Vivid Agents Arguing about Distributed Extended Logic Programs*

Michael Schroeder[1], Iara Móra[2], and José Julio Alferes[2,3]

[1] Institut für Rechnergestützte Wissensverarbeitung, University of Hannover, Lange Laube 3, 30159 Hannover, Germany, schroede@kbs.uni-hannover.de
[2] CENTRIA, Universidade Nova de Lisboa, 2825 Monte de Caparica, Portugal, {idm,jja}@di.fct.unl.pt
[3] D.M., Univ. Évora, Largo dos Colegiais, 7000 Évora, Portugal

Abstract. Argumentation semantics in extended logic programming has been defined in [5,12] for a single agent which determines its believes by an internal argumentation process. In this paper we extend the initial argumentation framework to a multi-agent setting including both argumentation and cooperation. We define inference for multi-agent systems and define an algorithm for inference. We sketch an argumentation protocol and line out by an example how it is implemented using vivid agents [15,13].

Keywords: Multi-Agent Systems, Argumentation, Logic Programming

1 Introduction

In the last 5-10 years researchers have been devoting much work to the semantics of logic programs. Among the different approaches that usually either emphasise an operational or a declarative view argumentation semantics turned out to be a very intuitive approach. Rather than defining a semantics in technical terms argumentation semantics uses the metaphor of argumentation as used in politics, law, discourse, etc. and formalizes a part of it sufficient to give a meaning to extended logic programs. Intuitively, argumentation semantics treats the evaluation of a logic program as an argumentation process, where a goal G holds if all arguments supporting G cannot be attacked anymore. Thus, logic programming is seen as a discourse involving attacking and counter-attacking arguments.

While argumentation in rhetorics comprises a variety of figures logic programming can be described in terms of two figures: Reductio ad absurdum- and ground-attack [6] or equivalent rebut and undercut [12]. The former classifies an argument that leads to a contradiction under the current believes and arguments, and the latter an argument that falsifies the premise of one of the current arguments. Argumentation semantics in extended logic programming has been defined in [5,12] for a single agent which

* Thanks to Luis Moniz Pereira, Wolfgang Nejdl, Daniela Plewe, and Gerd Wagner. The work is financially supported by JNICT project ACROPOLE PBIC/TIT/2519/95, the Europeu project BMFB/JNICT, and the Brasilian CAPES.

determines its believes by an internal argumentation process. In this paper we extend the initial argumentation framework to a multi-agent setting including both argumentation and cooperation. We define inference for multi-agent systems and define an algorithm for inference. We sketch an argumentation protocol and line out by an example how it is implemented using vivid agents [15,13].

2 Basic Definitions

Since Prolog became a standard in logic programming much research has been devoted to the semantics of logic programs. In particular, Prolog's unsatisfactory treatment of negation as finite failure led to many innovations. Well-founded semantics [8] turned out to be a promising approach to cope with negation by default. Subsequent work extended well-founded semantics with a form of explicit negation and constraints [11,4] and showed that the richer language, called WFSX, is appropriate for a spate of knowledge representation and reasoning forms [9,10,3].

Definition 1. Extended Logic Program, Integrity Constraint
An extended logic program is a (possibly infinite) set of rules of the form $L_0 \leftarrow L_1,\ldots,L_l$, *not* $L_{l+1},\ldots,$ *not* L_m $(0 \leq l \leq m)$ where each L_i is an objective literal $(0 \leq i \leq m)$. An objective literal is either an atom A or its explicit negation $\neg A$. Literals of the form *not*L are called default literals. The set of all objective literals is called the herbrand base $\mathcal{H}(P)$. A rule with head $L_0 = \bot$ is called *integrity constraint*. The symbol $\bot$ stands for falsity. A program P is inconsistent iff $P \models \bot$, otherwise it is consistent.

The following definitions for argumentation are based on [5,12]. In contrast to the latter we do not distinguish between strict and defensible rules. However, our results can be extended to this direction.

