PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
GRADUATE PROGRAM IN COMPUTER SCIENCE

ARGUMENTATION-BASED DIALOGUES FOR TASK REALLOCATION AMONG RATIONAL AGENTS

ALISON ROBERTO PANISSON

Dissertation presented as partial requirement for obtaining the degree of Master in Computer Science at Pontifical Catholic University of Rio Grande do Sul.

Advisor: Prof. Rafael Heitor Bordini

Porto Alegre
2015

Dados Internacionais de Catalogação na Publicação (CIP)

P192a Panisson, Alison Roberto
Argumentation-based diologues for task reallocation among rational agents / Alison Roberto Panisson. – Porto Alegre, 2015.
95 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Rafael Heitor Bordini.

1. Informática. 2. Sistemas Multiagentes. 3. Linguagens de Programação. I. Bordini, Rafael Heitor. II. Título.

CDD 005.13

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "*Argumentation-Based Dialogues for Task Reallocation Among Rational Agents*" apresentada por Alison Roberto Panisson como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 27/03/2015 pela Comissão Examinadora:

Prof. Dr. Rafael Heitor Bordini– Orientador — PPGCC/PUCRS

Profa. Dra. Profa. Dra. Renata Vieira– — PPGCC/PUCRS

Prof. Dr. Felipe Rech Meneguzzi– — PPGCC/PUCRS

Prof. Dr. Antônio Carlos da Rocha Costa– — FURG

Prof. Dr. Álvaro Freitas Moreira– — UFRGS

Homologada em 28/05/2015, conforme Ata No. 015 pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320–3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

I dedicate my work to my family and friends.

ACKNOWLEDGMENTS

I thank the project titled "Semantic and Multi-Agent Technologies for Group Interaction" and all its members for the help and encouragement to my research, in particular, my advisor Rafael H. Bordini. I thank the project sponsorship offered by Samsung Eletrônica da Amazônia Ltda. I thank the support of my family, my friends and my fiancee, both in the good days as well as in the bad days, as surely they are the ones who give us strength to pursue our dreams and goals.

Part of the results presented in this document 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.

DIÁLOGOS BASEADO EM ARGUMENTAÇÃO PARA REALOCAÇÃO DE TAREFAS ENTRE AGENTES RACIONAIS

RESUMO

Argumentação aparece em duas principais linhas de pesquisa no campo de sistemas multi-agentes: raciocínio baseado em argumentação e diálogos baseados em argumentação. Neste documento nós apresentamos uma abordagem que explora essas duas linhas de pesquisa. Primeiro, nós desenvolvemos um mecanismo de raciocínio baseado em argumentação em uma linguagem de programação orientada a agentes, a qual é baseada na arquitetura BDI. Este mecanismo de raciocínio é baseado no formalismo de *defeasible logic* e na noção da semântica *defeasible*. Usando este mecanismo de raciocínio baseado em argumentação, agentes podem raciocinar sobre incertezas e podem usar argumentos para dar suporte às suas alegações quando envolvidos em diálogos multi-agentes. Segundo, nós damos semântica operacional para um conjunto de atos de fala encontrados na literatura de diálogos baseados em argumentação. Esta semântica é também dada no contexto de linguagens de programação orientada a agentes inspiradas pela arquitetura BDI. Além disso, nós definimos um protocolo para diálogos baseados em argumentação para realocação de tarefas, considerando o mecanismo de raciocínio e a semântica operacional apresentados. Provou-se que o protocolo termina e que atinge soluções ideais, dados alguns pressupostos desse trabalho. Finalmente, nós descrevemos um domínio de aplicação usado como estudo de caso, e avaliamos nosso trabalho com alguns exemplos de problemas reais dentro desse cenário da aplicação.

Palavras Chave: Diálogos Baseado em Argumentação, Raciocínio Baseado em Argumentação, Semântica Formal, Programação Orientada a Agentes.

ARGUMENTATION-BASED DIALOGUES FOR TASK REALLOCATION AMONG RATIONAL AGENTS

ABSTRACT

Argumentation appears in two main lines of research in the field of multi-agent systems: argumentation-based reasoning and argumentation-based dialogues. In this document we present an approach exploring both of those lines of research. First, we develop an argumentation-based reasoning mechanism in an agent-oriented programming language based on the BDI architecture. This reasoning mechanism is based on a defeasible logic formalism and the notion of defeasible semantics. Using that argumentation-based reasoning mechanism, agents can reason under uncertainty and can use arguments to supports their claims when engaging in multi-agent dialogues. Second, we give operational semantics to a set of speech acts found in the literature on argumentation-based dialogues. That semantics is also given in the context of BDI-inspired agent-oriented programming languages. Next, we define a protocol for argumentation-based dialogues for task reallocation, using the reasoning mechanism and the operational semantics. We prove that the protocol terminates and that it can reach ideal solutions under certain assumptions. Finally, we describe an application domain used as case study, and we evaluate our work with some examples of real problems from that application scenario.

Keywords: Argumentation-Based Dialogues, Argumentation-Based Reasoning, Formal Semantics, Agent-Oriented Programming.

CONTENTS

1. INTRODUCTION

Argumentation has received significant interest in the multi-agent system community in recent years because it gives us means for allowing an agent to reconcile: (i) conflicting information within itself; (ii) its informational state with new perception of the environment; and (iii) conflicting information from multiple agents through communication [42].

Argumentation can be divided into two main lines of research in the multi-agent community: (i) argumentation focused on reasoning (nonmonotonic reasoning) over incomplete, conflicting, or uncertain information, where arguments for and against certain conclusions (beliefs, goals, etc.) are constructed and compared; 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.

In this dissertation we present our research, which explores argumentation-based reasoning and argumentation-based dialogues in agent-oriented programming languages based on "mental attitudes" (*e.g.*, BDI) such as AgentSpeak(L) and its extensions available in Jason [18]. Jason has been chosen because it has important characteristics for our research, as we will describe later.

The main contributions of our work cover the two areas mentioned above. First, we develop and implement argumentation-based reasoning in an agent-oriented programming language based on the BDI architecture, where agents can reason over inference rules, as well as they can recover the rules used in such inferences to justify their conclusions. Second, we identify, formalise and implement the main performatives/speech-acts used in the literature of argumentation-based dialogues. We follow the formalisation found in [87] to give operational semantics to these performatives and, as such formalisation makes references to the agents' mental attitudes, it can be easily implemented in any language based on the BDI architecture, as we do in Jason. Third, we define and formalise an argumentation-based protocol for task reallocation between agents, which considers a set of performatives formalised and the reasoning mechanism developed. Further, we define a decision-making process to support task reallocation which uses information from an ontology to decide on the possible courses of action; this is used by an agent that realises it cannot accomplish a task to which it was committed. Finally, we demonstrate in practice the techniques described in real scenarios developed for the SeaTeaMS Project (funded by Samsung Research Brazil and run at FACIN–PUCRS).

This document is organised as follows. In Chapter 2, we describe the background of our research, where we describe the main concepts used in the remainder of this document. In Chapter 3, we describe some related work. In Chapter 4, we present the argumentation-based reasoning mechanism developed in Jason, which we published in [56]. In Chapter 5, we give operational semantics to the performatives/speech-acts found in literature of argumentation-based dialogues and incorporated into Jason, which we published in [55]. In Chapter 6, we define the argumentation-based protocol and we make some efforts towards proving some properties of this

protocol. We published some of the ideas used in that protocol in [54]. In Chapter 7, we describe the application domain where we use the techniques developed in our research, including the decision-making process which uses information from a domain-specific ontology. In Chapter 8, we describe an evaluation of our work, where we implement three scenarios of the application domain. We use CArtAgO [74] artifacts and the Jason platform [18] to implement the techniques described in this work. Finally, in Chapter 9, we make some final remarks and point out possible directions for our work.

2. BACKGROUND

In this Chapter, we provide the background needed for understanding the remainder of our work. Initially, in Section 2.1, we give a brief account of argumentation in multi-agent systems. After that, in Section 2.2, we describe defeasible reasoning, a type of rule-based reasoning that bears significant relation to argumentation systems. In Section 2.3 we give a brief introduction to dialogue game. In Section 2.4 we describe argumentation-based dialogues where the interactions are formalised as dialogue games. In Section 2.5, we describe briefly the three major approaches to automated negotiation focusing on the argumentation-based approach, which move us towards demonstrating the benefits of using argumentation-based approaches in dialogues. Finally, in Section 2.6, we give a brief introduction to agent-oriented programming and Jason, the platform that will be used in our work.

2.1 Argumentation

Argumentation can be seen as the principled interaction of different, potentially conflicting arguments, for the sake of arriving at a consistent conclusion [42].

The survey presented in [42] states that argumentation in multi-agent systems has two main lines of research: (i) *autonomous agent reasoning*, such as belief revision and decision-making under uncertainty; and (ii) as a vehicle for facilitating *multi-agent interaction*, because argumentation naturally provides tools to designing, implementing and analysing sophisticated forms of interaction among rational agents.

According to Maudet *et al.* [42], argumentation lends itself naturally to two main sorts of problems encountered in multi-agent systems:

- **Forming and revising beliefs and decisions:** Argumentation provides means for forming beliefs and decisions on the basis of incomplete, conflicting, or uncertain information. This is because argumentation provides a systematic means for resolving conflicts among different arguments and arriving at consistent, well-supported standpoints;
- **Rational interaction:** Argumentation provides means for structuring dialogues between participants that have potentially conflicting viewpoints. In particular, argumentation provides a framework for ensuring that interaction respects certain principles (e.g., consistency of each participant's statements).

"As a reasoning mechanism, argumentation provides an alternative way to mechanise nonmonotonic reasoning. Argument-based frameworks view the problem of nonmonotonic reasoning as a process in which arguments for and against certain conclusions are constructed and compared. Nonmonotonicity arises from the fact that new premises may enable the construction of new arguments to support new beliefs, or stronger counterarguments against existing beliefs. As

the number of premises grows, the set of arguments that can be constructed from those premises grows monotonically. However, because new arguments may overturn existing beliefs, the sets of beliefs may grow nonmonotonically" [42].

Essentially, argumentation can be used both for theoretical reasoning (reasoning about what to believe) as well as practical reasoning (reasoning about what to do) [42, 66].

In the communication/interaction strand, an inherent, almost defining, characteristic of multi-agent systems is that agents need to communicate in order to achieve their individual or collective goals. Agent communication with argumentation techniques allows agents to exchange arguments to justify their stance and to provide reasons that defend their claims. This improved expressivity has many potential benefits, but it is often claimed that it should, in particular [42]:

- make communication more efficient by allowing agents to reveal relevant pieces of information when it is required during a conversation;
- allow for a verifiable semantics based on the agents' ability to justify their claims (and not on private mental states); and
- make protocols more flexible, by replacing traditional protocol-based regulation by more sophisticated mechanics based on commitments.

On the other hand, this improved expressivity comes with a price. According to [42], it poses some serious challenges when it comes to designing autonomous agents that actually communicate by means of argumentation, and makes more difficult:

- the integration with agents' reasoning, which requires to precisely specify what agents should respond to others on the basis of their internal state, but also on the basis of their goals (a strategy for the agent to participate in the interaction);
- to prove whether the protocols satisfy desirable properties.

Often, argumentation is treated abstractly, where the content of individual arguments is not relevant and an overall structure of the relations between arguments is used instead. Abstract argumentation frameworks have their origins in [27], which studies the acceptability of arguments. In [27], the focus is on the attack relation between arguments, and the sets of arguments that defend its members, representing the ones that, given a set of arguments, are acceptable. As most of the work found in literature make reference to properties and concepts defined in [27], we dedicate the next section to describe abstract argumentation. We describe abstract argumentation for completeness, because some of these definitions and concepts are used in the remaining of this document.

2.1.1 Abstract Argumentation

Dung showed in [27] that argumentation can be studied without consideration for the internal structure of the individual arguments (which became known as abstract argumentation). In his work, arguments are nodes in an argument graph and arcs in this graph represent attack relationships between arguments. Formally:

Definition 1 (Argumentation framework). *An argumentation framework is a pair* $AF = \langle A, R \rangle$ *where* A *is a finite set of arguments and* $R \subseteq AXA$ *is an* attack *relation (also known as* defeat *relation). An argument* α *attacks an argument* β *if* $(\alpha, \beta) \in R$.

A simple example is shown in Figure 2.1, where argument *a1* has two attackers (i.e., counterarguments) *a2* and *a4*; furthermore, *a2* is attacked by *a3* and *a4* is attacked by *a5*.

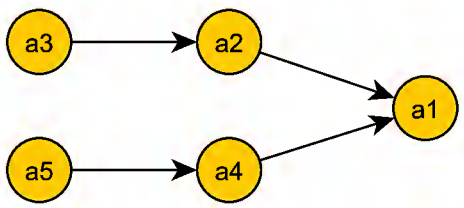

Figure 2.1 – Simple argument graph.

2.1.2 Evaluating an Argument in Abstract Argumentation

To evaluate an argument consists in checking whether this argument is acceptable or not given the other arguments and the attack relation. The acceptability of arguments is defined by a logical semantics, where it is considered how an argument interacts with the other arguments. Below, we have some important definitions to study the acceptability of arguments.

Definition 2 (Conflict-free [27]). *Let* $\langle A, R \rangle$ *be an argument framework and* S *a set of arguments (*$S \subseteq A$*).* S *is conflict-free if no argument in that set attacks another.* S *defends an argument if it attacks all the attackers of this argument.*

For example, in Figure 2.1, arguments $\{a3, a5\}$ defend $a1$. The collective acceptability of a set of arguments can be characterised by various different semantics, as defined below.

Definition 3 (Admissible set [27]). *Let* S *to be a conflict-free set of arguments in a framework* AF. S *is admissible if it is conflict-free and defends every element in* S.

An arguments is *admissible* if it is a conflict-free set that defends itself against all attackers. In Figure 2.1, the sets: $\emptyset$, $\{a3\}$, $\{a5\}$, $\{a3, a5\}$ and $\{a1, a3, a5\}$ are all admissible.

Definition 4 (Complete extension [27]). *An admissible set S is a complete extension if, and only if, all arguments defended by S are also in S (i.e., S is a fixed point).*

In the example in Figure 2.1 the only *complete extension* is the set $\{a1, a3, a5\}$.

The *complete extension* has some refinements, for example, for S a set of arguments:

- S is a *grounded extension* if it is the minimal complete-extension (S is the least fixed point);
- S is a *preferred extension* if it is a maximal complete extension (S is the maximal admissible set).

The *grounded extension* is unique and contains all arguments that are not attacked, as well as the arguments that are defended directly or indirectly by non-attacked arguments.

As for the definition of the acceptability of sets of arguments, we can define the status of individual arguments:

Definition 5 (Argument status [27]). *Let $\langle A, R \rangle$ to be an argumentation system, and $E_1, ..., E_n$ its extensions under a given semantics. For $\alpha \in A$ an individual argument, we say that:*

- *α is skeptically accepted iff $\forall E_i . \alpha \in E_i$, with $i = 1, ..., n$;*
- *α is credulously accepted iff $\exists E_i . \alpha \in E_i$;*
- *α is rejected iff $\nexists E_i . \alpha \in E_i$.*

The latest instantiation of Dung's abstract formalism [27] appears in Prakken's work [65]. Prakken [65] defines the structure of argument using two types of inference rules, *strict* and *defeasible* rules, respectively. Further, this work defines three types of attack between arguments, *undercutting* and *rebutting* attack (originally formalised in [63]), also a third type called *undermining* attack (inspired in [89]). In this latest instantiation, the author justifies that the structure of arguments permits more expressible representation of the attack relation.

2.2 Defeasible Reasoning

Defeasible reasoning is a simple and efficient approach to nonmonotonic reasoning where the objective is to formalise nonmonotonic inferences of the type "birds generally fly". Such inferences hold only if a defeasible theory contains no rule inferring contrary information, which will be explained throughout this section using a specific formalisation of defeasible reasoning called defeasible logic [51, 52].

Knowledge in a defeasible theory is organised as facts, rules, and a "superiority" relation. Rules are separated into strict rules, defeasible rules, and defeaters:

- **Facts**: facts are indisputable statements (e.g., "Alison is a graduate student");

- **Strict rules**: strict rules are rules in the classical logic sense, where if the premises are indisputable (i.e., facts) then so is the conclusion (e.g., "graduate students are students");
- **Defeasible rules**: defeasible rules are rules that can be defeated by contrary evidence (e.g., "graduate students usually study hard");
- **Defeaters or Undercutting Defeaters**: defeaters are rules that are used to prevent some conclusions from being derived rather than to draw particular conclusions;
- **Superiority relation**: superiority relation is a binary relation between rules which defines whether a rule is superior to another, and is used in case applying the rules would lead to contradicting conclusions.

Conclusions can be derived *strictly* or *defeasibly*. A conclusion is *strictly* derived if it is derived using only strict rules and facts contained in the knowledge base. A conclusion is *defeasibly* derived if it is derived using any clauses of the knowledge base including defeasible rules, presumptions[1] and defeaters [50].

As defeasible rules represent disputable knowledge, they can be defeated by contrary evidence (provided by other rules). The two types of defeat are: (i) *rebut*, where the conclusion of the rule is defeated because another rule derives the negation of that conclusion (i.e., a contrary conclusion can be obtained through another rule); and (ii) *undercut*, where the conclusion of the rule cannot be derived because an applicable defeater rule concludes the contrary (recall that the defeater cannot be used to conclude anything, it just prevents the conclusion of the contrary).

An answer to a query in a defeasible knowledge base can be of five types [50]:

- **definitely yes**: meaning that a conclusion is proved using only facts and strict rules, and therefore cannot be withdrawn when new knowledge is added to the available theory;
- **definitely no**: meaning that the negation of the queried conclusion can be proved using facts and strict rules;
- **presumably yes**: meaning that the conclusions can be defeasibly proved, so it might need to be withdrawn when new knowledge becomes available;
- **presumably no**: meaning that the negation of the query can be defeasibly proved, that is, although the query cannot be presently concluded, it might be concluded if new knowledge becomes available;
- **cannot tell**: it is not possible to answer the query either affirmatively or negatively (because both the queried formula and its conclusion can be defeasibly derived and the superiority relation does not favour one or the other).

[1] Presumptions are defeasible rules of the form $Head := true$ [50].

Defeasible logic is a nonmonotonic logic introduced by Nute [51, 52] as a way to formalise defeasible reasoning, and made practical as the d-Prolog programming language [50] (an extension of Prolog based on defeasible logic). Defeasible logic and defeasible Prolog (d-Prolog for short) have all types of knowledge representation mechanisms defined in the theory of defeasible reasoning, including facts, strict rules, defeasible rules, undercutting rules or defeaters, and superiority relation, as described above.

The representation in d-Prolog of defeasible rules and defeaters is possible through the introduction of the new binary infix functors `:=` and `:^`, respectively. It also introduces strong negation with the functor `neg`, which differs from the negation-as-failure operator `not`. D-Prolog also introduces a type of defeat by specificity, where more specific conclusions defeat more general ones. This is exemplified by the well-known Tweety triangle:

```
flies(X)      := bird(X).
neg flies(X)  := penguin(X).
bird(X)       := penguin(X).
penguin(tweety).
```

All clauses in the example are defeasible rules. If we make a query for whether "tweety flies" using `?-flies(tweety)` in d-Prolog, the answer will be "presumably no" because the rule for *penguin* is more specific than the rule for *bird*. The specificity is defined by two inferences where it is tested for the two rules in conflict whether one of them can be derived from the other, and if that is the case, that which is derived from the other is defeated for being less specific. In this example, a rule with a `penguin` premise is more specific than one requiring `bird` to be inferred because the rule `bird(X) := penguin(X)` says that normally penguins are birds, hence the class of penguins is more specific than that of birds (membership to the latter can be inferred from membership to the former).

Another characteristic of defeasible logic is having the so-called "preempting defeaters" [50] or "ambiguity blocking" [29], where defeasible rules that are rebutted by a superior rule are no longer available to rebut other rules.

An example of *preempting defeaters* is the knowledge base represented by Π below (where we use $\Rightarrow$ to refer to defeasible inferences):

$$\Pi = \left\{ \begin{array}{ll} a \Rightarrow b & x \Rightarrow e \\ b \Rightarrow c & e \Rightarrow \neg c \\ c \Rightarrow d & y \Rightarrow \neg e \\ a & \\ x & \\ y & \end{array} \right\}$$

In this example, we may conclude d using the inferences $\{a,\ a \Rightarrow b,\ b \Rightarrow c,\ c \Rightarrow d\}$, although there is a derivation $\{x,\ x \Rightarrow e,\ e \Rightarrow \neg c\}$ which rebuts the rule that concludes c; this

rule (the rule that derives $\neg c$) is defeated by rule $\{y,\ y \Rightarrow \neg e\}$ which prevents the use of that rule to rebut the inference of *d*.

