Argumentation and Artifacts for Intelligent Multi-agent Systems

Enrico Oliva

March, 2008

Coordinator: Prof. Paolo Bassi

1^{st} supervisor/tutor: Prof. Antonio Natali

2^{nd} supervisor: Prof. Andrea Omicini
3^{rd} supervisor: Dr. Ing. Mirko Viroli

Doctorate in Electronics, Computer Science and Telecommunications
ALMA MATER STUDIORUM – University of Bologna
ING-INF/05 Systems for information processing

Reviewers

Prof. Peter McBurney
University of Liverpool

Prof. Nicolas Maudet
University of Paris-Dauphin

Abstract

Reasoning under uncertainty is a human capacity that in software system is necessary and often hidden. Argumentation theory and logic make explicit non-monotonic information in order to enable automatic forms of reasoning under uncertainty. In human organization Distributed Cognition and Activity Theory explain how artifacts are fundamental in all cognitive process. Then, in this thesis we search to understand the use of cognitive artifacts in an new argumentation framework for an agent-based artificial society.

Acknowledgements

I owe many thanks to my family for their support over the years and in particular to my wife for her patient and constant support. I would like to thank Antonio Natali too for giving me the possibility to work and to study in this department. I would also like to thank Andrea Omicini for giving me the abstractions that are the bases of the thesis work. I like to thank Mirko Viroli for his helpful comments and for inspiring me to the Prolog implementation. I would also like to thank Peter McBurney for his enthusiastic support and for introducing me to argumentation theory that is a fundamental part of this thesis. Also many thanks to the department of Computer Science of Liverpool University where I spent six useful months for my research. Finally, I would like to thank all my colleagues of the department and all my friends from apiCE lab who gave me a valid support during these three years of phd.

"The uncreative mind can spot wrong answers, by it takes a creative mind to spot wrong questions" [1]

[1] Antony Jay, Management and Machiavelli, 1968, p. 87.

Contents

Chapter 1

Introduction

1.1 Relevance

Multi-Agent Systems (MAS) paradigm provides the instruments to represent and manage complex systems characterized by the presence of autonomously interacting and possibly heterogeneous entities. Following A&A meta-model a MAS is generally composed of computational entities of two sorts: agents and artifacts. An agent is an autonomous and proactive computational entity situated in an environment. An artifact is a reactive, non-autonomous computational entity that reacts to interaction events generating by agents or other artifacts, or changes of the environment.

In the context of complex software systems argumentation theory is requirement in applications that involve both theoretical and practical aspects of artificial intelligence and computer science. The Argumentation-based techniques range from non-monotonic reasoning, knowledge engineering to MAS communication, legal reasoning and alternative dispute resolution systems.

An agent-based society similar a human society could exploit cognitive artifact to support cognitive processes. The join among artifact abstraction and argumentation theory is an useful ground of research in order to build an intelligent multi-agent system. The artifacts could bring at architecture level some useful features of argumentation, enabling an easier composition of intelligent applications.

In this thesis we investigate the possibility to build cognitive artifact based on argumentation. We provide a model of argumentation exploitable by agents and artifacts. The capacity to build cognitive argumentation artifacts usable from agents opens new prospective to design MAS in direction to obtain intelligent artificial societies. For instance an automatic Alternative Dispute Resolution system could be obtained from such features.

The following critical questions the thesis aims at giving a response to underline the relevance of the thesis in both sectors of argumentation and multi-agent systems.

- Which general architecture can be developed for specifying communication and coordination among agent based on argument?
- Which argumentation framework could be exploited to implement multi-agent argumentation systems?
- Which general formalization could be exploited for an intelligent and automatic dialog mediation?
- Which general architecture could be exploited to realize Alternative and Online Dispute Resolution systems?

- How can an Alternative and Online Dispute Resolution systems be implemented in an agent society?

1.2 Thesis outline

This thesis goes from theoretic to practical studies. Table 1.1 shows the three sequence steps of thesis work: theory, model and coding with the respective results. We start from a theorization of an argumentation system based on first order logic representation. Moreover, we study the argument-based communication, defining a formal dialog system where we exploit a process algebra representation and the previous definition of arguments.

In the second step we design a model for multi-agent argumentation system. We express the desidered set of functionalities in order to exploit argumentation inside an agent society with two purposes: 1) enabling dialog based on arguments and 2) enabling coordination based on argument. To satisfy the requirements following A&A meta-model we propose two type of artifacts: Co-Argumentation Artifact and Dialog Artifact that encapsulate all the functionalities necessary to build intelligent agent societies. The artifacts have the role of *intelligent mediators* that automatically can make some inference operations helpful to agent coordination and communication.

The finally step is the implementation of the abstractions. To do that we exploit TuCSoN infrastructure for agent society programming the artifact by ReSpecT and Prolog languages. The reactive architecture of TuCSoN infrastructure is the winning strategy to build complicate software such as MAS, that impose a right scalable and flexible architecture. We foresee an ADR application scenario fitting

our theory, model and coding very well. This case study could be considered as a validation of the thesis work.

Table 1.1: Map of thesis contents

Theory	Argumentation System (AS)	Dialog System (DS)
Model	Co-Argumentation Artifcat (CAA)	Dialog Artifact (DA)
Code	TuCSoN	TuCSoN
Case Study	Argument Acceptance	Persuasion Dialog

Chapter 2

On Argumentation System

In this chapter are introduced argumentation theory and some open issue about argumentation. In the actual intelligent software systems closed to final user it is useful to have a representation of the uncertain and a system of communication both close to human. We focus our attention on the formalization of an Argumentation System (AS) to build non-monotonic knowledge, and a Dialog System (DS) to enable communication.

2.1 Argumentation Theory

Argumentation theory went up again 2,500 years old, to the time of Plato and Aristotle in Athens. The arts of communicating in public forums and persuasion were very important abilities for the Athenian government. Aristotle introduced the study of rhetoric that is generally understood to be the art or technique of persuasion through the use of oral, visual or written language. Argumentation is a part of the study of rhetoric in particular: how to create and present good arguments that

could be logically accepted. Argumentation is a form of communication, where at least one person supports the claims with evidences and reasoning. But it could also be a form of knowledge representation, where the relevant arguments lead to alternative, conflicting conclusions.

A society mainly evolves through interaction and communication among participating entities. Within a society, people argue in order to solve problems, to reduce conflicts, to exchange information and to inform each other of some pertinent facts. Argumentation is a useful feature of human intelligence that enables us to deal with incomplete and inconsistent information. People usually have only partial knowledge about the world (they are not omniscient) and often they have to manage conflicting information. In the same way, the entities that compose an artificial society should be able to deal with partial and conflicting knowledge. Correspondingly, an agent-based model for an artificial society should provide an adequate definition of knowledge with the purpose of providing a realistic reflection of a society. Also, it may be useful to share information in order to successfully deal with partial knowledge.

Actually, Argumentation or Argumentation Theory is a formal discipline within artificial intelligence, where the aim is to make a computer assist in or perform the act of argumentation. In addition, argumentation is used to provide a proof-theoretic semantics for non-monotonic logic, starting with the influential work of Dung (1995). Computational argumentation systems have found particular application in domains where formal logic and classical decision theory are unable to capture the richness of reasoning, domains such as law and medicine.

Argumentation is an important feature of human intelligence: the ability to understand and manipulate arguments is fundamental to understand a new prob-

lem, reason about actions and perform scientific research. Typically an argument is a sequence of inferences leading to a valid conclusion: a set of arguments is managed by an argumentation component that is particularly useful in the case of conflicting information.

Argumentation systems are now being applied for developing application in legal systems, negotiation among agents decision making etc. Argumentation is strictly connected with logic and reasoning. It makes use of formal and informal logic to enable reasoning and represent arguments. The study of logical form also includes the study of how premises should lead to conclusions assuming that the premises are correct. Informal logic is the study of reasoning from evidence to conclusions. The study of argumentation actually involves formal and informal logic.

Argumentation systems are a way to formalize non-monotonic reasoning, such as constructing and comparing arguments for and against certain conclusions. The idea is: the construction of an argument is monotonic and the non-monotonicity is expressed in term of interaction between conflicting arguments. An other view of inference under uncertainty leads to probabilistic argumentation system, that is the join between classic logic and probability theory. An argument can be seen such as a chain of possible events (premises) that makes the hypothesis (conclusions) true. The credibility of a conclusion can then be measured from the total probability calculated from the premises. Finally, a comparison between argument or a quantitative judgment is obtained by considering the supported argument probabilities.

Argumentation-based Communication

In a MAS, argumentation has a central role that allows agents to argue, justify positions and try to persuade another agent to endorse some statement. All these features are quite common in a real-world society, and enable complex global behaviours. Argumentation can be used to model the communication among agents in a MAS, in particular to model the dialog between two entities. A set of six primary dialogue types is identified by [44]: persuasion, inquiry, negotiation, information seeking, deliberation and eristic. All these dialogues can be captured in an argumentation framework [29], and they are developed strictly among two entities. In [9] an implementation of information-seeking dialog based on tuple centre architecture is presented. However, a definition for a dialogue says that a dialogue is a mutual conversation between two or more people. In a society there are forms of communication among multiple entities that enable humans to work together and achieve their goals. Following that definition, we can naturally extend the dialogue concept in MAS from two agents to N agents. The argumentation-based dialogues listed above, for instance, could be transformed in social discussions among agents.

2.2 Argumentation System

There are many existing formalisms to present argumentation theory such as defensible logic, default logic and auto-epistemic logic as special case. In our vision, an argument is a minimal set of facts that conduce to a conclusion. We use an underlining monotonic logic, with logical connective representing implication and

with a semantics and proof theory already provided with logic, to represent and define arguments.

Prakken and Vreeswijk [36], in their study of logics for defensible argumentation, observe that an argumentation system is generally composed of five elements (although sometimes implicitly): 1) a logical language; 2) an argument definition; 3) a concept of conflict among arguments; 4) a concept of defeated argument; 5) a concept of argument acceptability. In this section we define an argumentation system as a reference point for our work. We take inspiration from Dung's framework [10], and we also define the structure inside the arguments.

The underlining logic of our system is a first-order language, where Σ contains all well-formed formulae. The symbol $\vdash$ denotes classical inference (different styles will be used like deduction, induction and abduction), $\equiv$ denotes logical equivalence, and $\neg$ or *non* is used for logical negation.

2.2.1 Formal Model

Typically an argument has an internal structure, comprising of inferences that leads to a conclusion. It has three components: beliefs, inference rules and conclusions.

- **beliefs** are facts and rules that represent premises
- **inference rules** are labels that represent inference processes such as deduction or induction
- **conclusions** are facts that represent results of the inference process applied to the beliefs

In our system, we express the argument in predicate logic using the *logic tuple* notation. We take inspiration from Dung's framework [10], and we also define the structure inside the arguments. In [36] an argumentation system formalized in propositional logic is presented. Whereas we follow such an approach, we also try to extend it using predicative logic, which suits a logic programming framework. The object language of our system is a first-order language, where Σ contains all well-formed formulae. The symbol $\vdash$ denotes classical inference (different styles will be used like deduction, induction and abduction) $\equiv$ denotes logical equivalence, and $\neg$ or *non* is used for logical negation.

Definition 1 *An argument is a triple* $A = \langle B, I, C \rangle$ *where* $B = \{p_1, \ldots, p_n\} \subseteq \Sigma$ *is a set of* beliefs, $\vdash_I \in \{\vdash_d, \vdash_i, \vdash_a\}$ *is the inference style (respectively, deducion, induction, or abduction), and* $C = \{c_1, \ldots, c_n\} \subseteq \Sigma$ *is a set of* conclusions*, such that:*

1. *B is consistent*
2. $B \vdash_I C$
3. *B is minimal, so no subset of B satisfying both 1 and 2 exists*

For instance, a classical example of argument like *all men are mortal, Socrates is a man, Socrates is mortal*, in our representation becomes:

- $B = human(Socrates), human(X) \rightarrow mortal(X)$
- $I = \vdash_{MP}$ Modus Ponens
- $C \ni mortal(Socrates)$

Our formalization of the 'Socrates argument' can be easily mapped in a logic tuple. In the process of mapping, we add the predicate *argument* with the function *name* and other predicates such as *beliefs*, *infer* and *conclusions* to represent the triple $A = \langle B, I, C \rangle$.

$$argument(name, beliefs([human(Socrates)], [clause(mortal(X), [human(X)])]),$$
$$infer(MP), conclusions([mortal(Socrates)])).$$

A declarative representation of arguments could be useful to store and collect the arguments during the argumentation process. The formula *argument* in our system is the basic unit to represent an argument.

The inference rules we consider for deduction are Modus Ponens (MP), Multi-Modus Ponens (MPP) and Modus Tollens (MT).

$$\frac{B \;\; B \rightarrow C}{C} \tag{MP}$$

$$\frac{B_1 \;\; B_2 \;\; B_3 \;\; (B_1 \wedge B_2 \wedge B_3) \rightarrow C}{C} \tag{MPP}$$

The MP is a particular case of MMP with only one premise. Socrates argument is a example of MP deductive argument. Also, MT formula expresses a deductive inference.

$$\frac{\neg A \;\; B \rightarrow A}{\neg B} \tag{MT}$$

For example, all humans are mortal, but Eraclito is not mortal, than Eraclito is not human, in tuple form is:

$$argument(name, beliefs([non(mortal(eraclito))], [clause(mortal(X),$$
$$[human(X)])]), infer(MT), conclusions([non(human(eraclito))])).$$

The inference rule that we use for induction is θ-subsumption, as shown in (θ-su).

$$\frac{B}{R} \ \textit{where} \ \ R\theta \subseteq B \qquad (\theta\text{-su})$$

For example, $mortal(X) \leftarrow human(X)$, θ-subsumes $mortal(socrates) \leftarrow human(socrates)$ with $\theta = \langle X = socrates \rangle$, in tuple form looks like

$$argument(name, beliefs([mortal(socrates), human(socrates)]),$$
$$infer(Su), conclusions([clause(mortal(X), [human(X))])])).$$

This process derives a general rule R from specific beliefs B, but is not a legal inference in a strict sense. Currently, we do not consider it as a probability value that could be associated to the result of an induction process. Finally, the abductive reasoning is expressed with the inference rule shown in (Ab).

$$\frac{B \ \ A \rightarrow B}{A} \qquad (\text{Ab})$$

For example, all humans are mortal, Parmenide is a mortal, then Parmenide is a human, in tuple form looks like

$$argument(name, beliefs([mortal(parmenide)], [clause(mortal(X),$$
$$[human(X)])]), infer(Ab), conclusions([human(parmenide)]).$$

The definition of contrast is not trivial, because there are different types of attack well defined in [36]. Following those definitions, two possible types of attack are 'conclusions against conclusions' – called *rebuttals* – and 'conclusions against beliefs'—called *undercuts*.