Definition 2. Argument
Let P be an extended logic program. An argument for a conclusion L is a finite sequence $A = [r_n,\ldots r_m]$ of ground instances of rules $r_i \in P$ such that

1. for every $n \leq i \leq m$, for every objective literal L_j in the antecedent of r_i there is a $k < i$ such that L_j is the consequent of r_k.
2. L is the consequent of some rule of A;
3. No two distinct rules in the sequence have the same consequent.

A sequence of a subset of rules in A being an argument is called subargument. A rule $r \in P$ is called partial argument.

Definition 3. Undercut, Rebut
Let A_1 and A_2 be two arguments, then

- A_1 *undercuts* A_2 iff A_1 is an argument for L and A_2 is an argument with assumption *not* L, i.e. there is an $r : L_0 \leftarrow L_1,\ldots,L_l,$*not* $L_{l+1},\ldots,$*not* $L_m \in A_2$ and a $l+1 \leq j \leq m$ such that $L = L_j$.
- A_1 *rebuts* A_2 iff A_1 is an argument for L and A_2 is an argument for $\neg L$.

– A_1 *attacks* A_2 iff A_1 undercuts or rebuts A_2.

Definition 4. Coherent, Conflict-free
An argument is coherent if it does not contain subarguments attacking each other. A set *Args* of arguments is called conflict-free if no two arguments in *Args* attack each other.

Definition 5. Defeat, Acceptable
Let A_1 and A_2 be two arguments, then

– A_1 *defeats* A_2 iff A_1 is empty and A_2 incoherent or A_1 undercuts A_2 or A_1 rebuts A_2 and A_2 does not undercut A_1.
– A_1 *strictly defeats* A_2 iff A_1 defeats A_2 but not vice versa.
– A_1 is *acceptable* wrt. a set *Args* of arguments iff each argument undercutting A_1 is strictly defeated by an argument in *Args*.

Our notion of acceptability deviates from Prakken and Sartors definition [12] where an argument A_1 is accepted if each *defeating* argument is accepted. Our notion is more credolous and leads to more intuitive results.

Example 6. Consider the program $P = \{a \leftarrow not\ b; b \leftarrow not\ a; \neg a\}$, then $\neg a$ and $b \leftarrow not\ a$ are acceptabel, whereas $a \leftarrow not\ b$ is not. For Prakken and Sartors definition of acceptibility there is no acceptabel argument which contradicts the intuition of $\neg a$ being a fact.

Definition 7. Characteristic Function
Let P be an extended logic program and S be a subset of arguments of P, then $F_P(S) = \{A \mid A \text{ is acceptable wrt. } S\}$ is called *characteristic function*.

1. A is *justified* iff A is in the least fixpoint of F_P.
2. A is *overruled* iff A is attacked by a justified argument.
3. A is *defensible* iff A is neither justified nor overruled.

Argumentation is closely related to logic programming. While Dung uses argumentation to define a declarative semantics for extended logic programs, Prakken and Sartor's work is driven by their application in legal reasoning. To relate argumentation and extended logic programming, we review WFSX [4], a semantics for extended logic programs.

Definition 8. $\models$ Let P be an extended logic program, then

$$
\begin{aligned}
P &\models L \text{ iff } P,\emptyset,\emptyset,t \models L \\
P,M &\models L \text{ iff } P,\emptyset,\emptyset,M \models L \\
P,LA,GA,M &\models true \\
P,LA,GA,M &\models (L_1,L_2) \text{ iff } P,LA,GA,M \models L_1 \ \&\ P,LA,GA,M \models L_2 \\
P,LA,GA,M &\models not\ L \text{ iff } P,GA,GA,M \models \neg L \ \text{ or} \\
&\qquad M = t \ \&\ P,\emptyset,GA,tu \not\models L \ \text{ or} \\
&\qquad M = tu \ \&\ P,GA,GA,t \not\models L \\
P,LA,GA,t &\models L \text{ iff } L \notin LA \ \&\ L \leftarrow L_1,\ldots,L_l,\ not\ L_{l+1},\ldots,notL_m \in P \ \& \\
&\qquad P,LA\cup\{L\},GA\cup\{L\},t \models L_1,\ldots,L_l,\ not\ L_{l+1},\ldots,notL_m \\
P,LA,GA,tu &\models L \text{ iff } L \notin LA \ \&\ P,GA,GA,t \not\models \neg L \ \& \\
&\qquad L \leftarrow L_1,\ldots,L_l,\ not\ L_{l+1},\ldots,notL_m \in P \ \& \\
&\qquad P,LA\cup\{L\},GA\cup\{L\},tu \models L_1,\ldots,L_l,\ not\ L_{l+1},\ldots,notL_m
\end{aligned}
$$