2.3 Dialogue Games

Recently, dialogue games have been used as the basis for agent interaction protocols. According to [46], dialogue game protocols are more expressive than auctions and game-theoretic mechanisms, as dialogue games allow the participant to question and contest assertions, to support assertions with arguments and counter-arguments, etc. The authors of [46] also claim that these dialogues conducted by dialogue game protocols are more constrained than interactions using other agent communication languages (such as FIPA ACL [28], for example).

A formal dialogue game contains the following elements [44]:

Commencement Rules: Rules which define the circumstance under which the dialogue commences.

Locutions: Rules which indicate what utterances are permitted (locutions which permit participants to assert propositions, to question or to contest prior assertions, etc.).

Combination Rules: Rules which define the dialogue context under which particular locutions are permitted or not, or obligatory or not.

Commitments: Rules which define the circumstance under which participants express commitment to a proposition.

Termination Rules: Rules that define the circumstances under which the dialogue ends.

In [46] it was stated that the semantics for dialogue game protocols was still very immature. Further, the authors of that paper argue that the formal semantics for dialogue game protocols has the objective of enabling a better understanding of the formal properties of protocols and of dialogues conducted under them. According to [46], the study of the formal properties of dialogues and protocols is, like the development of formal semantics, still very immature, and considerable scope exists for further research in this area. The authors of [10] confirm this assertion, with [10] and [48] being some of the few examples of work found in the literature about formal properties of argumentation-based dialogues and protocols.

One property of great interest in dialogue games is *termination* [46] (examples of proves of such property can be found in [78, 60] where analyses for the termination of the protocols in [5, 7] are given).

2.4 Argumentation-Based Dialogues

In an argumentation-based dialogue, the agents trade propositions for which they have acceptable arguments, and accept proposition put forward by other agents if they find that the

arguments are acceptable. The locutions presented at each round and the way that they are exchanged define a formal dialogue game (a dialogue governed by a protocol) in which agents engage [5, 60].

In a dialogue, each agent has a knowledge base Σ, containing its knowledge or beliefs. In addition, each agent has a further knowledge base, the *commitment store* (CS), accessible to both agents[2] in a dialogue, containing commitments made by the agents during the dialogue (other names can be found for CS such as *dialogue obligation store* in [47] and *dialogue store* in [78]). Singh [82, 83] argues that agents are social entities, therefore, when involved in social interactions, they are committed to what they say. The CS is simply a subset of the knowledge base, and the union of all CSs can be viewed as the state of the dialogue at a given time [60].

How that dialogue will unfold depends on the message exchanges (what messages agents actually send and how they respond to message they receive). This aspect of the dialogue is specified by a protocol (stating the allowed message exchanges), and by some decision-making apparatus within the agent (the agent strategy depends on choices made by agent through reasoning).

A strategy in an argumentation dialogue specifies what utterances to make in order to bring about some desired outcome (e.g., to persuade the counterpart to perform a particular action). How the agent makes the choice is an aspect of its strategy, and relevance may come into its strategic thought [42].

The choices that an agent makes are referred to in the argumentation literature as *agent attitudes*. According to [57, 60, 6], agent attitudes can be separated into *assertion attitudes* and *acceptance attitudes*:

- Assertion attitudes: An agent may have two assertion attitudes:
  - a *confident* agent can assert any proposition p for which it can construct an argument for p;
  - a *thoughtful* agent can assert any proposition p for which it can construct an acceptable argument for p.

  That is, a thoughtful agent will only put forward propositions which, as far as it knows, are correct. A confident agent will not stop to check that this is the case.

- Acceptance attitudes: An agent may have three acceptance attitudes:
  - a *credulous* agent can accept any proposition p if it is backed up by an argument;
  - a *cautious* agent can accept any proposition p if it is unable to construct a strong argument for $\neg p$;
  - a *skeptical* agent can accept any proposition p if there is an acceptable argument for p.

[2]Note that most work in the literature consider dialogues of two agents only.

Clearly *skeptical* agents are more demanding than *credulous* ones in terms of the conditions they put on accepting new information.

In [6], further, the authors describe the "question attitudes", where an agent executes a `question` move[3] based on its argumentation systems (if the agent is able to construct an argument for or against the previous move).

Some of the locutions/performatives commonly used in argumentative dialogues are found in work such as [5, 57, 60], where its 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;
- `retract`: an agent that performs a `retract` utterance 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;
- `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 defend its claim, and presumably will provide the support set for its previous claim;
- `challenge`: the `challenge` performative is similar to the `question` performative, except that the sender of the message is committed to defending a claim contrary to the previous claim of another agent.

These performatives can also be found in the work by McBurney and Parsons [47], where they propose some performatives, that they consider necessary for argumentation, to be added to FIPA ACL [28]. In that work, they also give axiomatic and operational semantics to a protocol called *Fatio*. Other performatives, different from these, can also be found in the literature, as in [64].

Another, more recent, related work is [13], where the authors propose the use of some performatives for argumentation in AgentSpeak and give semantics to those performatives. That work is focused on negotiation (as in [7]) and uses an electronic trading scenario. That work is similar to [87] in treating the communication of a single message exchange and not as a sequence of interactions (*i.e.*, a dialogue).

[3]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 [42].

2.5 Negotiation

"Negotiation is a process that aims at finding some compromise or consensus on an issue between two or more agents having different goals" [10]. In the negotiation literature, the issue under negotiation is called the negotiation object. Generally the negotiation objects are represented as a set of offers. For example, if an agent negotiation about an allocation of resources (as in [33, 71, 78, 61]), the set of offers will contain all possible allocations. On the importance of negotiation for multi-agent applications, we can quote:

"In most agent applications, the autonomous components need to interact with one another because of the inherent interdependencies which exist between them, and negotiation is the predominant mechanism for achieving this by means of an exchange of offers" [4].

In the multi-agent systems field, the three major approaches to automated negotiation are: *game-theoretic approaches*, *heuristic-based approaches* and *argumentation-based approaches*. Further, the literature describes several limitations and emphasises that argumentation-based negotiation minimises these limitations [72].

Game theory has roots in the work of Von Neuman and Morgenstern [88]. It studies the interaction between self-interested economic agents (where economic is meant in the sense of people and organisations) and has been used to study interaction between self-interested computational agents [76].

Game-theoretic approaches to negotiation are based in dialogue-games (as described in the Section 2.3) that are interactions between two or more players, where each player makes a *move* by making some utterance in a common communication language, and according to some pre-defined rules. A dialogue-game protocol is defined in terms of a set of locutions, as well as different types of rules: commencement rules, combination rules, commitment rules, and termination rules. Commencement and termination rules specify when a dialogue starts and how it finishes. Commitment rules specify how the contents of commitment stores change as a result of different locutions. Finally, combination rules specify the legal sequences of dialogue moves [42].

However, classical game-theoretic approaches have some significant limitations from the computational perspective, as they generally assume that agents have unbounded computational resources and that the space of outcomes is completely known [72].

The *heuristic-based approaches* address some of the limitations of game-theoretic approaches using heuristics. Heuristics produce good enough (rather than optimal) outcomes and use more relaxed assumptions about the rationality and resources of the agents. In general, these methods offer approximations to the decisions made according to game-theoretic techniques.

While heuristic methods do indeed overcome some of the shortcomings of game-theoretic approaches, they also have a number of disadvantages [34]. Their models lead to outcomes that are sub-optimal because they adopt an approximate notion of rationality and because they do not examine the full space of possible outcomes, and it is very difficult to predict precisely how the system and the constituent agents will behave.

2.5.1 Argumentation-based Negotiation

In recent years, the prominence of argumentation-based dialogues in the literature has increased, in particular, argumentation-based negotiation, partly because of the promise to resolve the problems in other approaches (*game-theoretic* and *heuristic-based* approaches). The advantage of argumentation-based negotiation over others approaches is the additional exchange of meta-information that the approach permits as part of the offers, that allow more sophisticated forms of interaction than the other approaches [72].

Other approaches cited above, do not allow agents to exchange any additional information and in several negotiation situations, the agents may have incomplete information which limits their ability to negotiate. Thus, the agents might [72]:

- lack some of the information relevant to making a comparison between two potential outcomes;
- have limited resources preventing them from acquiring such information;
- have the information, but lack the time needed to process it in order to make the comparison;
- have inconsistent or uncertain beliefs about the environment;
- have unformed or undetermined preferences (e.g., about products new to them); or
- have incoherent preferences.

Finally, the game-theoretic and heuristic-based approaches assume that agent's utilities or preferences are *fixed.* One agent cannot directly influence another agent's preference model, nor any of its internal mental attitudes (e.g., beliefs, desires, goals, etc.) that generate its preference model. A rational agent would only modify its preferences upon receipt of new (reliable) information, however, as mentioned, traditional automated negotiation mechanisms do not facilitate the exchange of such information [72].

Argumentation-based negotiation approaches do not have the limitations mentioned above. Agents are capable of exchange additional information, or to "argue" about their beliefs and other mental attitudes during the negotiation process [72]. In the context of negotiation, the *argument* is viewed as a piece of information that may allow an agent to: (a) *justify* its negotiation stance; or (b) *influence* another agent's negotiation stance [35].

Thus, besides the agent accepting or refusing a proposal, it can offer a critique; this can help make the negotiation more efficient, because the agent can understand why its counterpart cannot accept a particular deal. In the next interaction, the agent can use the new information to make an alternative offer that has a higher chance of being accepted.

Besides criticism, another type of information that the agents can exchange in an argumentation-based negotiation is a justification of a proposal. This new information can change

the region of acceptability of an agent. This change can turn offers that previously were not acceptable into acceptable ones [35].

Also, more specific types of information can be exchanged by the agents in an argumentation-based negotiation, whereby they can make appeals [84] (or, as defined in [8], explanatory arguments), threats, and promises of rewards [9, 41, 81]. *Appeals* are used to justify a proposal; *threats* are used to warn about negative consequences in case the counterpart does not accept a proposal; and *rewards* are used to promise future rewards if the counterpart accepts the proposal.

2.6 Agent Oriented Programming and the Jason Platform

In the agent-oriented programming paradigm, the agents are computational entities with autonomous behaviour (i.e., able to make decisions and act without direct human intervention on unexpected circumstances). These computational entities are situated in an environment that they are to able to sense (through sensors), act upon it (through effectors), and communicate through message passing.

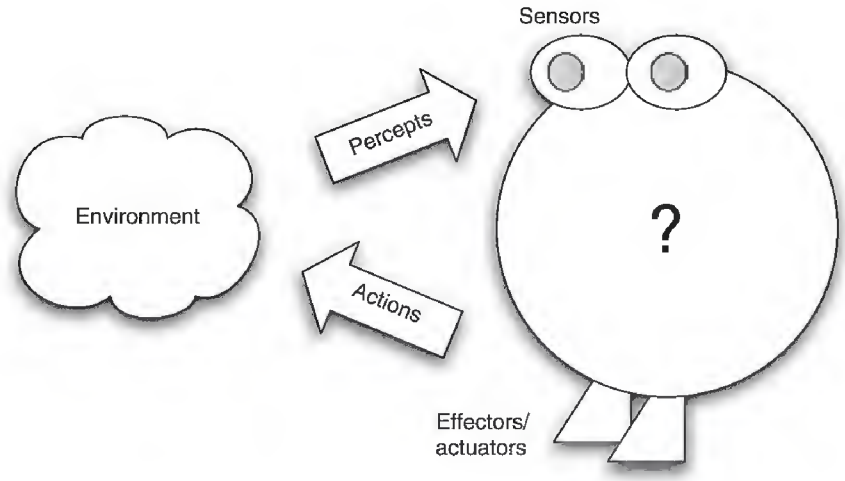

Figure 2.2 - Agent and Environment ([18]).

Generally, the systems built with the agent oriented programming paradigm are not composed for a single agent, but composed by a set of agents in the same environment (as shown in Figure 2.3). This characteristic makes evident the need for communications (interaction) between the agent in such systems. This interaction is generally about *beliefs*, *goals* and *plans*. This allows the existence of *social ability* for *cooperation* and *coordination* [18].

One of the most studied architectures for cognitive agents is the BDI (*Beliefs-Desires-Intentions*) architecture which provides a particular structure for agent internal states based on "mental attitudes". The internal state of a BDI agent is formed by: (i) *Beliefs* that represent the information about the world (including itself and other agents) available to that agent; (ii) *Desires*

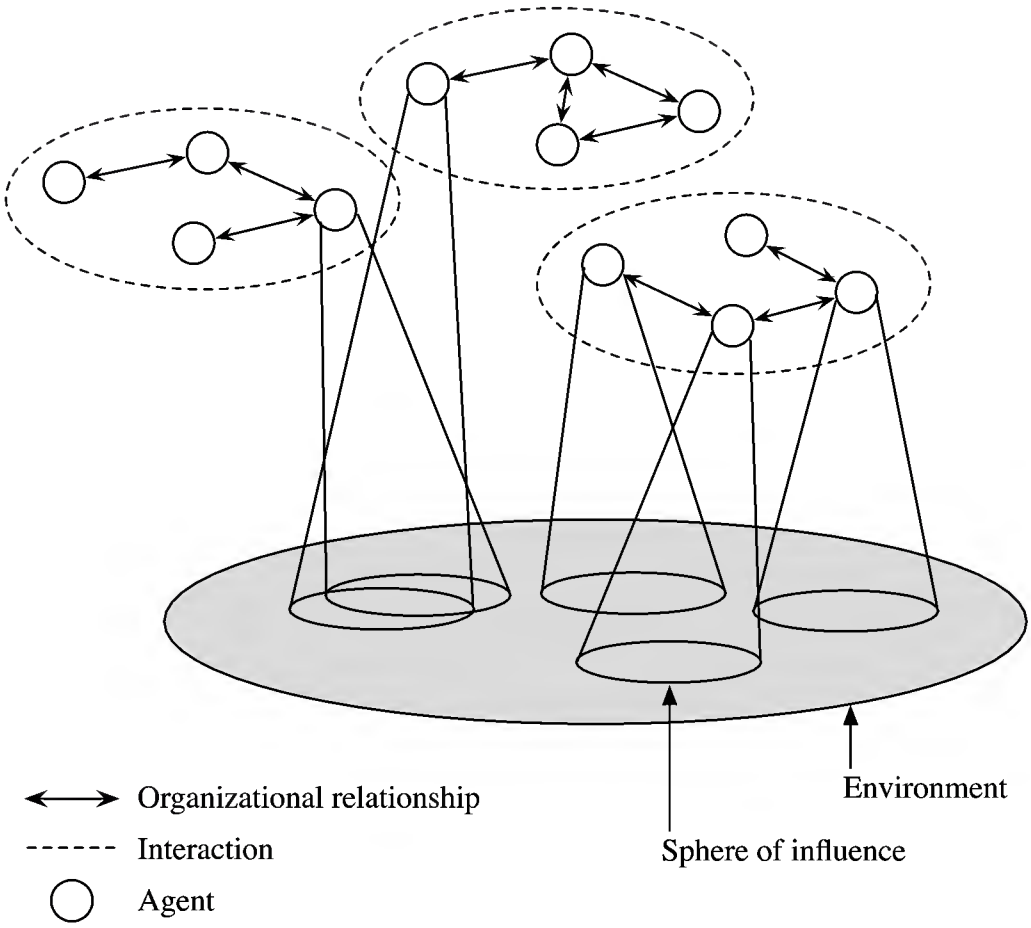

Figure 2.3 – Typical Structure of a Multi-Agent System (based on [36]).

representing the motivations of the agent, i.e., the states of the environment that the agent would like to reach; and (iii) *Intentions* which are desires that the agent is committed to achieve by following particular *plans* of action.

There exist many agent-oriented programming languages and platforms, such as Jason, Jadex, Jack, AgentFactory, 2APL, GOAL, Golog, and MetateM, as pointed out in [17]. Those languages differ in the agent architecture used, in the form of communication/interaction between them, and also on the programming paradigms that inspired or underlie each language.

Among the languages mentioned above, AgentSpeak(L), the language on which Jason is based, is one of the best-known languages inspired by the BDI architecture. AgentSpeak(L) is an abstract logic-based agent-oriented programming language introduced by Rao [73], and subsequently extended and formalised in a series of papers by Bordini, Hübner, and various colleagues.

AgentSpeak(L) is based in the Procedural Reasoning System (PRS) (Figure 2.4) where the agents are equipped with a library of pre-compiled plans. Plans in PRS have the following components: (i) a *goal* – the post-condition of the plan (the things that it achieves); (ii) a *context* – the pre-condition for the plan, defining what must be true of the environment in order for the plan

to be successful; and (iii) a *body* – the 'recipe' part of the plan – it may contain a list of actions and sub-goals in order to achieve the main goal.

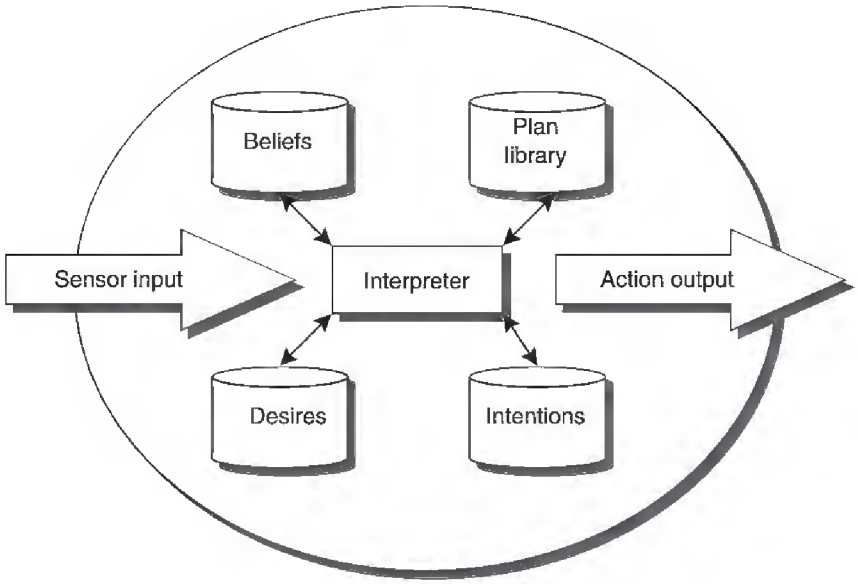

Figure 2.4 – The Procedural Reasoning System (PRS) ([18]).

Plans in AgentSpeak(L) have the following format:

```
triggering_event : context <- body.
```

where the *triggering_event* represents a new agent goal (or belief), which is to be pursued and has the format $!goal(Parameter)$, the *context* has preconditions needed to perform that plan to achieve that goal, and the *body* is a sequence of actions and sub-goals (which trigger others events and the use of other plans) to achieve the goal.

In particular, the main AgentSpeak(L) extensions available in Jason (a Java-based platform for the development of multi-agent systems), according to [18], are:

- **Strong negation:** Strong negation helps the modeling of systems where uncertainty cannot be avoided, allowing the representation of things that the agent believes to be true, believes to be false, and things that the agent is ignorant about;
- **Handling of plan failures:** Jason has a particular form of plan failure handling mechanism consisting of plans that are triggered by such failure, giving the programmer the chance to act so as to undo the effects of any action already done before the plan failed, if necessary, and then adopting the goal (that was not achieved) again, if the conditions are appropriate;
- **Belief annotations:** One interesting characteristic present in Jason is that it automatically generates annotations for all beliefs in the belief base about the source from where the belief was obtained (sensing the environment, communication with other agents, or a mental note created by the agent itself). The annotation has the following format:

$likes(john, music)[source(john)]$, stating that the source of the belief that john likes music is agent john itself;

- **Speech-act based communication:** Jason uses performatives based on speech acts in its communication language, which goes well with the availability of formal semantics of mental attitudes for the Jason extension of AgentSpeak; this permits rich communication;
- **Plan annotations:** programmers can add annotations to plan labels which can be used by elaborate selection functions, for example for defining preferences in case various different plans are applicable.

