantages that a MAS can offer in matters of information distribution, the argumentation framework used in the application improves the results obtained since the error obtained is lower.

The customer support application proposed in this work would be very useful integrated in a helpdesk support system of a company. On the one hand, this application is following a distributed approach with a MAS representing the operators of a call centre. This approach fits well on the structure of a call centre and allows to have the data distributed among agents. On the other hand, the agents (representing the operators of the call centre) try to obtain an agreement about the best solution to apply for an incidence after an argumentation dialogue. Therefore, this approach simulates a suitable way to make a decision between operators about the best solution of an incidence.

Finally, the results obtained by the customer support application using argumentation in MAS presented are better than the previous approaches since the prediction error obtained in different tests performed is lower.

ACKNOWLEDGEMENT

This work is supported by the Spanish government grants CONSOLIDER INGENIO 2010 CSD2007-00022, TIN2008-04446 and TIN2009-13839-C03-01 and by the GVA project PROMETEO 2008/051.

REFERENCES