Definition 2 *Let $A_1 = \langle B_1, I_1, C_1 \rangle$ and $A_2 = \langle B_2, I_2, C_2 \rangle$ are two distinct arguments, A_1 is an **undercut** for A_2 iff $\exists h \in C_1$ such that $h \equiv \neg b_i$ where $b_i \in B_2$*

Definition 3 *Let* $A_1 = \langle B_1, I_1, C_1 \rangle$ *and* $A_2 = \langle B_2, I_2, C_2 \rangle$ *are two distinct arguments,* A_1 *is a* ***rebuttal*** *for* A_2 *iff* $\exists h \in C_1$ *such that* $h \equiv \neg c_i$ *where* $c_i \in C_2$

From the algorithmic point of view, it is necessary to identify the opposite predicate: α defeats $\neg\alpha$ in order to find the contrast argument. In our framework we introduce $non/1$ operator that identifies the opposite predicate: $non(mortal(Socrates))$ is opposite to $mortal(Socrates)$. We also introduce another notion of undercut based on the principle of refutation. To find an attack to the rule, a counterexample is required that disproves its truth. An argument A_1 is attacked through a counterexample contained in the conclusion of another argument. In formula, we consider an implication with only one premise $A \rightarrow B \equiv \neg A \vee B$ the contrary is: $A\neg(\neg A \vee B) \equiv A \wedge \neg B$. An expression with A and the negation of B is a counterexample of the implication. For instance, the following argument undercuts the Socrates example by refuting the implication $mortal(X) \rightarrow human(X)$:

$$argument(name, beliefs([human(Eraclito), non(mortal(Eraclito))]),$$
$$infer(T), conclusions([human(Eraclito), non(mortal(Eraclito))])).$$

This type of attack is possible only with an explicit representation of the rules.

Finally inside the component there are the main algorithms to manipulate the conflict knowledge in order to decide the admissible subset of a set of arguments and to determine whether a new argument is acceptable or not. The definitions of acceptability and admissibility used in our framework are in agreement with [10]. The following definitions are the basic ones in our argumentation system and take inspiration from Dung's framework.

Definition 4 *An argument set S is a* ***conflict free*** *set iff there exists no* $A_i, A_j \in S$ *such that* A_i *attacks* A_j.

Definition 5 *An argument set S* ***defends collectively*** *all its elements if* $\forall$ *argument* $B \notin S$ *where B attacks* $A \in S \quad \exists\ C \in S$ *: C attacks B.*

Definition 6 *An argument set S is a* ***admissible*** *set iff S is conflict free and S defends collectively all its elements.*

Definition 7 *An argument set S is a* ***preferred extension*** *iff S is a maximal set among the admissible set of A.*

We consider also important argument extensions such as acceptability in order to determine whether a new argument is acceptable or not. In the context of preferred semantics the acceptance problem is divided in credulous acceptance or sceptical acceptance, if an argument is in some/all preferred extension.

Definition 8 *An argument A is* ***credulous acceptable*** *if* $A \in$ *at least one preferred extension.*

Definition 9 *An argument A is* ***sceptical acceptable*** *if* $A \in$ *all preferred extensions.*

2.3 Dialog System

In this section we present a novel formalization of a multi-agent dialog system. Our intention is to capture the rules which govern legal utterances, and which govern the effects of utterances on the commitment stores of the dialog. We use a process algebra approach in the style of [41] to represent the possible paths which a dialog may take, and to represent explicitly the operations to and from the commitment stores. We proceed by considering each element of a dialog system

in turn: 1) the communication language; 2) the interaction protocol; and 3) the protocol semantics.

Because a dialog is a dialectical exchange of arguments, we assume that arguments and counter-arguments are represented and expressed in the formal language defined above in Section 2.2. Agents may exchange arguments, along with facts, with one another in the form of instantiated parameters in their utterances.

2.3.1 Communication Language

The agents need to share a same communication language *CL* in order to exchange information. The role of *CL* as a language used for internal knowledge representation and reasoning is explained in [29]. We let F denote a set of terms representing *facts*, and $\mathscr{A}$ the set of terms representing all *arguments* able to be represented in Σ following the definition of an argument given in Definition 1. Our *CL* is defined in order to support all six primary dialogue types as identified by [44]: persuasion, inquiry, negotiation, information seeking, deliberation and eristic.

Definition 10 *Our communication language is a set of locutions L_c. A locution $l \in L_c$ is a term of the form $perf_{name}(Arg_1,\ldots,Arg_n)$ where $perf_{name}$ is a element of the set P of performatives and Arg_x is either a fact or an argument.*

An agent performing a dialog exploiting the communication language can utter a locution composed of facts and arguments. A fact is represented by syntax `fact(Terms)` and an argument with `argument(B,I,C)`. The definitions to manage attacking and undercutting arguments are provided by the underlying argumentation system given in Definition 1. In the example 1 an agent wants to communicate the classical example of argument like *All men are mortal, Socrates*

is a man, Socrates is mortal, and it uses an `Argue` locution with an `argument` parameter.

Example 1 `Argue(argument(name,beliefs([human(Socrates)],[clause(mortal(X),[human(X)])]),infer(MP),conclusions([mortal(Socrates)]))).`

Examples of performatives to support an instance of an *Information Seeking Dialog* could be: `OpenDialog`, `Ask`, `Tell`, `DontTell`, `Provide`, `Argue`, and so on. Further details about this form of dialog and its complete locutions are presented in [9].

2.3.2 Dialog Protocol

In our framework the dialog protocol is a complete description of all possible dialog paths, from the perspective of an external entity observing the dialog between the agents. The protocol indicates the possible paths of a dialog, specifies the source and target of each message and shows the relationship between utterances and the content of commitment stores. Our approach basically describes the step-by-step behaviour of an external entity acting as a mediator, hence enabling the allowed interactions. Therefore, we technically find it useful to model a dialog in terms of a process algebra with standard composition operators (sequence, parallel, iteration), and whose atomic actions represent either agent utterances or interactions with the commitment store (writing, reading or removing a commitment).

On the one hand, Prakken [33] proposes a general definition of locution where a move *m* is denoted by four elements: 1) identifier, 2) speaker (or source), 3) speech act and 4) intended recipient (or target). Following this model, we provide a definition of a speech act, as follows:

Definition 11 *An action A is defined by the syntax* $A ::= s : L_c | s[t_1, \ldots, t_n] : L_c$ *where s indicates the source, and* $[t1, \ldots, t_n]$ *indicates the (optional) targets of the message.*

On the other, beyond this, we include additional atomic operations K over commitment stores—many of them can actually occur into one argumentation artifact. To this end, the commitment store is viewed as a set of tuples as in [20]: such tuples are manipulated by the commands of the Linda language [13]—`in`, `rd` and `out`.

Definition 12 *A term action K has the syntax* $K ::= in(C,X) | out(C,X) | rd(C,X)$, *where C is a term representing the commitment store identifier, and X is a term representing the commitment.*

Specifically, the commands `in(C,t)`, `rd(C,t)` and `out(C,t)` respectively consume, read and put a tuple `t` in the commitment store `C`. These actions are useful to manage the private or public commitment store in relation to the dialog execution. In particular, they can operate, for example, as action-preconditions in order to restrict or constrain the next action choice, and thus enable only certain future dialog paths. If at a given time a sub-dialog is guarded by operation `rd(c,commit(a))`, for instance, then it is allowed to proceed only if `commit(a)` occurs in the commitment store.

Definition 13 *A protocol P is a composition of action from sets A and K, defined by syntax* $P ::= 0 | A | K | P.P | P + P | P \rhd P | (P \parallel P) | !P$ *where the symbols* $., +, \rhd, \|, !$ *denote respectively sequencencial composition, choice, left-priority-choice, parallel composition, and infinite replication operators.*

For example, an abstract dialog protocol definition is given by $D := (s : a_1 + s : a_2).(s : a_3 + s : a_4); s : a_5$ where agent s is only allowed to execute a sequence of three actions: the sequence composed of a first action consisting of either action a_1 or action a_2, then a second action consisting of either a_3 or a_4, and then a third action comprising a_5. A protocol specifies a set of actions histories, that the agents might execute. As another example of a protocol definition, consider $D := s : a_1 \parallel s : a_1 \parallel s : a_1 \parallel t : a_2 \parallel t : a_3$ where agent s can invoke a_1 at most three times, agent t can invoke a_2 and a_3 only once, but in whichever order.

To illustrate this framework, we present a specification for an Information-Seeking dialog (f is seen as a variable over the content of communication):

Example 2 (Information Seeking Dialog) `c:OpenDialog.`

```
s:OpenDialog.
!(c:Ask(f).
  s:Tell(f).(
     rd(perm(c,f)).
     s:Provide(f).
     s:Argue(perm(c,f),YES,A)
     +
     s:DontTell(f).
     s:Argue(perm(c,f),NO,B).
     c:Argue(perm(c,$\phi$),ADD,A). (
        s:Argue(perm(c,f),NO,B)
        +
        s:Accept(A,perm(c,f)).
```

```
        in(Accept(perm(c,f)))
) )   )
```

2.4 Operational Semantics

Following Hamblin [17], we assume that each agent is associated with a knowledge base, accessible to all agents, containing its commitments made in the course of the dialogue. Commitments are understood as statements which the associated agent must support, while they remain in the commitment store, if these statements are questioned or attacked by other agents. We can now use the notion of commitment store and the transition system given in Definition 15 to define an operational semantics for the dialog system. This semantics describes the evolution over time of the dialog state and the states of commitment store (seen as composition of all commitment stores). In essence, the commitment store is the knowledge repository of the dialog as a whole, and it is expressed in our framework as a multiset of terms.

Definition 14 *A commitment store C is a multiset of terms and it is defined by the syntax* $C ::= 0|(C|C)|X$ *where X is a term.*

Definition 15 *The operational semantics of our dialog system is described by a labelled transition system* $\langle S, \rightarrow, I \rangle$*, where* $S ::= (C)P$ *represents the state of dialog system (protocol P running with commitment store C), I is the set of interactions (labels) composed of* $i ::= \lambda : \theta$ *where* $\lambda ::= \tau|a$ *and* θ *is a term substitution, and* $\rightarrow$ *is a transition relation of the kind* $\rightarrow \subseteq S \times I \times S$*.*

A term substitution θ is of the kind $\{x/y\}$: when applied to a term t by syntax $t\{x/y\}$ it means t after applying the most general substitution between terms x and y—x should be an instance of y, otherwise the substitution notation would not make sense. As usual, we write $s \xrightarrow{i} s'$ in place of $\langle s, i, s' \rangle \in \leftarrow$, meaning the dialog system moves from state s to s' due to interaction i—either an action a or an internal step τ (an operation over the commitment store), involving a term substitution θ inside the protocol. We introduce a congruence relation $\equiv$, which syntactically equates similar states:

$$0.P \equiv P \qquad P.0 \equiv P \qquad (P.Q).R \equiv P.(Q.R) \qquad !P \equiv P|!P$$

$$0+P \equiv P \qquad P+Q \equiv Q+P \qquad (P+Q)+R \equiv P+(Q+R)$$

$$0 \rhd P \equiv P \qquad P \rhd 0 \equiv P \qquad (P \rhd Q) \rhd R \equiv P \rhd (Q \rhd R)$$

$$0 \parallel P \qquad P \parallel Q \equiv Q \parallel P \qquad (P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$$

Finally, we define operational rules that describe the behavior of the dialog system as follows:

$$
\begin{array}{rcll}
(C)out(x) & \xrightarrow{\tau:\theta_I} & (C|x)0 & (OUT) \\
(C|x)rd(y) & \xrightarrow{\tau:\{x/y\}} & (C|x)0 & (RD) \\
(C|x)in(y) & \xrightarrow{\tau:\{x/y\}} & (C)0 & (IN) \\
(C)a & \xrightarrow{a'\{a'/a\}} & (C)0 & (ACT) \\
(C)(P.Q) & \xrightarrow{\lambda:\theta} & (C')P'.Q\theta \quad \text{if} \quad (C)P \xrightarrow{\lambda:\theta} (C')P' & (SEQ) \\
(C)(P+Q) & \xrightarrow{i} & (C')P' \quad \text{if} \quad (C)P \xrightarrow{i} (C')P' & (SUM) \\
(C)(P \rhd Q) & \xrightarrow{i} & (C')P' \quad \text{if} \quad (C)P \xrightarrow{i} (C')P' & (LEFT) \\
(C)(P \rhd Q) & \xrightarrow{i} & (C')Q' \quad \text{if} \quad (C)P \not\rightarrow \quad (C)Q \xrightarrow{i} (C')Q' & (RIGHT) \\
(C)(P|Q) & \xrightarrow{i} & (C')(P'|Q) \quad \text{if} \quad (C)P \xrightarrow{i} (C')P' & (PAR) \\
(C)P & \xrightarrow{i} & (C')Q \quad \text{if} \quad P \equiv P' \quad (C)P' \xrightarrow{i} (C')Q & (EQUIV)
\end{array}
$$

Rule (OUT) provides the local semantic of `out` operation, expressing that x term is added to the commitment store C. Rules (RD) and (IN) similarly handle operation `rd` and `in`: the use of substitution operator guarantees that the term x in the commitment store is an instance of the term x to be retrieved. Rule (ACT) expresses that locution a' is executed that is an instance of the allowed one a, using the proper term substitution. Rule (SEQ) handles sequential composition and substitution: in a process $P.Q$, P is allowed to proceed (recursively), but, if any substitution is involved, this is applied to Q as well. Rules (SUM) and (PAR) provide the semantics for choice and parallel operators in the standard way. Rules (LEFT) and (RIGHT) provide the semantics for left-priority-choice: in a process $P \rhd Q$, P proceeds if allowed to, otherwise Q proceeds. Finally, rule (EQUIV) states that transitions can be applied modulo the congruence relation.

2.5 Issue on Argumentation

Building a software system in some domains requires the design and implementation of many programs to solve different computational tasks. In order to define a program, the programmer has to use both the knowledge of the domain and his/her own knowledge coming from experience. The knowledge representation is a key point to build flexible and autonomous software systems.

In the main stream approach to develop Object Oriented or Component Oriented software and in the new SOA architecture do not exist an explicit abstraction to make explicit knowledge of the domain does not exist. Through symbolic representation of knowledge some computational tasks are being realized by applying different forms of inference such as: deduction, abduction, induction, model generator, updating etc. This forms of computation are strategic to develop intelligent and complicated software systems.

In order to make all these ideas real it is necessary to provide: 1) first order entity in the project of software system to represent knowledge, 2) a reference logic theoryand 3) an infrastructure to make explicit symbolic knowledge representations.

Argumentation provides a form of non-monotonic knowledge representation, which is a generalization of non-monotonic logics. It could be a good choice to make explicit knowledge and conflicting knowledge, to much time hidden in software system. Also, a very useful innovation is to define a meta-model for software system where the knowledge is a first class entity of design.

The argumentation theory could cover some lacks in software systems making explicit conflicting knowledge and enabling form of reasoning under uncertain

such as practical reasoning [30] and common sense reasoning [12]. Moreover, argumentation is used in the construction of systems for legal reasoning, collective decision making and negotiation and, in general, it has a fundamental role in Alternative Dispute Resolution (ADR) systems.