The Jason platform, more generally, has the following features:

- **Distribution**: The platform permits easy distribution on a computer network; further details can be found in [18];
- **Environment**: In many cases, during development, a simulation of the target application environment will be needed, so Jason provides support for developing environments, which are programmed in Java;
- **Customisation**: programmers can customise important parts of the agent platform by providing application-specific Java methods for certain aspects of an agent and the agent architecture. These parts can be customised through methods of the agent class and agent architecture class (more details in [18]);
- **Language extensibility and legacy code**: The extension of the AgentSpeak language in Jason can be done through the so-called "internal actions", which are implemented in Java (or any other language using Java Native Interface);
- **Integrated Development Environment**: Jason is distributed as a plug-in for the jEdit and Eclipse IDEs. This facilitates the development of applications.

As mentioned above, the communication between agents is via message passing, and Jason's communication is based on speech acts and is performed by the pre-defined internal action '*.send*' that has the following format:

$$.send(receiver, illocutionary\_force, propositional\_content)$$

where $receiver$ is the name of an agent (each agent has a unique individual name in the multi-agent system) or a list of agent names, for whom the message is being sent. The $propositional\_content$ is a term in AgentSpeak (a literal, triggering event, plan, or a list of literals or plans). The $illocutionary\_force$ denotes the intention of the sender (often called *performative*), as in speech-act theory. The formal semantics of receiving such messages is given in [87], and a complete list of all the illocutionary forces available can be found in [18].

New illocutionary forces can be easily added, as well as the effects that each will have on the agent's mental state. This is an important feature for our work, because argumentation-based dialogue needs illocutionary forces that represent specific intentions of message senders, as well as the required effect in the mental state of receivers of the message. In Jason, agent plans can be written in AgentSpeak to give such semantics to new performatives, hence providing an elegant and transparent way for programming agents that are capable of argumentation.

3. RELATED WORK

In this chapter, we provide a literature review where we describe some of the most relevant work in the argumentation-based reasoning and argumentation-based dialogues literature within the multi-agent systems field. We cover a small part of this vast literature, focusing on the work that is most related to our research.

First, in Section 3.1, we describe in more depth a recent practical work on argumentation-based reasoning, which, to the best of our knowledge, was the first argumentation-based reasoning implemented in Jason (although using a different approach than we do). Further, also in Section 3.1, we describe the link between argumentation and defeasible reasoning which guided us to adapt Defeasible Prolog for Jason. In Sections 3.2 and 3.3, we describe general work that cover argumentation-based reasoning and argumentation-based dialogues, in particular those from which we use some concepts and formalisations or, rather, we describe them just for the sake of completeness (of an overall view of this research topic). Finally, in Section 3.4, we describe briefly some work on semantics for agent-oriented programming languages, which is important for the work we present in Chapter 5.

3.1 Reasoning Mechanism

Argumentation is a promising model for reasoning about inconsistent knowledge, based on the construction and the comparison of arguments. It may also be considered as an alternative method for handling uncertainty [2].

Reasoning mechanisms for argumentation-based dialogues, especially argumentation-based negotiation, refers to the capability of the agents to reason about the received assertion/proposals (the best offer, the acceptance or rejection of offers) and the capability of generating arguments to defend or attack offers. Several works considering reasoning mechanisms based on argumentation can be found in the literature ([2, 2, 11, 16, 66]), most of them based on abstract argumentation systems at the theoretical level.

As stated in [15], "we have a rich and well structured abstract theory, the challenge now is to put this work into practice and prove its usefulness in real applications". This author presents an approach for defeasible reasoning to enable argumentative capabilities in BDI agents. The work extends the Jason platform with a module for argumentation, which is decoupled from the BDI reasoning cycle, operating in a customised belief base of the agent and does not interfere in the execution of plans, creation of goals, or agent's commitments [15].

The module gives to the agents the capabilities of general argumentation and not just for a specific needed of a particular application. With it, the agent is capable of non-monotonic argumentation-based reasoning and can participate in dialogues and negotiation if the strategies and protocol which the agent must follow are programmed in the agent (but no example is given).

In other words, only the *reasoning mechanism* is implemented, and it requires the implementation of strategies and protocol to permit argumentation-based dialogues, which are the three characteristics mentioned in a recent overview of the area available in [25].

The work [15] is based on the latest instantiation of Dung's abstract formalism presented by Prakken's in [65], which uses two kinds of inferences: strict and defeasible rules (as defined in Section 2.2). The extension-based semantics (described in Section 2.1) is used, which provides two alternative types of justification, *skeptical* and *credulous*, depending on whether the argument belong to all extensions (skeptical) or belong to at least an extension (credulous). Using this classification the author defines the *justification state* of an argument as:

justified : corresponds to skeptical justification;

defensible : corresponds to credulous justification;

overruled : arguments that cannot be justified and are rejected.

Special beliefs are used in the belief base, $defeasible\_rule(RuleName, RuleText)$ and $strict\_rule(RuleName, RuleText)$, representing defeasible and strict rules, respectively. Another special belief predicate is used to represent contradictory and contrary information, $contradictory(Literal1, Literal2)$ and $contrary(Literal1, Literal2)$, respectively.

The module works in the following manner: an agent queries the argumentation module with the belief $why(Proposition)$ and the response is in format of a belief $because(X, Y)$ with X being either ***in*** or ***out***[1]. The possible responses are:

- $because(out, unknown)$: the proposition is not in KB (Knowledge Base);
- $because(in, Premise(premise\_type))$: the proposition is a premise, and is not the result of any form of reasoning;
- $because(out, \neg Proposition)$: the proposition is not in the KB, but its negation is an acceptable argument;
- $because(in, \neg Preposition)$: the proposition is not in the KB, but its negation is an overruled argument;
- $because(in, Rule)$: the proposition is accepted and is the result of applying rule $Rule$;
- $because(out, ListOfDefeats)$: returns a list the conclusions of the arguments that defeated the proposition;

A part of our work is closely related to this work, because we implement argumentation-based reasoning in Jason (as the work described above), but we use the defeasible logic formalism

[1]That is, whether the proposition is acceptable or not following the extension-based semantics.

through Prolog-like rules that Jason permits with some limitations such as disallowing the "cut" operator. This permits the agent to reason about such rules (defeasible and strict rules) during the executions of plans to achieve a goal as well as to query if an argument is acceptable or not.

The use of defeasible logic is justified by the connection made by [29] between argumentation and defeasible logic, as well as an existing implementation of defeasible logic called defeasible Prolog (d-Prolog for short).

In [29], a significant link between defeasible reasoning and argumentation is established, where Dung-like argumentation semantics is given for defeasible logic, providing a Dung-like argumentation system. The author argues that defeasible logic provides an efficient implementation platform for systems of argumentation.

The argumentation semantics proposed is for classical defeasible logic (as in [51, 52]) and provides an ambiguity blocking argumentation system. The paper presents the usual pieces of an argumentation system: *logical language* and definitions of *argument*, *conflict between arguments*, and the *status of arguments*. The language is the same as defeasible logic (presented in Section 2.2). Arguments of the type (R, h) are formed by a set of rules R that have as consequent the literal h.

The types of arguments depend on the rules used:

- A *supportive argument* is a finite argument where no defeaters are used;
- A *strict argument* is an argument in which only strict rules are used;
- An argument that in not strict is called *defeasible*.

The characterisation of conclusions of defeasible logic in argumentation terms also is defined (being p a literal):

- if p is *strictly proved* there is a strict supportive argument for p;
- if p *is not strictly proved* there is no strict argument for p;
- if p is *defeasibly derived* there is a supportive argument for p;
- if p *is not defeasible derived* there is no supportive argument for p.

The definition of attack is usual, where defeasible arguments can attack or undercut other defeasible arguments and can be attacked or undercut by strict arguments (strict argument cannot be attacked).

The paper defines the so called *defeasible semantics* which determines whether an argument is accepted or rejected in order to capture defeasible provability in defeasible logic [51, 52] with ambiguity blocking (in original defeasible logic). This is formally defined below.

Definition 6. *An argument A for p is acceptable in a set of arguments S if A is finite, and: (a) A is strict, or (b) every argument attacking A is undercut by S (i.e., it is proved that the premises of all the arguments that attack A cannot be proved, the so called ambiguity blocking).*

This definition is achieved with ambiguity blocking, also called *preempting defeaters* defined in Section 2.2.

Definition 7. *An argument A is rejected by the sets of arguments S and T when A is not strict and: (a) a proper subargument of A is in S, or (b) it is attacked by an argument supported by T (i.e., the attacking argument must be supported by the set of justified arguments).*

In relation to grounded semantics defined by Dung in [27], the paper defines that:

- if an argument A is justified under grounded semantics, then A is justified under defeasible semantics;
- if an argument A is rejected under grounded semantics, then A is rejected under defeasible semantics;
- if a literal p is justified under grounded semantics, then p is justified under defeasible semantics;
- if a literal p is rejected under grounded semantics, then p is rejected under defeasible semantics.

Another interesting related work is [66], where Rahwan and Amgoud present a framework for argumentation-based practical reasoning based in Dung's abstract argumentation framework [27]. In their work, agents argue about their beliefs, desires, and plans. The agents compare arguments based on decision-theoric notions (utility), *i.e.*, the worth of desires and the cost of resources are integrated into the proposed framework.

The desires and plans are represented by rules such as: $\varphi_1 \wedge ... \wedge \varphi_n \wedge \psi_1 \wedge ... \wedge \psi_n \Rightarrow \psi$ where φ denotes beliefs and ψ denotes desires. The rule means that if the agent believes in $\varphi_1 \wedge ... \wedge \varphi_n$ and desire $\psi_1 \wedge ... \wedge \psi_n$, then the agent will desire ψ as well.

The authors assume that a proposition is believed because it is *true* and *relevant.* Desires, on the other hand, are adopted because they are *justified* and *achievable.* Also, it is defined that an argument for a belief can be attacked by arguing that it is not consistent, or because there is a reason to believe the contrary. On the other hand, arguments for desires could be attacked by demonstrating that the justification for that desire does not hold, or that the plan intended for achieving it is itself not achievable.

The work extends others frameworks: (i) Dung's framework extended by Amgoud e Cayrol in [2] that handles beliefs; and (ii) for arguing about desires and plans, they extend work on argumentation-based desire generation and planning [1, 4, 32]. Moreover, they adapted the notion of attack and preference among arguments in order to capture the differences in arguing about beliefs, desires, and plans.

Our work has some similarity with [66]. We also use inference rules to represent arguments, and the idea of an agent believing something because there is no reason to believe the contrary is considered in our work too.

3.2 Argumentation Framework

The survey found in [72] focused in argumentation-based dialogues, especially argumentation-based negotiation, considers various aspects of the surveyed frameworks that the authors find important: the *communication language and domain language*, *protocol*, and *information store*.

Dialogues are, by definition, a form of interaction between agents. Elements of the *communication language* are usually referred to as *locutions*, *utterances*, or *speech acts* [80]. Several locutions can be found in literature (as described in Section 2.4).

In addition to the communication language, a *domain language* is also needed, which permits all the agents to "speak the same language" and understand each other in the interactions.

In multi-agent systems, the major proposals for agent communication language are: Knowledge Query and Manipulation Language (KQML) [43] and the Foundation for Intelligent Physical Agents' Agent Communication Language (FIPA ACL) [28]. FIPA ACL fails to capture all the utterances needed in negotiation interaction, and some works in the literature propose to add new locutions in order to address this problem (for example [47]).

Given a communication and domain language, an argumentation framework should also specify a *protocol* to constrain the use of the language. The protocol defines, at each stage of the dialogue process, which moves are allowed to each agent [72]. Normally, for rational agents the protocol is defined in terms of the conditions that need to be satisfied (like the context of a plan in Jason), and those that indicate the next move (or interaction) that the agent must make. Also, the protocol usually is represented as a finite-state machine. An important characteristic of protocols is *termination*.

In argumentation-based dialogues it is important to store the information exchanged by the agents because this makes it possible to prevent an agent from denying a promise it has previously made, for example. One type of information store that is common in the argumentation literature is the commitment store, which has its origin in the work of Hamblin [30] who used it as a way of tracking the claims made by participants in dialogue games.

In the work on the philosophy of dialogue (e.g., [90]) the focus is on commitment towards action, i.e. promises to initiate, execute, or maintain an action or course of action. Commitments to defend a claim if questioned, called propositional commitments, are viewed by the authors as special cases of such action commitments, and give rise to the *commitment store* mentioned in Section 2.4.

When an agent asserts a proposition p, it may not only be committed to believing that p holds, but also to defend p (if challenged), not to deny p, give evidence that p, and so on [90].

3.3 Argumentation-based Dialogues

In this section, we describe the related work on argumentation-based dialogues (most of them focused on argumentation-based negotiation), where the references cover the three main topics pointed out in [25]: (i) the *reasoning mechanism* that the agent uses for negotiating based on argumentation, (ii) the *protocol* that the agents use for conveying arguments and offers (in the case of argumentation-based negotiation), and (iii) the *strategy* that determines an agent's choices at each step of the dialogue.

Kakas and Moraitis, in [38], propose a protocol that is sensitive to the context and roles of the agents in which the agents can adapt their negotiation strategies and offers, as their environment changes, and when they exchange information within the negotiation. The authors also emphasise that argumentation-based negotiation removes many of the limitations of the alternative approaches (*game theoretic* and *heuristic based* approaches).

The agents can build arguments through a theory (formed by their knowledge base and goals), and the representation of their knowledge uses abduction. The work is based on argumentation systems with dynamic preference proposed by the same authors in [39, 37].

The agent has rules for representing the knowledge of the default and specific context, and the authors stress that the process of deliberation of the agent adapts their strategies, offers and counter-offers as the negotiation environment changes.

The protocol proposes that, when an agent cannot satisfy his own goal, it can consider, in a conciliation phase, the other agent's goals and searching for conditions under which it could accept the offer (extending the object of negotiation).

The main relevance of this work is the representation of knowledge through rules, whereas, in our work, the agents will have the representation of part of the knowledge by rules as in [15] (strict and defeasible rules). The change of strategy by the agent with changes in the environment is also relevant, as new information in the agent's theory (new rules, facts, etc.) changes the possible conclusions in defeasible reasoning.

Dimopoulos, Moraitis, and Amgoud, in [26], present a characterisation of outcomes of argumentation-based integrative negotiation. The authors argue the existence of two types of negotiation: *integrative negotiation* where all sides are looking for solutions that are "good" for everyone (the negotiation type that the authors treat); and *distributive negotiation* where each party tries to maximize his gain.

The work is presented in an abstract way, and the structure of arguments is not considered. The *negotiation theory*, which consist of a set of arguments, a function for each offer (or outcome), the arguments that support an offer, a non specified conflict relation among the arguments, and a preference relation between arguments are also all presented in an abstract way.

The authors assume the so-called "ideal" situation, where the agents have complete information about the negotiating agents or that the agents negotiate through a mediator, and that each

agent has a negotiation theory and an argumentation-based reasoning mechanism originally proposed in [2] and further extended in [3]. Furthermore, the authors ignore the arguments' support *beliefs* and consider only the support *offers* (the "object" of negotiation).

They propose the integration of the theories of all agents who are negotiating into a single theory (the so-called *aggregate argumentation system*) which contains the arguments that the agents have as a group. This integration permits a reasoning mechanism to identify the outcomes that are "good" for all negotiating agents. The authors argue that this approach allows for polynomial-time solutions in contrast to the intractability results known for general argumentation frameworks [26].

The authors demonstrate that the credulous conclusions of the aggregate theory are in direct correspondence to the *Pareto optimal* arguments of the theory. In the context of argumentation frameworks, Pareto optimality refers to the argument in the union theory for which there exists no preferred argument.

Amgoud and Vesic, in [10], analyse the role of argumentation in negotiation dialogues, proposing an abstract framework for argumentation-based negotiation where the impact of exchanging arguments on agents' theories is formally described, the different types of solutions in negotiation are investigated, and the added value of argumentation in negotiation dialogues discussed. The authors argue that argumentation can improve the quality of an outcome but never decrease it.

The authors further argue that the notion of *optimal solution* is not defined for argumentation-based negotiation. This is because the literature is not clear what kind of *solution* (or *outcome*) is reached by their dialogues and whether optimal solutions (when they exist) can be reached by a dialogue under such protocols. The paper proposes to cover this gap.

The authors argue that, by exchanging arguments, the theories of the agents (i.e., their mental states) may evolve and thus the status of offers may change. For example, an agent may accept an offer (which was rejected) after receiving a strong argument in favour of this offer. The exchange of information permits *optimal solutions* for both agents.

The paper characterises three types of outcomes (solutions): (i) local solutions; (ii) Pareto optimal solutions; and (iii) ideal solutions. Local and Pareto optimal solutions are the best outcomes at a given step of a dialogue, while an ideal solution is the best solution in general and is time-independent.

In that work, the concepts of *epistemic arguments* and *practical arguments* are presented. An epistemic argument justifies beliefs and is itself based only on beliefs, whereas a practical argument justifies an offer and is built from both beliefs and goals. Epistemic arguments can attack practical arguments, undermining the beliefs used as support of a practical arguments, and practical arguments cannot attack epistemic arguments. The status of an argument using that semantics are: *skeptical*, *credulously* and *rejected* (as in [15], discussed in Section 3.1). Based on this definitions, the offers can be partitioned into three classes: credulous, rejected, and non-supported. Credulous offers are strictly preferred to any non-supported offer, and a non-supported offer is better than a rejected one [10].

Agents in a framework use the same language, and the same definition of arguments, and recognise both arguments and conflicts between arguments. Its theory contains the offers, arguments, the relation between arguments and ordering over arguments. The negotiation is described as moves where the agents exchange offers. When the set of arguments change, the attack relation may change as well since new attacks may appear between the new arguments and the existing ones.

Outcomes/solution can be of two categories: time-dependent and global ones. Time-dependent solutions are good at a given step of a dialogue. Global solutions are the ideal outcomes that should be reached independently from protocol and dialogue. Time-dependent solution can be accepted in step t of a dialogue and no longer in step $t+1$. Optimal solution do not dependent on dialogue step, they are offers that an agent would choose if it had access to all arguments owned by the other agent (as in [26] where the theory of the agents are combined into a single theory to analyse the best outcomes). New arguments allow agents to revise their mental states, thus the best decision for an agent is the one it makes under *complete* information.

As in real life, where people, from the same information, do not necessarily draw the same conclusions, agents have different preferences on arguments, and can have different optimal solutions. In real life, it may also be the case that two people exchange arguments and at the end the negotiation fails; the same occurs between negotiating agents.

The paper proves that argumentation-based negotiation improves the quality of the outcome but never decreases it. They prove that: (i) only argumentative dialogues guarantee that ideal solutions will be reached, of course provided that the protocol is defined in an efficient way; (ii) the argumentative dialogue will lead to an outcome which is at least as good as the outcome that may be reached by a non-argumentative dialogue; (iii) the negotiation ends up with a failure or with a Pareto optimal solution, and that the agent can make more informed decisions; and (iv) arguing allows the agents to make better decisions and to reach outcomes in a rich context [10].

The two papers described above defined important characteristics that we take in consideration in our research, and reinforce the use of defeasible logic (defined in Section 2.2) as reasoning mechanism. This is because defeasible logic has this characteristic of changing the possible conclusions when new information is acquired and the union of the information provides the optimal solution (a conclusion that is guaranteed on that set of arguments). This situation, also, is described in [59], where the work shows how the beliefs of two agents that engage in an argumentation-based dialogue will converge over time (the new information – i.e., beliefs – changes the agents' conclusions).

Rahwan et al. in [71] investigate "interest-based negotiation", a form of argumentation-based dialogues where the agents can exchange information about their goals and alternative ways to achieve these goals [71]. The work proposes a model for reasoning about the interest-based negotiation protocol.

Also, the authors describe the three categories of automated negotiation (game theoretic, heuristic-based, and argumentation). They argue that the work (the framework proposed)

contributes to bridging the gap between the theory and the practice of argumentation-based negotiation.

The work assumes that the resources (in the system) are unique and indivisible, and the reallocation can benefit the agents (reallocation is referred to as a *deal*), and rational agents do not accept deals that result in loss of utility. The work proposes the use of *payments* in order to enable agents compensate each other for accepting deals that result in loss utility. They argue that reaching a deal depends not only on the protocol, but also the strategies of the agents.

The approach proposes that if an agent has goals but not the resource to achieve the goals, and has the information that the other agent has another goal which has the same sub-goal, the other agent will prefers this goal where both agents to achieve its goals. Also propose that new plans can be exchanged between the agents, and new plans can have more utility for the agent. A classical example of this situation is the painting/mirror problem presented by Parsons et al. [58], where an agent has the goal to hang a painting, and the other agent has the goal to hang a mirror. With the exchange of plans both agents can achieve their goals using resource reallocation and the plans previously unknown.

