Argumentation and Artifact for Dialogue Support

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

[1] ALMA MATER STUDIORUM–Università di Bologna, Cesena, Italy
[2] University of Liverpool, Liverpool L69 3BX UK

Abstract. Intelligent and autonomous software agents may engage in dialogue and argument with one another, and much recent research has considered protocols, architectures and frameworks for this. Just as with human dialogues, such agent dialogues may be facilitated by the presence of a mediator, able to summarise different positions, identify common assumptions and inconsistencies, and make appropriate interventions in the dialogue. Drawing on the theory of co-ordination artifacts in multi-agent systems, we propose a formal framework to explicitly represent the functions of a mediator artifact. We then describe an implementation of this framework using the TuCSoN coordination infrastructure for MAS, where the mediator artifact is realised by a tuple centre—a programmable tuple space.

1 Introduction

Proponents of public policy conversations and decision-making processes usually emphasise the need for a human moderator or mediator to be involved in the interaction, e.g., Forester [3]. The mediator may act to ensure fairness and equality of access by all participants, may assist participants to clarify their positions and to argue more effectively, and may even seek to reconcile opposing views. Similarly, the designers of computer-aided argumentation systems have also provided support for human mediators; for example, the developers of *Zeno* define their system as "a mediation system" [5, 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. Most of these mediator functions are better supported through some algorithmic procedure, rather than by some articulated process of rational deliberation.

In earlier work [8], we presented a conceptual framework for a central co-ordinating entity in an argumentation dialogue, called a *Co-Argumentation Artifact (CAA)*, to provide co-ordination services to the participating agents, allowing them to share, store and exchange arguments with one another. A CAA is an artifact, a computational entity used by the agents, specialised in argumentation reasoning. Vesting the CAA with its own argumentation capabilities meant that this entity, like the participants, could elaborate over the arguments stored. For example, the CAA could embody algorithms determining whether a particular argument is acceptable (under a specified semantics of argumentation) with respect to the global knowledge of all the participants. We reprise the CAA framework in Section 3.1.

It is easy to imagine that the CAA could undertake more sophisticated interventions in the dialogue, resembling complex, automated tasks of a human mediator. To this end, in this paper we extend our earlier concept of a central co-ordinating artifact to be a dialogue artifact (DA), acting as a mediator between the participating agents, enacting the agents participation in the dialogue. We do this, first, by articulating, in Section 2, the possible functions of the mediator artifact; for reasons of space, we do not consider all such functions here. We then present a formalisation of some of such mediator artifact functions in Section 4, drawing on recent work in the theory of communications artifacts in multi-agent systems [11, 18]. We follow this with a description of a prototype implementation we have undertaken in the TuCSoN coordination infrastructure, in Section 5. Finally, Section 6 concludes the paper.

2 Functionalities of Multi-agent Argumentation Support

In an agent and human society, argumentative reasoning system and dialogue system have a central role enabling exchange of knowledge, common sense reasoning, dispute resolution and argumentative communication between agents.

Our model of a multi-agent argumentation system follows the A&A meta-model [12, 17], which defines MAS as composed of two kinds of entities: agents, in charge of autonomous and proactive behaviour, and *artifacts*, providing function-oriented services to agents in a passive way. An agent is an autonomus computational entity situated in an enviroment, which is composed by artifacts; an artifact is hence a social construct shared by the agents that enables, constraints or mediates their activities.

In this work we want to propose a model that merges concepts from argumentation and artifact theories, and offers a coherent framework to cater for these reasoning and coordination abilities that agents need to exhibit. More precisely, we brings the theory of argumentation and of dialectical agents interaction to a level of operationalisation.

Dialogue participants, of course, need to be able to generate, evaluate, contest and defend arguments as they interact with one another through the dialogue. But the mediator 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 mediator 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 define two types of artifacts: Dialogue Artifacts (DA), to support dialectical agent interaction, and Co-Argumentation Artifacts (CAA), to support agent argumentative reasoning. The DA realises the mediator artifact enabling argumentative communication among a multiplicity of entities. DA exploits directly CAA functionalities in order to drive the dialectical interaction between agents through argumentative reasoning over the argument set stored inside the centralised CAA.