The inference operator has three parameters M, LA, and GA, where M is either t or tu indicating that we want to prove verity (t) and non-falsity (tu), respectively, LA and GA are lists of local and global ancestors that allow to detect negative and positive loops which lead to inference of non-falsity and failure, respectively; for details see [1,2,4]. For consistent programs the above inference operator yields the same results as the argumentation process:

Proposition 9. *Relation of $\models$ and Argumentation*
Let P be consistent. P,t $\models$ L, iff L is a conclusion from a justified argument. P,t $\models$ not L, iff L is a conclusion from an overruled argument. P,t $\not\models$ L and P,tu $\models$ L iff L is a conclusion from a defensible argument.

Proof sketch: Inference by argumentation is equivalent to WFSX given by bottom-up evaluation of extended logic programs [12], which is in turn equivalent to top-down inference [4]. For inconsistent programs argumentation semantics and WFSX slightly differ. By WFSX a and $\neg a$ can be inferred from the inconsistent program $\{a, \neg a\}$ which is not the case for argumentation semantics [12].

3 Multi-Agent Argumentation

There are two main reasons to extend the single-agent to a multi-agent approach:

1. Given an extended logic program, we want to distribute the program and its evaluation over a network. A semantics for implicitly parallel extended logic programs in terms of argumentation may be promising to be close to an operational semantics, since argumentation is naturally distributed.
2. Given several extended logic programs corresponding to agents with their individual view of the world we want to obtain their global believes.

I.e. in the first case, we partition a program and compute its semantics distributely to solve more complex problems or speed up evaluation. In the second case we exchange parts of several independent or overlapping programs and want to obtain a consensus among the programs concerning inference of literals.

Both approaches need an inference operation for MAS, the only difference being that in the first case we wish that the MAS proof procedure coincides with the single-agent one.

Definition 10. Multi-Agent System
Let Ag_i be extended logic programs, where $1 \leq i \leq n$. Then the set $\mathcal{A} = \{Ag_1, \ldots, Ag_n\}$ is called multi-agent system.

Definition 11. $\models_i$ Let $\mathcal{A} = \{Ag_1, \ldots, Ag_n\}$ be a MAS, then

$$\mathcal{A} \models_i L \text{ iff } \mathcal{A},\emptyset,\emptyset,t \models_i L$$

$$\mathcal{A},LA,GA,M \models_i true$$

$$\mathcal{A},LA,GA,M \models_i (L_1,L_2) \text{ iff } \mathcal{A},LA,GA,M \models_i L_1 \ \&\ \mathcal{A},LA,GA,M \models_i L_2$$

$$\mathcal{A},LA,GA,M \models_i not\ L \text{ iff } \mathcal{A},GA,GA,M \models_i \neg L \text{ or}$$
$$M = t\ \&\ \mathcal{A},\emptyset,GA,tu \not\models_i L \text{ or}$$
$$M = tu\ \&\ \mathcal{A},GA,GA,t \not\models_i L$$

$$\mathcal{A},LA,GA,t \models_i L \text{ iff } L \notin LA \ \&\ L \leftarrow L_1,\ldots,L_l,\ not\ L_{l+1},\ldots,notL_m \in Ag_i \ \&$$
$$\text{f.a. } 1 \leq k \leq l \text{ ex.} 1 \leq j \leq n \text{ s.t.}$$
$$\mathcal{A},LA \cup \{L\},GA \cup \{L\},t \models_j L_k \ \&$$
$$\text{f.a. } l+1 \leq k \leq m \text{ f.a.} 1 \leq j \leq n:$$
$$\mathcal{A},LA \cup \{L\},GA \cup \{L\},t \models_j notL_k$$