The author argue that "while much has been said about the intuitive advantage of argument-based negotiation over other forms of negotiation, very little has been done on making these intuitions precise" [71], which emphasizes the importance of research in this field.

This work is relevant because the exchange of plans that the agent do is facilitated, as proved in paper, by argumentation-based dialogues which permits the agents acquire information that makes it possible to suggest new plans to the other agent, cooperatively. Another work in the same line is [70], where the authors investigate an argumentation-based dialogue protocol in which agents exchange information about their underlying goals, and such information enables agents to discover mutual goals and thus increases the likelihood of reaching deals.

Rueda and Martínez in [77] propose an interaction language that allows argumentation-based dialogue among collaborative BDI agents. The knowledge that each agent uses for reasoning is formed by its specific knowledge and the knowledge shared with others members of the system. All members of the system are autonomous and rational entities, but have a collaborative attitude, in the sense that when their beliefs do not suffice to reach their goals they request collaboration.

Each agent elaborates arguments as part of its own planning process and justifies its proposals, counter-proposals, and rejections during the negotiation process [77].

The interaction language proposed in this work is presented in the form of *preconditions*, *meaning* (informal meaning), *response* (the possible responses for this locution) and *updates* (the effects over the belief base of the agents).

Hussain and Toni in [33] demonstrate the benefits of the use argumentation-based dialogues in a scenario of resource reallocation. It is assumed that the resources are unique and indivisible. In this work, the agents always justify their requests and responses. The requests are justified with the information that the agent needs the resource and does not have it. The responses given by agents depend on the information that they have, for example, if an agent knows who has

the resource requested, the agent can refuse to give the resource because it does not have it but inform that another agent has the resource, allowing the requesting agent to directly contact that agent.

In this work all interactions are through *tell* performatives, for example:
$tell(X, Y, (request(give(R))because\{needs(Y, R), \neg has(Y, R)\}))$
where agent Y sends to agent X the tell message requesting the resource R and uses as justification that it needs the resource and does not have. The knowledge of the agents is represented through inference rules over a set of *assumptions* based on [16]. Assumptions are represented as Prolog-like rules, for example, $asm(has(X, R)) = not(\neg has(X, R))$, i.e., the agent has the assumption that an agent has a resource when it does not have the information that the agent does not have the resource.

This work is relevant because the justification given by the agents permits reaching the goal faster, showing the benefits of argumentation-based dialogues. The representation and definition which performatives to use at each time step is also very interesting, because it is similar to plans in Jason, where the precondition (or plan context, in Jason) defines what performatives to use, and the argumentation generating is part of the agent's process of planning. Our research follows this line in the definition of the protocol and in the argument generation.

In [86], a model of an argumentation-based deliberation dialogue is presented which shows the benefits of argumentation schemes. The deliberation is over actions, goals, and norms, where the agents can formulate arguments that deal with potential conflicts between the proposed action and other actions, norms, and goals, using the critical questions: (i) is the action possible given other concurrent actions in the plan? (ii) is the action possible according to casual plan constraints? (iii) is there any conflicting norm that regulates actions or states of the world? and (iv) is the goal justified?

A support relation justifies an agent's commitment, and a defeat relation describes a conflict between a task of an agent and a task, a norm, or a goal of the opponent's plan. The authors argue that argumentation is useful in complex collaboration situations, where new information from argumentation-based dialogues allow the agents to reason about alternative plans. In our work, the agents use argumentation-based dialogues to reallocate task. Further, the new information from these dialogues will allow the agent to try alternative plans to solve the problems.

In [85], the authors propose an argumentation-based model for deliberative dialogues based on argumentation schemes. This model facilitates agreements about joint plans by enriching the quality of the dialogue through the exchange of relevant information about plan commitments and norms.

Rahwan and Larson introduce in [67] the so-called *argumentation mechanism design* (ArgMD) which enables the design and analysis of argumentation mechanism for self-interested agents. In that paper the authors also define the notion of a direct-revelation argumentation mechanism where the agents decide which arguments to reveal simultaneously.

In [67] it is argued that argumentation in multi-agents systems is an adversarial process, and the agents in these systems may have conflicting preferences over the arguments which end up being acceptable.

The direct-revelation argumentation mechanism consists in the agent revealing a set of arguments and the (centralised) mechanism for calculating the outcome using the skeptical (grounded) semantics. The agents' preferences are based on *utility* (how good is the outcome to this agent), where more arguments accepted in the outcome means more utility for that particular agent.

The same authors present in [68] a number of preference relations over argumentation outcome. The focus of the paper is *Pareto Optimality*, where it is analysed if an outcome can be improved for one agent without harming the other agent. This work extends previous work of the authors (such as [67]) where a comparison is made between different argumentation semantics using the notion of Pareto Optimality.

The paper uses the approach of *argumentation labelling* [21] where the arguments are labeled with *in*, *out*, or *undec*, depending on whether the argument is, respectively, acceptable, rejected, or undecided with respect to the *outcome* of the argumentation. The arguments are *in* if all their defeaters are *out* and an argument is *out* if at least one of its defeaters is *in*. The labelling approach corresponds to the complete extension, where all *in* arguments in the outcome are conflict-free, self-defending, and contain all arguments they defend.

In [68] the agents are interested in the status (i.e., labelling) of their own arguments and not a particular status of others agent's arguments. In previous work, the authors present the notion of *individual acceptability maximizing preferences* [67], where the agents want to maximize the number of their arguments that are accepted in the outcome of the dialogue. This paper presents other possible preferences of the agents, for example, minimizing rejections in the outcome (*Rejection minimizing preferences*), that minimize the uncertainty in the outcome (*Decisive preferences*), to have all arguments acceptable (*All-or-nothing preferences*), and an agent can prefer to defeat as many arguments as possible (*Aggressive preferences*).

The paper shows that every pareto optimal outcome is a preferred extension. The *grounded extension* is a pareto optimal outcome for agents that have *rejection-minimizing preferences*. The *semi-stable extension* characterises the pareto optimal outcome for agents with *decisive preferences*. An agent that have *all-or-nothing preferences*, there exists a pareto optimal *preferred extension*. For an agent that has *aggressive preferences*, all pareto optimal outcomes are *preferred extensions*.

Previous work of the same authors studies agents' strategic behaviour in argumentation over the *grounded semantics*. In [53], the authors analyse the *preferred semantics* which is a more credulous semantics and allows multiple outcomes (whereas *grounded semantics* produces only one outcome). The paper, also, proposes an analysis of two more refined semantics: the *ideal* and *skeptical-preferred semantics* (which produces only a unique extension).

The work assumes that the defeat relation is known and understood by all agents (abstract argumentation systems). The agents can only reveal arguments once. This mechanism, with this restriction, is called *direct mechanism.* The agents have focal arguments (the argument that is preferred to be accepted over the others), and the paper assumes that the agents do not contain direct or indirect defeats against their own focal argument. Also, the paper uses randomisation to chose among the several preferred possible semantics in the mechanism.

The insertion of randomisation into the mechanism (into ArgMD [67]) incentivises the agents to reveal all their arguments (the agents do not know which preferred extension will be selected, so the agents reveal the arguments with the higher probability to be accepted).

In [69] the authors present the first analysis of the case where agents can lie, using more realistic preference classes (focal arguments, as in [53]). In that paper each agent has a single focal argument it wishes to have accepted (to represent more naturally agent preferences). The paper contributes in providing the first comprehensive analysis of strategy incentives under grounded semantics when agents have focal arguments and to provide the first analysis of incentives when agents can lie in argumentation. [69] presents direct mechanisms for argumentation based on grounded semantics (as in [67]). The mechanism calculates the grounded extension given the arguments revealed by agents.

The authors argue that in many realistic dialogues, each agent is interested in the acceptance of a particular argument. This argument is called *focal argument.* The other arguments are called *instrumental arguments* towards the acceptance of the focal argument. The paper, also, argues that when all agents introduce their arguments, the mechanism demonstrates possible fallacies from the agents. When the agent does not know the indirect defeat for the argument presented by another agent, the mechanism can find this defeat relation and demonstrates to the agent that it is a fallacy.

3.4 Semantics to Agent-oriented Programming Languages

Some work on operational semantics for agent-oriented programming languages can be found in the literature, among which is [87], defining operational semantics for speech-act based communication, which serves as the basis for our work in defining the operational semantics to new speech acts (Chapter 5). In that paper, semantics is given for basic performatives that allow the communication between agents through simple message exchanges. We follow and extend that work with new performatives to allow argumentation-based dialogues, which also requires the exchange of sequences of interactions.

More recently, the work reported in [13] proposed the use of some performatives for argumentation in AgentSpeak and attempted to give semantics to those performatives. Unlike what we present in the Chapter 5, the work in [13], as well as [7], is focused on negotiation and uses an electronic trading scenario. Also, the work in [13] is similar to [87] in treating the communication of a single message exchange and not as a sequence of interactions (i.e., a dialogue) as in our work.

In [22], the authors propose an approach to the operational semantics of agent-oriented programming languages based on a game-theoretic approach to dialogue games (dialogue games have their origin in the philosophy of argumentation). The approach leads to a natural, uniform, and modular way of modeling all the components of the interpreter (agents), including the communication component and the communication protocol. The interpreter behaviour can be abstracted from the operational rules so as to define and prove useful properties.

Operational semantics allows agent programs to be precisely defined and interpreted [20]. "The effort of providing an abstract formalisation of agent behaviour is aimed at the application of formal methods in a rigorous definition and analysis of agents functionality" [22]. This may allow the demonstration of interesting properties of such systems.

McBurney and Parsons [45, 44, 49, 92] study argumentation-based dialogues between agents, and discuss the proof of some properties of dialogues under a given protocol, such as termination, dialogue outcome, and complexity.

The claimed advantage of the approach in [22] is the description in a uniform way of all aspects of an agent-oriented language, including communication. Another claimed advantage in [22] is the modularity, as the language interpreter is seen as composed of modules which are the players of a game.

4. ARGUMENTATION-BASED REASONING IN AGENT-ORIENTED PROGRAMMING LANGUAGE

In this chapter we describe the argumentation-based reasoning mechanism developed. We found reasons in literature for using defeasible logic [51, 52] and its practical implementation as Defeasible Prolog [50] (d-Prolog for short) as a basis for argumentation systems (as described in Section 3.1). We demonstrate that an adaptation of d-Prolog allows the implementation of argumentation-based reasoning in an agent-oriented programming language. The approach allows the agents to reason about rules (defeasible and strict rules) during the executions of plans to achieve their goals as well as to query if an argument is acceptable or not at runtime.

4.1 The approach for argumentation-based reasoning using defeasible logic

We have implemented defeasible reasoning (defeasible logic more specifically) in Jason through a set of Prolog-like rules that had to be modified in order to be processed in Jason (*e.g.*, the cut operator is not available). We have adapted the rule presentation representing defeasible and strict knowledge to a form similar to the approach in [15], as follows:

Facts: facts are represented as in the d-Prolog implementation of defeasible logic, where "Alison is a graduate student" is represented by a simple predicate such as `grad_student(alison);`

Strict Rules: the strict rules are represented as a special predicate `strict_rule(Head,Body)`, where for example "graduate students are students" is represented as `strict_rule(student(X),grad_student(X));`

Defeasible Rules: the defeasible rules are represented as a special predicate `defeasible_rule(Head,Body)`, where "graduate student usually studies hard" is represented as `defeasible_rule(studies_hard(X), grad_student(X));`

Defeater: the defeaters are represented with the predicate `undercut_rule(Head,Body);`

Superiority relation: the superiority relation is represented as `sup(`$Rule_1$`,`$Rule_2$`)` where $Rule_1$ is superior who $Rule_2$.

Another predicate is used to declare the complement of a proposition, for example, *good* is the complement of *bad*, in our representation we use the predicate $comp(good, bad)$ to define the complement. The adaptation of d-Prolog follows the example presented below. The rules are based on logic programming and the syntax and formal semantics of the AgentSpeak language extension can be found in [18].

```
strict_der(Content) :- Content.
strict_der([Content]):- strict_der(Content).
strict_der([First|Rest]):- strict_der(First) & strict_der(Rest).
strict_der(Content) :- strict_rule(Content,Condition) &
                       strict_der(Condition).
```

In this example we show the derivation of *strict rules* using Prolog-like rules in Jason, where first is checked if the queried content is a premise, after if it is a list of one element, after if it is a list of more elements, and finally if it is the *Head* of a strict rule and if the *Condition* (which derives the *Content*) is also strictly derived. These rules allow an agent to query if a content is strictly derived in its knowledge base (remember that strict knowledge is indisputably known).