In Multi-Agent Systems too the agents use arguments in order to exchange information, to resolve dispute and to inform each other of some pertinent facts. A MAS designer can provide an argumentation support within a social agent-based context to enable a consistent evolution of social knowledge. It can also provide agents with an instrument to enhance their ability to deal with their own partial and incomplete knowledge.

The open issue in argumentation theory is the research of an efficient computational model and scalable architecture to develop particular applications in domains like law and medicine, where formal logic and classical decision theory are unable to capture the richness of reasoning. Rahwan et al in [37] open a new perspective on the argumentation system laying the basis for a World Wide Argument Web. In their paper they show that semantic web, arguments and argumentation ontology are strictly connected, demonstrating how an ontology representation of arguments enables the description of web contents through a network of arguments on the Semantic Web. An interesting example of this use is provided by *Discourse DB*[1] forum powered by *Semantic MediaWiki*[2]. Nowadays, the open issue is building an infrastructure for large-scale argument representation, manipulation and evaluation, that could become pervasive in the Word Wide Web context.

[1] See http://discoursedb.org

[2] See http://ontoworld.org/wiki/Semantic_MediaWiki

Chapter 3

Argumentation and Artifact

In this chapter we propose desidered functionalities of artifacts for exploiting arguments. In building multi-agent systems the most hopeful meta-model is A&A that proposes agents and artifacts as first class abstraction to model the system. Two types of artifacts based on argumentation are proposed: Co-Argumentation Artifact for construction of common knowledge and Dialog Artifact to enable mediate argument-based communication.

3.1 A&A meta-model for MAS

Agent-based systems technology is the new paradigm for conceptualizing, designing, and implementing software systems. A Multi Agent System is a system composed of several software agents, collectively capable of reaching goals that monolithic systems find difficult to achieve. A most promising approach to design MAS is the A&A meta-model that re-interprets MAS in terms of two fundamental abstractions: agents and artifacts.

Agents are the active entities encapsulating control, which are in charge of the goals/tasks all together building up the whole MAS behaviour. Artifacts are instead the passive, reactive entities in charge of the services and functions that make individual agents work together in a MAS, and that shape agent environment according to the MAS needs. An artifact is used by agents, possibly featuring useful properties such as controllability, malleability, linkability, and situation [26]. More generally, to engineering software systems: (1) agents represent task-oriented or goal-oriented components that act pro-actively according to their task or goal; (2) artifacts represent resources or tools that are used by agents during their activities.

The design of agent-based artificial societies is based on the notion of artifact [27], which takes inspiration from Activity Theory [21], where any human activity within a society is enabled, constrained or mediated by artifacts. An artifact is social construct shared by agents of a MAS and is necessary to mediate interaction among agents and between agents and their environment. Unlike agents, artifacts are not meant to be autonomous or exhibit a proactive behaviour. Among the main properties of an artifact there are: (i) inspectability and controllability, i.e. the capability of observing and controlling artifact structure, state and behaviour at runtime and of supporting their on-line management, in terms of diagnosing, debugging, testing; (ii) malleability, i.e. the capability of artifact function to be changed / adapted at runtime (on-the-fly) according to new requirements or unpredictable events occurring in the open environment, (iii) linkability, i.e. the capability of linking together at runtime distinct artifacts as a form of composition, as a means to scale up with complexity of the function to provide, and also to support dynamic reuse, (iv) situation, i.e. the property of being immersed in the

MAS environment and be reactive to environment events and changes. A traffic light, for instance, is a sort of coordination artifact: drivers watching the signal know what they have to do to avoid accidents at an intersection, without any need for direct communication with one another.

In a social context, people have only partial knowledge about the world and use arguments in order to solve problems, to reduce conflicts or to exchange information. The same holds for intelligent agents in a multi agent system; here, however, it is not clear what could act as a support for argumentation between agents, external to the agents themselves. To this end, this work exploits the agents and artifacts (A&A) meta-model for MAS, exploring the use of artifacts for agent argumentation within a MAS.

3.2 Mediated Interaction

Mediation is useful to achieve cooperation between the entities and the coordination of the global system. In a MAS, in particular, mediation among agents has a central role to coordinate activities, to achieve social goals, and to support interaction. Moreover, in a system there are social properties that need to be expressed outside agents. Knowledge too, also according to Distributed Cognition [18], is not bounded inside each individual agent, but is instead distributed among agents and artifacts in the environment. Environment-based coordination and, more generally, mediated interaction frameworks and infrastructures based on forms of coordination / cooperation without direct communication are among the most promising lines of research in the MAS field.

In a human society the role of mediator exists: in a Dispute Resolution for

instance, the mediator ensures fairness and a correct resolution of the dispute. Proponents of public policy conversations and decision-making processes usually emphasize the need for a human moderator or mediator to be involved in the interaction, e.g., [11]. The mediator may act to ensure equality of access by all participants, assist participants to clarify their positions and to argue more effectively and even try to reconcile opposing views. Similarly, the designers of computer-aided argumentation systems have also provided support for human mediators; the developers of *Zeno*, for example, define their system as "a mediation system" [14, p.10]:

> *"a kind of computer-based discussion forum with particular support for argumentation. In addition to the generic functions for viewing, browsing and responding to messages, a mediation system uses a formal model of argumentation to facilitate retrieval, to show and manage dependencies between arguments, to provide heuristic information focusing the discussion on solutions which appear most promising, and to assist human mediators in providing advice about the rights and obligations of the participants in formally regulated decision making procedures."*

Just as with human interactions, and for the same reasons, many of the functions provided by mediators could be useful when software agents engage in argumentation with one another. The mediator is useful to coordinate agents that have to achieve a global goal. Some of these mediator functions require only limited intelligence that could, for example, support the storage and share arguments with the participants. These functionality could be automatically provided by an artifact.

3.3 Multi-agent Argumentation System

Our model of a multi-agent argumentation system following A&A meta-model exploits two types of entities: agents and artifacts. In particular in our model we consider intelligent dialog agents and two types of artifacts: a dialog artifact and a co-argumentation artifact. *Co-Argumentation Artifact (CAA)* is a central co-ordinating entity in an argumentation dialog, that provides co-ordination services to the participating agents allowing them to share, store and exchange arguments with one another. Vesting the CAA with its own argumentation capabilities means that this entity, like the participants, could undertake reasoning across the arguments it stores. The CAA, for example, could determine whether a particular argument is acceptable (under a specific semantics of argumentation) with respect to the global knowledge of all the participants.

It is easy to imagine that the CAA could undertake more sophisticated interventions in the dialog resembling complex, automated tasks of a human mediator. To this end, we extend our earlier concept of a central co-ordinating artifact to be a dialog artifact (DA), acting as a mediator between the participating agents. Dialog participants, of course, need to be able to generate, evaluate, contest and defend arguments as they interact with one another through dialog. But the dialog artifact also needs this argumentation functionality if it is to find common ground between different participants, or to clarify their differences. For example, if the dialog artifact is to convince two participants that their opposed positions in fact share common assumptions or that one position implies the other, then the mediator artifact may need – in an automated way – to create, present and defend a case to the participants.

Consequently, we have described two conceptual artifacts variously required by the entities in our system. The combination of both artifacts provides the support of basic and advanced functionality for automatic mediation services in a MAS. Some basic functionalities to support the exchange of arguments in a dialog between the participants include:

1. Storage of the dialog protocol (e.g. in a library of such protocols)
2. Storage of the specifications of the dialog protocol
3. Storage of the complete history of a dialog as it proceeds
4. The ability to refuse to allow agent utterances which do conform to the current protocol in use
5. The ability to suggest next moves which are legal according to the current protocol in use in a dialog
6. The ability to receive and store confidential information from the participating agents, such as their preferences in a negotiation. The mediator could then aggregate such information (across multiple agents), and/or seek to identify and reconcile differences.

Also, the central artifact could act as a sophisticated mediator of the discussion, by providing in an automated way the following services:

1. Seeking to resolve any dispute over the rules of the protocol
2. Providing rewards or penalties to agents for breaking the protocol rules
3. Having the power to admit or to expel agents to/from the dialog

4. Suggesting a new protocol, when needed.

5. Supporting multiple simultaneous bilateral interactions.

6. Assigning roles, rights and responsibilities to agents at run-time, as, for example, in an action protocol, assigning the role of winner to a particular agent near the end of the interaction.

7. Identifying conflicts and inconsistencies between commitments made by agents in a dialog, for example, if an agent commits to sell a car it is also trying to purchase.

8. Identifying agent utterances which are not relevant to the current state of the dialog, and refusing to permit these to be made.

9. Providing automated alerts to inform agents that dialogs on particular topics are about to start, or to end, or that particular commitments have just been made.

10. Combining different dialogs on the same topic.

More advanced functions of the CAA and DA combination could also include:

1. Annotation of protocols with their properties, for protocols stored in the protocol library, for instance, the possible outcomes of a protocol, its computational complexity, and so on.

2. Storing the outcomes of past dialogs, like for example the commitments remaining at the end of the dialog.

3. Tracking agent commitments across multiple dialogs.

4. Using previous dialogs to create an independent assessment of the reputation of participating agents.

5. Storage of the entire history of past dialogs. These may be required for regulatory or legal reasons, e.g. in stock market transactions.

In section 3.5 and 3.4, we present a formalization respectively of the DA and CAA which conceptually supports the basic functionalities listed above.

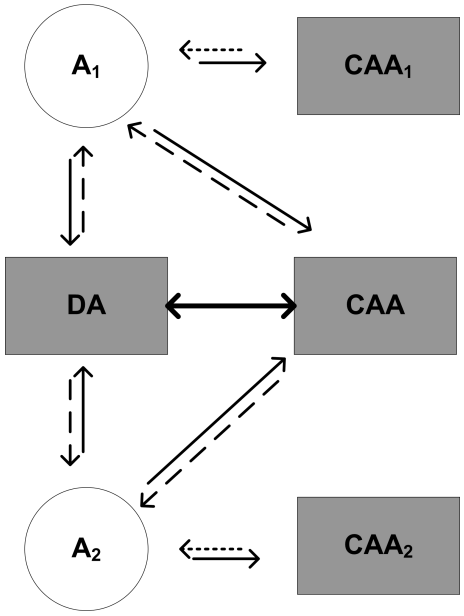

Figure 3.1: General Architecture of Multi-agent argumentation system

Figure 3.1 presents a possible architecture for a Multi-agent argumentation system. We propose a scalable architecture composed of a local CAA private for an agent and a global CAA common for the agent society. An agent exploits own local CAA to coordinate its mental state. Classically those functions (to store, to manage and to retrieve arguments) are provided by an internal argumentation component hidden inside the agent. Exploiting A&A vision we propose to make

an explicit representation of agent mental state by a co-argumentation artifact.

Global CAA and DA artifacts provide services and functionalities listed above for the entire agent society. Ideally, in the model DA and CAA are separate entities with separate and orthogonal functionalities. In an implementation scenario both common artifacts could collapse in one unique global entity without loss of generality.

To validate the architecture, we focus our attention on dispute resolution system in artificial society, where the dialog has a fundamental role in achieving a solution. In particular, the persuasion dialog based on arguments is exploited by agents to find an agreement.A dispute resolution is not an easy task and usually involves more entities (including mediator or arbitrator). Our architecture provides the desired abstraction and properties to realize a mediation service for dispute resolution in an agent society.

3.4 Co-Argumentation Artifact

In this section, combining multi-agent argumentation with the A&A meta-model we define a Co-Argumentation Artifact (CAA) as an artifact specialized in managing arguments and providing coordination services for argumentation process in a MAS. The CAA is a mediator of agent interaction and supports a simplified implementation of multi-agent argumentation system. It provides functionality that allows agents to exploit social commitment, enabling them to share, store and exchange arguments.

A simple example of social use of CAA is to fix social acceptance of the arguments: the goal is determining whether an argument is acceptable with respect

to the global knowledge of the community. The CAA applies an argumentation semantics over the shared arguments, which provides for the acceptance criteria. Another interesting example is the use of CAA as a commitment store during the dialog process. Tracing the commitments is fundamental for the next step of the discussion. Also, from the arguments stored during the dialog process the CAA could deduce or induce new knowledge. The introduction of the CAA model provides new support to design communications that involve more entities in a social context.

A similar type of artifact is the co-ordination artifact [27], specialised in providing a coordination service in MAS [40]. A typical use of a co-ordination artifact is enabling the exchange of information among agents in an open and dynamic environment—like a mailbox or a blackboard. Another interesting example is the use of the co-ordination artifact for knowledge mediation where the information can be manipulated by the artifact through either aggregation or induction process.

We define a CAA as a couple $CAA = \langle S, AC \rangle$ where S is the store of arguments and AC argumentation component is the collection of specifications to work over set of arguments.

Store

The class S is a collection of concrete representations of arguments, beliefs and argument sets. The store enables the agents and artifacts to write, read, search and consume information in form of arguments. Sharing arguments permits the CAA to calculate argument sets over a common knowledge and for instance fix a global argument acceptability - following acceptance argument definition pro-

vided in 2.2. Moreover, the stores could represent public or private information from agents or artifacts, the access of which is regulated by particular policies.

Argumentation Component

The argumentation component is a set of specifications that should be useful in principle in order to control a set of conflicting arguments. The main functions of this class are to calculate the preferred extensions of a set of arguments and to determine whether a new argument is valid and acceptable. Also, our goals are the utilization of these algorithms by each of the agents of an agent society and by artifacts embodying the social argumentation processes. This would be useful, for example, to identify subsets of arguments agreeable to all participants in a MAS. We adopt the argumentation system presented in the previous section with a tuple-based notation and the Prolog logic language to implement the algorithms. Prolog is very useful because of the uniform representation of code and data, both represented as first-order logic clauses, which makes writing (meta-)interpreters quite easy [39]. Our current specification of the AC follows the preferred semantic and it provides service to calculate: 1) conflict free sets, 2) preferred extensions and 3) admissible sets. In order to compute all these features, AC is composed of several modules and each module provides a specific set of constrains resulting from the analysis of the input argument set.

Argument base module contains all argument of the domain represented by S

Argumentation consistency check module verifies monotonicity of argument composition

Contrary module finds predicates that are in contrast

Argument set module makes argument division in sets be based on a given semantic

Prolog meta-interpreter works over argument set

3.5 Dialog Artifact

As mentioned above, the A&A meta-model for MAS as discussed in [22] views agents engaged in argumentative communication as making use of an abstraction, called a Co-Argumentation Artifact, to communicate, to exchange information, data and arguments, and to record their public commitments. The current work extends this abstraction by formally defining a Dialog Artifact (DA), able to support and mediate the communication between agents engaged in a dialog under the system defined in Section 2.3 above.

We define the Dialog Artifact as a triple $DA = \langle DP, CS, IC \rangle$, where: DP is a collection of specifications of dialog protocols; CS is a collection of commitments stores; and IC is a collection of specifications of interaction control (IC). We now define each of these components in turn.

