Arguing About Task Reallocation Using Ontological Information in Multi-Agent Systems

Alison R. Panisson, Artur Freitas, Daniela Schmidt,
Lucas Hilgert, Felipe Meneguzzi, Renata Vieira, and Rafael H. Bordini

Pontifical Catholic University of Rio Grande do Sul – PUCRS
Postgraduate Programme in Computer Science – School of Informatics (FACIN)
Porto Alegre – RS – Brazil
{alison.panisson,artur.freitas,daniela.schimidt}@acad.pucrs.br,
{lucas.hilgert,felipe.meneguzzi,renata.vieira,rafael.bordini}@pucrs.br

Abstract. Argumentation-based dialogue is claimed to be a rich form of agent interaction, which allows agents to make informed decisions using the additional information that is exchanged as arguments during the dialogue. In this paper, we demonstrate how agents can use domain specific knowledge provided by access to ontological information in the process of participating in argumentation-base dialogues. We also demonstrate how information obtained in a dialogue can be used by the agents to argue in subsequent dialogues. As an example scenario, we refer to an assisted living application where agents argue about task reallocation. We then define an agent decision-making process and a protocol; the decision-making and the protocol for argumentation-based dialogues are defined for cooperative agents in our scenario (i.e., task reallocation).

Keywords: Argumentation-based Dialogues, Task Reallocation, Multi-agent Systems

1 Introduction

Argumentation has received significant interest in the Multi-Agent System (MAS) community in recent years. The two main lines of research in the multi-agent community regarding argumentation are [17]: i) argumentation focused on reasoning (nonmonotonic reasoning) over incomplete, conflicting, or uncertain information, where agents construct arguments for and against certain conclusions (beliefs, goals, etc.) of their own; and ii) argumentation focused on communication/interaction between agents that allows the exchange of arguments to justify a stance and to provide reasons that defend claims made by individual participants.

The main claim appearing in the literature about the benefits of an argumentation-based approach to communication is the fact that this type of interaction allows agents to reach agreements in situations where other approaches would not (for example, in negotiation, where argumentation is compared to game-theoretic and heuristic-based approaches [27]). Such benefits are the consequence of the exchange of additional information (arguments) which allow agents to make more informed decisions and comparisons.

In this work, we explore argumentation in the communication aspect of its uses in MAS, where we model a decision-making process and an argumentation-based dialogue protocol for the reallocation of tasks between agents. In our approach, agents use information provided by an ontology that contains domain-specific information about the tasks. This approach is interesting because it allows the modularisation of applications, separating the domain-specific knowledge, which will be common to all agents in a given application.

Ontologies empower the execution of semantic reasoners, such as Pellet [31], which provide functionalities such as *consistency checking*, *concept satisfiability*, *classification* and *realisation*. Ontologies also allow people and software agents to share a common understanding of the structure of the available information and the reuse of domain knowledge.

The contribution of this paper is threefold. First, we define an ontology of tasks (the domain-specific knowledge in our case study) and agents query this information from within a MAS that runs distributed over a network. Second, we define a decision-making process using the information obtained from the ontology. An agent makes its decisions based on ontology queries and the results of the dialogues in which it participates; although in our framework dialogues with multiple agents is possible, in this work agents always engage in dialogues in pairs. Third, we define a protocol for argumentation-based dialogues between cooperative agents for task reallocation, where agents discuss over an assertion in order to decide whether it can be accepted by both, by none, or whether the agents have different preferences and therefore cannot reach an agreement about that assertion. The results of such dialogues are used in the decision-making processes.

The remainder of this paper is organised as follows. In the next section, we discuss briefly some related work. Then we describe the approach used to access ontological information, the scenario used to exemplify the remainder of the paper and the domain-specific ontology. Next, we describe a decision-making process and an argumentation-based dialogue protocol which are used by the agents in our domain. Then we exemplify our approach by discussing a case study. In the final section of this paper, we make some final remarks and discuss possible directions for future work.

2 Related Work

In argumentation-based dialogues, agents use speech-acts to exchange arguments. Speech acts for this purpose have long been discussed in the relevant literature [1, 24, 25]. Some of these speech-acts have actually been formalised in agent-oriented programming languages, for example in our previous work [22]. In that work, the changes by sending and receiving speech acts reflect in the MAS as whole, where several *dialogues* can be occurring simultaneously among different sets of agents. Our work uses a subset of the performatives found in the literature. We followed the literature using a structure that maintains the public knowledge that the agents introduce in the dialogue — the *commitment stores* [25]. Further, we define a protocol using the definition of "moves"[1], and agents take turns in making such moves, as in [13].