When an agent needs an argument (facts and inference rules used in the derivation of a *content*) to support a claim in a dialogue, for example, this information is accessible by an second parameter which we call *Arg*. We store each rule and fact, used in the derivation, using the internal action `.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).
```

As described in Section 3.1, this implementation has a well-defined semantics called *defeasible semantics* which defines the acceptability of the arguments. The adaptation of d-Prolog does not change this semantics and the status of any argument can be defined by this formalisation.

4.1.1 Example of Reasoning

An agent has submitted a paper to AAMAS conference and believes that its paper will be accepted, so it will buy its ticket to Paris because it has an argument that conclude *go to Paris to present the paper.*

$$defeasible\_rule(go\_to\_paris\_to\_present(X), accepted(X)).$$
$$defeasible\_rule(accepted(X), submitted(X)).$$
$$submitted(paper).$$

The plan to buy a ticket has the following format (in Jason platform):

$$+!buyTicket(paris): \quad defeasible\_der(go\_to\_paris\_to\_present(paper)) \\ <- buyTicket(paris).$$

Before the agent buys its ticket, its coauthor informs[1] it that the paper was submitted without a required field and this will invalidate its submission and so it will not go to Paris. The new knowledge received is:

$$strict\_rule(\neg go\_to\_paris\_to\_present(X), \neg accepted(X)).$$
$$strict\_rule(\neg accepted(X), incomplete(X)).$$
$$incomplete(paper).$$

The new information changes the conclusion of *go to Paris to present the paper*, and so the plan above no longer applies. The above example demonstrates our implementation, where more sophisticate reasoning are allowed, but to a simple example we argue that this is sufficient.

[1]We do not describe here how this knowledge is acquired; for the moment, it suffices to understand that this information becomes part of the agent's knowledge.

5. SEMANTICS OF SPEECH-ACT FOR ARGUMENTATION-BASED DIALOGUES

In this chapter, we extend the performatives normally available in implementations of the AgentSpeak agent-oriented programming language, such as ***Jason*** [18], as well as other agent programming languages, to enable argumentation-based dialogues between agents. We use a selection of performatives widely used in the argumentation-based dialogue literature (as described in Section 2.4), and we give operational semantics to them. In particular, we also define the semantics of a multi-agent system, i.e., at the social level, extending the work presented in [87], where operational semantics was given to basic speech acts, for which it sufficed to show how the mental attitudes of an individual agent were altered when a message with a particular speech act is received, whereas in this work both sending and receiving dialogue statements alter the social state of the argumentative system. Also, because we need to address the social perspective of a dialogue based on argumentation, we have adapted the operational semantics to include two separate transition systems, at the individual and social levels, and how one affects the other.

Here, we define the formal semantics of speech acts for argumentation-based dialogues building on a computationally grounded semantics for agent mental attitudes [19]. Those mental attitudes are directly involved in the semantics of the argumentation speech acts. This means that if a performative refers to agent beliefs or intentions, this can be concretely realised in a computational system (providing a more principled way to implement real-world argumentation-based agent systems). As a consequence, we also make the semantics more detailed, covering the relation to individual and social changes in the system states, although perhaps slightly more complex to specify. Furthermore, with the clear understanding of the semantics, it allows for agents in a multi-agent system to be independently designed by different programmers [6].

5.1 New Performatives for AgentSpeak

The performatives selected to enable argumentation-based dialogues in AgentSpeak are presented below, along with the intended (informal) meaning:

- `assert`: an agent that sends an `assert` message 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`: the sender declares, to all participants of the dialogue, that it accepts a previous claim of another agent. The receivers of the message become aware of this acceptance.

- `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.

- `question`: the sender desires to know the reasons for a previous claim from another agent. The receiver of the message is committed to defending its claim, so presumably it will provide the support set for its claim.

- `challenge`: the `challenge` performative is similar to `question`, except that the sender of the message is committed to defending a claim contrary to the previous claim of another agent.

Further, performatives `opendialogue` and `closedialogue` are used for creating and concluding dialogues, respectively; `justify` is used to justify agents' position in the dialogue; and two other performatives, `acceptdialogue` and `refusedialogue`, are used by the participants to accept or refuse taking part in a dialogue, respectively.

5.2 The Basis for the Operational Semantics

We define the semantics of speech acts for argumentation-based dialogues in AgentSpeak using operational semantics, a widely used method for giving semantics to programming languages [62]. The operational semantics is given by a set of inference rules that define a transition relation between configurations $\langle AG, D\rangle$ of the multi-agent system[1] where:

- The AG component is a set of tuples $\langle id, Conf\rangle$ representing each agent in the society, where each agent is identified by a unique identifier id and the agent current internal state is represented by $Conf$. The agent state is in fact given by a ***configuration*** of the operational semantics of AgentSpeak as formalised in the existing literature (e.g., [87]); we assume some familiarity with the semantics of AgentSpeak.

- The set of all dialogues in that society, D, is a set of tuples $\langle did, Ags, Status\rangle$ where:

  - did is a dialogue identifier (which is unique for each dialogue within that multi-agent system);
  - Ags is a set of tuples $\langle id, CS\rangle$, where id identifies a particular agent that is participating in the dialogue and CS is its ***commitment store***;
  - $Status$ represents the status of the dialogue and for the time being we assume it is one of only two values: `OPEN` if the dialogue is ongoing and `CLOSED` otherwise.

The agent configuration ($Conf$) is given by a tuple $\langle ag, C, M, T, s\rangle$, originally defined in [87], where:

- ag is a set of beliefs bs and a set of plans ps.

- An agent's circumstance C is a tuple $\langle I, E, A\rangle$ where:

[1] We use only components that are needed to demonstrate the semantics, but we emphasise the existence of other components such as roles, norms, etc.

- I is a set of *intentions* $\{i, i', \ldots\}$. Each intention i is a stack of partially instantiated plans.
- E is a set of *events* $\{(te, i), (te', i'), \ldots\}$. Each event is a pair (te, i), where te is a triggering event and i is an intention — a stack of plans in case of an internal event, or the empty intention T in case of an external event. For example, when the belief revision function (which is not part of the AgentSpeak interpreter but rather of the agent's overall architecture), updates the belief base, the associated events — i.e., additions and deletions of beliefs — are included in this set. These are called *external* events; internal events are generated by additions or deletions of goals from plans currently executing.
- A is a set of *actions* to be performed in the environment.

- M is a tuple $\langle In, Out, SI \rangle$ whose components characterise the following aspects of communicating agents (note that communication is typically asynchronous):
  - In is the mail inbox: the multi-agent system runtime infrastructure includes all messages addressed to this agent in this set. Elements of this set have the form $\langle mid, id, ilf, cnt \rangle$, where mid is a message identifier, id identifies the sender of the message, ilf is the illocutionary force of the message, and cnt its content: a (possibly singleton) set of AgentSpeak predicates or plans, depending on the illocutionary force of the message.
  - Out is where the agent posts messages it wishes to send; it is assumed that some underlying communication infrastructure handles the delivery of such messages. Messages in this set have exactly the same format as above, except that here id refers to the agent to which the message is to be sent.
  - SI is used to keep track of intentions that were suspended due to the processing of communication messages; the intuition is as follows: intentions associated with illocutionary forces that require a reply from the interlocutor are suspended, and they are only resumed when such reply has been received.

- When giving semantics to an AgentSpeak agent's reasoning cycle, it is useful to have a structure which keeps track of temporary information that may be subsequently required within a reasoning cycle. T is a tuple $\langle R, Ap, \iota, \varepsilon, \rho \rangle$ with such temporary information; these components are as follows:
  - R is the set of *relevant plans* (for the event being handled).
  - Ap is the set of *applicable plans* (the relevant plans whose contexts are true).
  - ι, ε, and ρ record a particular intention, event, and applicable plan (respectively) being considered along the execution of one reasoning cycle.

- The current step within an agent's reasoning cycle is symbolically annotated by $s \in \{\mathsf{ProcMsg}, \mathsf{SelEv}, \mathsf{RelPl}, \mathsf{ApplPl}, \mathsf{SelAppl}, \mathsf{AddIM}, \mathsf{SelInt}, \mathsf{ExecInt}, \mathsf{ClrInt}\}$. These labels stand for, respectively: processing a message from the agent's mail inbox, selecting an event from the set

of events, retrieving all relevant plans, checking which of those are applicable, selecting one particular applicable plan (the intended means), adding the new intended means to the set of intentions, selecting an intention, executing the selected intention, and clearing an intention or intended means that may have finished in the previous step.

- The semantics of AgentSpeak makes use of "selection functions" which allow for user-defined components of the agent architecture. We use here only the S_M functions, as originally defined in [87]; the *select message* function is used to select one message from an agent's mail inbox.

In the interests of readability, we adopt the following notational conventions in our semantics rules:

- If C is an AgentSpeak agent circumstance, we write C_E to make reference to the E component of C, and similarly for other components of the multi-agent system and of the configuration of each agent.

- We write AG^{id} to identify the agent represented by that id in the set of agents AG. We use this whenever the component corresponds to a set of tuples $\langle id, \ldots \rangle$. Also, if AG is a set of tuples $\langle id, Conf \rangle$, then we refer to a configuration ($Conf$) of one agent (identified by id) in AG by AG^{id}_{Conf}.

- We write $b[d(did), s(id)]$ to identify the origin of a belief related to a dialogue, where did is a dialogue identifier, and id an agent identifier (d refers to $dialogue$ and s refers to $source$). Whenever an agent makes a statement related to a dialogue, the dialogue identifier did is added as an annotation.

- We use two transitions to represent the state change of the multi-agent system, where the transition $\longrightarrow_{AS}$ (transition of the configuration of an individual agent) is part of the transition $\longrightarrow_{DS}$ (the transition of the multi-agent system). So each transition in the agent configuration also causes a transition in the multi-agent system configuration exactly in the component AG^{aid}_{Conf}, where aid refers to the identifier of the agent which went through the transition.

Also, we use aid to refer to the agent that is executing an internal action of interest or receiving a message. Finally, we make use of a function called CTJ (where CTJ stands for "care to justify") that returns TRUE if the agent wishes to justify its previous assertion (this depends on the agent's reasoning and makes reference to agents' autonomy).

5.3 Semantic Rules for new Internal Actions

In this section we give semantics to the internal actions .opendialogue and .closedialogue that are necessary to create a new dialogue and close a dialogue, respectively.

$$\frac{T_\iota = i[head \leftarrow \texttt{.opendialogue}(Agents);h]}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \\ (b) & \langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow_{AS} \langle ag', C', M', T, \mathsf{ProcMsg}\rangle\end{array}} \text{(ExecActOpenDialogue)}$$

(a) where:

$$\begin{aligned} D' &= D \cup \{\langle did, Ags, \texttt{OPEN}\rangle\} \text{ where} \\ &\quad Ags = \{\langle id, CS\rangle | id \in Agents \text{ and } CS = \{\}\} \\ &\quad \text{with } did \text{ a new dialogue identifier} \\ AG'^{aid}_{Conf} &= \text{the transition given by (b)} \end{aligned}$$

(b) where:

$$\begin{aligned} M'_{Out} &= M_{Out} \cup \{\langle mid, id, \texttt{opendialogue}, did\rangle\} \\ &\quad \text{for each } id \in (Agents \setminus \{aid\}) \\ M'_{SI} &= M_{SI} \cup \{(mid, i[head \leftarrow h], Agents)\}, \\ &\quad \text{with } mid \text{ a new message identifier;} \\ ag'_{bs} &= ag_{bs} + \texttt{dialogue}(did)[s(\texttt{self})] \\ C'_I &= C_I \setminus \{T_\iota\} \end{aligned}$$

Internal Action .opendialogue: The agent that intends to start a new dialogue performs the internal action .opendialogue($Agents$), where $Agents$ is a set of agents with whom the agent wants to have a dialogue. This action creates a new dialogue (in our implementation, an artifact) that is represented by a tuple $\langle did, Ags, Status\rangle$, with did being the *dialogue identifier* (that is unique in the system), Ags is the set of agents participating in the dialogue, and $Status$ is the dialogue status. Ags is a set of tuples of the form $\langle id, CS\rangle$, its cardinality being the number of agents in $Agents$, with id an agent identifier and CS its *commitment store* (where its positions in the dialogue are stored). The agent's commitment store CS is initially empty, i.e., CS is an empty set. The status can assume one of two values, either OPEN or CLOSED, depending on whether the dialogue is open or closed respectively; a dialogue always starts with an OPEN status.

After the dialogue was created, a message is sent to all agents requesting their participation, and the intention that started the action execution is suspended waiting for all replies from the agents (accepting or declining to participate in the dialogue).

$$\frac{T_\iota = i[head \leftarrow \texttt{.closedialogue}(did);h]}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \\ (b) & \langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow_{AS} \langle ag, C', M', T, \mathsf{ProcMsg}\rangle\end{array}} \text{(ExecActCloseDialogue)}$$

(a) where:

$$\begin{array}{rcl} D' & = & (D \setminus \{\langle did, Ags, \texttt{OPEN}\rangle\}) \cup \{\langle did, Ags, \texttt{CLOSED}\rangle\} \\ AG'^{aid}_{Conf} & = & \text{the transition given by (b)} \end{array}$$

(b) where:

$$\begin{array}{rcl} M'_{Out} & = & M_{Out} \cup \{\langle mid, id, \texttt{closedialogue}, did\rangle\} \\ & & \text{for each } Ags_{id} \in (D^{did}_{Ags} \setminus \{Ags^{aid}\}) \\ C'_I & = & (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{array}$$

Internal Action `.closedialogue`: This internal action is used by an agent when it decides to finish a dialogue. The action updates the status of the dialogue to `CLOSED` and sends a message to each agent to inform that the dialogue has been closed.

5.4 Semantic Rules for Sending the New Performatives

In this section, we give semantics for sending the new performatives which allow argumentation-based dialogues, showing how it affects the state of the agent and the state of the dialogue.

$$\frac{\begin{array}{c} T_\iota = i[head \leftarrow \texttt{.send}(did, \texttt{assert}, p);h] \\ p \notin CS \qquad \langle aid, CS\rangle \in D^{did}_{Ags} \end{array}}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \\ (b) & \langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow_{AS} \langle ag, C', M', T, \mathsf{ProcMsg}\rangle\end{array}} \text{(ExecActSndAssert)}$$

where:

$$\begin{array}{rcl} \textit{(a)}\ D'^{did}_{Ags} & = & (D^{did}_{Ags} \setminus \{\langle aid, CS\rangle\}) \cup \{\langle aid, CS'\rangle\} \\ & & \text{with } CS' = CS \cup \{p\} \\ AG'^{aid}_{Conf} & = & \text{the transition given by (b)} \\ \textit{(b)}\ M'_{Out} & = & M_{Out} \cup \{\langle mid, id, \texttt{assert}, p[d(did)]\rangle\} \\ & & \text{for each } Ags_{id} \in (D^{did}_{Ags} \setminus \{Ags^{aid}\}) \\ C'_I & = & (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{array}$$

Internal Action `.send` **with** `assert`: The action `.send` with performative `assert` updates the CS of the agent that performs the action and sends, to all agents in the dialogue, a message stating that the sender is willing to defend this claim.

The agent can use *assertion attitudes* as defined in [57, 60], but in any case the agent can only assert a formula it did not previously assert; that is, an agent cannot assert again formulas that are already in its CS.

Another important point to be noticed is that an assertion is always made to a particular dialogue (identified by *did*) and not to a specific agent; this is because the agents will introduce new claims to be defended to all agents participating in the dialogue and not to an individual agent.

$$\frac{T_\iota = i[head \leftarrow .\texttt{send}(tid, \texttt{accept}, p[d(did)]);h]}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \\ (b) & \langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow_{AS} \langle ag, C', M', T, \mathsf{ProcMsg}\rangle\end{array}} \text{(ExecActSndAccept)}$$

where:

(a)	D'^{did}_{Ags}	$=$	$(D^{did}_{Ags} \setminus \{\langle aid, CS\rangle\}) \cup \{\langle aid, CS'\rangle\}$
			with $CS' = CS \cup \{p\}$
	AG'^{aid}_{Conf}	$=$	the transition given by (b)
(b)	M'_{Out}	$=$	$M_{Out} \cup \{\langle mid, id, \texttt{accept}, p[d(did)]\rangle\}$
			for each $Ags_{id} \in (D^{did}_{Ags} \setminus \{Ags^{aid}\})$
	C'_I	$=$	$(C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\}$

Internal Action `.send` **with** `accept`: The action `.send` with performative `accept` updates the CS of the agent that performs the action and sends, to all agents in the dialogue, a message stating that the agent accepts the claim made by another agent identified by *tid*. Note that p (the formula that was accepted) has the annotation $[d(did)]$, this means that p has been previously asserted in that dialogue (identified by *did*). In other words, an agent can only accept a claim made by another agent in that same dialogue.

$$\frac{\begin{array}{c}T_\iota = i[head \leftarrow .\texttt{send}(did, \texttt{justify}, S);h] \\ S \notin CS \qquad \langle aid, CS\rangle \in D^{did}_{Ags}\end{array}}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \\ (b) & \langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow_{AS} \langle ag, C', M', T, \mathsf{ProcMsg}\rangle\end{array}} \text{(ExecActSndJustify)}$$

where:

(a)	D'^{did}_{Ags}	$=$	$(D^{did}_{Ags} \setminus \{\langle aid, CS\rangle\}) \cup \{\langle aid, CS'\rangle\}$
			with $CS' = CS \cup \{S\}$
	AG'^{aid}_{Conf}	$=$	the transition given by (b)
(b)	M'_{Out}	$=$	$M_{Out} \cup \{\langle mid, id, \texttt{justify}, S[d(did)]\rangle\}$
			for each $Ags_{id} \in (D^{did}_{Ags} \setminus \{Ags^{aid}\})$
	C'_I	$=$	$(C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\}$

Internal Action `.send` **with** `justify`: The action `.send` with performative `justify` updates the CS of the agent that performs the action and sends, to all agents in the dialogue, a message with an argument which defends the agent's position. The justification is always made to a particular dialogue (identified by *did*) and not to a specific agent; this is because the agent will introduce new information (defending a previous position) to be considered by all agents

participating in the dialogue and not just the individual agent(s) who might have questioned the previous position.

$$\frac{T_\iota = i[head \leftarrow .\texttt{send}(did, \texttt{retract}, p[d(did)]);h]}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \\ (b) & \langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow_{AS} \langle ag, C', M', T, \mathsf{ProcMsg}\rangle\end{array}} \text{(ExecActSndRetract)}$$

where:

(a)	D'^{did}_{Ags}	$=$	$(D^{did}_{Ags} \setminus \{\langle aid, CS\rangle\}) \cup \{\langle aid, CS'\rangle\}$
			with $CS' = CS \setminus \{p\}$
	AG'^{aid}_{Conf}	$=$	the transition given by (b)
(b)	M'_{Out}	$=$	$M_{Out} \cup \{\langle mid, id, \texttt{retract}, p[d(did)]\rangle\}$
			for each $Ags_{id} \in (D^{did}_{Ags} \setminus \{Ags^{aid}\})$
	C'_I	$=$	$(C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\}$

Internal Action `.send` **with** `retract`: The agent performs this internal action to retract a previous claim that the agent itself asserted in that dialogue. The agent's CS is updated with the removal of the given formula. A message is sent to each agent in the dialogue informing the decision of that agent to retract its previous claim.

$$\frac{T_\iota = i[head \leftarrow .\texttt{send}(id, \texttt{question}, p[d(did)]);h]}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D,\rangle \\ (b) & \langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow_{AS} \langle ag, C', M', T, \mathsf{ProcMsg}\rangle\end{array}} \text{(ExecActSndQuestion)}$$

where:

(a)	AG'^{aid}_{Conf}	$=$	the transition given by (b)
(b)	M'_{Out}	$=$	$M_{Out} \cup \{\langle mid, id, \texttt{question}, p[d(did)]\rangle\}$
	C'_I	$=$	$(C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\}$

Internal Action `.send` **with** `question`: The action `.send` with performative `question` is used when an agent wants to question another agent about an assertion it has previously made. This message is sent only to the agent that previously made the assertion.

$$\frac{T_\iota = i[head \leftarrow \texttt{.send}(tid, \texttt{challenge}, p[d(did)]);h]}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \\ (b) & \langle ag, C, M, T, \mathsf{ExecInt}\rangle \longrightarrow_{AS} \langle ag, C', M', T, \mathsf{ProcMsg}\rangle\end{array}} \text{(ExecActSndChallenge)}$$

where:

(a)
$$\begin{aligned} D'^{did}_{Ags} &= (D^{did}_{Ags} \setminus \{\langle aid, CS\rangle\}) \cup \{\langle aid, CS'\rangle\} \\ &\quad \text{with } CS' = CS \cup \{\neg p\} \\ AG'^{aid}_{Conf} &= \text{the transition given by (b)} \end{aligned}$$

(b)
$$\begin{aligned} M'_{Out} &= (M_{Out} \cup \{\langle mid, tid, \texttt{challenge}, p[d(did)]\rangle\}) \\ &\quad \cup \{\langle mid, id, \texttt{assert}, \neg p[d(did)]\rangle\} \\ &\quad \text{for each } Ags_{id} \in (D^{did}_{Ags} \setminus \{Ags^{aid} \cup Ags^{id}\}) \\ C'_I &= (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{aligned}$$

Internal Action `.send` **with** `challenge`: The action `.send` with the performative `challenge` is performed when an agent wants to challenge another agent about an assertion it previously made. Differently from the question performative, when an agent makes a challenge move it is willing to defend a claim contrary to the claim of the other agent.

The message with a performative `challenge` is sent only to the agent that made the previous claim. As the agent is willing to defend its claim, messages are sent to all agents in the dialogue with the respective `assert` messages.

5.5 Semantic Rules for Receiving the New Performatives

In this section we give semantics for receiving the new performatives that allow argumentation-based dialogues, showing how they affect the state of the agent and the state of the dialogue.