Dialog Protocols

The class DP is a collection of formal specifications of dialog protocols, with each protocol specified using a labeled process algebra, as in Definition 13. Protocols in DP may also be annotated with identifiers and with their properties, such as their termination complexity. When agents engage in dialog using a protocol in the collection DP, they make utterances according to the permitted sequences

defined by the protocol specification. Accordingly, the Dialog Artifact is able to verify that utterances proposed by agents in a dialog are valid under the protocol; the DA is also able to use the specification to suggest potential legal utterances to participating agents at each point of the dialog.

Commitment Stores

For any particular collection of agents and any particular dialog they undertake, the collection *CS* specifies a set of stores representing the private and public Commitment Stores of each participant, together with a central Commitment Store for the dialog as a whole. The Dialog Artifact can support the dialog by holding these stores. The private Commitment Stores are also held by the DA to record confidential information entrusted to it by the participants, such as their private valuations of some scarce resource (in the case of Negotiation dialogs) or arguments based on privileged information (in the case of dialogues over beliefs). Sharing such information with the DA may allow the DA to reason across these stores in a manner which does not reveal the private information of individual agents.

We can classify these various types of stores according to the access permissions (write-, read-, and delete-permissions) holding on each store, as shown in Table 3.1. The cells of this table indicate the access permissions pertaining to different types of Commitment Stores (the rows of the table), depending on the agent seeking access (the columns of the table). The Dialog Artifact may also store other relevant information, such as the sequence of locutions exchanged in the current dialog, which would be stored in the Central Commitment Store. These stores do not have an algebraic structure but a declarative representation of the contents

Type	Agent A	All Agents	Mediator Artifact
Private Commitment Store of Agent A	R/W/D	-	R
Public Commitment Store of Agent A	R/W/D	R	R
Central Commitment Store	-	R	R/W/D

Table 3.1: Commitment Stores - Read (R), write (W) and delete (D) Permissions

with a proper classification.

Interaction Control

The third component of the Dialog Artifact, denoted as *IC*, is a collection of specifications for interaction control. We roughly follow the pattern MVC (Model View Control), where the model is the dialog specification in *DP*, the view is the *CS* component with dialog trace and the control is represented by *IC* specification. The control rule of the dialog is represented by the label transition system introduced in previous section, modelling the evolution over time of the agent interaction protocol. Three operators can be used to control the dialog:

$$next^I(s) = \left\{ i : s \xrightarrow{i} s' \right\} \; next^S(s) = \left\{ s' : \exists i, s \xrightarrow{i} s' \right\} \; next^{IS} = \left\{ (i, s') : s \xrightarrow{i} s' \right\}$$

Operator $next^I(s)$ yields the next admissible interactions i from state s. Operator $next^S(s)$ yields the states reachable from s in one step. Operator $next^{IS}$ yields couples (i, s) instead.

The component *IC* realizes the above three operators in order to identify which potential utterances for any agent at any point in the dialog are legal. The basic primitives `in`, `rd`, `out` to manage arguments and facts in commitment stores allow the *IC* to identify which constraints on the future course of dialogs are created

by the existing commitments. For instance, the *IC* could permit only one utterance in a chosen point basing the decision on state of commitment store. Also, it can work with argument set over some advanced structures such as conflict free sets and preferred extensions presented in section 2.2 to determine, for instance, an argument acceptability.

DA & CAA Functionalities

It is straightforward to see that all six basic functionalities of a multi-agent argumentation system listed in Section 3.3 can be performed by a Dialog Artifact defined as a triple $DA = \langle DP, CS, LI \rangle$ and Co-Argumentation Artifact $CA = \langle S, AC \rangle$ as above. Basically the CAA is exploited by DA such as its Central Commitment Store. The collection *DP* provides the functionalities of items 1 and 2, the storage of protocols and their formal specifications; the Central Commitment Store of the collection *CS*, which could be realized by *CAA*, provides storage for the history of a dialog, item 3; similarly, the private Commitment Store components of the collection *CS* realized by *CAA* provides storage for confidential information communicated from agents to global DA & CAA, item 6; the formal specification of a protocol in *DP* (as given by the process algebra formalism we have used above) permits the DA to identify potential utterances which do not conform to the protocol, item 4; and both the formal protocol specifications in the collection *DP* and the logics of interaction in *IC* permit the DA to suggest possible legal next moves, item 5.

Chapter 4

Implementation of Dialog and Argumentation Artifact

Logic programming and meta logic programming are two useful techniques to prototype quickly complicated software systems with rational behavior.TuCSoN infrastructure following a Linda like coordination model provides a programmable environment based on logic tuples. In this chapter TuCSoN infrastructure and logic programming techniques are exploited in order to realize the artifacts.

4.1 The TuCSoN Infrastructure

The technological support to build artifacts is provided here by TuCSoN, a coordination infrastructure for MAS introduced in [28]. TuCSoN provides MAS with coordination abstractions called *tuple centres* where agents write, read and consume logic tuples via simple communication operations (`out`, `rd`, `in`, `inp`, `rdp`). As programmable tuple spaces [24], tuple centers can play the role of au-

tomatic agent mediators, where coordination rules are expressed in terms of logic specification tuples of the ReSpecT language—an event driven language over the multi-set of tuples [23].

Tuple centres can play the role of agent coordinators, where coordination rules are expressed in terms of tuples, and also be considered such as a general support for artefacts. As a coordination artifact, a tuple centre is also a container of knowledge declaratively represented through logic tuples, and is equipped with Turing-equivalent computational power through the ReSpecT specification language. There, MAS coordination is obtained by governing the exchange of logic tuples through the tuple centres by properly programming their reactive behaviour.

4.2 CAA Implementation

So, in order to realize a CAA, an obvious choice is to exploit a TuCSoN logic tuple centre. In fact, on the one hand a typical argumentation process is composed of two parts: (1) knowledge representation; and (2) computation over the set of arguments. On the other hand, the tuple centre architecture is also composed of two parts: an ordinary tuple space where the information are stored in form of tuples, and a behaviour specification that defines the computation over the tuple set. Thus, a TuCSoN tuple centre could support the argumentation process by representing knowledge declaratively in terms of logic-tuple arguments, and by specifying the computation over argument set in term of ReSpecT specification tuples. Therefore, our first experimental implementation of a CAA is built as a TuCSoN tuple centre programmed with an argumentation component algorithm (Section 3.4) and with arguments represented by logic tuples (Section 2.2).

Agents use the CAA and whenever a new argument is added to the tuple centre as a logic tuple the CAA reacts and re-calculates the conflict free sets, the admissible sets and the preferred extensions, representing them too in terms of logic tuples in the tuple centre. A complete implementation of ReSpecT reactions joined with the argumentation component implementation is provided in appendix A.3.

Argumentation Component Implementation

From a practical point of view, computational model is based on predicative logic and logic programming. Each argument has its own context, where the argument is true. The context is provided in the argument and is composed only by the set of beliefs – facts and rules – directly declared in the tuple. The connection between the premises and the conclusion is expressed in terms of the corresponding inference process, which is specified in the argument too.

The programs to manage, verify and compare arguments are meta-interpreters written in Prolog. We have created a library composed of interpreters for each type of inference rules supported: MP, MT, Su and Ab. When the component has to evaluate an argument, the program looks for the correct interpreter and checks if the conclusion is a consequence of the premises.

Meta-Interpreter for Argument Check

The following interpreter for argument check (1) has the argument name as its input parameter, (2) asserts all of its facts and rules and (3) verifies its correctness in the different sorts of inference.

```
check_argument(Name):-
```

```
    argument(Name,_,beliefs(facts(F),rules(R)),infer(I),conclusion(C)),
    assert_list(F),
    assert_list(R),
    check_conclusion(I,C).
check_conclusion(mt,[T|C]):-proveMT(T).
check_conclusion(mp,[T|C]):-proveMP(T).
contrary(non(P),P):-!.
contrary(P,non(P)).
```

The contrary term is a support to find opposite predicate. We also add specific relation of opposition like *old vs. young* that in predicate form looks like `contrary(old(X),young(X))` and vice versa; or add the definition of contrary for the subset like a number.

```
% Meta-interpreter for Modus-Ponens
proveMP([]):-!.
proveMP([Goal1|Goal2]):-
  !,
  proveMP(Goal1),
  proveMP(Goal2).
proveMP(Goal):-
  write('call:'),write(Goal),nl,
  (my_clause(Goal,Body);call(Goal)),!,
  proveMP(Body).

% Meta-interpreter for Modus-Tollens
proveMT([]):-!.
proveMT([Goal1|Goal2]):-
  !,
  proveMT(Goal1),
```

```
  proveMT(Goal2).
proveMT(Goal):-
  write('call:'),write(Goal),nl,
  contrary(Goal,NegGoal),
  my_clause(Head ,[NegGoal|T]),contrary(Head,NegHead),NegHead.
```

Example 3 *Check of argument in Modus Ponens and execution trace*

```
argument(arg1,1,beliefs(facts([man(john),age(90,john)]),
                rules([my_clause(old(X),[human(X),age(A,X),A>80]),
                       my_clause(human(X),[man(X)])])),
                infer(mp),conclusion([old(john)])).
?- check_argument(arg1).

assert:man(john)
assert:age(90, john)
assert:my_clause(old(_G385), [human(_G385), age(_G395, _G385), _G395>80])
assert:my_clause(human(_G385), [man(_G385)])
prove:old(john)
call:old(john)
call:human(john)
call:man(john)
call:age(_G430, john)
call:90>80
Yes
```

Example 4 *Check of argument in Modus Tollens*

```
argument(arg3,1,beliefs(facts([non(mortal(eraclito))]),
                rules([my_clause(mortal(X),[human(X)])])),
                infer(mt),
                conclusion([non(human(eraclito))])).
```

```
?- check_argument(arg3).
Yes
```

Meta-Interpreter for Argument Management

Managing the argument set requires in particular an ability to calculate: (1) the relations of undercut and attack between arguments; (2) the conflict-free sets; and (3) the preferred extensions. Undercut and attack relations are found by comparing the 'conclusion vs. conclusion' and 'conclusion vs. beliefs' (and vice versa) between two different arguments. The operation of comparison is done in the argumentation component with the `check/4` predicate. Each argument has to be compared with the others to find all the relations; if we have N arguments we have to do $\approx \sum_{i=0}^{N} N^2$ comparisons. At the end of this process, tracing the `attack(from,to)` and `undercut(from,to)` we obtain a *defeat graph* where the relations are the arcs and the arguments are the nodes, according to Dung [10].

The core of the argumentation component is represented by the interpreters that manage the arguments in order to find the conflict free sets, the admissible sets and the preferred extensions.

Conflict Free Set

The problem of a conflict free set is already known in graph theory with the name of stable set or independent set. It is in the class of NP-hard problem, for which it is very unlikely to find an efficient algorithm. Our idea is to build an algorithm that works incrementally, trying to avoid the complexity of a growing amount of information, because we foresee a dynamic and distributed scenario where agents

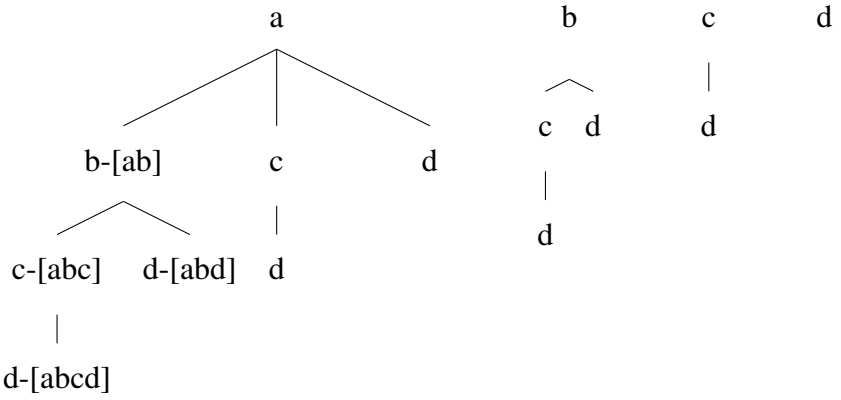

Figure 4.1: Search trees generated for 4 arguments

share their own arguments at different times.

To solve the conflict free problem, we adopt a constraint-based approach. Our algorithm is based on a standard backtracking strategy. The constraint is the absence of conflicts among arguments (undercut, rebuttal). A solution is consistent if the set of arguments satisfies the constraints. In order to limit the degree of backtracking, consistency is checked before each argument is added to the solution. When the consistency check fails, the algorithm stores partial results, and starts backtracking. Then, it recursively tries to add all the remaining arguments.

In order to limit the size of the search space, a branching strategy is used in the phase of set instantiation. The logic program constructs search trees with decreasing depth for all input elements, so that algorithm tries to find all possible solutions around each argument. After such a search process, the selected argument is removed from the next search space. For example, if we consider a list of four input arguments `[a,b,c,d]`, the resulting search trees are shown in figure 4.1. There, the possible partial solutions are denoted in square brackets.

The algorithm can also be used in a dynamic context with inputs in succession.

To find a new solution, after each update we have to insert new arguments in each existing conflict free set, and run the algorithm again. The following Prolog code has been tested in tuProlog 1.3.0 [1] and shows the main predicates implementing the conflict free set division.

```
selection(X,[X|Rest],Rest).
selection(X,[Head|List],Rest) :-
    selection(X,List,Rest).

turn(ArgumentSet):-
        selection(Name,ArgumentSet,RestArgumentSet),
        argument(Name,_,beliefs(facts(F),rules(R)),_,conclusion(C)),
        newconflictfree(RestArgumentSet,[Name],F,C,[Name]).

newconflictfree(Arguments,Result,Facts,Conclusions,ConflictFree):-
        selection(Name,Arguments,RestArguments),
        argument(Name,_,beliefs(facts(F),rules(R)),_,conclusion(C)),
        check(Facts,F,Conclusions,C),
        append1(Facts,F,NewFacts),
        append1(Conclusions,C,NewConclusions),
        add2end(Name,ConflictFree,NewConflictFree),
        newconflictfree(RestArguments,NewConflictFree,NewFacts,
                        NewConclusions,NewConflictFree).

check(FL,F,CL,C):-
        not(control(FL,C)),
        not(control(F,CL)),
        not(control(CL,C)).

newconflictfree(_,[],_,_,_):-!,fail.
```

```
newconflictfree(_,R,_,_,_):-
        mem(P),
        notsubsetset(R,P),
        retract(mem(P)),
        assert(mem([R|P])),!,
        mem(P1),
        fail.
```

Admissible Set and Preferred Extension

An admissible set of arguments is a conflict free set that defeats collectively all its elements, referring back to definition 6. The notion of 'collectively defends' is useful to find a subset of arguments that is more consistent than the conflict free set. The Preferred Extension is the largest set among the admissible sets.

We have to find a conflict free set, where if an argument is attacked, then there exists another argument in the same set that attacks the attacker. This is an indirect form of defense, which we call collective defense.

Our algorithm to resolve the admissible set problem directly uses the conflict free set calculated in the previous section. Also, the algorithm looks only for undercut relations because each argument defends itself from a rebuttal attack but not from an undercut. In a graph representation, the rebuttal relation is a bidirectional arc; on the contrary the undercut relation is a one-direction arc.