Some protocols for argumentation-based dialogues can be found in the literature. In [2], the authors present a protocol called PARMA which defines the rules for argumentation-based persuasive dialogue over actions, where the participants rationally propose, attack, and defend an action or course of action. In [4] a protocol for conflict resolution between agents is proposed; in that protocol, the generation and evolution of arguments apply assumption-based argumentation [10]. Black et al. [6] propose a dialogue-game inquiry protocol that allows two agents to share knowledge in order to construct an argument for a specific claim. In [8], a protocol is introduced in a declarative way, determining which speech acts are legal in a particular state, and this model is used to analyse a formal disputation. In our work, we define a protocol for argumentation-based dialogues between cooperative agents, where the agents discuss about a certain topic (the first assertion made in the dialogue). The dialogue ends when the agents agree about the first assertion, agree about the complement of the first assertion, or otherwise the dialogue eventually ends in disagreement. We also use declarative rules to define which moves are allowed after one another. This protocol is used in a decision-making process that obtains the required knowledge from a domain ontology.

Approaches to integrate ontology information with agent-oriented programming languages can be found in work such as AgentSpeak-DL [19] which extends agents' belief base with *description logic*; JASDL [15] which is an AgentSpeak-DL implementation that extended Jason to provide agents with ontology manipulation capabilities using the OWL API; CooL-AgentSpeak [16] which is an extension of AgentSpeak-DL with plan exchange and ontology services. The CooL-AgentSpeak implementation uses a CArtAgO artifact functioning as an ontology repository tool which stores a possibly dynamic set of ontologies and offers services such as ontology matching/alignment. In our work too a CArtAgO artifact is used to implement an interface between ontologies and MAS. Using this approach, the agents can share ontologies that are available to them, while in CooL-AgentSpeak agents do not share knowledge from their ontologies. This is an important aspect of our work, where we intend to use the ontology as a knowledge repository from which agents can query specific domain knowledge that is common to all of them.

[1] The name "moves" is from game-theoretic approaches to agent argumentation where a dialogue is treated as an adversarial game. In that context, each *move* corresponds to a communicative action made by an agent [17]

3 Accessing Ontological Information

To provide access to ontological information (i.e., domain-specific knowledge developed by a knowledge engineer) in a MAS, we developed a CArtAgO [29] artifact. CArTAgO is a platform that provides MAS with support to the notion of artifacts. Artifacts are function-oriented computational abstractions which provide services that agents can exploit to support their activities [28]. An artifact makes its functionalities available and exploitable by agents through a set of operations and observable properties. Operations represent computational processes executed inside artifacts, which can be triggered by agents or other artifacts. Observable properties are artifact attributes that are directly mapped into the belief base of the agents that observe (i.e., focus on) an artifact.

Our artifact uses the OWL API, which is an open source Java API [12], for creating, querying, manipulating, and serialising ontologies coded in OWL (Web Ontology Language). These functionalities are made available to the agents through a set of operations such as *load the ontology*, *add instances* and *add concepts*, for example. In our work, we make use the following operations in particular:

- **isInstanceOf** (instance, concept)
  - checks whether the instance belongs to the given concept, returning a boolean value.
- **getInstances** (instance, property)
  - returns the instances (Set<OWLNamedIndividual>) that are targeted by the given instance and property.

As a design and implementation decision, each instance of the proposed artifact can load and encapsulate exactly one OWL ontology. However, each workspace can have any number of instances of this artifact, where each instance makes reference to an ontology, and the agents in the same workspace of the artifacts can observe and manipulate any number of such artifacts. Thus, each MAS using this type of artifact can handle multiple ontologies, all shared by the agents that enter the workspace where those artifact instances are located.

Using an artifact to access information from ontologies is an alternative to the approach of representing all the knowledge in platform-specific mechanisms such as the belief base of an agent, for example. However, agents can still use their regular knowledge representation approach simultaneously with the information provided by the ontology (or completely replace the native approach to knowledge representation).

4 Scenario and Ontology Description

The scenario used here is the development of a complex software system, in particular an assisted-living application to provide functionalities such as activity recognition and task reallocation among agents representing human users through the use of planning, agent and semantic technologies. More specifically, this multi-agent application is designed to provide the following functionalities for its users:

- allocate tasks and commitments considering the context of patient care;
- detect if the person responsible for each of the activities of the elderly patient is following their scheduled appointments/commitments;
- detect problems that may prevent the person in charge to attend to their obligations;
- reallocate tasks among users whenever required/possible (using an argumentation-based approach);
- send reminders for users to monitor the patient's schedule.