$$\frac{\begin{array}{c}S_M(M_{In}) = \langle mid, id, \texttt{acceptdialogue}, did\rangle \\ (mid, i, Set) \in M_{SI} \text{ (for some intention } i) \qquad id \in Set\end{array}}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D, \rangle \\ (b) & \langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow_{AS} \langle ag, C', M', T, \mathsf{ExecInt}\rangle\end{array}} \text{(AcceptDialogue)}$$

(a) where:
$$AG'^{aid}_{Conf} = \text{the transition given by (b)}$$

(b) where:
$$\begin{aligned} M'_{In} &= M_{In} \setminus \{\langle mid, id, \texttt{acceptdialogue}, did\rangle\} \\ M'_{SI} &= (M_{SI} \setminus \{(mid, i, Set)\}) \cup \{(mid, i, Set')\} \\ &\quad \text{with } Set' = Set \setminus \{id\}; \\ C'_I &= \begin{cases} C_I \cup \{i\} & \text{if } Set' = \{\} \\ C_I & \text{if } Set' \neq \{\} \end{cases} \end{aligned}$$

Receiving an `acceptdialogue` **Message:** The sender of the message is removed from the set of agents that are expected respond. If all agents have now responded, the intention can be resumed, otherwise the intention continues suspended.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, \texttt{refusedialogue}, did\rangle \\ (mid, i, Set) \in M_{SI} \text{ (for some intention } i) \qquad id \in Set \\ (a) \qquad \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \\ (b) \quad \langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow_{AS} \langle ag, C', M', T, \mathsf{ExecInt}\rangle \end{array}}{} \text{(RefuseDialogue)}$$

(a) where:

$$\begin{aligned} D'^{did}_{Ags} &= D^{did}_{Ags} \setminus \{\langle id, CS\rangle\} \\ AG'^{aid}_{Conf} &= \text{the transition given by (b)} \end{aligned}$$

(b) where:

$$\begin{aligned} M'_{In} &= M_{In} \setminus \{\langle mid, id, \texttt{refusedialogue}, did\rangle\} \\ M'_{SI} &= (M_{SI} \setminus \{(mid, i, Set)\}) \cup \{(mid, i, Set')\} \\ &\quad \text{with } Set' = Set \setminus \{id\}; \\ C'_I &= \begin{cases} C_I \cup \{i\} & \text{if } Set' = \{\} \\ C_I & \text{if } Set' \neq \{\} \end{cases} \end{aligned}$$

Receiving a `refusedialogue` **Message:** The sender of the message is removed from the set of agents that are expected respond, as well as from the set of agents participating in the dialogue. If all agents have replied, the intention can be resumed. It should be useful for the agent to check, after resuming the intention, if the set of agents remaining in the dialogue is sufficient for it to continue the dialogue given the purposes for which it originally opened this dialogue.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, sid, \texttt{assert}, p[d(did)]\rangle \\ (a) \qquad \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D, \rangle \\ (b) \quad \langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow_{AS} \langle ag', C', M', T, \mathsf{ExecInt}\rangle \end{array}}{} \text{(Assert)}$$

where:

$$\begin{aligned} &\textit{(a)} \quad AG'^{aid}_{Conf} = \text{the transition given by (b)} \\ &\textit{(b)} \quad M'_{In} = M_{In} \setminus \{\langle mid, sid, \texttt{assert}, p[d(did)]\rangle\} \\ &\qquad ag'_{bs} = ag_{bs} + p[d(did), s(sid)] \\ &\qquad C'_E = C_E \cup \{\langle +p[d(did), s(sid)], \mathsf{T}\rangle\} \end{aligned}$$

Receiving an `assert` **Message:** The claim asserted in the dialogue is added to the belief base of the receiver with an annotation of the dialogue identifier $d(did)$ and the identifier of the agent that asserted the claim as the source of that information $s(sid)$. The agent that received the message can react to this claim because of the event generated by the belief addition, as usual in AgentSpeak. Whether an agent accepts or not the claim made by another agent depends on its *acceptance attitude* as described in [57, 60], which depends on if the agent has or not an acceptable argument to or against the claim.

$$\frac{S_M(M_{In}) = \langle mid, sid, \texttt{accept}, p[d(did)]\rangle}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D, \rangle \\ (b) & \langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow_{AS} \langle ag', C', M', T, \mathsf{ExecInt}\rangle\end{array}} \quad \text{(Accept)}$$

where:

(a)	AG'^{aid}_{Conf}	=	the transition given by (b)
(b)	M'_{In}	=	$M_{In} \setminus \{\langle mid, sid, \texttt{accept}, p[d(did)]\rangle\}$
	ag'_{bs}	=	$ag_{bs} + p[d(did), s(sid)]$
	C'_E	=	$C_E \cup \{\langle +p[d(did), s(sid)], \mathsf{T}\rangle\}$

Receiving an `accept` Message: This message means an agent (identified by *sid*) accepts a claim previously made, as part of this dialogue, by another agent. The receiver of the message becomes aware of this acceptance.

$$\frac{S_M(M_{In}) = \langle mid, sid, \texttt{retract}, p[d(did)]\rangle}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D, \rangle \\ (b) & \langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow_{AS} \langle ag', C', M', T, \mathsf{ExecInt}\rangle\end{array}} \quad \text{(Retract)}$$

where:

(a)	AG'^{aid}_{Conf}	=	the transition given by (b)
(b)	M'_{In}	=	$M_{In} \setminus \{\langle mid, sid, \texttt{retract}, p[d(did)]\rangle\}$
	ag'_{bs}	=	$ag_{bs} - p[d(did), s(sid)]$
	C'_E	=	$C_E \cup \{\langle -p[d(did), s(sid)], \mathsf{T}\rangle\}$

Receiving a `retract` Message: This message means an agent, identified by *sid*, is withdrawing its earlier assertion. The formula is removed from belief base of the receiver of the message, with the appropriate source and dialogue annotation.

$$\frac{\begin{array}{c}S_M(M_{In}) = \langle mid, sid, \texttt{question}, p[d(did)]\rangle \\ \mathsf{CTJ}(p) = \texttt{TRUE}\end{array}}{\begin{array}{ll}(a) & \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \\ (b) & \langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow_{AS} \langle ag, C, M', T, \mathsf{ExecInt}\rangle\end{array}} \quad \text{(Question)}$$

where:

(a)	D'^{did}_{Ags}	=	$(D^{did}_{Ags} \setminus \{\langle aid, CS\rangle\}) \cup \{\langle aid, CS'\rangle\}$
			with $CS' = CS \cup \{Sp\}$
	AG'^{aid}_{Conf}	=	the transition given by (b)
(b)	M'_{In}	=	$M_{In} \setminus \{\langle mid, sid, \texttt{question}, p[d(did)]\rangle\}$
	M'_{Out}	=	$M_{Out} \cup \{\langle mid, id, \texttt{justify}, Sp[d(did)]\rangle\}$
			for each $Ags_{id} \in (D^{did}_{Ags} \setminus \{Ags^{aid}\})$
			where $Sp \models p$ *and* $Sp \in ag_{bs}$

Receiving a `question` Message: If the agent can or wants to reply, in keeping with agent autonomy (this is represented in the semantics through a CTJ function which is meant to be agent specific), then the agent's CS will be updated with the support of the previous claim p, and the support will be also sent to all other agents in the dialogue.

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, sid, \texttt{justify}, Sp[d(did)]\rangle \\ \hline (a) \qquad \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D, \rangle \end{array}}{(b) \quad \langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow_{AS} \langle ag', C', M', T, \mathsf{ExecInt}\rangle} \quad \text{(Justify)}$$

where:

(a)	AG'^{aid}_{Conf}	$=$	the transition given by (b)
(b)	M'_{In}	$=$	$M_{In} \setminus \{\langle mid, sid, \texttt{justify}, Sp[d(did)]\rangle\}$
	and for each $p \in Sp$:		
	ag'_{bs}	$=$	$ag_{bs} + p[d(did), s(sid)]$
	C'_E	$=$	$C_E \cup \{\langle +p[d(did), s(sid)], \mathsf{T}\rangle\}$

Receiving a `justify` Message: This is similar to the assert performative, except for the fact that the content of the message is a set of formulæ that justify the previous claim of the sender of the message (identified by sid).

$$\frac{\begin{array}{c} S_M(M_{In}) = \langle mid, sid, \texttt{challenge}, p[d(did)]\rangle \\ \mathsf{CTJ}(p) = \texttt{TRUE} \\ \hline (a) \qquad \langle AG, D\rangle \longrightarrow_{DS} \langle AG', D'\rangle \end{array}}{(b) \quad \langle ag, C, M, T, \mathsf{ProcMsg}\rangle \longrightarrow_{AS} \langle ag', C, M', T, \mathsf{ExecInt}\rangle} \quad \text{(Challenge)}$$

where:

(a)	D'^{did}_{Ags}	$=$	$(D^{did}_{Ags} \setminus \{\langle aid, CS\rangle\}) \cup \{\langle aid, CS'\rangle\}$
			with $CS' = CS \cup \{Sp\}$;
	AG'^{aid}_{Conf}	$=$	the transition given by (b)
(b)	M'_{In}	$=$	$M_{In} \setminus \{\langle mid, sid, \texttt{challenge}, p[d(did)]\rangle\}$
	M'_{Out}	$=$	$M_{Out} \cup \{\langle mid, id, \texttt{justify}, Sp[d(did)]\rangle\}$
			for each $Ags_{id} \in (D^{did}_{Ags} \setminus \{Ags^{aid}\})$
			where $Sp \models p$ *and* $Sp \in ag_{bs}$
	ag'_{bs}	$=$	$ag_{bs} + \neg p[d(did), s(sid)]$

Receiving a `challenge` Message: If the agent can or wants to reply (we assume the CTJ function determines whether that is the case or not), the agent's CS is updated with the support of the previous claim p and the support is also sent to all other agents in the dialogue. In addition to the question performative, this rule adds that the agent identified by sid (i.e., the sender of the message) is willing to defend the claim contrary to the previous claim that is being challenged.

6. PROTOCOL FOR ARGUMENTATION-BASED DIALOGUE

In this chapter we describe and formalise the argumentation-based dialogues we propose as a dialogue game. Towards this formalisation, we describe the protocol and the dialogue rules which consider the agents' strategy participating in these dialogues. At the end, we make initial efforts towards proving some properties of this protocol considering the agents' strategies (formalised as dialogue rules). This protocol has been developed specifically for the domain of task reallocation in cooperative multi-agent systems. However, we argue that the protocol can be used for similar domains with similar characteristics (especially cooperativeness).

6.1 Dialogue Game

In this section we describe the dialogue game in terms of components and specification; this model is based in [45] and the work in [44] discussed in Section 2.3. As described in [45], the topic of discussion in a dialogue needs to be represented in some logical language; in our approach the representation follows the predicates described in Chapter 4. The elements that correspond to the dialogue game specification in our domain are:

Commencement Rule: In our approach, an agent can start a dialogue when it needs to reallocate a task which is attributed to it and which the agent is failing to complete. The agent starts the dialogue executing an `assert` move suggesting it will not execute the task (when the agent has information that the task does not need to be executed) or an `assert` move suggesting that another agent executes the task. To start a dialogue, the agent needs to have an argument which allows it to conclude the assertion (the subject of the dialogue), and this is formally defined below (in Section 6.4).

Locutions: In our approach, the agents can use the set of locutions described in the Chapter 5, following the protocol described below (in Section 6.3).

Combination Rules: In our approach, the combination rules depend on the strategy of the agent (corresponding the agent attitudes to assert and to accept claims in the dialogue [60, 57]). The combination rules are formally described in Section 6.4.

Commitments: In our approach, the update of commitments of the participants is formally defined in the semantics of the speech acts used (as described in Chapter 5), where each speech-act/performative introduces or removes commitments of participants in accordance with the literature [5, 57, 60].

Termination Rules: In our approach, the dialogue ends when either the *opponent* executes the `accept` move, accepting the `subject` of the dialogue; or the *proponent* closes the dialogue, because it cannot make the *opponent* accept the `subject` of the dialogue (the *proponent* either accepts the justification from *opponent* which introduces reasons for it not to accept the first assertion or its arguments fail to convince the *opponent* to accept the `subject`). These rules are formally defined in Section 6.4.

6.2 Agent Configuration

In this Section, we describe the agent configuration. We assume that two agents (as in [5, 23, 24, 49]) will participate of an argumentation-based dialogue (in this section, we will use a and b to refer to these two agents). The agent that introduces the subject of the dialogue is called *proponent*, and the other agent participating in the dialogue is called *opponent* (we use Pr and Op, respectively, to refer to them as in [23, 24]). Each agent has a knowledge base which contains facts and rules. The agents are capable of generating acceptable arguments from this knowledge base, as well as evaluating the acceptability of the arguments (when new information is available) as described in the Chapter 4.

We assume that agents exchange arguments for and against the subject (the predicate that will be discussed in the dialogue). Considering the application domain and that agents are cooperatives in this domain, the agents are not allowed to question or to challenge the support of arguments. For example, if an agent argues that its owner cannot execute a task because they are late, it would not be appropriate (considering the cooperative system and the application domain) to doubt this (i.e., we assume that agents do not lie).

The agents rationally decide about the next move to play (e.g., accept, question, etc.) based on their argumentation systems (i.e., depending on whether the agents have or not an argument for or against a certain claim). These decisions taken by the agents correspond to their strategies. Each agent has a commitment store (CS) which is accessible to all agents participating in the dialogue, but only the owner agent can update the information in its commitment store (the other agents can only *read* its contents).

The commitment store of each agent is updated following the semantics presented in Chapter 5, depending on the performative used in that interaction. We use CS_a to represent the *commitment store* of agent a at the current moment.

The agents can build an acceptable argument S which supports a claim/predicate p (denoted as $S \models p$) from its knowledge base and the commitment store of the other participant. For example, agent a can build an acceptable argument S (which supports a predicate p) from its knowledge base (KB_a) and from the commitment store of b (CS_b) (denoted $(KB_a \cup CS_b) \models S$) [1].

We use the notation $\overline{p}$ to describe the complement and contrary of a predicate, for example, we can describe the complement of p as $\overline{p}$ where $\overline{p} = \neg p$. We can have p as "good", and "good" is the contrary of "bad"; in this case "bad" can be also denoted by $\overline{p}$.

[1] The commitment store of the agent a (CS_a) is a subset of the knowledge base of a (KB_a), formally $CS_a \subseteq KB_a$.

6.3 Dialogue Game Protocol

In this section, we describe the dialogue game protocol which restricts the moves allowed by agents. The sequence of moves allowed is represented in Figure 6.1, where the white circle represents the start move and the black circle the finish move.

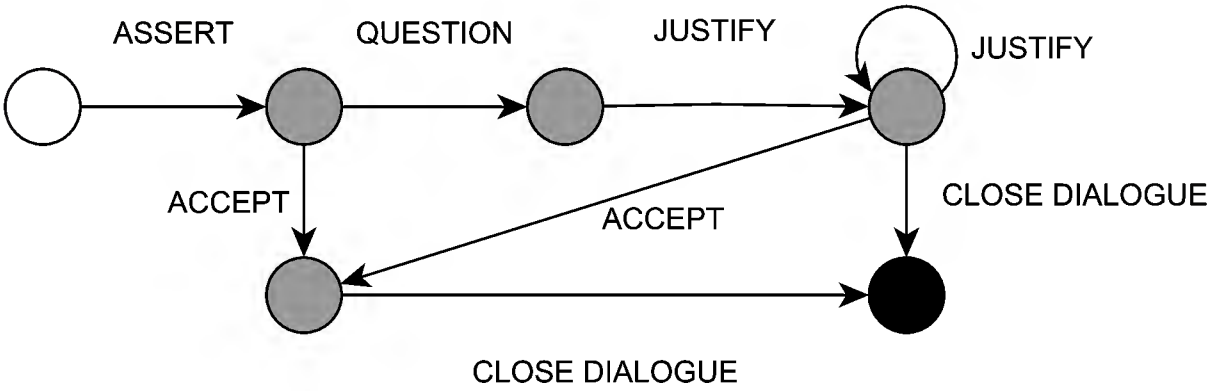

Figure 6.1 - Allowed Moves.

Regarding Figure 6.1, we can make the following considerations:

- A dialogue game starts with an agent executing an `assert` move, introducing the `subject` of the dialogue.
- When an agent receives an `assert` message, it can either accept, executing an `accept` move, or question, executing a `question` move. The choice of the agent depends on its strategy, as described in the next section.
- When an agent receives a `question` message, it can only execute a `justify` move (the agent is committed to provide an argument which supports the previously asserted predicate).
- When an agent receives a `justify` message, it can either accept the `subject` of the dialogue (executing an `accept` move) or provide an argument not to accept the `subject` of the dialogue (executing a `justify` move). This choice depends on the agent's strategy, as described in the next section. Further, after receiving a `justify` message, the agent may close the dialogue as well.
- When an agent receives an `accept` message it may close the dialogue.

Figure 6.1 describes all possible allowed moves by the agents. The protocol also restricts who can execute the moves, as well as who can close the dialogue:

- As described, the dialogue starts with an agent executing an `assert` move, introducing the `subject` of the dialogue. We call the agent that starts the dialogue the *proponent*.
- An agent can never repeat a move with the same content; this is guaranteed by the semantic specification and the structure which maintains the information introduced in the dialogue as commitments (the *commitment store*).
- The dialogue can be closed only by the *proponent*, so there is only a single agent that can execute the *closedialogue* move in a particular dialogue, namely the *proponent*.
- The dialogue ends when the proponent executes a *closedialogue* move. We prove that a dialogue following our protocol will always terminate in the next sections.

6.4 Dialogue Rules

Now, in this section, we describe the *dialogue rules* (which are defined based on the protocol) that govern the interactions between the agents considering their strategies, where each agent moves by performing the allowed utterances. These rules (which correspond to a dialogue game [45]) are expressed as if-then rules, which are then easy to implement.

The dialogue rules specify the moves the other player can make next, and so specify the *protocol* under which the dialogue takes place [5] (in our case, the protocol is the one specified in Section 6.3).

Definition 8 (Dialogue). *A dialogue is formally represented as a tuple* $\langle \mathtt{MO}, \mathtt{DI} \rangle$, *where* `MO` *is a finite set of moves, containing all moves made thus far in that dialogue which are exchanged according to* `DI`, *a set of dialogue rules that also take into account the strategy of each agent.*

Definition 9 (Dialogue Move). *We denote a move in* `MO` *as* $\mathtt{M}^{\mathtt{i}}(\alpha, \beta, \mathtt{cont}, \mathtt{t})$, *where* `i` *is the type of move made by agent* α *and addressed to agent* β *at time* `t` *regarding content* `cont`. *We consider the following set of types of moves, denoted by* `P`, *which contains the performatives presented and formalised in Chapter 5:* `assert`, `accept`, `question`, `justify`, *and* `closedialogue`. *The content of a move (*`cont`*) can be an argument (a set of predicates and rules) or a predicate (for example, in an* `assert` *move the content is a predicate and in a* `justify` *move the content will be an argument that supports a claim/predicate uttered in a previous* `assert` *move).*

The dialogue rules in `DI` indicate the possible moves that an agent can make following a previous move of the other agent. The formalisation we give here follows the work in [14]). To define the dialogue rules, we use a set of condition (denoted by `C`) which reflect the agents' strategies. Formally, we have:

Definition 10 (Dialogue Rules). *Dialogue rules can assume one of two forms:*

- *First, we have dialogue rules that specify which moves are allowed given the previous move and conditions (corresponding to the combination rules of the dialogue game).*

$$\bigwedge_{\substack{0<k\leq n_i,\\ i,j\in P}} (M^i(\alpha,\beta,\mathtt{cont},t) \wedge C_k \Rightarrow M^j_k(\beta,\alpha,\mathtt{cont}_k,t')$$

where P *is the set of move types,* M^i *and* M^j *are in* MO, $t < t'$ *and* n_i *is the number of allowed communicative acts that* β *could perform after receiving a move of type* i *from* α.

- *Second, we have the initial conditions (corresponding to the commencement rules of the dialogue game), which do not require that any move was previously executed.*

$$\bigwedge_{\substack{0<k\leq n,\\ j\in P}} (C_k \Rightarrow M^j_k(\alpha,\beta,\mathtt{cont}_k,t_0)$$

where t_0 *is the initial time and* n *is the number of allowed moves that* α *could make initially.*

1 - Initial Rule

The first move (corresponding to the commencement rule) introduces the subject of the dialogue and we will make reference to that statement using the term `subject` (where `subject(p)` means that the predicate `p` is the subject of the dialogue). In our approach, each dialogue has only one `subject`.

$C_{in1} \Rightarrow \mathtt{assert}(\alpha,\beta,p)$

where:

$C_{in1} = \exists S,\ S \models p : (KB_\alpha \cup CS_\beta) \models S \wedge \mathtt{subject}(p)$

The dialogue starts when an agent needs to argue about a given subject, for example, in our domain (application) the agents will start an argumentation-based dialogue when they need to reallocate a particular task. To start a dialogue the agents need to have an argument to propose that another agent executes that task (i.e., it will try to reallocate the task to another participant of the collaborative team using the application), or to propose that the task will not be executed at all (i.e., the agent will try to postpone the task). The initial rule restricts that an agent needs to have an argument (corresponding to the *thoughtful* attitude described in the section 2.4) that defends its claim in order to start an argumentation-based dialogue (as the agent will be committed to defending the initial assertion, i.e., the subject of the dialogue).

2 - Assert Rules

We have two dialogue rules that restrict the possible next move for agents to respond to an `assert` move:

$\texttt{assert}(\alpha, \beta, \texttt{p}) \wedge \texttt{C}_{as1} \Rightarrow \texttt{accept}(\beta, \alpha, \texttt{p})$

$\texttt{assert}(\alpha, \beta, \texttt{p}) \wedge \texttt{C}_{as2} \Rightarrow \texttt{question}(\beta, \alpha, \texttt{p})$

Where:

$\texttt{C}_{as1} = \nexists \texttt{S}, \texttt{S} \models \overline{\texttt{p}} : (\texttt{KB}_{\beta} \cup \texttt{CS}_{\alpha}) \models \texttt{S}$

$\texttt{C}_{as2} = \exists \texttt{S}, \texttt{S} \models \overline{\texttt{p}} : (\texttt{KB}_{\beta} \cup \texttt{CS}_{\alpha}) \models \texttt{S}$

The options of the agent are: (i) to accept the previous claim (the subject asserted in the dialogue), where condition $\texttt{C}_{as1}$ means that the agent will accept a claim if it has no argument against it (corresponding to the *cautious attitude* introduced in [60, 57] and described in Section 2.4); and (ii) when the agent has an argument against the previous assertion, $\texttt{C}_{as2}$, the agent will question the other agent to provide the support of its previous claim.

3 - Question Rule

The dialogue rule which restricts the moves after an agent receives a `question` message is:

$\texttt{question}(\alpha, \beta, \texttt{p}) \wedge \texttt{C}_{qs1} \Rightarrow \texttt{justify}(\beta, \alpha, \texttt{S})$

Where:

$\texttt{C}_{qs1} = \exists \texttt{S}, \texttt{S} \models \texttt{p} : (\texttt{KB}_{\beta} \cup \texttt{CS}_{\alpha}) \models \texttt{S}$

As the agent has asserted a predicate p previously (which allowed the *question* move), the agent is committed to defend its claim in the dialogue, so it will provide the support to this claim.

4 - Justify Rules

We have four dialogue rules to restrict the moves to respond to a `justify` move:

$\texttt{justify}(\alpha, \beta, \texttt{S}) \wedge \texttt{C}_{js1} \Rightarrow \texttt{accept}(\beta, \alpha, \texttt{p})$

$\texttt{justify}(\alpha, \beta, \texttt{S}) \wedge \texttt{C}_{js2} \Rightarrow \texttt{justify}(\beta, \alpha, \texttt{S}')$

$\texttt{justify}(\alpha, \beta, \texttt{S}) \wedge \texttt{C}_{js3} \Rightarrow \texttt{closedialogue}(\beta, \alpha)$

$\texttt{justify}(\alpha, \beta, \texttt{S}) \wedge \texttt{C}_{js4} \Rightarrow \texttt{justify}(\beta, \alpha, \texttt{S}')$

Where:

$C_{js1} = \nexists S', S' \models \overline{p} : (KB_\beta \cup CS_\alpha) \models S' \wedge S' \notin CS_\beta \wedge Op(\beta) \wedge subject(p)$

$C_{js2} = \exists S', S' \models \overline{p} : (KB_\beta \cup CS_\alpha) \models S' \wedge S' \notin CS_\beta \wedge Op(\beta) \wedge subject(p)$

$C_{js3} = \nexists S', S' \models p : (KB_\beta \cup CS_\alpha) \models S' \wedge S' \notin CS_\beta \wedge Pr(\beta) \wedge subject(p)$

$C_{js4} = \exists S', S' \models p : (KB_\beta \cup CS_\alpha) \models S' \wedge S' \notin CS_\beta \wedge Pr(\beta) \wedge subject(p)$

The agent will accept the subject of the dialogue, according to condition C_{js1}, if the justification received from the proponent has changed the agent's conclusion, otherwise the agent will justify why it cannot accept the subject, C_{js2}. The agent cannot accept the subject because it still has an argument against it, even after receiving this new information. In the case where the agent that receives the justify from the opponent but cannot itself reach the same conclusion given the new information received (i.e., the agent does not have an acceptable argument for the subject), the agent closes the dialogue, C_{js3}. In the final case, the agent sends the new argument[2] to support the subject of the dialogue, C_{js4}.