The algorithm basically works by subtracting from each conflict free set the arguments attacked but not defended by elements of the same set. The remaining sets represent the solution called admissible sets. The three basic steps that the algorithm does for each conflict free set are: (1) to find defeat arguments with respect to the general set, (2) to find defenders from attackers in the general set,

and (3) to remove defeat arguments without defender. Following, the Prolog code that calculates the admissible sets, again tested in tuProlog 1.3.0.

```
admissible(_,[],[]).
admissible(TotalArguments,[ConflictFreeSet|Rest],Solution):-
      %to find set of attacker to conflict free
      findundercat(TotalArguments,ConflictFreeSet,Attacker,Defeat),
      %it find the defend argument that block the attack
      findundercat(ConflictFreeSet,Attacker,AttackerFromCF,DefeatOut),
      removelist(DefeatOut,Attacker,AttackerNotDefeat),
      findundercat(AttackerNotDefeat,Defeat,AF,DF),
      removelist(DF,ConflictFreeSet,Sol),
      Solution=[Sol|Result],
      admissible(TotalArguments,Rest,Result).

findundercat([],_,[],[]):-!.
findundercat([H|T],CF,A,D):-
      argument(H,_,beliefs(facts(F),rules(R)),infer(_),conclusion([C])),
      contrary(C,P),!,
(argument(Element,_,beliefs(facts([P]),rules(_)),infer(_),conclusion(_))->
(member(Element,CF)->(A=[H|R1],D=[Element|R2]);(A=R1,D=R2));(A=R1,D=R2)),
      findundercat(T,CF,R1,R2).
```

The predicate `findundercut(+General,+Reference,-Attackers,-Defeats)` is used to find the undercut relation among two sets: (1) general (the set with all arguments) and (2) reference (a conflict free set).

The next step is to find the preferred extensions. We use the previous results, and find the preferred extensions by looking for the maximal admissible set, in accordance with the previous definition 7.

4.3 DA Implementation

In order to realize the *DA*, we exploited TuCSoN, a logic tuple centre with programmable behaviour. Agents utter a locution by means of a `out(move(Dialog, AgentID,Locution))` in the tuple space. The automatic actions executed over the commitment store are represented by the term `cs(ID,out(commit(...)))`— where `out` could be replaced by `in` or `rd` operations.

Commitment Store

The *CS* class is composed of `commit` tuples that are put in the tuple space as facts and arguments express in logic tuple notation.The *CS* could be realized by several CAA: one private for an agent and one common for the DA and the agent society. Each CAA has a different access policy, expressed by ReSpecT reaction, that makes the artifact readable or writable only by specific category of agents or artifacts. The DA is connected with CAA by the likability function of TuCSoN infrastructure provided by the `link` operation.

Dialog Protocol

The dialog is written in terms of tuples `dialog(name,AList)` where `AList` is the list of actions reifying in tuple form the operators choice `act(A1)+(act(A2))`, parallel `par(A1,A2)` and sequence `A1,A2`. There are two types of actions *A*: 1) the action `act(_)` expresses the locutions of communication language and 2) the action `cs(_)` expresses the move versus the commitment store.

Example 5 *Example of Dialog State (DP component)*

```
dialogsession(infoseek,close)
```

```
participant(infoseek,2)
dialog(infoseek,[act(C,opendialog(C,T)),
       act(T,opendialog(C,T)),act(C,ask(Arg)+
       (act(T,tell(arg1),cs(T,out(commit(arg1))))))])
currentpar(infoseek,0)
```

Example 5 shows a dialog protocol composed by some basic information on dialog state and few steps of the *information seeking dialog* protocol. The tuples that form the *DP* component are: `participant` (number of participants), `dialog` (dialog protocol), `dialogstate` (actual protocol dialog state), and `currentpar` (actual number of participant). In addition, an open dialog session also uses tuple `session(AgentID,infoseek,open)` for each dialog participant.

Interaction Control

The complete *IC* implementation is shown in appendix A.2, where the reactions implementing the control of dialog interaction are presented. In particular, the code implements the dialog state transition after an agent action, the search of next admissible move after an agent request, and also makes the automatic interaction with the commitment store automatically executing `cs` actions possible. Such mechanisms make it possible for a dialog to be driven automatically by the state of the commitment store. Figure 4.3 shows the ReSpecT implementation of the $next^I$ operator.

The engine of process algebra management is implemented exploiting a transition system defined by the predicates `transition(Currentstate, Action,`

```
%reacts from agent next moves request
reaction(rd(nextmoves(Dialog,S)),(
  rd_r(dialogstate(Dialog,S)),
  out_r(findall(S,Dialog))
)).
reaction(out_r(findall(S,Dialog)),(
  in_r(findall(S,Dialog)),
  findall(A,transition(S,A,Q),L),%collect all next legal moves
  out_r(nextmoves(Dialog,L))
)).
```

Figure 4.2: Implementation of $next^I$ operator in ReSpecT

```
Newstate).

transition(cs(Id,A),cs(Id,A),zero).
transition(act(Id,A),act(Id,A),zero).
transition([Act],A,zero):-!,transition(Act,A,zero).
transition([Act,Act2],A,Act2):-!,transition(Act,A,zero).
transition([Act|S],A,S):-transition(Act,A,zero).
transition(S1+S2,A,R1):-transition(S1,A,R1).
transition(S1+S2,A,R2):-transition(S2,A,R2).
```

Future state of dialog `Newstate` is calculated exploiting the current action and the current dialog state. Next admissible locutions are calculated exploiting a second order query by `findall(Object,Goal,List)` predicate and by `transition(S,A,Q)` predicate as `Goal`. Basically the `findall` collects in a `List` all the solutions

of the query `Goal`.

In our engine for dialog execution the action `cs` are executed and consumed automatically by the artifact. When the next admissible move is a `cs` action, the ReSpecT reactions try to perform it over the right commitment store. If the action is well executed, then the move is consumed and the dialog state consequently advance. The actions (k-OUT, k-RD, k-IN) model an automatic interaction versus the commitment store. They could be used as pre- or post-condition of locutions in communication language in order to automatically conduct the dialog through a right sequence of actions.

```
reaction(out_r(findall(S,Dialog)),(
in_r(findall(S,Dialog)),
findall(cs(Id,Commit),transition(S,cs(Id,Commit),Q),L),
out_r(nextmoves(Dialog,L))
)).
reaction(out_r(nextmoves(D,[H|T])),(
in_r(nextmoves(D,[H|T])),
out_r(excom(H)),
out_r(looknext(D,T))
)).
```

Figure 4.3: Collection of next *cs* actions

Figure 4.3 presents the first step of automatic action execution by collecting all the next admissible `cs` moves. The next step to perform automatically the admissible actions is implemented by the reactions presented in 4.4.

```
%Automatic Action execution on the Commitment Store
reaction(out_r(excom(cs(Id,out(A)))),(
out_r(A),
  in_r(excom(cs(Id,out(A)))),
in_r(dialogstate(Dialog,S)),
  out_r(transition(S,cs(Id,Act),C,Dialog))
)).
reaction(out_r(excom(cs(Id,in(A)))),(
in_r(A),
out_r(excom(cs(Id,in(A)))),
in_r(dialogstate(Dialog,S)),
  out_r(transition(S,cs(Id,Act),C,Dialog))
)).
reaction(out_r(excom(cs(Id,rd(A)))),(
rd_r(A),
in_r(excom(cs(Id,rd(A)))),
in_r(dialogstate(Dialog,S)),
        out_r(transition(S,cs(Id,Act),C,Dialog))
)).
```

Figure 4.4: Automatic Action execution on the Commitment Store

Chapter 5

Case study over ADR Systems

5.1 Overview of Alternative Dispute Resolution

In a social context conflicts are often inevitable. From a human point of view the different culture, own interest and partial consciousness are often the causes of disputes. People develop systems and methods in order to settle conflicts in a fair way. They provide norm systems, infrastructures (such as court) and methods (such as trial) to achieve the dispute resolution.

In a global business process scenario there is a increasingly necessity for a speed-up of the processes, and for faster conflict resolution. The new systems have to support legal process, for instance when a negotiation is broken, or to combine mediation and legal service to avoid litigation.

Alternative Dispute Resolution (ADR) is usually considered to be alternative to litigation. It can also be used as a colloquialism for allowing a dispute to drop or as an alternative to violence. ADR is generally classified into at least four subtypes: negotiation, mediation, collaborative law and arbitration. Walker and

Daniels [42] underline that legal negotiation is a part of traditional dispute resolution system rather than a component of the ADR movement. The legal negotiation happens directly among agents that represent the disputants in a context similar to courtroom.

Arguments have a central role in the process of formal legal system and in the trial. [35] shows a survey of logic in computational model on legal argument. It presents the main architecture of legal arguments with in background a four layer architecture: 1) logical layer, 2) dialectical layer, 3) procedural layer and 4) strategic layer. Disputants use arguments in order to persuade the other parts of the dispute and also the decision makers - juries, judges, clients and attorneys. [32] considers the use of argument in ADR systems and it is presents an analysis of arguments in different context such as arbitration, mediation and multi-party facilitation. Argumentation plays an important part in conflict resolution system, since it drives the ADR to obtain a successful solution of the dispute. Argumentation process promotes the values of justice, equality and community that are desirable in a dispute resolution system.

In an open agent, society holds the same role as human society: it is undesirable to resolve dispute by litigation. The develop of a system for internal disputes resolution in virtual organisation is purposed by Jeremy Pitt et al in [32]. It provides a norm-government MAS and an ADR protocol specification for virtual organization exploited by intelligent agents.

ADR supplies a theoretical bases of Online Dispute Resolution (ODR) defined in [38]. ODR has the purpose to extend ADR process, moving it to virtual environment and providing computation and communication support. In ODR what is crucial is the role of technology, which used to facilitate the resolution of dis-

putes between parties. It provides a structured communication and an informed environment that helps the successful conclusion of the conflict.

ODR could be seen such as an instance of ADR system, communication infrastructure and Artificial Intelligence (AI) techniques aiming at supporting the parties towards agreements. The reasoning and argument ability of the parties are realized by AI methods.

In this section we aim at providing a framework for conflict resolution in an agent-based society. We want to supply a supporting infrastructure in order to manage arguments, to retrieve information and to bargain.

In Walton and Godden [45] is shown that argument-based dialogs, in particular persuasion dialogs, contribute to the realization of effective dispute resolution systems. The main type of dialog usually considered by ADR is negotiation, but it could be interpreted such as a particular kind of communication for the purpose of persuasion. In argumentation theory are present both types of dialog: persuasion dialog and negotiation dialog. These two types of dialog have a different structure and different goals and in ODR systems have to be managed by different procedural rules.

A complete classification of six primary dialogs models in which argumentation occur is presented by Walton in *The New Dialectic* [44]. The set of these dialogue is composed by: persuasion, inquiry, negotiation, information seeking, deliberation and eristic. All these dialogues can be captured in an argumentation framework [29], and they are developed strictly among two entities.

Fundamental problem in ODR and ADR systems is that it is difficult to structure and process the information that is exchanged between negotiating parties. In order to solve this problem, we propose to build an ADR system to use A&A

meta model with Co-Argumentation Artifact and Dialog Artifact abstraction explained in the previous sections. Our framework provides a structured information based on logic tuple and the control of dialog processes through a mediate form of communication over a programmable infrastructure. These two characteristics are useful in order to build MAS in a scalable and flexible architecture and also to build ADR that supports multi-party dialog session.

5.2 Architecture for ADR

We propose our architecture for MAS based on A&A metamodel to design a ADR/ODR application. An ADR system, especially on-line, exploits the form of negotiation, arbitration or mediation to achieve a solution. Typically in that system the entities involved are more than two: two participants and a third entity to help the dispute resolution like in mediator and arbitrator procedure. The parties involved choose the procedure, terms and condition of their dispute.In [32] are presented an arbitration protocol and concepts of decision making through formation and voting protocol.

The parties, in order to find a solution, have the possibility to share any pertinent argument, make demands and evaluate the acceptability of an argument with respect to normative context. To do that are a multi-party dialog protocol and an impartial computation over the shared knowledge necessary. When the dispute involves an increasing number of participants, it is necessary to introduce a mediate form of communication in order to have a scalable system. Also, in the role of mediation there are often evaluations that could be done automatically.

In that scenario our architecture provides the correct abstractions: 1) Dialog

Artifact 2) Co-Argumentation Artifact to make a flexible system. In the DA we store the arbitration, mediation or negotiation protocol. The parties exploit the DA to take part in the discussion, which drives the dialog ground on the commitments. The advantages are the management of dialog between multiple entities and the automatic interaction with commitment/argument store. The CAA provides the right abstraction to make a commitment/argument store and where possible, to evaluate automatically the argument validity as to the normative context. Also, it provides default function to exchange information, data and arguments, and to record their public commitments in private or public form. In a bargain among three or more entities, for instance, through a CAA the final set of arguments stored during the bargain represent a form of contract among the parties.

In the following paragraph we focus on the persuasion dialog, that is among the most common and useful dialog in ADR. An interesting observation in [45] underlines that a negotiation dialog can naturally include or shift to persuasion dialog almost in two points: 1)to follow an offer and 2) to follow a rejection of an offer. In both cases are provided reasons (by argument) to proof the acceptability or unaccetabilitiy of an offer. A dialog model for persuasion could be composed of: 1) commitment store for each participant, 2) inference rule to draw conclusion from commits in the commitment store made by the participants and 3) practical rules that govern the sequence of locutions and their consequences.

5.3 Persuasion Dialog

In persuasion dialog the goal of a participant is to prove his thesis and to rationally persuade the other parties. With the word “persuasion” we do not mean a psycho-

logical persuasion, but rather a rational persuasion supported by arguments. In Walton & Krabbe [44] they observe that disputes are a subtype of persuasion dialog where the parties disagree about a single proposition φ. At the start of the dialog a party believes in φ and the other believes in $\neg\varphi$, therefore they have a contrary opinion about a proposition. Generally, the moves that are allowed in the dialog are: asking question, answering question, putting forward arguments. Following Walton [45], a proponent in a persuasion dialog is successful when: 1) the responded has committed all the premises of the argument 2) each argument is corrected 3) the chain of argument has the proponent thesis as its conclusion

In [34] is presented a survey of formal systems of persuasion dialogue that points out the crucial role of regulating interaction among agents rather than design of behaviour in individual agent within a dialogue. Among the main approaches to design persuasion dialogue and communication between agents based on arguments, we keep inspiration from Parson and McBurney [29] approach and Prakken [33] approach. And also from [2], where it is shown how each move of the dialog could be specified by rational rules, dialog rules and update rules making explicit the relation with the commitment store.