Among these characteristics of the application, we focus on the task reallocation. In particular, we demonstrate agents using information provided by an ontology to engage in argumentation-based dialogues about task reallocation. We only briefly explain the remainder of the application to provide an overall understanding of the application as a whole. Considering this scenario, we created an ontology to represent collaborative tasks that correspond to caring for an elderly person and the interactions with the members of the group of people who care for the elderly person (the members of the extended family of that person plus some professional carers). The ontology allows agents to reason and query information about the typical tasks and commitments of that group of people.

4.1 Task Ontology

We use a task ontology to represent task-related knowledge, such as location, temporal characteristics, and execution properties. It gives the information of who is involved in each type of task execution, where the tasks normally take place, when they happen, what changes in the environment they cause, what is required for their execution, and so on. Agents can use this information to make decisions at various moments, for example during the execution of plans to achieve some goal and in the reallocation of tasks among agents. Logical rules and semantic reasoners may be applied over the ontology to infer new knowledge about tasks. Such knowledge may be required by agent programmers to implement task reasoning mechanisms, such as techniques for task recognition, allocation, and negotiation.

Figure 1 shows the main concepts, properties, and examples of instances in our task ontology. Its main concepts are *Task*, *TaskPurpose*, *Person*, *Location*, *Object*, and *TimeInterval*. A *Task* may contain restrictions based on the locations where it can happen, as represented in the *Location* concept (e.g., *InternalLocation* and *ExternalLocation*). The *Person* concept includes information that a *Task* might be assigned to any number of persons that will execute it or participate in its execution. *Persons* can be subdivided according to the necessity to specify *Task* restrictions, e.g. *Adult* is a subclass of *Person*, which can be used to specify tasks that only adults can execute. The *Object* concept represents the objects involved in the task execution; *TaskPurpose* represents the task classification according to their speciality (e.g., entertainment, personal hygiene, etc.); and *TimeInterval* helps modelling tasks that have particular temporal characteristics, such as a specific time when they might happend.

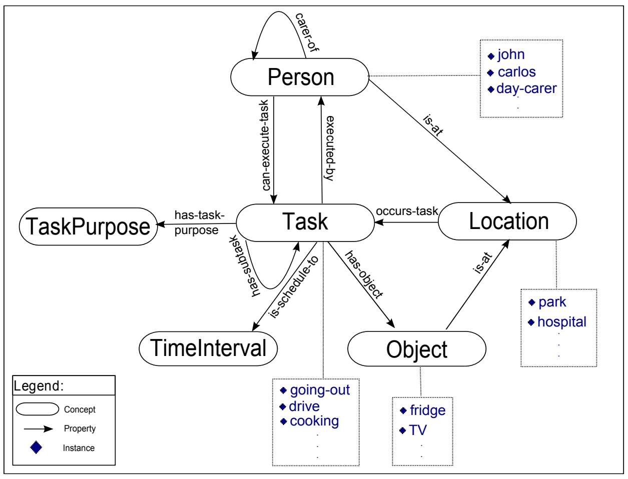

Fig. 1. Overview of the Task Ontology.

The *Task* concept is specialised in various dimensions to address the task representation needs for collaborative groups. For example, in terms of its *complexity*, a *Task* may be classified as *SimpleTask* or *CompositeTask* (if it is divided into other tasks through the *has-subtask* property). Also, a *Task* can be reallocatable between members of the group and/or temporally, as addressed by the *ReallocatableTask* concept, that considers the *ReallocatableResponsible* and *ReallocatableTime* subconcepts. In our approach, a *Task* can be reallocated in two occasions: (*i*) when a person cannot execute it at the scheduled time — in this case, a *ReallocatableTime* task will be reallocated to a different time, but it will be assigned to the same person; and (*ii*) when a *ReallocatableResponsible* task cannot be reassigned to the same person at a different time — in this case, the *Task* has a property called *can-be-reallocated-to* to identify the persons who can execute it. Information about *Persons* may be used to define characteristics that people must have in order to perform certain tasks, such as *AdultTask*, that represents tasks that only adults can perform (e.g., tasks that involve driving a car). Also, when a *Task* is assigned to a *Person* and assisted by another, it is classified as an *AssistedTask*.