The CAA realises coordination services based on argumentative reasoning between multi entities. For example, the CAA could determine whether a particular argument is acceptable (under a specified semantics of argumentation) with respect to the global knowledge of all the participants. Briefly the CAA is composed of arguments, represented in a suitable computational form, and a collection of algorithms deployed over this argument set. The CAA is well introduced in a recent paper by Oliva et al [8].

The combination of both artifacts DA and CAA provides the support of basic and advanced functionality for automatic mediation services in a MAS based on argumentation.

2.1 Dialogue Artifact

The dialogue artifact requires some basic functionality to support the exchange of arguments in a dialogue between the participants. This basic functionality includes:

1. Storage of the dialogue protocol (e.g. in a library of such protocols)
2. Storage of the specifications of the dialogue protocol
3. Storage of the complete history of a dialogue as it proceeds
4. The ability to refuse to allow agent utterances which do not 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 dialogue
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 dialogue artifact could act as sophisticated mediator of the discussion, by providing in an automated way the following services:

- Seeking to resolve any disputes over the rules of the protocol
- Providing rewards or penalties to agents for breaking the protocol rules
- Having the power to admit or to expel agents to/from the dialogue
- Suggesting a new protocol, when needed.
- Supporting multiple simultaneous bilateral interactions.
- 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.
- Identifying conflicts and inconsistencies between commitments made by agents in a dialogue, for example, if an agent commits to sell a car it is also trying to purchase.

– Identifying agent utterances which are not relevant to the current state of the dialogue, and refusing to permit these to be made.
– Providing automated alerts to inform agents that dialogues on particular topics are about to start, or to end, or that particular commitments have just been made.
– Combining different dialogues on the same topic.

More advanced functions could also include:

– 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.
– Storing the outcomes of past dialogues, for example, the commitments remaining at the end of the dialogue.
– Tracking agent commitments across multiple dialogues.
– Using previous dialogues to create an independent assessment of the reputation of participating agents.
– Storage of the entire history of past dialogues. These may be required for regulatory or legal reasons, e.g., in stock market transactions.

In this paper, we present a Dialogue Artifact supporting dialectical argumentation in a MAS scenario, which provides some of the basic functionalities listed above. The DA is based over two formal systems: an argumentation system and a dialogue system, explained in the next sections.

3 Argumentation and Dialogue: Formal Definitions

In our earlier work [8], we introduced an argumentation system and an artifact abstraction to support the co-ordination functions necessary to support agent argumentation. We now extend that framework to handle argumentation dialogues, drawing on the theory of organisations and roles in multi-agent systems of Omicini *et al.* [11] and the formal language used to define an agent interaction protocol, in the work of Viroli *et al.* [18]. In our approach we describe a dialogue in terms of a labelled process algebra, where labels denote roles, as in [11], and the process algebra specifies the interaction protocol, as in [18].

We assume that the interaction is between a finite number N of intelligent software agents, and that each agent has a range of possible utterances (or actions) at each step in the dialogue (i.e., this is a multi-move protocol). Formally, a multi-agent dialogue system for argumentation is composed of two parts: an argumentation system, and a dialogue system. The definition of the argumentation system is discussed based on the work in [8], and builds on various earlier argumentation frameworks.

3.1 Argumentation System

Prakken and Vreeswijk [16] 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

Deductive Inference		Inductive Inference	
MP	$\frac{A \quad A \to B}{B}$	θ-su	$\frac{B}{R}$ *where* $R\theta \subseteq B$
MT	$\frac{\neg A \quad B \to A}{\neg B}$	Abductive Inference	
MMP	$\frac{B_1,\ldots B_n \quad (B_1,\ldots B_n) \to C}{C}$	Ab	$\frac{B \quad A \to B}{A}$

Table 1. Deductive Inference: (MP) Modus Ponens, (MMP) Multi-Modus Ponens and (MT) Modus Tollens; Inductive and Abductive Inference: (θ-su) θ-subsumption, (Ab) Abductive

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 [2], and we also define the structure inside the arguments.

