Co-Argumentation Artifact for Agent Societies

Enrico Oliva[1], Peter McBurney[2], and Andrea Omicini[1]

[1] Alma Mater Studiorum–Università di Bologna, Cesena, Italy
[2] University of Liverpool, Liverpool L69 3BX UK

Abstract. In a social context, people have a partial knowledge about the world and use arguments in order to solve problems, to reduce conflicts, or to exchange information. Argumentation is a dialogic process, and could occur through direct interaction, or through supports of some sorts—like blackboards, or electronic fora. The same holds for intelligent agents in MAS (multi-agent system), where however it is not clear what could work as an actual support for argumentation between agents, external to agents themselves. To this end, this work exploits the A&A (agents and artifacts) meta-model for MAS, exploring the use of artifacts for agent argumentation within a MAS. Along this line, the first aim of this work is to design an argumentation component based on Dung's preferred semantics, combining it with artifact abstraction in order to realise a social support for argumentation in MAS. Using argumentation within the A&A meta-model, we introduce here the notion of Co-Argumentation Artifact (CAA) as an artifact specialised in managing arguments and providing a coordination service for argumentation process in a MAS. In order to give concreteness to our proposal, we also discuss a first CAA deployment based on logic programming and tuple centres exploiting the TuCSoN infrastructure.

1 Introduction

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, or to inform each other of some pertinent facts. Argumentation is a useful feature of human intelligence that provides us with the basic mechanism to work with incomplete information. People usually own partial knowledge about the world (they are not omniscient) and sometimes 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 succesfully deal with partial knowledge.

A novel approach to the design of agent-based artificial societies is based on the notion of coordination artifact [1], which takes inspiration from Activity Theory [2] where any human activity within a society is enabled / constrained / mediated by artifacts. Coordination artifacts are social constructs shared by agents of a MAS, and are necessary to mediate interaction among agents, and between agents and their environment. A traffic light, for instance, is a sort of coordination artifact: people watching the signal know what they have to do with no need of direct communication with one another to avoid accidents at an intersection.

Argumentation is an important feature of human intelligence: the ability to understand and manipulate arguments is fundamental to understand a new problem, to reason about actions, and to perform scientific research. 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.

In this paper we elaborate on the idea of social support for argumentation in a MAS, by coupling the A&A meta-model for MAS with argumentation theory. In particular, we introduce the notion of Co-Argumentation Artifact (CAA), as a social support based on argumentation theory able to manage conflicting information exchange during the social argumentation processes in a MAS. Agents use a CAA to be guided during the discussion or to find some new information in agreement with their internal goals. A CAA would be useful, for example, to identify the largest subset of arguments agreeable to all participants in the society.

In this paper, we first introduce our reference notion of Argumentation System (Section 2), then (Section 3) we discuss the properties of an Argumentation Component. In Section 4 the new Co-Argumentation Artifacts abstraction is defined and explained, and a simple example built on top of the TuCSoN coordination infrastructure is discussed.

2 Argumentation System

In this section we introduce the system of argumentation that is the reference for our approach. An argument, in classic logic, is a sequence 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 [3], and we also define the structure inside the arguments. In [4] 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. We assume that Σ contains formulas of a predicate language L distinct in F facts and R rules. The symbol $\vdash$ denotes classical inference process for deduction, induction and abduction, and $\equiv$ denote logical equivalent.

Definition 1. *An argument is a triple* $A = \langle B, I, C \rangle$ *where* $B = b_1, \ldots, b_n, r_1, \ldots, r_n$ *with* $b_i \in F$ *and* $r_i \in R$, $I = \{\vdash_d \textit{Deduction}, \vdash_i \textit{Induction}, \vdash_a \textit{Abduction}\}$ *and* $C = c_1, \ldots, c_n, r_1, \ldots, r_n$ *with* $c_i \in F$ *and* $r_n \in R$ *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 a 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 (MMP) and Modus Tollens (MT). The Multi-Modus Ponens (MMP) in formula is:

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

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

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

For example, all human 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, that we use for induction is the rule: θ-subsumption, that in formula is:

$$\frac{B}{R} \;\; where \;\; R\theta \subseteq B$$

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

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

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 a probability value that could be associated to the result of an induction process. Finally the abductive reasoning is expressed with the inference rule:

$$\frac{B \quad A \rightarrow B}{A}$$

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

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

The definition of contrast is not trivial because there are different type of attack well defined in [4]. 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 $mortal(Socrates)$. Also we 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 [3]. 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 exist no $A_i, A_j \in S$ than A_i attack A_j.*

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

3 Argumentation Component

The argumentation component is a system that should be useful in principle in order to control a set of conflicting arguments. Argumentation component is formed by: (1) the concrete representation of arguments and (2) the deployment of algorithms that work over the arguments set. The main functionalities of this component 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 deployment of these algorithms within each of the agents of an agent society, and within artifacts embodying the social argumentation processes. This would be useful, for example, to identify the largest subset 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 [5].