In OWL, a class C can be declared with certain conditions (i.e., every instance of C has to satisfy those restrictions, and/or every instance that satisfies those restrictions can be inferred as belonging to C). OWL class restrictions [3] can be defined by elements such as cardinality and logic restrictions (e.g., the *universal* and *existential quantifiers*). In our ontology, the concepts were defined based on a series of restrictions and other logical characteristics, e.g. the *CompositeTask* concept is equivalent to a task that has *sub-tasks*, and a *SimpleTask* is equivalent to a task (a subclass of *Task*) that is not a *CompositeTask* (i.e., it has no sub-tasks). These restrictions allow task recognition to be conducted by means of the classification capability by semantic reasoners. These concept definitions use the existential quantifier and negation, as follows:

$$CompositeTask \equiv Task \sqcap \exists has\text{-}subtask.Task$$

$$SimpleTask \equiv Task \sqcap \neg CompositeTask$$

5 Arguing About Task Reallocation

Using the information provided by the ontology artifact in their decision-making process, the agents decide if a particular task can be transferred to another time slot, as well as which persons (members of the group using the application) can execute this particular task if it cannot be postponed. In the latter case, it is argued with the group members the reallocation of the task to one of them. As previously described, the scenario corresponds to a MAS designed to monitor and support the execution of tasks in an assisted living context. The agents have the goal of helping the users to conclude the tasks for which they are responsible. For example, if an agent detects a failure that could negatively impact the completion of a task, it can try to find an alternative manner to conclude that task (transferring temporally or transferring to another person who would be able to execute it to successfully). The knowledge of this domain is described in an ontology designed by a knowledge engineer (in our example, the task ontology), and can be reused in other applications, or in some cases it is even possible that the application (i.e, the MAS) can be used in another domain by simply changing the domain ontology.

5.1 Speech Acts for Argumentation-based Communication

In our work, agents argue about task reallocation using a subset of the speech-acts found in the literature on argumentation-based dialogue [1, 24, 25]. The speech-acts used and the informal meaning are:

- *assert:* an agent that performs an assert utterance declares, to all participants of the dialogue, that it is committed to defending this claim. The receivers of the message become aware of this commitment.
- *accept:* an agent that performs an accept utterance declares, to all participants of the dialogue, that it accepts the previous claim of another agent. The receivers of the message become aware of this acceptance.
- *question:* an agent that performs a question utterance desires to know the reasons for a previous claim of another agent. The receiver of the message is committed to defending its claim, and provides the support set for it.

- *justify:* the justify message is similar to the assert message, and it is the response to the question message previously uttered, where the agent provides the support to its previously claim uttered.
- *retract*: the agent declares, to all participants of the dialogue, that it is no longer committed to defending its previous claim. The receivers of the message become aware of this fact.

The formal semantics of some these speech acts/performatives for agents based on the BDI architecture (such as Jason [7] agents) is found in our work [22] which specifies the exact effects of sending and receiving such speech acts in the agent's mental state, as well as in the MAS as a whole. We use that semantics in our work so we refer the reader to [22] for details of the semantics formalisation.

In addition to the speech acts presented above, the agents exchange messages to start and to end the dialogues. We do not present that simple message exchange, so we assume that any protocol can be used to start a dialogue, and we further assume that the dialogue ends when the agents reach an agreement or no agent can execute further moves.

5.2 Rules to Update the Commitment Store

The Commitment Store (CS) is a data structure accessible to all agents in a dialogue; it contains, for each agent, all the commitments made by that agent during the dialogue (the CS is sometimes also called *dialogue obligation store* [18] and *dialogue store* [30]). The CS of an agent is simply a subset of its knowledge base, and the union of the CSs can be viewed as the state of the dialogue at a given time [25].

In the course of the dialogue, the agents use rules that define how the CSs are updated. These rules are implicit in the semantic definition used in this work and presented in [22], and can be summarised as follows:

- *assert*: the agent's CS is updated with the content asserted (assume the content is a formula p): $CS = CS \cup \{p\}$;
- *accept*: the agent's CS is updated with the accepted content p: $CS = CS \cup \{p\}$;
- *question*: has no effects over the CS;
- *justify*: the agent's CS is updated with the justification content (a support set of rules and facts S): $CS = CS \cup S$;
- *retract*: the agent's CS is updated removing the retracted content p: $CS = CS \setminus \{p\}$.

5.3 Argument Generation and Evolution

We assume that our agents have an internal rule-based argumentation mechanism capable of generating and evolving argument positions. We have implemented an argumentation-based reasoning mechanism [23] in Jason Platform [7] adapting d-Prolog [20] (an implementation of defeasible logic formalism [21]). We will consider our previous work as reasoning mechanism, where we will present the basic of this work to the understanding of the remaining of the document and we refer to [23] for the full work. Although we use our previous work as reasoning mechanism, others work in literature can be used, as the approach of Berariu [5], which implements a decoupled module in Jason Platform based in the work of Prakken [26].