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 (argument). *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,* deduction*,* 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*

The types of inference we consider for deduction, induction and abduction are shown in Table 1. Modus Ponens (MP) is a particular case of Multi-Modus Ponens (MMP) with only one premise. The inference process θ-subsumption derives a general rule R from specific beliefs B, but is not a legal inference in a strict sense.

For defeat of arguments, the definition is not straightforward because there are different type of attack well defined in [16]. Following those definitions, two possible types of attack are 'conclusions against conclusions' – called *rebuttals* – and 'conclusions against beliefs'—called *undercuts*.

Definition 2 (undercut). *Let* $A_1 = \langle B_1, I_1, C_1 \rangle$ *and* $A_2 = \langle B_2, I_2, C_2 \rangle$ *be 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 (rebuttal). *Let* $A_1 = \langle B_1, I_1, C_1 \rangle$ *and* $A_2 = \langle B_2, I_2, C_2 \rangle$ *be 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* $c_i \in C_2$

The definitions of acceptability and admissibility used in our framework are those of Dung in [2]. The following definitions are the basic ones in our argumentation system and follow from Dung's framework.

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

Definition 5 (collective defense). *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 (admissible set). *An argument set S is a* admissible *set iff S is conflict free and S defends collectively all its elements.*

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

An argument is acceptable in the context of preferred semantics if an argument is in some/all preferred extensions (credulous/sceptical acceptance).

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

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

For further details, including an implementation and examples of this argumentation framework, we refer the interested reader to our earlier paper [8].

3.2 Dialogue System

In this section we present a novel formalisation of a multi-agent dialogue system. Our intention is to capture the rules that govern legal utterances, as well as the effects of utterances on the commitment stores of the dialogue. We use a process algebra approach in the style of [18] to represent the possible paths that a dialogue may take, and to represent explicitly the operations to and from the commitment store. We proceed by considering each element of a dialogue system in turn: (1) the communication language; (2) the interaction protocol; and (3) the protocol semantics.

Because a dialogue 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 3.1. Agents may exchange arguments, along with facts, with one another in the form of instantiated parameters in their utterances.

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 [14]. We let F denote a set of terms representing *facts*, and $\mathscr{A}$ the set 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 [19]: persuasion, inquiry, negotiation, information seeking, deliberation and eristic.

Definition 10 (communication language). *Our* communication language *is a set of locutions* L_c*. A locution* $l \in L_c$ *is a expression 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 dialogue using 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 Example 1 an agent wants to communicate the classical example of argument like *All men are mortal, Socrates is a man, Socrates is mortal*, so it uses a `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 Dialogue* could be: `OpenDialogue`, `Ask`, `Tell`, `DontTell`, `Provide`, `Argue`, and so on. Further details about this form of dialogue and its complete locutions are presented in [1] (see also Example 2).

Dialogue Protocol In our framework the dialogue protocol is a complete description of all possible dialogue paths, from the perspective of an external entity observing the dialogue between the agents. The protocol indicates the possible paths of a dialogue, 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. Hence, technically, we find it useful to model a dialogue 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 [15] 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 provides a definition of a speech act, as follows:

Definition 11 (action). *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 [7]: such tuples are manipulated by the commands of the Linda language [4]—`in`, `rd` and `out`.

Definition 12 (term action). *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 consumes, reads and puts a tuple `t` in the commitment store `C`. These actions are useful to manage the private or public commitment store in relation to the dialogue 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 dialogue paths. For instance, if at a given time a sub-dialogue is guarded by operation `rd(c,commit(a))`, then it is allowed to proceed only if `commit(a)` occurs in the commitment store.