$$\mathcal{A},LA,GA,tu \models_i L \text{ iff } L \notin LA \ \&\ \text{f.a.} 1 \leq j \leq n: \mathcal{A},GA,GA,t \not\models_j \neg L \ \&$$
$$L \leftarrow L_1,\ldots,L_l,\ not\ L_{l+1},\ldots,notL_m \in Ag_i \ \&$$
$$\text{f.a. } 1 \leq k \leq l \text{ ex.} 1 \leq j \leq n \text{ s.t.}$$
$$\mathcal{A},LA \cup \{L\},GA \cup \{L\},tu \models_j L_k \ \&$$
$$\text{f.a. } l+1 \leq k \leq m \text{ f.a.} 1 \leq j \leq n:$$
$$\mathcal{A},LA \cup \{L\},GA \cup \{L\},tu \models_j notL_k$$

$$\mathcal{A} \models L \text{ iff } \{\bigcup_{i=1}^{n} \mathcal{A}\} \models L$$

While 1. to 4. of $\models_i$ are identical to $\models$ items 5. and 6. differ. Item 5 states that an agent can infer L if it has a rule for L whose body can be proven with the help of the other agents. This cooperative prove means that any objective literal in the body has to be proven by a least one agent, whereas default literals have to be infered by all of them, i.e. to infer L no agent should object to the default assumptions. Item 6. is similar to 5. for mode *tu* . Multi-agent inference is closely related to the single agent semantics. For $n = 1$, i.e. $\mathcal{A} = \{Ag_1\}$, $\models$ and $\models_1$ coincide. But in general this is not the case. The main difference being that for multi-agent inference default assumptions are treated locally, whereas conclusions via arguments are subject to global consensus.

Proposition 12. $\models$ *is not equivalent to* $\models_i$.

Example 13. For $Ag_1 = \{a \leftarrow not\ b\}$, $Ag_2 = \{b\}$ and $\mathcal{A} = \{Ag_1,Ag_2\}$ we have $\mathcal{A} \models_1 not\ b,not\ a$, $\mathcal{A} \models_2 b$, and $\mathcal{A} \models b,not\ a$. At first sight it seems puzzling that Ag_1 does not infer a. The reason is that concluding a requires Ag_2 to agree to *not b* which is obviously not the case.

However, if an agent has complete knowledge about a literal single and multi-agent semantics are equivalent.

Definition 14. Partial and Complete Definition
Let $\mathcal{A} = \{Ag_1,\ldots,Ag_n\}$ be a MAS. Ag_i *defines L partially* iff $L \in \mathcal{H}(Ag_i)$. Ag_i *defines L completely*, iff Ag_i is the only agent defining L partially.

Proposition 15. *Let Ag_i be an agent that defines L completely, then $\mathcal{A} \models L$ iff $\mathcal{A} \models_i L$.*

Proposition 16. *Let* $\{a_1, \ldots, a_m\}$ *be all indices of agents partially defining L, then* $\mathcal{A} \models L$ *iff ex. j such that* $\mathcal{A} \models_{a_j} L$.

Since previous approaches did not tackle multi-agent argumentation there was no need for cooperation. With two or more agents involved in the process it may be the case that an agent needs the support of another one in order to counter-argue. For example, in a trial a prosecutor may need the help of a witness to defeat the arguments of the defender.

Definition 17. Successful Cooperation
Let $\mathcal{A}$ be a MAS. Agent $Ag_i \in \mathcal{A}$ successfully cooperates wrt. a partial argument $A \in Ag_i$ iff for all objective literals L in the body of A there is an agent $Ag_j \in \mathcal{A}$ such that $\mathcal{A} \models_j L$.

Proposition 18. *If an agent successfully cooperates wrt. a partial argument A and* A' *is equal to A with all objective literals removed from A's body, then* A' *is an argument.*

To infer conclusions the agents have to agree on the justifi ed arguments. To obtain one any possible attack has to be overruled.

