Negotiation via co-operation and competition in *ALIAS*

Anna Ciampolini [a] Evelina Lamma [b] Paola Mello [a] Francesca Toni [c] Paolo Torroni [a]

[a] *University of Bologna*
V.le Risorgimento 2, 40136 Bologna, Italy
email: {aciampolini,pmello, ptorroni}@deis.unibo.it
[b] *University of Ferrara*
Via Saragat 1, 44100 Ferrara, Italy
email: elamma@ing.unife.it
[c] *Imperial College*
London, UK
email: ft@doc.ic.ac.uk

Abstract

This paper presents ALIAS, an agent architecture based on intelligent logic agents, where the main form of agent reasoning is abduction. The system is particularly suited for solving problems where knowledge is incomplete, where agents may need to make reasonable hypotheses about the problem domain and other agents. ALIAS agents are pro-active, exhibiting a goal-directed behavior, and autonomous, since each one can solve problems using its own private knowledge base. ALIAS agents are also social, because they are able to interact with other agents, in order to cooperatively solve problems. The coordination mechanisms are modeled by means of LAILA, a logic-based language which allows to express intra-agent reasoning and inter-agent coordination. We show how LAILA can be used to implement inter-agent dialogues, e.g., for negotiation. In particular, LAILA is well-suited to coordinate the process of negotiation aimed at exchanging resources between agents, thus allowing them to execute the plans to achieve their goals.

Received 12 February 2001

Keywords: Logic agents, abductive reasoning, dialogue, negotiation, coordination

1. Introduction

In the last decade the interest for multi-agent systems (MAS) and the agent paradigm has considerably increased [40], leading to a shift in the nature of computing: away from building stand-alone applications, to the development of distributed systems, built largely from pre-existing components [52,35]. In this setting, MAS have been proposed as systems that enclose properties such as intelligence, autonomy in reasoning and in activity, communication abilities, and all categories that could make a software component somewhat self-contained, easy to be used but also combined with other ones in building an application.

In complex, distributed and dynamic environments, it is very hard to make assumptions about the available knowledge. In fact, one of the basic principles of multi-agent systems is locality: each agent embodies a private knowledge base which is managed and directly accessed only by the agent itself. Moreover, in real cases such as, for instance, WEB applications, the rapid changes of the scenario, where agents may appear and disappear and no global control is applied to the system, implies that it is also difficult to make assumptions about knowledge unicity and completeness.

In such situations, it is often the case that an agent requires some sort of guess about a computation which cannot be performed locally, since for it only incomplete information is known. In this respect, the Closed World Assumption [8] can be no longer assumed, and some form of open or abductive reasoning has to be considered. Abduction has been widely recognized as a powerful mechanism for hypothetical reasoning in presence of incomplete knowledge [10,16,26]. Abduction is generally understood as reasoning from effects to causes, and also captures other important issues such as reasoning with defaults and beliefs (see for instance [30,44]). In a single agent context, hypotheses are made provided that they are consistent with the agent's current knowledge. In a multi-agent context, the need emerges for specific protocols for global consistency maintenance, as agents interacting with one another, although reasoning with their own local knowledge base, cannot completely set aside other agents beliefs. In particular, the hypotheses generated by an individual agent on behalf of a second one need to be re-checked for consistency with the beliefs of the second.

In this paper we present a logic-based agent architecture, Abductive LogIc AgentS (ALIAS, for short), where agents are intelligent and have social abilities. In particular, intelligence stems from the agents' capability of performing abductive reasoning with their own knowledge, which is specified in a logic programming-based paradigm.

Interaction and cooperation with oneanother are amongst the fundamental social cabilities of agents. In the literature, communication and coordination languages such as KQML[17] and Linda[21,22] have been proposed in order to provide a standard infrastructure for agent social behavior. With reference to social ability, ALIAS agents can coordinate their reasoning with other agents, following several coordination schemes. In particular, ALIAS agents could either "collaborate" or "compete" in the solution of problems.