3.1 Computational Model

The computational model that we propose to manage the argument set 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 evaluates an argument, the program looks for the correct interpreter and checks if the conclusion is a consequence of the premises.

3.2 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 for 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 is: `contrary(non(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 1. Check of argument in Modus Ponens

```
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 2. 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
```

3.3 Meta-Interpreter for Argument Management

Managing the argument set requires in particular an ability to calculate (1) the relations of undercut and attack between argument, (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

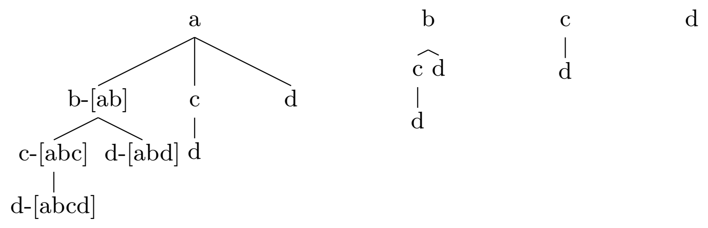

Fig. 1. Search trees generated for 4 arguments

`undercut(from,to)` we obtain a *defeat graph* where the relations are the arcs and the arguments are the nodes, according to Dung [3].

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 which is very unlikely to find an efficient algorithm.

Our idea is to build an algorithm that works incrementally, to try to avoid the complexity of an increase information number—and also, because we foresee a dynamic and distributed scenario where agents share their own arguments at different times.

To solve the conflict free problem, we adopt a constraint-based approach. Our algorithm is based over 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, the 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 1. There, the possible partial solutions are denoted in square brackets.

The algorithm also can be used in a dynamic context with subsequent inputs. 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 [6] 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 argument 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 are the solution called Admissible set. 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 we find the preferred extensions by looking for the maximal admissible set, in agreement with the previous definition 7.

4 Co-Argumentation Artifact

The main contribution of the paper is the combination of multi-agent argumentation with the A&A meta-model, exploiting agents and artifacts as the two fundamental abstractions for MAS. In a Multi-Agent System (MAS), argumentation has a central role that allows agents to argue, to justify positions, and to try to persuade another agent. 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 [7], that are: persuasion, inquiry, negotiation, information seeking, deliberation and eristic. All these dialogues can be captured in a argumentation framework [8], 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. For instance, the argumentation-based dialogues listed above could be transformed in social discussions among agents.

In a social context any action and activity are mediated through *artifacts*, in accord with Activity Theory (AT)[2]. Mediation is useful to achieve cooperation between the entities and the coordination of the global system. In particular in a MAS, 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, is not bounded inside each individual agent, but is instead distributed across agents and artifacts in the environment.

An artifact is a computational entity used by agents, possibly featuring useful properties such as controllability, malleability, linkability and situation [10]. The artifact abstraction is introduced in the A&A meta-model for MAS, where agents and artifacts are two basic building block to design the system, more generally to engineering software systems: 1) agents represent task-oriented or goal-oriented components that act pro-actively following their task or goal, 2) artifacts represent resources or tools that are used by agents during their activities.

In this work, 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 permits 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 to determine 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 [1], specialised to provide a coordination service in MAS. 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 by either aggregation or induction process.

In this work, we developed a hybrid distributed system combining the argumentation component with a multi-agent co-ordination artifact, which is what we call CAA. Our current implementation of the CAA follows a preferred semantic, providing service to calculate preferred extensions and admissible sets.

The technological support to build co-ordination artifacts is provided here by TuCSoN, a coordination infrastructure for MASs introduced in [11]. TuCSoN provides programmable tuple spaces called tuple centres where agents write, read and consume logic tuples via simple communication operations (out,rd,in,inp,rdp). Tuple centres can play the role of agent coordinators, where coordination rules are expressed in terms of tuples. In particular, coordination in TuCSoN is expressed through the ReSpecT specification language [12]: there, coordination laws are encapsulated in the coordination media, with obvious benefits for the engineering of open and dynamic system like MAS. 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.