Definition 13 (protocol). *A* protocol *P is a composition of action from sets A and K, defined by syntax $P ::= 0 \mid A.P \mid K.P \mid P+P \mid (P \parallel P) \mid !P$ where the symbols $.,+,\parallel,!$ denote respectively sequence (action prefix), choice, parallel composition, and infinite replication operators, and the symbol 0 denotes the empty protocol.*

For example, an abstract dialogue 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 invokes a_1 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 Dialogue (f is seen as a variable over the content of communication):

Example 2 (Information Seeking Dialogue). This protocol involves two agents: an agent `s` *controlling information, and an agent* `c` *trying to persuade* `s` *to give him the permission to access. Operation* `rd(permission(c,f))` *is the instruction by which the protol instance checks whether* `c` *can be given the permission: if this is the case, permission is provided and the dialogue ends; otherwise,* `c` *should try to persuade* `s` *by arguing an* `ADD`*, which can then by either accepted or refused by* `s`*.*

```
c:Opendialogue.
s:Opendialogue.
c:Ask(f).(
     rd(permission(c,f)).
     s:Tell(f).
     s:Provide(f).
     s:Argue(permission(c,f),YES,A).
     s:Enddialog.
     c:Enddialog
    +
     rd(not(permission(c,f))).
     s:DontTell(f).
     s:Argue(permission(c,f),NO,B).
     c:Argue(permission(c,f),ADD,A).(
        s:Argue(permission(c,f),NO,A).
        s:Enddialog.
        c:Enddialog
       +
        s:Accept(A,permission(c,f)).
        out(accept(permission(c,f))).
        s:Provide(f).
        s:Enddialog.
        c:Enddialog
)   )
```

3.3 Operational Semantics

Following Hamblin [6], we assume that each agent is associated to 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 either 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 dialogue system. This semantics describes the evolution over time of the dialogue 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 dialogue as a whole, and it is expressed in our framework as a multiset of terms.

Definition 14 (commitment store). *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, and 0 is the empty set.*

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

As usual, we write $s \xrightarrow{i} s'$ in place of $\langle s, i, s'\rangle \in \rightarrow$, meaning the dialogue 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). We introduce a congruence relation $\equiv$, which syntactically equates similar states:

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

We use also notation $t\{x/y\}$, to mean term 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. Finally, we define operational rules that describe the behavior of the dialogue system as follows:

$$(C)out(x).P \xrightarrow{\tau} (C|x)P \qquad (K-OUT)$$
$$(C|x)rd(y).P \xrightarrow{\tau} (C|x)P\{x/y\} \qquad (K-RD)$$
$$(C|x)in(y).P \xrightarrow{\tau} (C)P\{x/y\} \qquad (K-IN)$$
$$(C)(P+Q) \xrightarrow{i} (C')P' \quad \text{if} \quad (C)P \xrightarrow{i} (C')P' \qquad (OP-SUM)$$
$$(C)(P|Q) \xrightarrow{i} (C')(P'|Q) \quad \text{if} \quad (C)P \xrightarrow{i} (C')P' \qquad (OP-PAR)$$
$$(C)a.P \xrightarrow{a'} (C)P\{a'/a\} \qquad (ACT)$$

Rule (K-OUT) provides the semantic of `out` operation, expressing that x term is added to the commitment store C, and process continuation can carry on. Rules (K-RD) and (K-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.

Rules (OP-SUM) and (OP-PAR) provide the semantics for choice and parallel operators in the standard way. Finally rule (ACT) expresses that locution a' is executed that is an instance of the allowed one a, and accordingly process continuation P can carry on.

4 The Dialogue Artifact

As mentioned above, the A&A meta-model for MAS as discussed in [8] 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 Dialogue Artifact (DA), able to support and mediate the communication between agents engaged in a dialogue under the system defined in Section 3 above.

Definition 16 (Dialogue Artifact). *A* Dialogue Artifact *is a triple* $DA = \langle DP, CS, IC \rangle$*, where*

- *DP is a collection of specifications of dialogue protocols*
- *CS is a collection of commitment stores*
- *IC is a collection of specifications of interaction control (IC)*

The *DP*, *CS* and *IC* components are in turn defined in the following subsections.

Dialogue Protocols The class *DP* is a collection of formal specifications of dialogue protocols, with each protocol specified using a labelled 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 dialogue using a protocol in the collection *DP*, they make utterances according to the permitted sequences defined by the protocol specification. Accordingly, the Dialogue Artifact is able to verify that utterances proposed by agents in a dialogue 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 in the dialogue.