Proposition 19. *A is justified iff any attack is overruled.*

The above proposition is the core for an algorithm. Additionally we make use of a dialogue tree [12] where each agent stores the dialogues it has been involved in. To fully implement the algorithm sketched below, the dialogue trees have to be extended to include dialogues on cooperation.
Algorithm:
Proponent, Argumentation:
1. On receipt of a query to infer L, cooperate for arguments for L and then propose these arguments
2. On receipt of an agreement, add it in the dialogue tree and check tree's leaves
3. On receipt of an oppose, cooperate for counter-arguments and then propose them. If there are none, answer initial request negatively
Opponents, Argumentation:
1. On receipt of a proposal, cooperate for counter-arguments and then oppose. If there are none, then agree with proposal
Both, Cooperation for L:
1. For all objective literals L' in the body of a partial argument for L ask all agents to infer L'
2. On receipt of a reply, update cooperation process.

To implement the above algorithm we need agents with an expressive knowledge system and reaction rules such as vivid agents [15,13].

4 Vivid Agents

A *vivid agent* is a software-controlled system whose state is represented by a knowledge base, and whose behavior is represented by means of *action* and *reaction rules*. Following [14], the state of an agent is described in terms of mental qualities, such as beliefs

and intentions. The basic functionality of a vivid agent comprises a knowledge system (including an update and an inference operation), and the capability to represent and perform actions in order to be able to generate and execute plans. Since a vivid agent is `situated' in an environment with which it has to be able to communicate, it also needs the ability to react in response to perception events, and in response to communication events created by the communication acts of other agents. Notice that the concept of vivid agents is based on the important distinction between action and reaction: actions are first planned and then executed in order to solve a task or to achieve a goal, while reactions are triggered by perception and communication events. Reactions may be immediate and independent from the current knowledge state of the agent but they may also depend on the result of deliberation. In any case, they are triggered by events which are not controlled by the agent. A vivid agent without the capability to accept explicit tasks and to solve them by means of planning and plan execution is called *reagent*. The tasks of reagents cannot be assigned in the form of explicit ('see to it that') goals at run time, but have to be encoded in the specification of their reactive behavior at design time.

We do not assume a fixed formal language and a fixed logical system for the knowledge-base of an agent. Rather, we believe that it is more appropriate to choose a suitable knowledge system for each agent individually according to its domain and its tasks. In the case of diagnosis agents, extended logic programs proved to be an appropriate form of the knowledge base of an agent because it is essential for model-based diagnosis to be able to represent negative facts, default rules and constraints.

4.1 Specification and Execution of Reagents

Simple vivid agents whose mental state comprises only beliefs, and whose behavior is purely reactive, i.e. not based on any form of planning and plan execution, are called *reagents*. A reagent $R = \langle X, EQ, RR \rangle$, on the basis of a knowledge system K consists of

1. a knowledge base $X \in L_{\mathrm{KB}}$,
2. an event queue EQ being a list of instantiated event expressions, and
3. a set RR of *reaction rules*, consisting of epistemic and physical reaction and interaction rules which code the reactive and communicative behavior of the agent.

A multi-reagent system is a tuple of reagents $\mathcal{S} = \langle R_1, \ldots, R_n \rangle$

Operational Semantics of Reaction Rules Reaction rules encode the behavior of vivid agents in response to perception events created by the agent's perception subsystems, and to communication events created by communication acts of other agents. We distinguish between epistemic, physical and communicative reaction rules, and call the latter *interaction rules*. We use L_{PEvt} and L_{CEvt} to denote the perception and communication event languages, and $L_{\mathrm{Evt}} = L_{\mathrm{PEvt}} \cup L_{\mathrm{CEvt}}$. The following table describes the different formats of epistemic, physical and communicative reaction rules:

$$
\begin{aligned}
\mathit{Eff} &\leftarrow \mathsf{recvMsg}[\varepsilon(U), S], \mathit{Cond} \\
\mathsf{do}(\alpha(V)), \mathit{Eff} &\leftarrow \mathsf{recvMsg}[\varepsilon(U), S], \mathit{Cond} \\
\mathsf{sendMsg}[\eta(V), R], \mathit{Eff} &\leftarrow \mathsf{recvMsg}[\varepsilon(U), S], \mathit{Cond}
\end{aligned}
$$

The event condition $\mathsf{recvMsg}[\varepsilon(U), S]$ is a test whether the event queue of the agent contains a message of the form $\varepsilon(U)$ sent by some perception subsystem of the agent or by another agent identified by S, where $\varepsilon \in L_{\text{Evt}}$ represents a perception or a communication event type, and U is a suitable list of parameters. The epistemic condition $Cond \in L_{\text{Query}}$ refers to the current knowledge state, and the epistemic effect $Eff \in L_{\text{Input}}$ specifies an update of the current knowledge state.