5.3.1 CAA

The CAA provides the services to read, store and consume arguments and beliefs such as a commitment store. And also following the general architecture for MAS propose in 3.4 we provide a private and a public CAA. In particular for persuasion dialog we exploit the ability of the CAA, in order to automatically calculate arguments and beliefs acceptability following the agent attitudes and the

argumentation semantic. In [29] are introduced agent attitudes in order to provide some acceptability criteria. An agent may have one of three acceptance attitudes about proposition: a *credulous* agent can accept any formula for which there is an argument S, a *cautious* agent can accept any proposition for which there is an argument if no stronger rebutting argument exists, a *skeptical* agent can accept any proposition for which there is an acceptable argument S. Exploiting our argument definition and referring to our argumentation system, the argument acceptance is resolved following the argumentation preferred semantics. In the context of preferred semantics the argument acceptance is divided in credulous acceptance or sceptical acceptance, if an argument is in some/all preferred extension.

The CAA validates the arguments committed verifying their correctness and also it evaluates their acceptability verifying which of the preferred sets it belongs to. In particular for persuasion dialog the CAA supports the agent attitudes and reacts from the tuple `rd(cs(Y,acceptable(Attitude,P)))` where `Attitude` could be `credulous`, `cautious` or `skeptical`, in order to verify the acceptability of *P* formula grounded on agent attitude.

5.3.2 DA

The DA is the abstraction that encapsulates the rules of dialog and it coordinates the entities during the persuasion process. We follow the definition of this artifact provided in 3.5, that supplies all components definition. We also propose a dialog persuasion protocol formalized with our process algebra. The commitment store of DA is provided by CAA correctly implemented, and we suppose for simplicity that it is in the same space of the DA.

Communication Language

The more common locutions on persuasion dialog that can be found in literature are well collected in [34] and they are briefly listed here:

- claim φ (assert): The agent asserts a formula φ to start the persuasion.
- why φ (challenge): The agent asks for reasons about the φ formula.
- concede φ (accept): The agent accepts the validity of φ.
- reject φ(retract): The agent does not commit the φ. In some cases it retracts the formula from the commitment store previously stored.
- *S* since φ (argue): The agent provides reasons for φ formula by an argument.

Our communication language is composed of the following set of locutions: *assert*, *why*, *accept*, *reject*, *retract*, and *argue*.

Dialog Protocol

To make a persuasion dialog concrete, a persuasion protocol is defined, typically among two parties: proponent and respondent. We formalize through our process algebra a generic persuasion dialog protocol keeping inspiration from [31, 2] and adding repetition rule proposed by [34]. Exploiting the expressive ability of the defined process algebra we can express the agent attitude directly in the protocol enabling an automatic checking of those properties. The dialog is partially driven through the state of commitment store by the actions (in, out, rd) that are specifiable in the protocol.

```
dialog_persuasion(X,Y,P):= X:assert(P).dialog_response(X,Y,P)

dialog_response(X,Y,P):=
    Y:accept(P) +
    Y:reject(P) +
    Y:why(P).
       X:argue(argument(Name,bel(B),inf(I),conc(P))).
       dialog_argue(X,Y,P)

% Evaluation of chain argument support of P assertion
dialog_argue(X,Y,P):=
    Y:accept(P) +
    Y:reject(P) +
Y:argue(argument(Name,bel(B1),inf(I1),conc(P1))).(
     X:retract(P) +
     X:argue(argument(Name,bel(B2),inf(I2),conc(P2))).dialog_argue(X,Y,P)))
```

Figure 5.1: Persuasion dialog without interaction with the CS

In figure 5.1 is shown a dialog protocol for persuasion where an agent can accept or reject an assertion *P*, based on its attitudes, by an internal evaluation of facts and argument acceptability. Then starts a phase of arguing conclude with an acceptance or rejection of the assertion *P*. The relation among dialog and commits is not explicitly expressed. In a dialog each move could be specified by rational rules, dialog rules and update rules [2]: the rational rules specify the preconditions for playing a move; The update rules specify the modification of commitment store; the dialog rules specify the next moves. With our process algebra we have the expressive power to cover the three types of dialog rules. For instance we propose a modified version of the persuasion protocol in the figure 5.2 where we provide an automatic evaluation of some preconditions(rationality) and the consequent modification of the commitment store (update). In that version of the

dialog the DA automatically drives the sequence of action through the state of the commitment store. In the choice points some locutions are automatically chosen. We exploit also the ability of the CAA to find beliefs acceptability following the agent attitudes and the argumentation semantic.

This protocol formalization is very flexible and opens a lot of courses of actions. The problems could be the termination of the dialog and the determination of the dialog results. The dialog is partially automated through DA and CAA infrastructure. The agents have every time the control over their own actions and they can decide in every moment to suspend the dialog. From the responder agent point of view this type of dialog could be interpreted such as a dialog to build a new argument. The new argument is composed of the tree branches from last locution to the first assertion. A new belief is accepted if there is a course of action from the last uttered locution to the first assertion.

5.3.3 Example of Run

In this section we provide an example of simulation of a simplified version of persuasion dialog exploiting TuCSoN infrastructure. The purpose is to show the use of the infrastructure. In order to perform the dialog simulation TuCSoN provides useful tools: *CLIAgent* to simulate agents interaction and *Inspector* to inspect current state of tuple space . The Inspector tool shown in figure 5.3 allows users to observe and debug the communication state and the behaviour of a tuple centre. In particular, it makes it possible to inspect the tuple set, the pending query set, the triggered reaction set and the behaviour specification set.

The TuCSoN *CLIAgent* tool allows users to invoke the commands of the TuC-

```
dialog_persuasion(X,Y,P):= X:assert(P).dialog_response(X,Y,P)

dialog_response(X,Y,P):=
   rd(cs(Y,acceptable(P))).Y:accept(P) +
   rd(cs(Y,acceptable(not(P)))).Y:reject(P) +
   Y:why(P).X:argue(argument(Name,bel(B),inf(I),conc(P))).dialog_argue(X,Y,P).(
        rd(_,acceptable(B).Y:accept(P) +
        rd(_,acceptable(not(B)).X:retract(P).in(X,assert(P))
        +else dialog_argue(X,Y,P)
   )

% Evaluation of chain argument support of P assertion
dialog_argue(X,Y,P):=
   Y:accept(P).out(Y,commit(P)) +
   Y:reject(P) +
Y:argue(argument(Name,bel(B1),inf(I1),conc(P1))).(
     rd(_,acceptable(B1)).out(_,commit(P1)) +
     X:retract(P).in(X,assert(P))+
     X:argue(argument(Name,bel(B2),inf(I2),conc(P2))).(
       rd(_,acceptable(B2)).out(_,commit(P2)) +
       dialog_argue(X,Y,P)
     )
```

Figure 5.2: Persuasion dialog with CS interaction: Automatic evaluation of acceptability

SoN coordination language. For our purpose we exploit the *CLIAgent* to utter agent locution in the form `out(move(Dialog,Id,Locution))`.

The rules to manage the dialog in the DA are programmed with the ReSpecT code in appendix A.2, for the commitment store is considered the same tuple space of dialog and the initial dialog state is expressed by the tuple

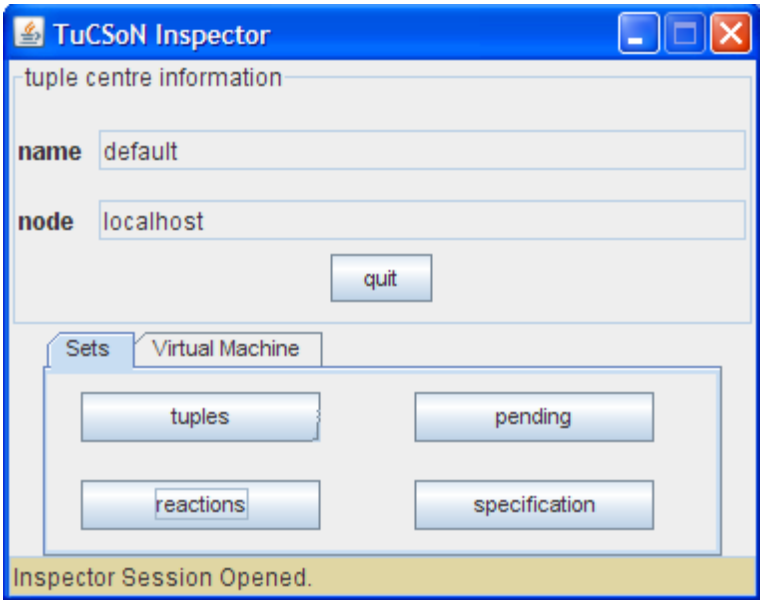

Figure 5.3: Inspector tool

```
dialogstate(persuasion,[act(X,assert1(P)),
(act(Y,accept(P))+act(Y,reject(P)))+act(Y,assert1(non(P)))+
act(Y,why(P),act(X,argue(argument(N,bel(B),inf(I),conc(C)))),
(act(Y,accept(N))+ act(Y,reject(N)))]).
```

The locutions that could be uttered in that dialog are: *assert*, *accept*, *reject*, *why*, and *argue*. We start the simulation sending an *assert* locution in the tuple centre from agent *Paul* by the CLIAgent shown in the figure 5.4. After that move the infrastructure reacts and it calculates next dialog state.

```
move(persuasion,paul,assert1(safe))
dialogstate(persuasion,
['+'('+'('+'(act(_4,accept(safe)),act(_4,reject(safe))),
   act(_4,assert1(non(safe)))),act(_4,why(safe))),
```

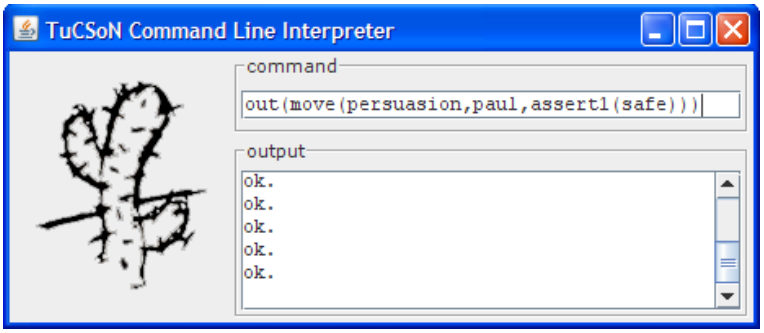

Figure 5.4: CLIAgent

```
act(paul,argue(argument(_3,bel(_2),inf(_1),conc(_0)))),
'+'(act(_4,accept(_3)),act(_4,reject(_3)))])
```

The responder agent *Olga* can ask the tuple `rd(nextlocutions(persuasion,L))` for the possible admissible next locutions and the tuple centre responds by new tuple `nextlocution`.

```
nextlocution(persuasion,
    [act(_2,accept(safe)),act(_2,reject(safe)),
    act(_1,assert1(non(safe))),act(_0,why(safe))])
```

At this point the responder chooses a move either from the state of commitment store or independently from our knowledge base, for instance in this case the choice could be *why(safe)*. The figure 5.5 shows the state of the tuple centre after *Olga* locution by the inspector tool. The new `dialogstate` expresses the remaining locution constrained by previous logical unification of *paul* and *olga* identifiers.

```
dialogstate(persuasion,[act(paul,
```

```
argue(argument(_3,bel(_2),inf(_1),conc(_0)))),
'+'(act(olga,accept(_3)),act(olga,reject(_3)))])
```

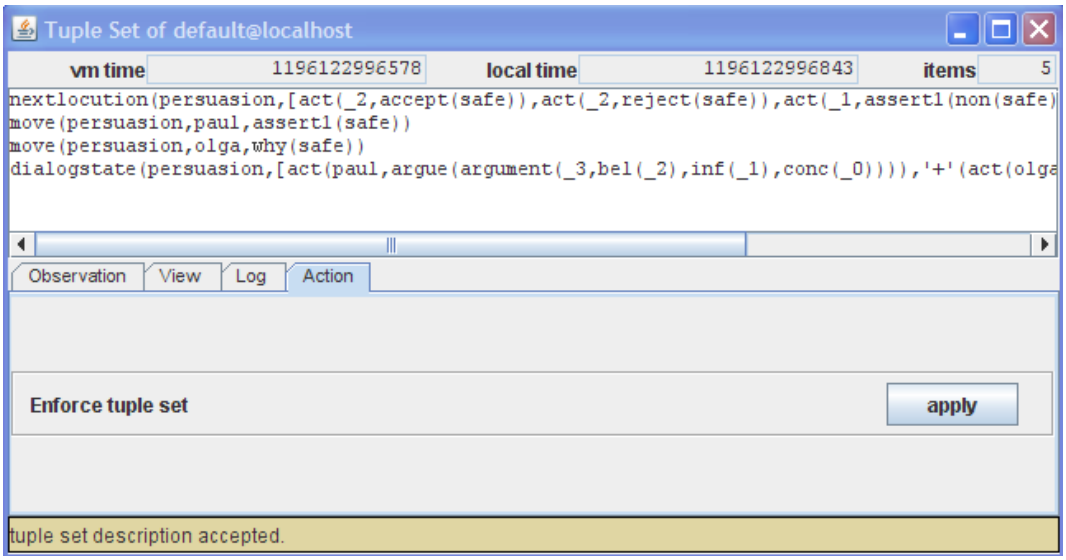

Figure 5.5: Tuple Set

5.4 Argument Acceptance

A tool to find argument socially acceptable could be useful in ADR process. The key idea is to form a common knowledge acknowledged by all the participants whereby the parties or third figures such as mediator, arbitrator or attorney use to resolve their disputes. In a collaborative divorce, for instance, the parties look for an agreement with the support of the attorneys. No one imposes a resolution on the parties. An automatic evaluation of argument agreement could be useful in order to speed up the solution or may eliminate in similar cases the figure of the attorneys. More easily we present a example of dispute resolution among friend

that have to choose the activity of Saturday night.

We present an application of CAA in a multi-agent context where agents have to decide whether their arguments are socially acceptable. We use the argumentation system presented in Section 2.2 with preferred semantics, and either credulous or sceptical acceptance. An argument is considered as accepted in the credulous definition if it is contained at least in one preferred extension, and in the sceptical definition if it is contained in every preferred extension. In [5] an algorithm is presented that resolves the credulous and the sceptical decision problems based on an argumentation game formalised with a dialog between two entities. The algorithm could be applied either inside each agent simulating a dialog game, or between two agents. In order to extend the solution to N agents, we propose to use the A&A meta-model by adopting the CAA abstraction.

We foresee a scenario where a group of agents argue about what to do on Saturday night. For instance, the agents are conditioned from the past history of the place where to go, or the possible company. Each agent has its arguments about whether to go or not to go to, say, the El Farol Bar. In order to make a personal evaluation the agents may benefit from social information that could be retrieved asking other agents. Besides, when the agents share their arguments, a form of social knowledge is implicitly generated, which provides agents with a social point of view on the Saturday night problem. Also, sharing knowledge and arguments gives the group more chances to take congruent decisions.

More generally, social contexts typically introduce the need to represent and store social knowledge. Since shared, social knowledge belongs in principle to every agent, so to no agent in particular, it should be stored and maintained outside agents: in short, this is what makes it useful to introduce in this scenario the

notion of artifact, as an abstraction that agents can use to share, compare and store information.