So, in order to realize a CAA, an obvious choice is then to exploit a TuCSoN logic tuple centre as a co-ordination artifact. 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. So, our first experimental implementation of a CAA is built as a TuCSoN tuple centre programmed with an argumentation component algorithm (Section 3), and with arguments represented by logic tuples (Section 2). 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.

4.1 Example: Argument Acceptance

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 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 [13] 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 El Farol Bar. In order to make a personal evaluation the agents may have some benefit from social information that could retrieve by 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 over 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 divides in three sequent steps. First step, the agents share own arguments writing the arguments in the CAA. Second step, the CAA reacts and calculates the conflict free and preferred extension over the shared arguments. Third and final step, the agents evaluate credulous or sceptical acceptability based on common sets calculated in the CAA. Then, each agent can consult the CAA to undestand the "social acceptability" of its own arguments, but also the other agent's arguments, and possibly deliberate its course of actions based on a shared view of arguments. Also, the CAA keeps track of the overall argumentation process, and could be exploited by an external observe 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 the table 1 the arguments possessed and shared by the three agents are shown. Some arguments are in favor to go out if the conclusions are `play(1)`, or vice versa `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. An external observer can look inside the CAA through the Inspector utility provide by `TuCSoN`, and consult the argument sets. In the following we

Table 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)])).`
...

show the sets computed after the last argument insertion. The sets are composed with the arguments name.

```
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]])
```

We can 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 are present the own arguments. For instance, `agent1` may want to consider the social acceptability of argument `musicB` that in its own knowledge is accepted, because the argument belong to its 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 sets. These sets are calculated from more information than it is available to each individual agent, and in some context they could be considered as more reliable. In any case, the agents can decide in autonomy which use to make of these information—either to use or to ignore them.

In general, the use of a component external to agents to support argumentation within a social agent-based context, like the CAA, easily provides MAS designers with a tool for encapsulating and consistently handling the evolution of social knowledge, and agents with an instrument to enhance their ability to deal with their partial and incomplete knowledge.

References

1. Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L.: Coordination artifacts: Environment-based coordination for intelligent agents. In Jennings, N.R., Sierra, C., Sonenberg, L., Tambe, M., eds.: 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004). Volume 1., New York, USA, ACM (19–23 July 2004) 286–293
2. Nardi, B.A.: Context and Consciousness: Activity Theory and Human-Computer Interaction. MIT Press (1996)
3. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77**(2) (1995) 321–358
4. Prakken, H., Vreeswijk, G.: Logical systems for defeasible argumentation. In Gabbay, D.M., Guenther, F., eds.: Handbook of Philosophical Logic, Volume 4. Second edn. Kluwer Academic, Dordrecht, The Netherlands (2002) 219–318
5. Sterling, L., Shapiro, E.: The art of Prolog: advanced programming techniques. MIT Press, Cambridge, MA, USA (1994)
6. aliCE Research Group: tuProlog home page. `http://tuprolog.alice.unibo.it/`
7. Walton, D.N., Krabbe, E.C.W.: Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning. SUNY Press (1996)
8. Parsons, S., McBurney, P.: Argumentation-based communication between agents. In Huget, M.P., ed.: Communication in Multiagent Systems. Volume 2650 of Lecture Notes in Computer Science., Springer Berlin (September 2003) 164–178
9. Doutre, S., McBurney, P., Wooldridge, M.: Law-governed linda as a semantics for agent dialogue protocols. In Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M., eds.: AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (July 2005) 1257–1258
10. Omicini, A., Ricci, A., Viroli, M.: *Agens Faber*: Toward a theory of artefacts for MAS. Electronic Notes in Theoretical Computer Sciences **150**(3) (29 May 2006) 21–36 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.
11. Omicini, A., Zambonelli, F.: Coordination for Internet application development. Autonomous Agents and Multi-Agent Systems **2**(3) (September 1999) 251–269

12. Omicini, A., Denti, E.: Formal ReSpecT. Electronic Notes in Theoretical Computer Science **48** (jun 2001) 179–196
13. Cayrol, C., Doutre, S., Mengin, J.: On decision problems related to the preferred semantics for argumentation frameworks. Journal of Logic and Computation **13**(3) (Jun 2003) 377–403