In our approach, when an agent needs an argument (fact and rules used in the derivation of a *content*), this information is accessible through of queries in its belief base, returned as a list unified in the parameter which we call *Arg*. We store each rule and fact, used in the derivation, using the internal action[2] `.concat` (which concatenates a list with the new element – a rule or fact). Thus, depending on the strategy of the agent, it can verify if it has a strict or defeasible argument, using `strict_der(Arg,Content)` and `def_der(Arg,Content)`, or if this distinction is not necessary, the agent can use the predicate `argument(Content,Arg)`.

```
argument(Content,Arg):- strict_der(Arg,Content)
                        | def_der(Arg,Content).
```

This implementation has a well-defined semantics called *defeasible semantics* [11], which defines the acceptability of the arguments. This semantics is unique[3] and it is compared with the *grounded semantics* of Dung [9] in [11].

Regarding our scenario, we will present an simple argumentation-based reasoning example, which the agent use to decide about do not execute a task. We will suppose that an agent, named *ag*, is committed to execute a task, named *t*1, but by any reason he cannot execute this task. The agent *ag* believes that the task *t*1 can be transferred, so it believes that the task *t*1 not need be executed.

```
defeasible_rule(¬execute(ag,t1),can_be_transferred(t1)).
can_be_transferred(t1).
```

The plan to cancel the agent commitment to execute the task *t*1 has the following format (in Jason platform):

```
+!cancel_task(t1): argument(¬execute(ag,t1),Arg))
                <- cancel_task(t1,Arg).
```

Before the agent to cancel its commitment with the task *t*1, it is informed that the patient (centralized person of the application, which the others members take care) needs that the task *t*1 to be executed by some reason. The reason is dependent of the task, in this case we will consider that the patient has pain and the task is take the patient to the physiotherapy.

```
strict_rule(execute(ag,t1),has_pain(patient)).
has_pain(patient).
```

[2] All internal actions available in Jason Platform can be found in [7].
[3] It generates only one set of acceptable arguments.

The new information changes the conclusion of that the task not need to be executed by agent *ag*, and so the above plan no longer applies.

In the remain of the paper we assume that the agents only accept propositions/claims for which they do not have an acceptable argument against (i.e., the so called *cautious* attitude [24, 25]), and agents assert propositions/claims which they have an acceptable argument for (i.e., the so called *thoughtful* attitude [24, 25]).

5.4 Decision-Making for Task Reallocation Using Ontological Information

Decision-making can be seen as a process whereby an agent looks for the information available to it in order to decide which course of action to take [14]. In this section, we describe our agent decision-making process for task reallocation, which uses information provided by the ontology described earlier in this paper. It is important to note that the tasks are assigned to users and not to agents, therefore the agents argue about the reallocation to their users, using the information available to them.

The process of task reallocation starts when the system detects that a user could not execute a task because of a particular problem (e.g., the user is late, or the system recognises that the user cannot execute the task because another more important task/commitment was created with conflicting times, etc.). Then the system generates an event which is treated by the agent responsible to help that particular user. The decision-making process proceeds as follows:

- **Step 1**: The user agent checks if the task is temporally reallocatable (this is domain-specific knowledge and is provided by the ontology, accessed by the agent using the artifact). If that is the case, the agent tries to reallocate the task to another time slot, and the decision-making process goes to Step 2. In case the task is not temporally reallocatable, the user agent tries to reallocate it to another user of the application — the process goes to Step 4.
- **Step 2**: The user agent starts a dialogue with the agents representing those involved in the task [4], suggesting that the task be transferred to another time slot (following the protocol for argumentation-based dialogue that we introduce in the next section). There are three possible results to this dialogue: (i) the dialogue ends with agreement about transferring the task to another time slot — the decision-making process goes to Step 3; (ii) the dialogue ends with agreement about not transferring the task to another time slot; and (iii) the agents cannot reach an agreement about postponing the task. In both the last two cases the decision-making process goes to Step 4.

[4] In our domain, generally, the task has two people involved, the person responsible for the task (who will usually try to reallocate it if needed) and the elderly family member who requires constant care.