Commitment Stores For any particular collection of agents and any particular dialogue 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 dialogue as a whole. The Dialogue Artifact can support the dialogue 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 dialogues) or arguments based on privileged information (in the case of dialogues over beliefs). Sharing such information with the DA may allow the DA to elaborate over such stores while not revealing private information of individual agents.

We can classify the various types of stores according to the access permissions (write, read, and delete permissions) holding on each store, as shown in Table 2. The

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	R/W/D

Table 2. Commitment Stores - Read (R), write (W) and delete (D) Permissions

cells of the 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 Dialogue Artifact may also store other relevant information, such as the sequence of locutions exchanged in the current dialogue, which would be stored in the Central Commitment Store. These stores do not have an algebraic structure, but rather a declarative representation of the contents with a proper classification.

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

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

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 *IC* component realises the above three operators in order to identify which potential utterances are legal for any agent at any point in the dialogue. 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 dialogues are created by the existing commitments. For instance, the *IC* could permit only one utterance in a choice point basing the decision on state of commitment store. Also, it could work with an argument set over some advanced structures such as conflict free sets and preferred extensions presented in Section 3.1 to determine for instance the acceptability of an argument.

DA Functionalities It is straightforward to see that all six basic functionalities of the central Dialogue Artifact listed in Section 2.1 can be performed by a Dialogue Artifact defined as a triple $DA = \langle DP, CS, LI \rangle$ as above. 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* provides storage for the history of a dialogue (item 3); similarly, the private Commitment Store components of the collection *CS* provide storage for confidential information communicated from agents to the DA

(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).

5 TuCSoN Implementation

The technological support to build a *DA* is provided by the TuCSoN coordination infrastructure for MAS introduced in [13]. 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`). In particular, `inp`, `rdp` respectively consume and read matching tuples in the same way `in`, `rd`; unlike `in`, `rd` they fail if the tuple is not present when the request is served. As programmable tuple spaces [10], tuple centers can play the role of agent mediator, where coordination rules are expressed in terms of logic specification tuplesof the ReSpecT language—an event driven language over the multi-set of tuples [9]. Since tuple centre can be used as a general-purpose support for MAS artifacts, we exploited TuCSoN logic tuple centres in order to implement *DA*.

In this framework, agents utter a locution by means of an `out(move(Dialogue, AgentID,Locution))` in the tuple centre. 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. The *CS* class is composed of `commit` tuples that are put in the tuple space as facts and arguments expressed in logic tuple notation.

The dialogue is written in terms of tuples `dialogue(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`. Figure 1 shows a dialogue protocol composed by some basic information on dialogue state and few steps of the *Information Seeking Dialogue* protocol. The tuples that form the *DP* component are: `participant` (potential number of participants), `dialogue` (dialogue protocol), `dialoguestate` (actual protocol dialogue state), and `currentpar` (actual number of participants). In addition, an open dialogue session also uses tuple `session(AgentID,infoseek, open)` for each dialogue participant.

The key idea of the *IC* implementation is shown in figure 3, where the reactions implementing the control of dialogue interaction are presented. In particular, the code

```
dialoguesession(infoseek,close)
participant(infoseek,2)
dialogue(infoseek,[act(C,openDialogue(C,T)),
       act(T,openDialogue(C,T)),act(C,ask(Arg)+
       (act(T,tell(arg1),cs(T,out(commit(arg1))))))])
currentpar(infoseek,0)
```

Fig. 1. Example of Dialogue State (*DP* component)

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

Fig. 2. Implementation of $next^l$ operator in ReSpecT

implements the dialogue state transition after an agent action, the search of next admissible move after an agent request, and also makes it possible the automatic interaction with the commitment store executing `cs` actions. Such mechanisms make it possible for a dialogue to be driven automatically by the state of the commitment store. FAs an example, Figure 2 shows the ReSpecT implementation of the $next^l$ operator.