Physical Reaction: $\mathsf{do}(\alpha(V))$ calls a procedure realizing the action α with parameters V.

Communicative Reaction: $\mathsf{sendMsg}[\eta(V), R]$ sends the message $\eta \in L_{\text{CEvt}}$ with parameters V to the receiver R.

Both perception and communication events are represented by incoming messages. In general, reactions are based both on perception and on knowledge. Immediate reactions do not allow for deliberation. They are represented by rules with an empty epistemic premise, i.e. $Cond = \mathsf{true}$. Timely reactions can be achieved by guaranteeing fast response times for checking the precondition of a reaction rule. This will be the case, for instance, if the precondition can be checked by simple table look-up (such as in relational databases or fact bases).

Reaction rules are triggered by events. The agent interpreter continually checks the event queue of the agent. If there is a new event message, it is matched with the event condition of all reaction rules, and the epistemic conditions of those rules matching the event are evaluated. If they are satisfiable in the current knowledge base, all free variables in the rules are instantiated accordingly resulting in a set of triggered actions with associated epistemic effects. All these actions are then executed, leading to physical actions and to sending messages to other agents, and their epistemic effects are assimilated into the current knowledge base.

5 Argumentation Protocol

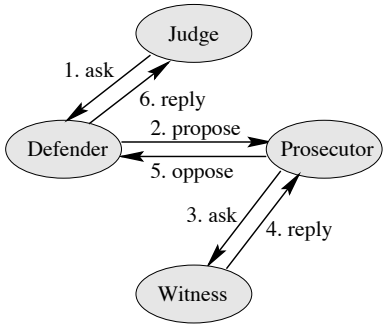
To implement the MAS's inference operation, the agents evolve an argumentation process. In the light of speech act theory and subsequent agent communication languages such as ACL, KIF, or KQML [7] we can identify five relevant speech acts of the dialogues:

1. An agent requests inference of a literal L by sending a token $ask(L)$,
2. and replies to a request by $reply(L)$.
3. An agent proposes a conclusion L to other agents by sending the token $propose(L, A, LA, GA, M)$ where A is the agent's argument A for conclusion L. LA, GA, and M are parameters for local and global ancestors and the mode due to definition 11
4. An agent opposes an argument by $oppose(L, A, LA, GA, M, L', A', LA, GA', M')$ where $L' \leftarrow A'$ is an argument attacking the previously received $L \leftarrow A$.
5. In case the received argument is acceptable the agent acknowledges this by sending $agree(L)$, where L is the conclusion of A.

Using the above speech acts we defi ne an argumentation protocols in terms of reaction rules. We assume that a proposing agent sends its conclusion to all agents, but the opposition against a conclusion occurs only between the proponent and the opposer.

Consider a trial with a judge, a defender, a prosecutor and a witness. The judge has no knowledge, while the defender knows that the accused is by default not guilty $\neg guilty \leftarrow not\ guilty$, i.e. explicit negation is derived by default. The prosecutor knows that the accused is guilty if there is evidence given by a witness: $guilty \leftarrow seen$. Finally the witness saw the accused committing the crime and knows therefore *seen*. If the judge asks the defender to defend the accused an argumentation process as shown on the right evolves.

The defender proposes not guilty by its default assumption. The prosecutor asks the witness to cooperate by asking whether the witness saw the accused. With the subsequent testimony the prosecutor generates the counter-argument that the accused is guilty because he/she was seen. The defender cannot attack the counter-argument anymore and therefore answers to the judge accordingly.