5 - Accept Rule

The dialogue rule that restricts the moves when an agent receives an accept message is:

$accept(\alpha, \beta, p) \wedge C_{ac1} \Rightarrow closedialogue(\beta, \alpha)$

Where:

$C_{ac1} = subject(p) \wedge Pr(\beta)$

When the agent receives an accept move it will close the dialogue. Only the proponent will receive an accept move, when the opponent accepts the subject of the dialogue.

6.5 Properties of the Protocol

After we have defined the protocol, it is interesting to demonstrate its effectiveness through the properties commonly found in the literature. One of the more important properties to be proved over a protocol is that a dialogue, following such protocol, will always terminate.

Theorem 1 (Termination). *Any argumentation-based dialogue, following the protocol defined above, eventually terminates.*

Proof. Considering that at least one agent (the agent α) needs to have p as acceptable (where we use p as the subject of the dialogue), the initial configurations are restricted to two:

- In the first case, where $KB_\alpha \models p$ and $KB_\beta \models p$, agent α will introduce p in the dialogue using the assert move; as agent β has no argument against (i.e., no argument for $\overline{p}$), following

[2]The argument is new because, as defined in the protocol, the agent cannot repeat a move with the same content.

the dialogue rule condition C_{as1} the agent will accept p executing the accept move. Agent α receives the accept message and, following the dialogue rule condition C_{ac1}, closes the dialogue.

- In the second case, where $KB_\alpha \models p$ and $KB_\beta \models \overline{p}$, agent α, as before, will introduce p in the dialogue using the assert move. As agent β has an argument against (i.e., an argument to $\overline{p}$), agent β, following the dialogue rule condition C_{as2}, will execute the question move. Agent α receives the question message and, following the dialogue rule condition C_{qs1}, executes the justify move with the argument which support p (the existence of this argument is guaranteed by initial dialogue rule condition C_{in1} which allowed the agent to start the dialogue). The agent β receives this new information and has two options: (i) When the new information changes the acceptability of p to agent β (the agent has no acceptable argument for $\overline{p}$ which has not yet been introduced in the dialogue), then the agent, following the dialogue rule condition C_{js1}, accepts p (executing the accept move). The process continues as in the first case, where agent α receives the accept message and closes the dialogue; (ii) Otherwise, even considering the new information received, agent β still has an argument for $\overline{p}$ which has not yet been introduced in the dialogue, then the agent, following the dialogue rule condition C_{js2}, will execute a justify move. The agent α receives the justify message and has two options: (i) Agent α closes the dialogue, following the dialogue rule condition C_{js3}, because it has no acceptable argument to support p that has not yet been introduced in the dialogue; (ii) The agent α introduces a new acceptable argument, considering the new information, following the dialogue rule condition C_{js4}, support p. As the knowledge bases of the agents are finite and the justify move cannot be repeated with the same content, as formally defined in the semantic rule $ExecActSndJustify$ in Chapter 5 and in the dialogue rule conditions C_{js2} and C_{js4}, eventually agent α will close the dialogue, because it has no acceptable argument that has not yet been introduced in the dialogue or agent β will accept p because it has no acceptable argument for $\overline{p}$ that has not been introduced in the dialogue, and α will close the dialogue as in the first case.

Therefore, any argumentation-based dialogue following that protocol eventually terminates. □

The termination of a dialogue is an important and well-known property of protocols. However, as described in [10], except the termination of each dialogue generated under those protocols, nothing is said on their quality. Therefore, we will also prove such properties demonstrating that the dialogues following our protocol will always end with the best solution (*ideal* solution[3] [10]) when it exists.

As described in [10], the *ideal solution* is the best solution in the general and is time-independent. Considering our reasoning mechanism, the ideal solution is the result achieved when the agents have all information related to the subject (all arguments to and against the subject). This definition is characterised as *integrative*, where both sides are looking for solutions that are

[3]The *ideal* solution is the best solution in general and is time-independent [10].

"good" for everyone [26]. "Ideal" is also called, following [26], the situation in which the agents have complete information about the other agents, suggesting the integration of the theories of all agents into a single theory (the so-called *aggregate argumentation system*). According to [26] this integration of all arguments (agent theories) allows a centralised reasoning mechanism to identify the outcomes that are "good" for all agents participating in the dialogue.

Our work differs from [26] because we do not have a centralised reasoning mechanism, but with the exchange of all arguments related to the `subject` the agents can, rationally, reach the same conclusion of a centralised reasoning mechanism.

Definition 11 (Ideal Solution). *The ideal solution is the conclusion resulting from the all information related to* `subject` *(arguments and preferences regarding* `subject`*). The conclusion is whether the* `subject` *is acceptable, it is not acceptable, or it is not possible to determine the acceptability of the* `subject` *(i.e., the ideal solution does not exist).*

As in the real life, arguing does not necessarily lead to an agreement, it may be the case that two agents will exchange arguments and at the end the dialogue fail to agree about the `subject` [10]. This disagreement is caused by different preferences between the agents, but even with the disagreement, in the worst case, argumentation-based dialogue improves the choices made by each agent (using the additional information exchanged). Therefore, as described in [10], argumentation may improve the quality of the outcome but never decrease it.

To prove that the agents, using our protocol, strategies, and reasoning mechanism defined will reach the *ideal solution* (when it exists), we will consider the union of the knowledge bases (as in [26]).

Theorem 2 (Ideal Solution). *The ideal solution for a dialogue with subject* $\mathtt{p}$ *is for both agents to agree about* $\mathtt{p}$ *when* $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \mathtt{p}$ *and for both agents to conclude* $\overline{\mathtt{p}}$ *if* $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \overline{\mathtt{p}}$*; otherwise, the ideal solution will not exist (when neither* $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \mathtt{p}$ *nor* $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \overline{\mathtt{p}}$ *hold).*

Proof. First, we will prove that the *ideal solution* $\mathtt{p}$ is agreed upon by both agents when $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \mathtt{p}$. As $\mathtt{p}$ is always acceptable to the proponent (we refer to the proponent in this proof as α), the cases where $\mathtt{KB}_\alpha \models \overline{\mathtt{p}}$ and $\mathtt{KB}_\beta \models \overline{\mathtt{p}}$, and the case where $\mathtt{KB}_\alpha \models \overline{\mathtt{p}}$ and $\mathtt{KB}_\beta \models \mathtt{p}$ do not exist. Therefore, the possible cases are limited to two:

1. $\mathtt{KB}_\alpha \models \mathtt{p}$ and $\mathtt{KB}_\beta \models \mathtt{p}$. In this case the proponent agent will start the dialogue asserting $\mathtt{p}$ (using the `assert` move) and the other agent will execute the `accept` move (the other agent does not have an acceptable argument against, as for both agents $\mathtt{KB} \models \mathtt{p}$), hence terminating the dialogue with both agents agreeing on $\mathtt{p}$, i.e., the ideal solution in this case. Note that, as we are assuming in this part of the proof that $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \mathtt{p}$, agreement on $\mathtt{p}$ is indeed the ideal solution.

2. $\mathtt{KB}_\alpha \models \mathtt{p}$ and $\mathtt{KB}_\beta \models \overline{\mathtt{p}}$. In this case the agent identified by α starts the dialogue using the move `assert`, the agent identified by β will execute the move `question` because, as yet, $\overline{\mathtt{p}}$ is

acceptable to it. The agent α does a `justify` move; if this argument changes the conclusion of agent β, the agent accepts the `subject` (executing the move `accept`). Otherwise, agent β will send its arguments for not accepting the subject (doing another `justify` move). The agents will exchange arguments until a new argument from α changes the conclusion of β and agent β accepts the `subject` of the dialogue (executing the `accept` move), given that $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \mathtt{p}$. It follows that the ideal solution p is reached.

The part of the proof for when $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \overline{\mathtt{p}}$ is similar to the one above. The difference is that, in case two (i.e., when agents initially disagree), at a certain moment agent β will introduce an argument (using the `justify` move) where the new information will change the acceptability of the `subject` to agent α, and it will then close the dialogue. Given that $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \overline{\mathtt{p}}$, it follows that $\overline{\mathtt{p}}$, the *ideal solution*, is reached through the dialogue following our protocol.

I the last case, where the *ideal solution* does not exist, the agents will exchange arguments with the `justify` move and the dialogue will terminate in disagreement. As before, the agents initially disagree about the `subject`, at a certain moment agent α will introduce an argument (using the `justify` move) where the new information will not make the `subject` acceptable for agent β. Agent β will justify its position executing a `justify` move hence not accepting the `subject`, when the new information does not change the acceptability of `subject` for agent α. At this moment, if agent α does not satisfy the dialogue rule condition $\mathtt{C}_{js3}$, i.e., it has no new argument which supports the `subject`, the agent will close the dialogue and the dialogue will ends in disagreement. Otherwise, a new round of `justify` moves will occur. Given that neither $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \mathtt{p}$ nor $(\mathtt{KB}_\alpha \cup \mathtt{KB}_\beta) \models \overline{\mathtt{p}}$ hold, i.e., the *ideal solution* does not exist, the dialogue ends with the agents disagreeing about the `subject`. □

An important point to be noticed, in the proof above, is that the non existence of the *ideal solution* is consequence of agent's preferences, as described before. When individual preferences are considered, an agent's own argument will be acceptable even with the argument against presented by the other agent. For example, agent α executes the `justify` move, introducing an argument to p in the dialogue, agent β receives this information and uses the `justify` move because it has an argument to $\overline{\mathtt{p}}$, agent α receives this information but its own argument is still acceptable. At this moment the agent detects that there exist different preferences between the agents and, following the dialogue rule condition $\mathtt{C}_{js3}$, the agent closes the dialogue.

7. APPLICATION DOMAIN

In this Chapter we describe the application (demo) developed in the SeaTeaMS Project (Semantic and Multi-Agent Technologies for Group Interaction). The SeaTeaMS Project has as objective the research and development of a programming framework which allows the development of complex applications that integrate multiple autonomous entities, both agents and humans. The applications that can be developed using the proposed framework have the purpose of coordinating the execution of tasks by members of the group. For example, when a member will fail to execute one of their tasks, the system can try to reallocate this task to another member of the group.

7.1 Application

The application developed to demonstrate the effectiveness of the framework is in the context of assisted-living. The application provides 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 the patient is following their appointments/commitments;
- detect problems which may prevent someone responsible for the patient to do their tasks;
- reallocate tasks among users if required (using an argumentation-based approach);
- send reminders for users to monitor the patient 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/application, a member of the project has created an ontology [79] 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.

In the next Sections we describe some aspects of the application which are necessary in our work. In Section 7.2 we describe how the agents access the information from the ontology and then, in Section 7.3, we describe a little of the ontology developed in the project. Next, in Section 7.4, we describe how the agents use the information provided by the ontology in a decision-making process to argue about the task reallocation.

7.2 Accessing Ontological Information

To provide access to ontological information (i.e., domain-specific knowledge developed by a knowledge engineer) in a MAS, it has been developed in the context of the project a CArtAgO [75] 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 [74]. 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.

The artifact developed uses the OWL API, which is an open source Java API [31], 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 related by the given instance through 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, if necessary).

7.3 Task Ontology

An ontology was developed 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 programs to implement task reasoning mechanisms, such as techniques for task recognition, allocation, and negotiation.

In the 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 subsumes 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* who needs to be 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 [12] 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.

7.4 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 successfully). The knowledge of this domain is described in an ontology designed by a knowledge engineer, based on the task ontology, and can be reused in other applications. 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.

7.4.1 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 [40]. 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 on behalf of 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 will not be able to 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 for helping 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 [1], suggesting that the task be transferred to another time slot (following the protocol for argumentation-based dialogue defined in Chapter 6). 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

[1]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.

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.

- **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 decision-making 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 from 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, otherwise it goes to Step 7.

- **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 and negotiations using the provided information. The process ends.

8. EVALUATION

To evaluate our work, we implemented three use cases of the SeaTeaMS Project that we will describe in this chapter, corresponding to the scenario implemented as a *Demo* application that resulted from the project. We will consider three cases where the task reallocation is started by the system. To do so, we start describing the scenario and a typical day (where nothing fails) in the routine of elder person assisted by his family, then we describe the three cases where the system tries to reallocate the task because it identifies failure in their fulfilment.

8.1 Scenario Description

"João is an elder man (76 years old) who suffers from severe arthritis and has early signs of dementia. He has trouble walking (he sometimes falls over if left to walk alone), cannot lift heavy weights or perform fine motor actions with his hands, and should not be trusted to take medicine nor leave his home without being accompanied. He lives by himself in his own house. João has two sons, Paulo and Stefano. Paulo lives next door with his wife Jane and two children, Pedro (12) and Maria (14). Stefano lives in the same city, but 10 km away from João's house. João has some professional help, in particular two professional carers who collaborate with the family to take care of him. There is a day carer and a night carer with some time in between the end of the day shift and the start of the night shift. João has to take various different medicines 4 times a day, and has physiotherapy sessions in general three times a week. The sessions are flexible and can be postponed to the next day, unless João is in a particularly bad day, in which case he must be taken to physiotherapy as it alleviates the most acute pain from which he suffers occasionally. Paulo is assigned as responsible for taking João to the physiotherapy clinic (if he is not available for that in a particular day, another adult member of the group with a driving license and a car must take over the task)."

The following sequence exemplifies a typical day in the life of João and his extended family:

1. João wakes up;
2. He takes a bath, with the help of the day carer;
3. The day carer prepares breakfast;
4. After breakfast, João has to take his pills (the family and carers need to be notified if it appears that João did not take the required medicine at the appropriate time);
5. João and the day carer go for a walk in a park;
6. When they get home, João watches TV while the day carer cleans the house and prepares lunch;

7. After lunch, João has to take a second set of pills;
8. He takes a nap after lunch, in the meantime the day carer loads the dishwasher;
9. At 15:00h João has to go to physiotherapy with Paulo;
10. When João gets back home, he has a snack, and the day carer finishes his shift;
11. Just before the day carer leaves, Pedro comes home from school and spends time with his grandfather, playing a memory game with him on the smart TV;
12. João has dinner with Paulo (who is the responsible for providing the dinner) and his family;
13. After dinner, João has to take more pills;
14. Between dinner and bedtime, João spends time with his family, and the night carer arrives;
15. Before going to bed, João takes yet another set of pills;
16. João goes to sleep;
17. The night carer loads the washer-drier and watches over João's sleep (in case he needs to go to the toilet, for example); the night carer only leaves when the day carer arrives.

The three cases presented bellow describe situations in which the normal daily schedule could not be met due to changes in the individual schedules of the people related to the scenario.

Case 1

As a first example of a problem with the usual schedule, consider that Pedro has to stay longer at school to do a school project with his colleagues. This means that he will not be at home to look for João (grandpa) during the interval between the day and night carers' shifts. The application recognises this because:

(a) Pedro puts the new appointment in his Google calendar, or;

(b) Pedro is still at school when he should have left to go home, as recognised by the GPS in his Galaxy S5 phone.

In this case, the application reminds Pedro of his obligation and starts to check (through negotiation between the agents that represent the family members) if another member of the family can perform this task instead. As the application detects that Maria (the sister) is able to perform the task, it asks Pedro if he wants it to ask her to do it. If Pedro confirms the proposed course of action, the application asks Maria to perform the task. If she accepts it, the task is reallocated. Otherwise, there is a conflict. In both cases (reallocation or conflict), Paulo and/or Jane (whoever is the assigned responsible) are notified so as to become aware of the reallocation or to resolve the conflict.

Case 2

Paulo is supposed to take João to physiotherapy that particular date but his boss has just added a meeting with him at the time he should be leaving the office to go home to pick up João. Paulo's agent concludes that Paulo will postpone the task since physiotherapy sessions have a flexible schedule and normally can be postponed. The agent confirms with Paulo and tells João's agent that Paulo cannot make it so the physiotherapy will be left for tomorrow. João's agent says that this is not acceptable because it noted signs that João was in more pain than usual that particular day (for example, there was a note from the carer saying that extra painkillers were administered). Paulo's agent then informs him that it will try to reallocate the task to Jane (the children are not considered for this task as only adults can perform it). Jane's agent questions the request but when informed about João being in too much pain the agent proposes to reallocate Jane's next appointment and asks if she agrees to take João to physiotherapy.

Case 3

In the third case, Paulo gets stuck in the traffic on his way home from work and cannot cook dinner for João at the usual time. This situation is recognised by the application if it verifies that Paulo is not moving or it has not reached a specific point at a predefined time stamp (which can be understood as moving too slowly).

Similar to the previous case, in this situation the agent observing Paulo initially warns him about the failure in the plan, then it tries to negotiate the task with other members of the family. As the children are not allowed to cook and Paulo's wife is away in business, the application asks Paulo if he wants to ask Stefano to perform the task (as it was identified that he can do it). If Paulo accepts the proposal, the application asks Stefano if he can take care of João and the children and provide the dinner. If Stefano accepts the proposal the task is reallocated, otherwise, there is no known alternative solution. In both cases Paulo (which is the assigned responsible) receives a notification in order to become aware of the reallocation or that he will need to resolve the problem himself.

8.2 Implementation

We implemented the techniques described in Chapters 4, 5 and 6 in a modular way, in order to make them independent of one another. The argumentation-based reasoning mechanism and the performative semantics implementation are independent of application. The protocol, the decision-making process and the strategy are implemented specifically to our scenario/application (but they can be used to similar domains/applications).

- The reasoning mechanism was implemented using the adaptation of d-Prolog, as described before; it is an AgentSpeak file that can be included in any agent. When that file is included, the agents can use the corresponding reasoning.

- The semantics presented in the Chapter 5 has been implemented extending the internal action *.send* (for sending messages) and a *.asl* file (which can be included as the previous one) with plans similar to the KQML plans available with Jason. That AgentSpeak file is used to treat the receiving of messages related to the performatives to which we gave semantics. The corresponding dialogue system has been implemented using a CArtAgO artifact [75] to coordinate agent interactions.