Here, we consider agents with different knowledge bases composed only of arguments, and an empty CAA only containing the algorithms proposed in the argumentation component. The arguments acceptance is driven generally by a system process divide in to three sequential steps. First, the agents share their own arguments writing the arguments in the CAA. Secondly, the CAA reacts and calculates the conflict free and preferred extension over the shared arguments. Thirdly and finally, the agents evaluate credulous or sceptical acceptability based on common sets calculated in the CAA. Then, each agent can consult the CAA to understand the "social acceptability" of its own arguments, but also the other agent's arguments, and possibly deliberate its course of action based on a shared view of arguments. Also, the CAA keeps track of the overall argumentation process, and could be exploited by an external observer to understand the social behaviour of agents sharing arguments and behaving accordingly.

In particular, in our example the CAA is implemented as a TuCSoN tuple centre called `saturdayNight`, which processes and combines knowledge expressed by arguments from various agents. In Table 5.1 the arguments possessed and shared by the three agents are shown. Some arguments are in favor of going out if the conclusion is `play(1)`, or vice versa is `play(-1)`. The support of conclusions should contain the motivation to do the choice, for instance: a favorite kind of music `music(rock)`, a previous nice night `result(1)` or a good company `willgo(susan)`. Different sets of arguments represent different opinions and motivations that bring an agent to make a decision. The sharing of the arguments enables the composition and completion of the information.

The sets calculated in CAA are expressed with the tuples `conflictfreeset`, `admissibleset` and `preferredset` and calculated using the algorithm explained in section 3.4. An external observer can look inside the CAA through the Inspector utility provide by TuCSoN, and consult the argument sets. In the following we show the sets computed after the last argument insertion. The sets contains the argument names.

```
conflicfreeset([[argB,argC,argD,musicB,companyB,dayA,dayB,typemusic],
[argB,argC,argD,musicB,companyB,day,dayA,typemusic],
[argB,argC,argD,music,companyB,dayA,dayB],
[argB,argC,argD,music,companyB,day,dayA],
[argA,companyA,day,typemusic],
[argA,music,musicA,companyA,day]])

admissibleset([[argB,argC,argD,companyB,dayA,typemusic],
[argB,argC,argD,companyB,day,dayA,typemusic],
[argB,argC,argD,music,companyB,dayA],
[argB,argC,argD,music,companyB,day,dayA],
[argA,companyA,day,typemusic],
[argA,music,companyA,day]])

preferredset([[argA,music,companyA,day],
[argA,companyA,day,typemusic],
[argB,argC,argD,music,companyB,day,dayA],
[argB,argC,argD,companyB,day,dayA,typemusic]])
```

One should observe that the global preferred sets are different from the ones that each agent could calculate based on its own arguments only. Agents could then read the `preferredset` tuple, and verify in which set its own arguments occur.

`agent1`, for instance, may want to consider the social acceptability of argument `musicB` that in its own knowledge is accepted, because the argument belong to their own preferred set `([argB,argC,day,musicB])`. Vice versa, when considering the common preferred extension, the argument is no longer (socially) acceptable because it does not belong to a common set. These sets are calculated from more information than the ones available to each individual agent, and in some context they could be considered as more reliable. In any case, agents can autonomously decide what to do with such information—either use it or ignore it.

Table 5.1: Arguments by Agent1, Agent2, and Agent3

Agent1

```
argument(argB,1,beliefs(facts([result(-1)])),infer(t),conclusion([play(-1)])).

argument(argC,1,beliefs(facts([result(1)])),infer(t),conclusion([play(-1)])).

argument(day,1,beliefs(facts([today(sunday)])),infer(t),conclusion([today(sunday)])).

argument(musicB,1,beliefs(facts([non(music(rock))])),infer(t),conclusion([play(-1)])).

argument(dayB,1,beliefs(facts([non(today(sunday))])),infer(t),conclusion([play(-1)])).

...
```

Agent2

```
argument(music,1,beliefs(facts([music(rock)])),infer(t),conclusion([music(rock)])).

argument(argD,1,beliefs(facts([result(-1)])),infer(t),conclusion([play(-1)])).

argument(companyA,2,beliefs(facts([willgo(susan)])),infer(t),conclusion([play(1)])).

argument(companyB,2,beliefs(facts([non(willgo(susan))])),infer(t),conclusion([play(-1)])).

argument(musicA,1,beliefs(facts([music(rock)])),infer(t),conclusion([play(1)])).

...
```

Agent3

```
argument(argA,1,beliefs(facts([result(1)])),infer(t),conclusion([play(1)])).

argument(typemusic,1,beliefs(facts([imtired(yes)])),infer(t),conclusion([non(music(rock))])).

argument(dayA,1,beliefs(facts([today(sunday)])),infer(t),conclusion([play(-1)])).

argument(company,1,beliefs(facts([willgo(susan)])),infer(t),conclusion([willgo(susan)])).

...
```

Chapter 6

Conclusions

6.1 Related works and Discussion

In literature there are several logical models to formalize reasoning among arguments that are called by Prakken and Vreeswijka in [36] Argumentation Systems. Different logical models of arguments formalize "*commonsense*" reasoning. For instance giving rise to two conflicting arguments, accepting some premises or building a counterargument could be considered commonsense reasoning. They are typically human actions whence to study inference patterns.

To represent our arguments we use a meta logic approach based on first order logic (FOL) where the conflicts and reasoning are computed at argument level. The arguments are build with classical inference exploiting monotonic logic and the commonsense reasoning is brought at the meta level. We follow an abstract framework for defeasible argumentation developed in several articles from Dung et al [10] that is a completely abstract version of Bondarenko et al [4] framework.

Another approach to build arguments is based on defeasible logic. It is a

non-monotonic logic that provides directly at language level three types of special rules to model undercutting, rebuttal and defeater relation to express conflicts. The fitness of defeasible logic for argumentation is stressed out by Governatori et al in [15] and Garcia et al [12]. Moreover, there are also some other approaches that extend defeasible logic and normal logic adding probability and certainty factors, [16] in order to increase the expressiveness, and modalities to weight arguments, [8] in order to resolve conflicts. An unifying approach to computational models of argument using Labelled Deductive Systems (LDS) is proposed in [7].

Argumentation-based communication between agents is exploited to define forms of rationality that drive the agents to accept or reject statements. Different dialog formalizations are presented by Prakken et al [33] and Parsons et al [31], both based on arguments. Walton and Krabbe in [44] presents six main type of dialog: persuasion, inquiry, negotiation, information seeking, deliberation and eristic that can be captured in a argumentation framework as that one presented in [31]. Our dialog formalization is based on process algebra. We exploit a dialog system similar to the generic framework for dialog presented by Maudet et al in [19]. We use a common and private board (mediator artifacts) to represent commitment stores, the rules of dialog expressed by a process algebra with commitment store interaction and the inferences automatically calculated by the infrastructure. We present a formalization of some of these mediator artifact functions drawing on the theory of communications artifacts in multi-agent systems [25, 41]. In earlier work [22], we presented a conceptual framework for a central co-ordinating entity in an argumentation dialog, called a Co-Argumentation Artifact (CAA), to provide co-ordination services to the participating agents.

There is also the line of research based on argumentation schemes by Walton

[43], which tries to decrease the gap between human linguistic argumentation and logic based approaches. The main drawback consists in the hardness of building computational models for them. Rahwan et al [37] wrote an interesting paper that drives a new direction of argumentation theory in the context of World Wide Web application. It extends the Argument Interchange Format (AIF) [6] ontology with Argumentation Schemes [43], in order to use both logic-based reasoning and schemes based reasoning. We share with AIF a very similar definition of argument composed of three components: beliefs, inference rules and conclusions. Our meta logic approach with implementation in TuCSoN infrastructure fits very well in the new application scenario of AIF ontology. An ontology expressed by RDF meta data model could be equivalently represented in FOL and reified in tuple form in the tuple center. In this way we can exploit by our artifacts all the AIF-RDF ontology power, opening new innovative scenarios. For instance agent and humans could collaborate exchanging arguments in WWAW context through a Co-Argumentation Artifact. Moreover, our meta reasoning approach emerge to be very useful to reason over the set of web arguments. The use of TuCSoN coordination model inside a web architecture is already introduced by Omicini et al in the paper [28].

6.2 Conclusions

In this thesis we have presented a completed work from theory to implementation level. At the theory level we define a formal argumentation and dialog system following respectively a meta logic approach based on FOL and an algebraic approach based on process algebra representation. At the model level following

A&A meta-model for MAS we have proposed a conceptual architecture for multi-agent argumentation system in which the agents are assisted by an automatic mediation, provided by Co-Argumentation Artifact and Dialog Artifact abstraction. At the coding level we reported on a prototype implementation of these ideas exploiting programmable Tuple Space framework TuCSoN. We provide an implementation of the desired functionality of the two artifacts. The case study of ADR shows how it can efficiently applicate our system on an agent-based context. ODR extends ADR moving it to the virtual environment of the Internet. Exploiting TuCSoN infrastructure we provide a framework that enables the dispute resolution on the Internet and an automatic mediation that makes the dispute resolution process easier. The persuasion dialog and argument acceptance are two concrete realizations of ADR/ODR systems.

6.3 Further Research

In future work, we want to promote a formalization of argument composition of different type of arguments with dfferent inference rules. We hope to instrument an appropriate artifact to safely combine, if possible, arguments based on different inference rules such as strict and presumptive.

Moreover, we hope to formalize more of the potential functions of the mediator which we listed above in Section 3.5. Some of these functions will be straightforward to formalize, to identify conflicts between commitments or provide automated alerts to agents concerning upcoming dialogs. Others, however, such as run-time assignment of rights and responsibilities to dialog participants, will be more challenging.

Finally, we desire to concretely open the possibility to add, in our artifacts, arguments based on presumptive inference, and to generalize our argument representation with the AIF ontology. The vision is to validate presumptive argument through an automatic revision of critical questions. We aim at obtaining an advanced tool for a WWAW based on artifact, where agents and humans could indistinctly interact.Our tool encapsulates and consistently handles the evolution of Web social knowledge based on argument ontology, and it provides agents and humans with an instrument to dialog and to enhance their ability to deal with their own partial knowledge.

Appendix A

Implementation and Example

A.1 Example of Prolog Meta-Interpreter for legal reasoning

In this section a meta-interpreter is proposed written in Prolog for the production of legal argument. That program follows the idea presented by Trevor et al in [3]. We exploit a labelled theory `term:label` to record the extra logic information require to organize the explanation. The labels `class`, `data`, `condition`, `qual` identify different roles of Prolog clauses and the results of computation are a set of arguments based on Stephen Toulmin argument schema.

```
%Labelled theory
age(john,80).
male(john).
greater_than(A,70):condition :-
A > 70.
```

```
man(X):-
human(X):class,
male(X):data.
old(X) :-
man(X):class,
age(X,A):data,
greater_than(A,70):condition,
not(tibetan(X)):qual.

%Meta-iterpreter for argument generation
body_list(',',(Goal,OtherGoals),[Goal|L]) :-
body_list(OtherGoals,L).
body_list(LastGoal,[LastGoal]).

make_arguments([Goal|OtherGoals],C):-
argument(Goal,C),
make_arguments(OtherGoals,C).
make_arguments([],C).

argument(':'(G,class),C):- C1 is C+1,clause(G,B),
    B=\=true,meta(G,C1),nl,write(context(C,G,C1)),!.
argument(':'(G,class),C):- nl,write(context(C,G)),!.
argument(':'(G,data),C):- G,nl,write(data(C,G)),!.
argument(':'(G,qual),C):- G,nl,write(rebuttal(C,G)),!.
```

```
%argument(':'(G,condition),C):- nl, write('condition'),!.
argument(G,C).

%warrant.claim,if,data,cond
correct_type(G,data).
correct_type(G,condition).

append1([],L,L).
append1([H|T],L,[H|O]):-append1(T,L,O).

make_worrant([],[]).
make_worrant([':'(Goal,Type)|OthersGoals],[Goal|R]):-
correct_type(Goal,Type),!,
make_worrant(OthersGoals,R).
make_worrant([_|OthersGoals],Lout):-
make_worrant(OthersGoals,Lout).

type_basis(G,data).
type_basis(G,condition).
type_basis(G,class).

make_basis([],[]).
make_basis([':'(Goal,Type)|OthersGoals],[Goal|R]):-
type_basis(Goal,Type),!,
make_basis(OthersGoals,R).
```

```
make_basis([_|OthersGoals],Lout):-
make_basis(OthersGoals,Lout).

make_worrants(Claim,Count,Lin,Lout) :-
make_worrant(Lin,Lout),!,
append1([Claim,if],Lout,Lout1),
nl,write(warrant(Count,Lout1)),
make_basis(Lin,LoutB),
append1([Claim,if],LoutB,LoutB1),
nl,write(basis(Count,LoutB1)).

%qual-rebuttal
%class-context
%data-data

meta_interpreter(G):-
tell('argument.pl'),
meta(G,1),!,
told.

meta(X,C) :-
clause(X,B),
nl,write(claim(C,X)),
body_list(B,L),!,
make_arguments(L,C),
```

```
clause(X,B1),
body_list(B1,L1),!,
make_worrants(X,C,L1,A).
```

We exec the following java command the labeled theory `java -cp 2p.jar;./alice.tuprolog.Agent meta.pl meta_interpreter(old(john)).` and we generate a list of possible arguments that follow Toulmin schema.

```
claim(1,old(john))
claim(2,man(john))
context(2,human(john))
data(2,male(john))
warrant(2,[man(john),if,male(john)])
basis(2,[man(john),if,human(john),male(john)])
context(1,man(john),2)
data(1,age(john,80))
rebuttal(1,not(tibetan(john)))
warrant(1,[old(john),if,age(john,_142),
        greater_than(_142,70)])
basis(1,[old(john),if,man(john),age(john,_142),
      greater_than(_142,70)])
```