- **Step 3**: The user agent informs the user that the task has been transferred to another time slot. Further, the elder user (or other participants of the dialogue) are also informed about the reallocation. The process ends.
- **Step 4**: The user agent checks which other members of the group of users are allowed to execute the task (this information is also provided by the ontology). If the list of members that can execute the task is empty, then the decision-making process goes to Step 7; otherwise, it goes to Step 5.
- **Step 5**: The user agent selects one member of the list of members who can execute the task and starts a dialogue with the agent of that user, suggesting that the selected member executes the task instead (following our argumentation-based dialogue protocol). As before, there are three possible results to this dialogue: (i) the dialogue ends with agreement about the selected member executing the task — the process goes to Step 6; (ii) the dialogue ends with agreement that the selected member cannot execute the task either; and (iii) the agents cannot reach an agreement about the selected member executing or not the task. In the last two cases, that selected member is removed of the local list of members that can be asked to execute the task and the process goes back to the beginning of Step 5 if the list is not empty, and to Step 7 otherwise.
- **Step 6**: The user agent informs the user about the suggestion of the system (to reallocate the task to that selected person) and waits for confirmation from the other member. If the other member accepts the suggestion (confirms that they will execute the task), the user is informed and the process ends. If the other member rejects the suggestion, that member is removed of the local list of the members who can be asked to execute the task, the user is informed, and the decision-making process goes to Step 5.
- **Step 7**: The user agent informs the user that the task can be neither postponed nor transferred to another member of the group. The user will have to personally make the necessary decisions using the provided information. The process ends.

5.5 Protocol for Task Reallocation Using Argumentation

Next, we define a protocol for argumentation-based dialogues between cooperative agents for task reallocation, where the agents exchange arguments seeking an agreement about the initial assertion (the subject of the dialogue); i.e., the task being postponed or another person taking charge of it.

- **Step 1**: The dialogue starts with one agent executing the move *assert* containing the subject of the dialogue. The protocol goes to Step 2.
- **Step 2**: The agent receives an *assert* message and checks if this assertion is acceptable to it (i.e., the agent does not have an argument against this assertion). If the agents accept the previous assertion (making the *accept* move) the protocol goes to Step 5. Otherwise, in case the assertion is not acceptable, the agent executes a *question* move and the protocol goes to Step 3.

- **Step 3**: The agent receives a *question* message and provides the justification to its previous assertion (*i.e.*, the agent provides the (defeasible) proof that allowed its deduction of that claim) using the *justify* move. The protocol goes to Step 4.
- **Step 4**: The agent receives a *justify* message and checks if this new information changes its conclusion about the acceptance of the previously questioned assertion. In the case where the new information changes the agent's conclusion (i.e., the previous assertion is, now, acceptable) the agent executes the *accept* move and the protocol goes to Step 5. Otherwise, the agent either questions something in the justification and the protocol goes to Step 3, or it executes the *justify* move hence not accepting the previous assertion and the protocol goes to Step 4.
- **Step 5**: The dialogue ends in one of three possible results: (i) the agent that started the dialogue accepts the arguments of another agent, and it retracts its initial assertion because it now agrees that the initial assertion is not acceptable; (ii) the agents that received the initial assertion accept either the arguments or the initial assertion directly, because they now agree that the initial assertion is acceptable; or (iii) none of the agents can execute more moves. In all cases the protocol goes to step 6.
- **Step 6**: The agent that started the dialogue closes it.

6 Example

Back to our assisting-living scenario, suppose the task of taking the elder person to a physiotherapy session, under the responsibility of John (i.e., the task involves the elderly person being cared for, we will call him Carlos). At a certain moment in time the application recognises that John will not execute the task (*e.g.*, a work-related meeting has just been scheduled to a conflicting time). Then the system generates an event to John's agent and the process continues as described below:

John's agent queries the ontology using the operation *isInstanceOf* to verify if the task to *take to physiotherapy* is temporally reallocatable (i.e., it is an instance of the *ReallocatableTime* concept). In our case, this returns *true* and the agent, following the decision-making process for task reallocation, starts a dialogue with Carlos's agent. Initially John's agent uses the *assert* move, stating that John will not execute the task (this assertion, following the *thoughtful* attitude assumed, is possible because John's agent has an acceptable argument which uses the defeasible rule that if a task is temporally reallocatable the user does not need to execute the task at that time).

Carlos's agent receives the *assert* message and checks if it can accept that John does not execute the task. Carlos's agent has the information that Carlos is in great pain that particular day, and has a strict rule that if Carlos is in more pain than usual, then John needs to take Carlos to the physiotherapy (i.e., the task cannot be postponted as usual). With this information, Carlos's agent cannot accept this assertion and executes the *question* move to gather the reasons why John cannot execute the task.

John's agent receives the *question* message and as the agent is committed to defending its assertion (the content of the previous assertion is in its CS) it executes the *justify* move explaining that, as the task can be transferred, John does not need to execute the task.

Carlos's agent receives the *justify* message and checks if the new information changes its conclusion that John needs to execute the task. As this new information does not change its conclusion, Carlos's agent executes the *justify* move, informing that John needs to execute the task because Carlos is in pain.