- The protocol is used to restrict the next move of the agents, and the strategy of the agents was implemented as AgentSpeak plans. The strategy for the agents can be rewritten or replaced easily.

Others AgentSpeak files are used to define the specific rules of the domain (which represent inferences based on the application domain) and plans that are specific for the application.

8.3 Solution for Scenario Case 1

As described above, Pedro either puts a new appointment in his Calendar, or remains at school when he should have left to go home. The application recognises the failure in routine and, following the decision-making process described in Section 7.4.1, starts the argumentation-based dialogue. The process is the following:

Following the decision-making process the Pedro's agent checks the ontology to verify if the task can be postponed using the artifact which returns false, because the task to keep João company cannot be postponed.

After checking if the task can be postponed and in case the result is *FALSE*, Pedro's agent checks which users can execute the task (described in Section 7.2). In our example, the users that can execute the task are "Pedro" and "Maria". Then, Pedro's agent starts a dialogue with Maria's agent using the `assert` move, suggesting that Maria executes the task. Maria's agent receives the `assert` message and accepts it, because it does not have an acceptable argument against. Maria's agent queries if Maria really can execute (in this case Maria says yes) and the users are informed. If Maria's agent had an argument against, the agent would perform a `question` move and the process would continue until Maria's agent accepts to execute the task, or Pedro's agent is convinced that Maria cannot execute the task either.

8.4 Solution for Scenario Case 2

In Case 2, Paulo has a new commitment and he will not be able to take João to physiotherapy (as described above). Then Paulo's agent will try to reallocate the task. First Paulo's agent checks if the task can be postponed, in this case the task belongs to the concept *ReallocatableTime* in the ontology. With this information Paulo's agent initiates a dialogue with João agent, asserting

that Paulo cannot execute the task. João's agent receives the message and executes a `question` move, because the agent has an argument against, with information that João is in severe pain, and this implies that someone needs to take João to the physiotherapy.

Paulo's agent receives the `question` message and provides the justification using the `justify` move with the argument which allowed the initial assert, more specifically: that a physiotherapy appointment can be transferred. João's agent receives this message, but this information does not change its conclusion, then João's agent executes the `justify` move with the strict rule that João is in severe pain, and this implies that someone needs to take João to the physiotherapy. Paulo's agent receives this new information and concludes that someone needs to take João to physiotherapy. Paulo's agent closes the dialogue and checks in the ontology (using the same artifact operation described in the previous case) to get the information of who can execute the task. In this case "Paulo", "Jane" and "Stefano" can execute this task (because it is a task that only adults can do). Paulo's agent starts a new dialogue with Jane's agent using the `assert` move, suggesting that Jane executes the task.

Jane's agent receives the `assert` message and executes a `question` move, because it knows that the task can be postponed. Paulo's agent sends the justification (argument) that Jane needs to execute the task because João is in pain. Jane's agent receives this new information and accepts to execute the task[1].

8.5 Solution for Scenario Case 3

In Case 3, Paulo gets stuck in traffic on his way home from work and cannot cook dinner for João at the usual time. Paulo's agent checks if the task can be postponed, in this case it cannot, then Paulo's agent checks who can execute the task, in this case "Paulo", "Jane" and "Stefano" can execute. Paulo's agent starts a dialogue with Jane's agent using the `assert` move, suggesting that Jane executes the task. Jane's agent has an argument against, because Jane has another appointment at that time, and executes the `question` move. Paulo's agent executes the `justify` move, stating that Paulo will be home late. The new information does not change the conclusion of Jane's agent, and it executes the `justify` move, arguing that Jane has a meeting and the two tasks overlap.

The new information does not change the conclusion of Paulo's agent, but the agent does not have any further arguments, so the agent closes the dialogue and starts a new dialogue with Stefano's agent (the next in the list) using the `assert` move, suggesting Stefano to prepare João's dinner. Stefano's agent receives this message and accepts to execute the task, because it does not have an argument against[2].

[1] Jane's agent, as always, queries Jane to have the confirmation.
[2] Stefano's agent, as always, queries Stefano to have the confirmation.

8.6 Final Remarks About the Solutions

All the solutions follow the decision-making process and the protocol presented in the previous Chapter. As the implementation refers to an application, we have some interactions with the users of the application before the agent makes some decisions. For example, (i) when an agent perceives that the user will not be able to execute the task (the user has another appointment or is late) the agent always queries the user if they want to try to reallocate the task; (ii) when an agent accepts an argument for its user execute a task, the agent queries the user, if the user objects to the reallocation, the agent that is trying to reallocate the task starts a dialogue with the next agent of the list of agents that can execute the task. If the list is empty, the agent informs its user that it cannot do anything else to help reallocate the task and that the users need to solve the problem by themselves.

9. CONCLUSION

In this dissertation we presented our research on argumentation in multi-agent systems. We have contributed in the direction of practical argumentation frameworks in an agent-oriented programming language. Our work has been implemented in the SeaTeaMS Project, as described before.

The main contributions of our work are:

1. the adaptation of defeasible-Prolog (d-Prolog) for the Jason Platform as a reasoning mechanism, which allows argumentation-based reasoning in agents developed in this language (this work has been published in [56]);

2. the operational semantics formalisation and implementation of performatives/speech-acts in an agent-oriented programming language, which can be used to implement argumentation-based dialogues for other agent-oriented languages based on the BDI architecture (this work has been partially published in AAMAS conference [55]); and

3. the definition and formalisation of an argumentation-based protocol as a dialogue game, considering the agents' strategy.

Further, we have demonstrated these techniques in the practical implementations of real scenarios. Towards the practical use of the techniques developed in this work, we have participated in the implementation of an ontology artifact to access information from a domain-specific ontology and a decision-making process, which uses the result of queries in this ontology.

As future work we intend to define a more general protocol for argumentation-based dialogues, allowing more complex interactions. We also intend to explore the argumentation schemes field, which has been claimed as a promising field of research [91]. Our intention in exploring argumentation schemes follows the formalisation of argumentation schemes related to the organisation in a multi-agent system, where agents can use these pattern of reasoning in argumentation-based reasoning and argumentation-based dialogue (as in this work).

BIBLIOGRAPHY

[1] Amgoud, L. "A formal framework for handling conflicting desires." In: 7th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Nielsen, T. D.; Zhang, N. L. (Editors), 2003, pp. 552–563.

[2] Amgoud, L.; Cayrol, C. "A reasoning model based on the production of acceptable arguments", *Annals of Mathematics and Artificial Intelligence*, vol. 34–1-3, Maio 2002, pp. 197–215.

[3] Amgoud, L.; Dimopoulos, Y.; Moraitis, P. "A unified and general framework for argumentation-based negotiation". In: 6th international joint conference on Autonomous agents and multiagent systems, 2007, pp. 158:1–158:8.

[4] Amgoud, L.; Kaci, S. "On the generation of bipolar goals in argumentation-based negotiation." In: 1th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS), Rahwan, I.; Moraitis, P.; Reed, C. (Editors), 2004, pp. 192–207.

[5] Amgoud, L.; Maudet, N.; Parsons, S. "Modeling dialogues using argumentation." In: 4th International Conference on MultiAgent Systems, 2000, pp. 31–38.

[6] Amgoud, L.; Maudet, N.; Parsons, S. "An argumentation-based semantics for agent communication languages". In: 15th European Conference on Artificial Intelligence (ECAI), 2002, pp. 38–42.

[7] Amgoud, L.; Parsons, S.; Maudet, N. "Arguments, dialogue, and negotiation". In: 14th European Conference on Artificial Intelligence (ECAI), 2000, pp. 338–342.

[8] Amgoud, L.; Prade, H. "Generation and evaluation of different types of arguments in negotiation." In: 10th International Workshop on Non-Monotonic Reasoning (NMR), Delgrande, J. P.; Schaub, T. (Editors), 2004, pp. 10–15.

[9] Amgoud, L.; Prade, H. "Formal handling of threats and rewards in a negotiation dialogue." In: 2th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS), Parsons, S.; Maudet, N.; Moraitis, P.; Rahwan, I. (Editors), 2005, pp. 88–103.

[10] Amgoud, L.; Vesic, S. "A formal analysis of the role of argumentation in negotiation dialogues", *Journal of Logic and Computation*, vol. 22–5, Out 2012, pp. 957–978.

[11] Atkinson, K.; Bench-Capon, T. "Practical reasoning as presumptive argumentation using action based alternating transition systems", *Artif. Intell.*, vol. 171–10-15, jul 2007, pp. 855–874.

[12] Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; Stein, L. A. "OWL Web Ontology Language Reference", Technical Report, W3C, 2004, 62p.

[13] Bedi, P.; Vashisth, P. "Extending speech-act based communication to enable argumentation in cognitive agents". In: *Advances in Computing, Communication and Control*, Berlin: Springer, 2011, pp. 25–40.

[14] Bentahar, J.; Alam, R.; Maamar, Z. "An argumentation-based protocol for conflict resolution". In: Workshop on Knowledge Representation for Agents and MultiAgent Systems (KRAMAS 2008), 2008, pp. 19–35.

[15] Berariu, T. "An argumentation framework for bdi agents". In: *Intelligent Distributed Computing VII*, Zavoral, F.; Jung, J. J.; Badica, C. (Editors), Springer International Publishing, 2014, *Studies in Computational Intelligence*, vol. 511, pp. 343–354.

[16] Bondarenko, A.; Dung, P. M.; Kowalski, R. A.; Toni, F. "An abstract, argumentation-theoretic approach to default reasoning.", *Artif. Intell.*, vol. 93, Nov 1997, pp. 63–101.

[17] Bordini, R. H.; Dastani, M.; Dix, J.; Seghrouchni, A. E. F. "Multi-Agent Programming: Languages, Tools and Applications". Springer Publishing Company, Incorporated, 2009, 1st ed., 389p.

[18] 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, 292p.

[19] Bordini, R. H.; Moreira, A. F. "Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L)", *Annals of Mathematics and Artificial Intelligence*, vol. 42–1-3, Maio 2004, pp. 197–226.

[20] Bracciali, A.; Mancarella, P.; Stathis, K.; Toni, F. "On modelling declaratively multiagent systems". In: Declarative Agent Languages and Technologies (DALT 2004), 2005, pp. 53–68.

[21] Caminada, M. "On the issue of reinstatement in argumentation". In: *Logics in artificial intelligence*, Springer, 2006, pp. 111–123.

[22] Costantini, S.; Tocchio, A. "A dialogue games framework for the operational semantics of logic agent-oriented languages". In: 11^{th} International Conference on Computational Logic in Multi-agent Systems, 2010, pp. 238–255.

[23] Dignum, F.; Dunin-Keplicz, B.; Verbrugge, R. "Agent theory for team formation by dialogue". In: *Intelligent Agents VII Agent Theories Architectures and Languages*, Springer, 2001, pp. 150–166.

[24] Dignum, F.; Dunin-Keplicz, B.; Verbrugge, R. "Creating collective intention through dialogue", *Logic Journal of IGPL*, vol. 9–2, Jun 2001, pp. 289–304.

[25] Dimopoulos, Y.; Moraitis, P. "Advances in argumentation based negotiation". Bentham Science Publishers, 2014, chap. 4, Negotiation and Argumentation in Multi-Agent Systems, pp. 82–125.

[26] Dimopoulos, Y.; Moraitis, P.; Amgoud, L. "Characterizing the outcomes of argumentation-based integrative negotiation", *IEEE/WIC/ACM InternationalConference on Web Intelligence and Intelligent Agent Technology*, vol. 2, Dez 2008, pp. 456–460.

[27] Dung, P. M. "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games", *Artificial Intelligence*, vol. 77, Nov 1995, pp. 321–357.

[28] FIPA, T. "Fipa communicative act library specification", *Foundation for Intelligent Physical Agents, http://www.fipa.org*, 2008.

[29] Governatori, G.; Maher, M. J.; Antoniou, G.; Billington, D. "Argumentation semantics for defeasible logic.", *J. Log. Comput.*, vol. 14–5, Jul 2004, pp. 675–702.

[30] Hamblin, C. L. "Fallacies". Methuen, London, UK, 1970, 326p.

[31] Horridge, M.; Bechhofer, S. "The OWL API: A Java API for OWL ontologies", *Semantic Web*, vol. 2–1, jan 2011, pp. 11–21.

[32] Hulstijn, J.; van der Torre, L. "Combining goal generation and planning in an argumentation framework". In: 10^{th} International Workshop on Non-Monotonic Reasoning (NMR), 2004, pp. 212–218.

[33] Hussain, A.; Toni, F. "On the benefits of argumentation for negotiation-preliminary version". In: 6^{th} European workshop on multi-agent systems (EUMAS-2008), 2008, pp. 1–21.

[34] Jennings, N. R.; Faratin, P.; Lomuscio, A. R.; Parsons, S.; Wooldridge, M. J.; Sierra, C. "Automated negotiation: prospects, methods and challenges", *Group Decision and Negotiation*, vol. 10–2, Mar 2001, pp. 199–215.

[35] Jennings, N. R.; Parsons, S.; Noriega, P.; Sierra, C. "On argumentation-based negotiation". In: Int. Workshop on Multi-Agent Systems, 1998, pp. 1–8.

[36] Jennings, N. R.; Wooldridge, M. "On agent-based software engineering", *Artificial Intelligence*, vol. 117, Nov 2000, pp. 277–296.

[37] Kakas, A.; Moraitis, P. "Argumentation based decision making for autonomous agents". In: 2^{th} International Joint Conference on Autonomous Agents and Multiagent Systems, 2003, pp. 883–890.

[38] Kakas, A.; Moraitis, P. "Adaptive agent negotiation via argumentation". In: 5^{th} International Joint Conference on Autonomous Agents and Multiagent Systems, 2006, pp. 384–391.

[39] Kakas, A. C.; Moraitis, P. "Argumentative agent deliberation, roles and context.", *Electr. Notes Theor. Comput. Sci.*, vol. 70–5, Jun 2002, pp. 39–53.

[40] Kaufman, M. "Local decision-making in multi-agent systems", Ph.D. Thesis, Oxford University, 2010, 185p.

[41] Kraus, S.; Sycara, K.; Evenchik, A. "Reaching agreements through argumentation: A logical model and implementation", *Artificial Intelligence*, vol. 104, Nov 1998, pp. 1–69.

[42] Maudet, N.; Parsons, S.; Rahwan, I. "Argumentation in multi-agent systems: Context and recent developments." In: 3th International Workshop on Argumentation in Multi-Agent Systems, Maudet, N.; Parsons, S.; Rahwan, I. (Editors), 2006, pp. 1–16.

[43] Mayfield, J.; Labrou, Y.; Finin, T. W. "Evaluation of kqml as an agent communication language." In: Agent Theories, Architectures, and Languages (ATAL), Wooldridge, M.; Müller, J. P.; Tambe, M. (Editors), 1995, pp. 347–360.

[44] Mcburney, P.; Parsons, S. "Games that agents play: A formal framework for dialogues between autonomous agents", *Journal of Logic, Language and Information*, vol. 11, Maio 2001, pp. 1–22.

[45] Mcburney, P.; Parsons, S. "Dialogue games in multi-agent systems", *Informal Logic*, vol. 22, Jan 2002, pp. 1–8.

[46] McBurney, P.; Parsons, S. "Dialogue game protocols". In: *Communication in Multiagent Systems*, Springer, 2003, pp. 269–283.

[47] McBurney, P.; Parsons, S. "Locutions for argumentation in agent interaction protocols." In: Agent Communication, van Eijk, R. M.; Huget, M.-P.; Dignum, F. (Editors), 2004, pp. 209–225.

[48] McBurney, P.; Parsons, S. "Syntax and semantics of the fatio argumentation protocol". In: 3th International Joint Conference on Autonomous Agents and Multi-agent Systems, 2004, pp. 1–10.

[49] Mcburney, P.; Van Eijk, R. M.; Parsons, S.; Amgoud, L. "A dialogue game protocol for agent purchase negotiations", *Autonomous Agents and Multi-Agent Systems*, vol. 7–3, Fev 2003, pp. 235–273.

[50] Nute, D. "Defeasible Prolog". Artificial Intelligence Programs, University of Georgia, 1993, 9p.

[51] Nute, D. "Handbook of logic in artificial intelligence and logic programming". In: *Handbook of logic in artificial intelligence and logic programming*, Gabbay, D. M.; Hogger, C. J.; Robinson, J. A. (Editors), New York, NY, USA: Oxford University Press, Inc., 1994, chap. Defeasible logic, pp. 353–395.

[52] Nute, D. "Defeasible logic". In: Handbook of Logic in Artificial Intelligence and Logic Programming, 2001, pp. 353–395.

[53] Pan, S.; Larson, K.; Rahwan, I. "Argumentation mechanism design for preferred semantics." In: 3th International Conference on Computational Models of Argument, Baroni, P.; Cerutti, F.; Giacomin, M.; Simari, G. R. (Editors), 2010, pp. 403–414.

[54] Panisson, A. R.; Farias, G.; Freitas, A.; Meneguzzi, F.; Vieira, R.; Bordini, R. H. "Planning Interactions for Agents in Argumentation-Based Negotiation". In: 11th International Workshop on Argumentation in Multiagent Systems (ArgMAS), 2014, pp. 1–15.

[55] Panisson, A. R.; Meneguzzi, F.; Fagundes, M.; Vieira, R.; Bordini, R. H. "Formal semantics of speech acts for argumentative dialogues". In: 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2014, pp. 1437–1438.

[56] 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 (ArgMAS), 2014, pp. 1–15.

[57] Parsons, S.; McBurney, P. "Argumentation-based dialogues for agent co-ordination", *Group Decision and Negotiation*, vol. 12–5, 2003, pp. 415–439.

[58] Parsons, S.; Sierra, C.; Jennings, N. "Agents that reason and negotiate by arguing", *Journal of Logic and Computation*, vol. 8, Jun 1998, pp. 261–292.

[59] Parsons, S.; Sklar, E. "How agents alter their beliefs after an argumentation-based dialogue." In: 2th International Workshop on Argumentation in Multiagent Systems (ArgMAS), Parsons, S.; Maudet, N.; Moraitis, P.; Rahwan, I. (Editors), 2005, pp. 297–312.

[60] Parsons, S.; Wooldridge, M.; Amgoud, L. "An analysis of formal inter-agent dialogues". In: 1th International Conference on Autonomous Agents and Multi-Agent Systems, 2002, pp. 394–401.

[61] Pilotti, P.; Casali, A.; Chesnevar, C. "A belief revision approach for argumentation-based negotiation with cooperative agents". In: 9th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2012), Valencia, Spain, 2012, pp. 1–18.

[62] Plotkin, G. D. "A structural approach to operational semantics", *J. Log. Algebr. Program.*, vol. 60-61, Fev 2004, pp. 17–139.

[63] Pollock, J. L. "Defeasible reasoning", *Cognitive science*, vol. 11–4, Abr 1987, pp. 481–518.

[64] Prakken, H. "On dialogue systems with speech acts, arguments, and counterarguments". In: *Logics in Artificial Intelligence*, Springer, 2000, pp. 224–238.

[65] Prakken, H. "An abstract framework for argumentation with structured arguments", *Argument and Computation*, vol. 1–2, Set 2011, pp. 93–124.

[66] Rahwan, I.; Amgoud, L. "An argumentation based approach for practical reasoning." In: 5th International joint conference on Autonomous agents and multiagent systems (AAMAS), Nakashima, H.; Wellman, M. P.; Weiss, G.; Stone, P. (Editors), 2006, pp. 347–354.

[67] Rahwan, I.; Larson, K. "Mechanism design for abstract argumentation". In: 7th Int. Joint Conference on Autonomous Agents and Multiagent Systems, 2008, pp. 1031–1038.

[68] Rahwan, I.; Larson, K. "Pareto optimality in abstract argumentation." In: 23th Conference on Artificial Intelligence, Fox, D.; Gomes, C. P. (Editors), 2008, pp. 150–155.

[69] Rahwan, I.; Larson, K.; Tohmé, F. "A characterisation of strategy-proofness for grounded argumentation semantics". In: 21th International Jont Conference on Artifical Intelligence, 2009, pp. 251–256.

[70] Rahwan, I.; Pasquier, P.; Sonenberg, L.; Dignum, F. "On the benefits of exploiting underlying goals in argument-based negotiation." In: 22th Conference on Artificial Intelligence, 2007, pp. 116–121.

[71] Rahwan, I.; Pasquier, P.; Sonenberg, L.; Dignum, F. "A formal analysis of interest-based negotiation", *Annals of Mathematics and Artificial Intelligence*, vol. 55–3-4, Fev 2009, pp. 253–276.

[72] Rahwan, I.; Ramchurn, S. D.; Jennings, N. R.; Mcburney, P.; Parsons, S.; Sonenberg, L. "Argumentation-based negotiation", *The Knowledge Engineering Review*, vol. 18–04, Nov 2003, pp. 343–375.

[73] Rao, A. S. "AgentSpeak(L): BDI agents speak out in a logical computable language". In: 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away: agents breaking away, 1996, pp. 42–55.

[74] Ricci, A.; Piunti, M.; Viroli, M. "Environment programming in multi-agent systems: An artifact-based perspective", *Autonomous Agents and Multi-Agent Systems*, vol. 23–2, 2011, pp. 158–192.

[75] Ricci, A.; Viroli, M.; Omicini, A. "CArtAgO: an infrastructure for engineering computational environments in MAS". In: 3th International Workshop "Environments for Multi-Agent Systems" (E4MAS), Weyns, D.; Parunak, H. V. D.; Michel, F. (Editors), 2006, pp. 102–119.

[76] Rosenschein, J. S.; Zlotkin, G. "Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers". Cambridge, Massachusetts: MIT Press, 1994, 253p.

[77] Rueda, S. V.; Martínez, M. V. "Interaction among bdi argumentative agents: a dialogue games approach". In: XI Congreso Argentino de Ciencias de la Computación, 2005, pp. 1–12.

[78] Sadri, F.; Toni, F.; Torroni, P. "Logic agents, dialogues and negotiation: An abductive approach". In: Symposium on Information Agents for E-Commerce (AISB), 2001, pp. 1–8.

[79] Schmidt, D. "Ontologias para representação de tarefas colaborativas em sistemas multi-agentes", Master's Thesis, Pontifical Catholic University of Rio Grande do Sul, 2015, 72p.

[80] Searle, J. R. "Speech Acts: An Essay in the Philosophy of Language". Cambridge University Press, 1969, 203p.

[81] Sierra, C.; Jennings, N. R.; Noriega, P.; Parsons, S. "A framework for argumentation-based negotiation". In: 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages, 1998, pp. 177–192.

[82] Singh, M. P. "Agent communication languages: Rethinking the principles", *IEEE Computer*, vol. 31-12, 1998, pp. 40–47.

[83] Singh, M. P. "A social semantics for agent communication languages". In: *Issues in agent communication*, Springer, 2000, pp. 31–45.

[84] Sycara, K. P. "Persuasive argumentation in negotiation", *Theory and Decision*, vol. 28-3, Maio 1990, pp. 203–242.

[85] Toniolo, A.; Norman, T. J.; Sycara, K. P. "Argumentation schemes for collaborative planning." In: Agents in Principle, Agents in Practice, Kinny, D.; jen Hsu, J. Y.; Governatori, G.; Ghose, A. K. (Editors), 2011, pp. 323–335.

[86] Toniolo, A.; Norman, T. J.; Sycara, K. P. "On the benefits of argumentation schemes in deliberative dialogue." In: 11th International Conference on Autonomous Agents and Multiagent Systems, van der Hoek, W.; Padgham, L.; Conitzer, V.; Winikoff, M. (Editors), 2012, pp. 1409–1410.

[87] Vieira, R.; Moreira, A.; Wooldridge, M.; Bordini, R. H. "On the formal semantics of speech-act based communication in an agent-oriented programming language", *J. Artif. Int. Res.*, vol. 29-1, Jun 2007, pp. 221–267.

[88] Von Neumann, J.; Morgenstern, O. "Theory of games and economic behavior". Princeton university press, 1944, 776p.

[89] Vreeswijk, G. A. "Abstract argumentation systems", *Artificial intelligence*, vol. 90-1, Nov 1997, pp. 225–279.

[90] Walton, D.; Krabbe, E. "Commitment in Dialogue: Basic concept of interpersonal reasoning". Albany NY: State University of New York Press, 1995, 235p.

[91] Walton, D.; Reed, C.; Macagno, F. "Argumentation Schemes". Cambridge University Press, 2008, 456p.

[92] Wooldridge, M. "Properties and complexity of some formal inter-agent dialogues", *Journal of Logic and Computation*, vol. 13, Jun 2003, pp. 347–376.