To implement the above scenario we assume two meta predicates $argument/2$ and $partial\_argument/3$. If the defender receives a query of the judge it generates an argument for its conclusion and proposes it to the prosecutor (see fi gure 1 rule 1), the prosecutor asks the witness for cooperation in case it receives a proposal by the defense and has a partial argument that needs further elaboration (2). If the witness is asked and can serve the query it responds accordingly (3). The prosecutor assimilates testimonies of the witness and informs itself to check arguments in the light of the new testimony (4). On the receipt of the check-message the prosecutor verifi es whether there are open proposals of the defense and arguments to counter-attack the proposal, which are then sent. If the defense receive an oppose to its proposal it reports to the judge its failed proposal.

If executed the example leads to a trace as shown in fi gure 2. Though the example is small it gives a favor of the full implementation. It contains proposing, opposition and cooperation. For the sake of simplicity it does not consider full dialogue of repeated proposal and opposition and it does not implement the full semantics. However, the example shows how dialogues can be easily expressed by reaction rules which are executable. For the full implementation of the algorithm the concept of dialogue trees [12] has to be generalized to include multiple agents and cooperation. Vivid agents have a further advantage as tested for implemented legal reasoning. They are formally underpinned, which allows to verify protocols. In the proof theory lined out in [15,13] we can prove the following proposition stating that inference by $\models_i$ is equivalent to a reply transition eventually reached by the vivid agent:

Proposition 20. *Let* $\mathcal{A} = \{Ag_1, \ldots, Ag_n\}$ *be a MAS. Then* $\mathcal{A} \models_i L$ *iff* $(Ag_i, RR_i, [ask(L)|EQ]) \rightarrow^* \xrightarrow{reply(L)} (Ag_i, RR_i, EQ')$.

1. $send(propose(L,B), prosecutor)$,
   $proposed(L,B) \longleftarrow$
   $recv(ask(L), judge)$,
   $i\_am(defender)$,
   $argument(L,B)$.
2. $send(ask(C), witness)$,
   $open\_proposed(L,B) \longleftarrow$
   $recv(propose(L,B), defender)$,
   $i\_am(prosecutor)$,
   $partial\_argument(L,B,C)$.
3. $send(reply(B), A) \longleftarrow$
   $recv(ask(B), A)$,
   $i\_am(witness), B$.
4. $send(check\_for\_oppose, prosecutor), B \longleftarrow$
   $recv(reply(B), witness)$,
   $i\_am(prosecutor)$.
5. $send(oppose(L,B,C,D), defender) \longleftarrow$
   $recv(check\_for\_oppose, prosecutor)$,
   $i\_am(prosecutor)$,
   $open\_proposed(L,B)$,
   $member(not\ C, B)$,
   $argument(C,D)$.
6. $send(reply(notL), judge) \longleftarrow$
   $recv(oppose(L,B,C,D), prosecutor)$,
   $proposed(L,B)$.

Fig. 1. Reaction rules of the trial.

1.	*defender*	$\longleftarrow$	*judge*	$ask(\neg guilty)$
2.	*defender*	$\longrightarrow$	*prosecutor*	$propose(\neg guilty, [not\ guilty])$
3.			*defender*	$assimilates\ proposed(\neg guilty, [not\ guilty])$
4.	*prosecutor*	$\longleftarrow$	*defender*	$propose(\neg guilty, [not\ guilty])$
5.	*prosecutor*	$\longrightarrow$	*witness*	$ask(seen)$
6.			*prosecutor*	$assimilates\ open\_proposed(\neg guilty, [not\ guilty])$
7.	*witness*	$\longleftarrow$	*prosecutor*	$ask(seen)$
8.	*witness*	$\longrightarrow$	*prosecutor*	$reply(seen)$
9.	*prosecutor*	$\longleftarrow$	*witness*	$reply(seen)$
10.	*prosecutor*	$\longrightarrow$	*prosecutor*	$check\_for\_oppose$
11.			*prosecutor*	$assimilates\ seen$
12.	*prosecutor*	$\longleftarrow$	*prosecutor*	$check\_for\_oppose$
13.	*prosecutor*	$\longrightarrow$	*defender*	$oppose(\neg guilty, [not\ guilty], guilty, [seen])$
14.	*defender*	$\longleftarrow$	*prosecutor*	$oppose(\neg guilty, [not\ guilty], guilty, [seen])$
15.	*defender*	$\longrightarrow$	*judge*	$reply(not\ \neg guilty)$
16.	*judge*	$\longleftarrow$	*defender*	$reply(not\ \neg guilty)$

Fig. 2. Trace of the trial example.