John's agent receives the *justify* message and checks if the new information changes its conclusion about not executing the task. As this information is strict and cannot be defeated by other arguments, John's agent accepts (executing the *accept* move) that the task needs to be executed, hence executing also a *retract* move and then closes the dialogue.

After closing the dialogue with Carlos's agent, and concluding that the task cannot be postponed, John's agent, following the decision-making process for task reallocation, tries to reallocate the task to another member of the group.

John's agent queries the ontology about the members of the group who can execute this particular task, using the artifact operation *getInstances*. The operation returns a list of all members who are adults since the *take to physiotherapy* task has a relation *can-be-relocated-to* with the class *Adult*. In our example, we will assume that the members who can execute the task are John and Jane (John's wife). Then John's agent starts a dialogue with Jane's agent suggesting that Jane executes the task (using the *assert* move).

Jane's agent receives the message but it has an, as yet, acceptable argument for Jane not to execute the task (the information that the task can be transferred to another time). Then, Jane's agent executes the *question* move, because it cannot accept the assertion.

John's agent receives the question message and executes the *justify* move, sending the information that Carlos is in pain and John has had an urgent meeting just scheduled, hence the request to reallocate the task to Jane.

Jane's agent receives the justify message, and checks if this new information changes its conclusion about whether to execute the task. In our case, as the inference for when Carlos is in pain is strict compared to inferences about postponing tasks, Jane's agent has no argument against executing the task and accepts the argument (using the *accept* move).

Jane's agent asks her to confirm if she is willing to execute the task instead of John, and Jane confirms that she really can do that at that time. John's agent closes the dialogue and informs him that the task of taking Carlos to physiotherapy has been transferred to Jane. The process ends.

7 Final Remarks

Argumentation-based approaches to communication in MAS provide several advantages, as demonstrated in this work. The exchange of arguments allows the agents to obtain more information about the position of the other agents in the dialogue, and this information can be used to make informed decisions.

In this work, we demonstrated that information obtained by agents in an argumentation-based dialogue can be used in subsequent dialogues, leading to outcomes that could not be achieved without such information. Another contribution of this work is the use of information provided by ontologies in a decision-making process and in an argumentation process as well, using a CArtAgO artifact to facilitate the access to the ontologies. The integration of agent platforms with ontologies enables the modularisation of knowledge, where domain specific knowledge can be accessed from an ontology, and this information can be reused for any application in that domain.

Furthermore, our contribution includes the definition of a decision-making process for task reallocation which uses the outcomes of argumentation-based dialogue in such process. We have also introduced a protocol to be used for such argumentation-based dialogues for cooperative agents reallocate tasks.

As future work, we intend to explore the use of other argumentation-based dialogues protocols in the decision-making process. In the development of the application described above, we have modularised the decision-making process, the speech-acts effects formalisation presented in [22], the dialogue protocol, and agents' *strategy* in participating in dialogues (i.e., the agent *attitudes* [24, 25]). Therefore, our approach allows for individual modules to be replaced and its effects tested separately.

Acknowledgements

Part of the results presented in this paper were obtained through research on a project titled “Semantic and Multi-Agent Technologies for Group Interaction”, sponsored by Samsung Eletrônica da Amazônia Ltda. under the terms of Brazilian federal law No. 8.248/91.

References

1. Amgoud, L., Maudet, N., Parsons, S.: Modeling dialogues using argumentation. In: ICMAS. pp. 31–38. IEEE Computer Society (2000)
2. Atkinson, K., Bench-Capon, T.J.M., McBurney, P.: A dialogue game protocol for multi-agent argument over proposals for action. Autonomous Agents and Multi-Agent Systems 11(2), 153–171 (2005)
3. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference. Tech. rep., W3C (February 2004)
4. Bentahar, J., Alam, R., Maamar, Z.: An argumentation-based protocol for conflict resolution. In: In KR2008 - Workshop on Knowledge Representation for Agents and MultiAgent Systems (KRAMAS 2008) (2008)
5. Berariu, T.: An argumentation framework for bdi agents. In: Zavoral, F., Jung, J.J., Badica, C. (eds.) Intelligent Distributed Computing VII, Studies in Computational Intelligence, vol. 511, pp. 343–354. Springer International Publishing (2014)