A.2 ReSpecT Implementation of DA

The Control of Interaction: Checking agent legal locution, Making dialog protocol transition and executing automatically *cs* actions are the basic function here

implemented in ReSpecT.

```
transition(cs(Id,A),cs(Id,A),zero).
transition(act(Id,A),act(Id,A),zero).
transition([Act],A,zero):-
           !,transition(Act,A,zero).
transition([Act,Act2],A,Act2):-
           !,transition(Act,A,zero).
transition([Act|S],A,S):-transition(Act,A,zero).
transition(S1+S2,A,R1):-transition(S1,A,R1).
transition(S1+S2,A,R2):-transition(S2,A,R2).
%Start reaction
reaction(out(move(Dialog,Id,Act)),(
   in_r(dialogstate(Dialog,S)),
   out_r(transition(S,act(Id,Act),C,Dialog))
)).
reaction(out_r(transition(S,A,S1,Dialog)),(
   transition(S,A,S2), %make the state transition
   in_r(transition(S,A,S1,Dialog)),
   out_r(dialogstate(Dialog,S2)),
   out_r(findall(S2,Dialog))
)).
reaction(out_r(findall(S,Dialog)),(
   in_r(findall(S,Dialog)),
   %collect all next commits
```

```
   findall(cs(Id,Commit),transition(S,cs(Id,Commit),Q),L),
   out_r(nextcsmoves(Dialog,L))
)).
reaction(out_r(nextcsmoves(D,[H|T])),(
   in_r(nextcsmoves(D,[H|T])),
   out_r(excommit(H)), %call execution commit
   out_r(looknext(D,T))
)).
reaction(out_r(looknext(D,[E])),(
   in_r(looknext(D,T)),
   out_r(nextcsmoves(D,T))
)).
reaction(out_r(looknext(D,T)),(
   T==[], in_r(looknext(D,[])),
   in_r(nextcsmoves(D,[]))
)).
%Implementation of K-OUT, K-IN and K-RD
reaction(out_r(excommit(cs(Id,out(A)))),(
   out_r(A), in_r(excommit(cs(Id,out(A)))),
   in_r(dialogstate(Dialog,S)),
   out_r(transition(S,cs(Id,Act),C,Dialog))
)).
reaction(out_r(excommit(cs(Id,in(A)))),(
   in_r(A),  out_r(excommit(cs(Id,in(A)))),
   in_r(dialogstate(Dialog,S)),
```

```
   out_r(transition(S,cs(Id,Act),C,Dialog))
)).
reaction(out_r(excommit(cs(Id,rd(A)))),(
   rd_r(A),  in_r(excommit(cs(Id,rd(A)))),
   in_r(dialogstate(Dialog,S)),
   out_r(transition(S,cs(Id,Act),C,Dialog))
)).
```

A.3 ReSpecT Implementation of CAA

In this appendix we provide the complete code for CAA functionality. The procedures to calculate argument sets (conflict free set, admissible set and preferred extension) are called by ReSpecT reaction. The first reaction is fired by agents putting an *argument* in the CAA. The CAA reacts collecting all arguments present in the tuple space and using that list to calculate the argument sets. The use of ReSpecT reaction in A&A perspective is well explained by Omicini in [23].

```
reaction(out_r(argument(Name,_,beliefs(facts(F),
  rules(R)),infer(_),conclusion([C]))),(
pre,
out_r(rdall_collect(argument(Name1,_,beliefs
  (facts(F1),rules(R1)),infer(_),conclusion([C1])),
    [])),
in_r(out_r(argument(Name,_,beliefs(facts(F),
   rules(R)),infer(_),conclusion([C])))
)).
```

```
reaction(out_r(rdall_collect(T,L)),(
current_tuple(rdall_collect(T1,_)),
in_r(T1), %leggo la tupla dal tuple space
in_r(rdall_collect(T,L)),
out_r(rdall_collect(T,[T1|L])))).

reaction(out_r(rdall_collect(T,_)),(
current_tuple(rdall_collect(T1,_)),
no_r(T1),
in_r(rdall_collect(T,L)),
out_r(start(L)),
out_r(rdall_restore(L)) )).

reaction(out_r(rdall_restore([])),(
in_r(rdall_restore([])),
)).

reaction(out_r(rdall_restore([H|T])),(
in_r(rdall_restore([H|T])),
out_r(H),
out_r(rdall_restore(T)) )).

%calculation of preferred extension
reaction(out_r(start(TotalARg)),(
```

```
 not(turn(TotalArg)),
 rd_r(mem(ConflictFreeSet)),
 assert(mem(ConflictFreeSet)),
 out_r(conflicfreeset(ConflictFreeSet)),
 preferred(TotalArg,ConflictFreeSet,PreferredSet),
 out_r(admissibleset(PreferredSet)),
 eliminatesubset(PreferredSet,PreferredSet,[],_,Preferredmaxset),
 out_r(preferredset(Preferredmaxset))
)).

contrary(non(P),P):-!.
contrary(P,non(P)).

%append1([1,2],[6,7],X). - X / [1,2,6,7]
append1(L1,L2,L3) :- L1=[], L3=L2.
append1(L1,L2,L3) :- L1=[H1|T1],
    append1(T1,L2,T3), L3=[H1|T3].


add2end(X,[H|T],[H|NewT]):-add2end(X,T,NewT).
add2end(X,[],[X]).

%give back a element of the list and
%the rest in order [a,b,c,d]-> a [b,c,d]
selection(X,[X|Rest],Rest).
```

```
selection(X,[Head|List],Rest) :-
    selection(X,List,Rest).

subset([A|X],Y) :- member(A,Y),subset(X,Y).
subset([],Y).  % The empty set is a subset of every set.

%notsubset(+List,+List)
notsubsetset(R,[H|T]):-
    not(subset(R,H)),
    notsubsetset(R,T).
notsubsetset(_,[]).

turn(ArgumentSet):-
    selection(Name,ArgumentSet,RestArgumentSet),
    argument(Name,_,beliefs(facts(F),rules(R)),_,conclusion(C)),
    newconflictfree(RestArgumentSet,Result,F,C,[Name]).

newconflictfree(Arguments,Result,Facts,Conclusions,ConflictFree):-
    selection(Name,Arguments,RestArguments),
    argument(Name,_,beliefs(facts(F),rules(R)),_,conclusion(C)),
    check(Facts,F,Conclusions,C),
    append1(Facts,F,NewFacts),
    append1(Conclusions,C,NewConclusions),
    add2end(Name,ConflictFree,NewConflictFree),
    newconflictfree(RestArguments,NewConflictFree,
```

```
            NewFacts,NewConclusions,NewConflictFree).

check(FL,F,CL,C):-
    not(control(FL,C)),
    not(control(F,CL)),
    not(control(CL,C)).

newconflictfree(_,[],_,_,_):-!,fail.
newconflictfree(_,R,_,_,_):-
    mem(P),
    notsubsetset(R,P),
    retract(mem(P)),
    assert(mem([R|P])),!,
    mem(P1),
    fail.

removelist([],L,L).
removelist([H|T],List1,Result):-
    delete(H,List1,R),
    removelist(T,R,Result).
eliminatesubset([],CF,L,_,L).
eliminatesubset([H|T],CF,Newset,Sol,R):-
    delete(H,CF,NewS),
    (notsubsetset(H,NewS)->Sol=[H|Newset];Sol=Newset),
    eliminatesubset(T,CF,Sol,Result,R).
```

```
preferred(_,[],[]).
preferred(TotalArguments,[ConflictFreeSet|Rest],Solution):-
    findundercat(TotalArguments,ConflictFreeSet,Attacker,Defeat),
    %to find set of attacker to conflict free
    findundercat(ConflictFreeSet,Attacker,AttackerFromCF,DefeatOut),
    %it find the defend argument that block the attack
    removelist(DefeatOut,Attacker,AttackerNotDefeat),
    findundercat(AttackerNotDefeat,Defeat,AF,DF),
    removelist(DF,ConflictFreeSet,Sol),
    Solution=[Sol|Result],
    preferred(TotalArguments,Rest,Result).

findundercat([],_,[],[]):-!.
findundercat([H|T],CF,A,D):-
    argument(H,_,beliefs(facts(F),rules(R)),infer(_),conclusion([C])),
    contrary(C,P),!,
    (argument(Element,_,
      beliefs(facts([P]),rules(_)),infer(_),conclusion(_))->
    (member(Element,CF)->
      (A=[H|R1],D=[Element|R2]);(A=R1,D=R2));(A=R1,D=R2)),
    findundercat(T,CF,R1,R2).

control([],_):-fail,!.
control([T|C],C2):-
```

```
contrary(T,CT),
(member(CT,C2)->true;
(control(C,C2))).
```

Bibliography

[1] aliCE Research Group. tuProlog home page. `http://tuprolog.alice.unibo.it/`.

[2] Leila Amgoud, Nicolas Maudet, and Simon Parsons. An argumentation-based semantics for agent communication languages. In Frank van Harmelen, editor, *ECAI*, pages 38–42. IOS Press, 2002.

[3] T. J. M. Bench-Capon, F. P. Coenen, and P. Orton. Argument Based Explanation of the British Nationality Act as a Logic Program. *Computers, Law and Artificial Intelligence*, 2(1):53, 1993.

[4] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63 – 101, 1997.

[5] Claudette Cayrol, Sylvie Doutre, and Jérôme Mengin. On decision problems related to the preferred semantics for argumentation frameworks. *Journal of Logic and Computation*, 13(3):377–403, Jun 2003.

[6] C. Chesñevar, J. Mcginnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 21:293 – 316, 2006.

[7] C. I. Chesñevar and G. R. Simari. Modelling Inference in Argumentation through Labelled Deduction: Formalization and Logical Properties. *Journal Logica Universalis*, 1(1):93 – 124, January 2007.

[8] Carlos I. Chesñevar, Guillermo R. Simari, Teresa Alsinet, and Lluís Godo. A logic programming framework for possibilistic argumentation with vague knowledge. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 76–84, Arlington, Virginia, United States, 2004. AUAI Press.

[9] Sylvie Doutre, Peter McBurney, and Micheal Wooldridge. Law-governed Linda as a semantics for agent dialogue protocols. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 1257–1258, Utrecht, The Netherlands, 25–29 July 2005. ACM Press.

[10] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.

[11] J. Forester. *The Deliberative Practitioner: Encouraging Participatory Planning Processes*. MIT Press, Cambridge, MA, USA, 1999.

[12] A. J. García and G. R. Simari. Defeasible Logic Programming: An Argumentative Approach. *CoRR*, cs.AI/0302029, 2003.

[13] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[14] T. F. Gordon and N. Karacapilidis. The Zeno argumentation framework. In *Proceedings of the Sixth International Conference on AI and Law*, pages 10–18, New York, NY, USA, 1997. ACM Press.

[15] G. Governatori, M. J. Maher, G. Antoniou, and D. Billington. Argumentation Semantics for Defeasible Logics. In *Pacific Rim International Conference on Artificial Intelligence*, pages 27 – 37, 2000.

[16] R. Haenni and N. Lehmann. *ABEL: An Interactive Tool for Probabilistic Argumentative Reasoning*, volume 2711, pages 588 – 593. Springer Berlin / Heidelberg, April 2004.

[17] C. L. Hamblin. *Fallacies*. Methuen, London, UK, 1970.

[18] David Kirsh. Distributed cognition, coordination and environment design. In *European Conference on Cognitive Science*, pages 1–11, 1999.

[19] N. Maudet and F. Evrard. A generic framework for dialogue game implementation. In *In Proceedings of the 2nd Workshop on Formal Semantics and Pragmatics of Dialogue*, pages 185 – 198, University of Twente, The Netherlands, 1998.

[20] Peter McBurney and Simon Parsons. Posit spaces: a performative theory of e-commerce. In M. Wooldridge J. S. Rosenschein, T. Sandholm and

M. Yokoo, editors, *Proceedings of AAMAS 2003*, pages 624–631, New York City, NY, USA, 2003. ACM Press.

[21] B. A. Nardi. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996.

[22] Enrico Oliva, Peter McBurney, and Andrea Omicini. Co-argumentation artifact for agent societies. In I. Rahwan, C. Reed, and S. Parsons, editors, *Proceedings of the Fourth International Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2007)*, pages 115–130, AAMAS 2007, Honolulu, Hawai'i, USA, 2007.

[23] Andrea Omicini. Formal ReSpecT in the A&A perspective. In Carlos Canal and Mirko Viroli, editors, *5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06)*, pages 93–115, CONCUR 2006, Bonn, Germany, 31 August 2006. University of Málaga, Spain. Proceedings.

[24] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, November 2001.

[25] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. An algebraic approach for modelling organisation, roles and contexts in MAS. *Applicable Algebra in Engineering, Communication and Computing*, 16(2-3):151–178, August 2005. Special Issue: Process Algebras and Multi-Agent Systems.

[26] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. *Agens Faber*: Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer*

Sciences, 150(3):21–36, 29 May 2006. 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.

[27] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, and Luca Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 286–293, New York, USA, 19–23 July 2004. ACM.

[28] Andrea Omicini and Franco Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, September 1999.

[29] Simon Parsons and Peter McBurney. Argumentation-based communication between agents. In M-P. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *LNCS*, pages 164–178. Springer, Berlin, September 2003.

[30] Simon Parsons, Carles Sierra, and Nick Jennings. Agents that Reason and Negotiate by Arguing. *Journal of Logic and Computation*, 8(3):261 – 292, 1998.

[31] Simon Parsons, Micheal Wooldridge, and Leila Amgoud. Properties and Complexity of Some Formal Inter-agent Dialogues. *Journal of Logic and Computation*, 13(3):347 – 376, 2003.

[32] D. Pitt, J.and Ramirez-Cano, L. Kamara, and B. Neville. Alternative Dispute Resolution in Virtual Organizations. In *Proceedings of The Eighth Annual International Workshop "Engineering Societies in the Agents World" (ESAW 07), Athens, Greece*, 2007.

[33] Henry Prakken. Coherence and flexibility in dialogue games for argumentation. *Journal of Logic and Computation*, 15(6):1009–1040, 2005.

[34] Henry Prakken. Formal systems for persuasion dialogue. *Knowl. Eng. Rev.*, 21(2):163–188, 2006.

[35] Henry Prakken and Giovanni Sartor. *Computational Logic: Logic Programming and Beyond. Essays In Honour of Robert A. Kowalski, Part II.*, chapter The Role of Logic in Computational Models of Legal Argument: A Critical Survey, pages 342–380. Lecture Notes in Computer Science 2048. Springer, Berlin, 2048 edition, 2002.

[36] Henry Prakken and G. Vreeswijk. Logical systems for defeasible argumentation. In D. M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, Volume 4*, pages 219–318. Kluwer, Dordrecht, 2002.

[37] I. Rahwan, F. Zablith, and C. Reed. Laying the foundations for a World Wide Argument Web. *Argumentation in Artificial Intelligence*, 171(10-15):897 – 921, July-October 2007.

[38] T. Schultz, G. Kaufmann-Koheler, D. Langer, V. Bonnet, and J. Harms. Online dispute resolution: State of the art, issues, and perspectives. Technical report, Faculty of Law and Centre Universitaire Informatique, University of Geneva, October 2001. Draft Report.

[39] Leon Sterling and Ehud Shapiro. *The art of Prolog: advanced programming techniques*. MIT Press, Cambridge, MA, USA, 1994.

[40] Mirko Viroli and Andrea Omicini. Coordination as a service. *Fundamenta Informaticae*, 73(4):507–534, 2006. Special Issue: Best papers of FOCLASA 2002.

[41] Mirko Viroli, Alessandro Ricci, and Andrea Omicini. Operating instructions for intelligent agent coordination. *Knowledge Engineering Review*, 21(1):49–69, March 2006.

[42] G. B. Walker and S. E. Daniels. Argument and alternative dispute resolution systems. *Argumentation*, 9(5):693 – 704, 1995.

[43] D. N. Walton. *Argumentation Schemes for Presumptive Reasoning*. Lawrence Erlbaum Associates., Mahwah, NJ, 1996.

[44] D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY Press, 1996.

[45] Douglas Walton and David M. Godden. Persuasion dialogue in online dispute resolution. *Artificial Intelligence and Law*, 13:273–295, 2005.