6 Comparison and Conclusion

The work presented in this paper is based on work by Dung [5,6] and Prakken and Sartor [12] on argumentation. Dung defines a declarative semantics for extended logic programs using the metaphor of argumentation. Our work continues this line of research in that we extend the single-agent approach to a multi-agent one. Prakken and Sartor are motivated by legal reasoning and define a rigorous and concise framework similar to Dung's. Prakken and Sartor also deal only with a single agent and therefore do not tackle the issue of cooperation. Defeasible priorities as used in [12] can be easily added in the implementation by defining the meta predicate *argument* accordingly. In contrast to ours, Dung's and Prakken and Sartor's work is not implemented.

In this article we showed the connection between top-down inference of WFSX and bottom-up argumentation. We extended top-down inference from single to multi-agent and discussed the relation. We designed an argumentation language to specify argumentation protocols for multi-agent systems and implemented multi-agent inference by vivid agents. For a full implementation dialogue trees have to be extended to include cooperation.

References

1. J. J. Alferes, C. V. Dam´asio, and L. M. Pereira. Top–down query evaluation for well–founded semantics with explicit negation. In A. Cohn, editor, *Proc. of the European Conference on Artifi cial Intelligence'94*, pages 140–144. John Wiley & Sons, August 1994.
2. J. J. Alferes, C. V. Dam´asio, and L. M. Pereira. A top-down derivation procedure for programs with explicit negation. In M. Bruynooghe, editor, *Proc. of the International Logic Programming Symposium'94*, pages 424–438. MIT Press, November 1994.
3. J. J. Alferes, C. V. Dam´asio, and L. M. Pereira. A logic programming system for nonmonotonic reasoning. *Journal of Automated Reasoning*, 14(1):93–147, 1995.
4. J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*. (LNAI 1111), Springer-Verlag, 1996.
5. P. M. Dung. An argumentation semantics for logic programming with explicit negation. In *Proc. of the 10th International Conference on Logic Programming*, pages 616–630. MIT Press, 1993.
6. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artifi cial Intelligence*, 77(2):321–357, 1995.
7. T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, and C. Beck. Specification of the KQML agent communication language. Technical report, The DARPA Knowledge Sharing Initiative, External Interfaces Working Group, Baltimore, USA, 1993.
8. Allen Van Gelder, Kenneth Ross, and John S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceeding of the 7th ACM Symposium on Principles of Databse Systems*, pages 221–230. Austin, Texas, 1988.
9. L. M. Pereira, J. N. Apar´ cio, and J. J. Alferes. Non–monotonic reasoning with logic programming. *Journal of Logic Programming. Special issue on Nonmonotonic reasoning*, 17(2, 3 & 4), 1993.

10. L. M. Pereira, C. V. Dam´asio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *2nd Int. Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 334–348, Lisboa, Portugal, June 1993. MIT Press.
11. Lu´ s Moniz Pereira and Jos´e J´ulio Alferes. Well founded semantics for logic programs with explicit negation. In *B. Neumann (Ed.), European Conference on Artifi cial Intelligence*, pages 102–106. John Wiley & Sons, 1992.
12. Henry Prakken and Giovanni Sartor. Argument-based extended logig prpgramming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 1997.
13. Michael Schroeder, Rui Marques, Gerd Wagner, and Jos´e Cunha. CAP - Concurrent Action and Planning: Using PVM-Prolog to implement vivid agents. In *Proceedings of the fi fth Conference on Practical Applications of Prolog*, 1997. to be published.
14. Yoav Shoham. Agent-oriented programming. *Artifi cial Intelligence*, 60(1):51–92, 1993.
15. Gerd Wagner. A logical and operational model of scalable knowledge-and perception-based agents. In *Proceedings of MAAMAW96, LNAI 1038*. Springer-Verlag, 1996.