6. Black, E., Hunter, A.: A generative inquiry dialogue system. In: Durfee, E.H., Yokoo, M., Huhns, M.N., Shehory, O. (eds.) AAMAS. p. 41. IFAAMAS (2007)
7. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology). John Wiley & Sons (2007)
8. Brewka, G.: Dynamic argument systems: A formal model of argumentation processes based on situation calculus. Journal of Logic and Computation 11(2), 257–282 (2001)
9. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence 77, 321–357 (1995)
10. Dung, P.M., Kowalski, R.A., Toni, F.: Assumption-based argumentation. In: Argumentation in Artificial Intelligence, pp. 199–218. Springer (2009)
11. Governatori, G., Maher, M.J., Antoniou, G., Billington, D.: Argumentation semantics for defeasible logic. J. Log. Comput. 14(5), 675–702 (2004)
12. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semant. web 2(1), 11–21 (Jan 2011)
13. Kakas, A., Moraitis, P.: Adaptive agent negotiation via argumentation. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 384–391. AAMAS '06, ACM, New York, NY, USA (2006)
14. Kaufman, M.: Local Decision-making in Multi-agent Systems. Ph.D. thesis, Oxford University (2010)
15. Klapiscak, T., Bordini, R.H.: JASDL: a practical programming approach combining agent and semantic web technologies. In: The 6th international workshop on Declarative Agent Languages and Technologies. vol. 5397, pp. 91–110. Springer (2008)
16. Mascardi, V., Ancona, D., Barbieri, M., Bordini, R.H., Ricci, A.: CooL-AgentSpeak: Endowing AgentSpeak-DL agents with plan exchange and ontology services. Web Intelligence and Agent Systems 12(1), 83–107 (2014)
17. Maudet, N., Parsons, S., Rahwan, I.: Argumentation in multi-agent systems: Context and recent developments. In: Maudet, N., Parsons, S., Rahwan, I. (eds.) ArgMAS. Lecture Notes in Computer Science, vol. 4766, pp. 1–16. Springer (2006)
18. McBurney, P., Parsons, S.: Locutions for argumentation in agent interaction protocols. In: van Eijk, R.M., Huget, M.P., Dignum, F. (eds.) AC. Lecture Notes in Computer Science, vol. 3396, pp. 209–225. Springer (2004)
19. Moreira, A.F., Vieira, R., Bordini, R.H., Hübner, J.F.: Agent-oriented programming with underlying ontological reasoning. In: Proceedings of the 3rd international workshop on Declarative Agent Languages and Technologies. pp. 155–170. DALT'05, Springer-Verlag, Berlin, Heidelberg (2006)
20. Nute, D.: Defeasible Prolog. Research report (University of Georgia. Artificial Intelligence Programs), Artificial Intelligence Programs, University of Georgia (1993)
21. Nute, D.: Defeasible logic. In: Handbook of Logic in Artificial Intelligence and Logic Programming. pp. 353–395. Oxford University Press (2001)
22. Panisson, A.R., Meneguzzi, F., Fagundes, M., Vieira, R., Bordini, R.H.: Formal semantics of speech acts for argumentative dialogues. In: Proceedings of the Thirteenth International Conference on Autonomous Agents and Multiagent Systems. pp. 1437–1438 (2014)
23. Panisson, A.R., Meneguzzi, F., Vieira, R., Bordini, R.H.: An Approach for Argumentation-based Reasoning Using Defeasible Logic in Multi-Agent Programming Languages. In: 11th International Workshop on Argumentation in Multiagent Systems (2014)

24. Parsons, S., McBurney, P.: Argumentation-based dialogues for agent coordination. group decision and negotiation. Group Decision and Negotiation (2004)
25. Parsons, S., Wooldridge, M., Amgoud, L.: An analysis of formal inter-agent dialogues. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. pp. 394–401. AAMAS '02, ACM, New York, NY, USA (2002)
26. Prakken, H.: An abstract framework for argumentation with structured arguments. Argument and Computation 1(2), 93–124 (2011)
27. Rahwan, I., Ramchurn, S.D., Jennings, N.R., McBurney, P., Parsons, S., Sonenberg, L.: Argumentation-based negotiation (2004)
28. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: An artifact-based perspective. Autonomous Agents and Multi-Agent Systems 23(2), 158–192 (Sep 2011)
29. Ricci, A., Viroli, M., Omicini, A.: CArtAgO: an infrastructure for engineering computational environments in MAS. In: Weyns, D., Parunak, H.V.D., Michel, F. (eds.) 3rd International Workshop "Environments for Multi-Agent Systems" (E4MAS). pp. 102–119 (2006)
30. Sadri, F., Toni, F., Torroni, P.: Logic agents, dialogues and negotiation: An abductive approach. In: In Proceedings AISB'01 Convention. AISB (2001)
31. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. Web Semant. 5(2), 51–53 (Jun 2007)