6 Conclusions

In this paper we propose a conceptual architecture for a multi-agent dialogue system in which participants are assisted by a mediator, called a *Dialogue Artifact*. To the best of our knowledge, there is no other research which combines at an operative level dialogue and argumentation reasoning through the use of a mediation artifact. The functions of such a mediator are the basic functionalities we have identified as part of a longer list of potential mediation or moderation functions in agent argumentation dialogues. Our Dialogue Artifact is an extension of our previous concept of a Co-Argumentation Artifact (CAA), and builds on that earlier work. We also draw on the recent theory of communication artifacts in MAS to formalise the properties of the Dialogue Artifact. Our paper also reported on a prototype implementation of these ideas we have undertaken in the TuCSoN coordination framework. In future work, we plan to formalise more of the potential mediator functions as listed in Section 2. While some of such functions will be straighforward to formalise – e.g., identifying conflicts between commitments, providing automated alerts to agents concerning upcoming dialogues – others, such as run-time assignment of rights and responsibilities to dialogue participants, are likely to result more challenging.

Also we aim at extending the underlying argumentation system by introducing *labels*. In fact, labelled arguments should make it possible to capture different sorts of certainty resulting from the different types of inference applied. Moreover, we plan to exploit labels to fix preferred ordering in an arguments set and to define stricter attack relation.

Acknowledgments

We are grateful for partial financial support from the EC's *Information Society Technologies* programme through project ASPIC (IST-FP6-002307).

References

1. S. Doutre, P. McBurney, and M. 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.
2. 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.
3. J. Forester. *The Deliberative Practitioner: Encouraging Participatory Planning Processes*. MIT Press, Cambridge, MA, USA, 1999.
4. D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
5. 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.
6. C. L. Hamblin. *Fallacies*. Methuen, London, UK, 1970.
7. P. McBurney and S. 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.
8. E. Oliva, P. McBurney, and A. 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.
9. A. Omicini. Formal ReSpecT in the A&A perspective. In Carlos Canal and M. 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.
10. A. Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, November 2001.
11. A. Omicini, A. Ricci, and M. 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.
12. A. Omicini, A. Ricci, and M. 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.
13. A. Omicini and Franco Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, September 1999.
14. S. Parsons and P. 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.
15. H. Prakken. Coherence and flexibility in dialogue games for argumentation. *Journal of Logic and Computation*, 15(6):1009–1040, 2005.
16. H. 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.
17. A. Ricci, M. Viroli, and A. Omicini. Programming MAS with artifacts. In Rafael P. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3862 of *LNAI*, pages 206–221. Springer, March 2006.

18. M. Viroli, A. Ricci, and A. Omicini. Operating instructions for intelligent agent coordination. *Knowledge Engineering Review*, 21(1):49–69, March 2006.
19. D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY Press, 1996.

```
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(Dialogue,Id,Act)),(
   in_r(dialoguestate(Dialogue,S)),
   out_r(transition(S,act(Id,Act),C,Dialogue))
)).
reaction(out_r(transition(S,A,S1,Dialogue)),(
   transition(S,A,S2), %make the state transition
   in_r(transition(S,A,S1,Dialogue)),
   out_r(dialoguestate(Dialogue,S2)),
   out_r(findall(S2,Dialogue))
)).
reaction(out_r(findall(S,Dialogue)),(
   in_r(findall(S,Dialogue)),
   findall(cs(Id,Commit),transition(S,cs(Id,Commit),Q),L), %collect all next commits
   out_r(nextcsmoves(Dialogue,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(dialoguestate(Dialogue,S)),
   out_r(transition(S,cs(Id,Act),C,Dialogue))
)).
reaction(out_r(excommit(cs(Id,in(A)))),(
   in_r(A),  out_r(excommit(cs(Id,in(A)))),
   in_r(dialoguestate(Dialogue,S)),
   out_r(transition(S,cs(Id,Act),C,Dialogue))
)).
reaction(out_r(excommit(cs(Id,rd(A)))),(
   rd_r(A),  in_r(excommit(cs(Id,rd(A)))),
   in_r(dialoguestate(Dialogue,S)),
   out_r(transition(S,cs(Id,Act),C,Dialogue))
)).
```

Fig. 3. Control of Interaction: Checking agent legal locution, Making dialogue protocol transition and executing automatically *cs* actions are the basic function here implemented in ReSpecT