In the *collaborative* case, the solution of a (possibly complex) task is distributed among several agents. Let us consider, as an example, a group of medical doctors, each one expert in a particular area (e.g. gastroenterology, ematology, etc.) who have to collaborate in order to formulate a diagnosis for a given set of symptoms. Each expert can be modeled by an abductive agent whose task is to find a hypothesis (i.e., a disease) as an explanation for a sub-part of the symptoms (i.e., those relevant to his/her area) given as observations to the agent. In some cases, the hypotheses made by an agent can be inconsistent with other hypotheses made by a collaborative agent (for instance, a certain symptom s does not occur when the disease d is present).

In the *competitive* case, instead, several agents could be asked to solve the same problem. Let us consider again a group of medical doctors, each one expert in a particular area, who have to find a diagnosis for the symptom. Different, alternative diagnoses can be proposed by the different agents, and the *best* one can be chosen according to some policy (for instance, the one with major incidence, or the most plausible one, with respect to the clinical history of the patient).

Agent interaction and coordination is specified in ALIAS by using a logic language named LAILA (Language for AbductIve Logic Agents) that allows to model agent structures, especially from the viewpoint of social behaviour. In particular, LAILA provides agents with the possibility of expressing different cooperation schemata by means of high level declarative operators, such as the communication operator $>$, the competition operator ;, the collaboration operator &, while a down-reflection operator $\downarrow$ allows a local abductive resolution to be triggered.

Notice that the ALIAS architecture is layered, in particular reasoning and social behaviour are kept separate and correspond to different levels of abstraction: the upper level specifies, by means of LAILA expressions, possibly complex schemata of interaction among agents and can also exploit the reasoning capabilities local to the agents (located at the lower level, and specified by abductive logic programming).

We apply the ALIAS architecture and LAILA to implement a form of negotiation among agents. Negotiation plays a central role in e-commerce multi-agent applications. In such a domain, when it is natural to assume that agents are self-interested and lack complete knowledge and resources, it is likely that they need to negotiate in order to obtain the information or resources they need. The dialogue protocols used for negotiation have to be properly designed in order to ensure that each part will have a positive payoff out of the negotiation process. To this purpose, we rely on the dialogue performatives and protocols introduced in [47], and show how negotiation can be also addressed in ALIAS.

The paper is organized as follows. In Section 2 we sketch briefly the architecture of the system. Section 3 discusses abduction as an agent reasoning paradigm and describes how abduction is exploited within ALIAS. Section 4 defines the coordination language by giving syntax and operational semantics. Section 5 is devoted to introduce the dialogue performatives (dialogue moves) useful for negotiation, and addresses negotiation by exploiting ALIAS and LAILA cooperation schemata in particular. Related work and discussion follow.

2. The architecture of *ALIAS*

In this section we introduce *ALIAS* (Abductive LogIc AgentS), an agent architecture where several agents, each enclosing a local knowledge base, can either autonomously reason using its own local knowledge base or dynamically join other agents to cooperatively solve goals using abductive reasoning. As we will see later on, the reasoning of agents can be coordinated following either a competitive or a collaborative model, or a combination of them.

The inner structure of each agent, shown in Figure 1, is basically composed of two modules: the *Abductive Reasoning Module*, $\mathcal{ARM}$, and the *Agent Behavior Module*, $\mathcal{ABM}$. Each agent is characterized by two knowledge bases: the *abductive knowledge base*, encapsulated within the $\mathcal{ARM}$, and the *behavior knowledge base*, encapsulated within the $\mathcal{ABM}$. The abductive knowledge base is represented by an *abductive logic program* (details are given in Section 3), describing the knowledge local to an agent and used to perform abductive reasoning, e.g., for planning towards the achievement of the agent's goals or for explaining the agents' observations. The behavior knowledge base is a set of logic clauses in *LAILA* (see Section 4), used to describe the desired actions and interactions of the agent within its environment. In particular, the social behavior of each agent can be expressed within the $\mathcal{ABM}$, by means of collaborative/competitive queries submitted to other agents. Sometimes the agent's behavior may require the abductive explanation of an observation: this situation requires an interaction between local $\mathcal{ABM}$ and $\mathcal{ARM}$, in order to locally start an abductive computation.

In our framework, each ALIAS multi-agent application is represented by a set of abductive agents, each modeled by its knowledge bases (located respectively at $\mathcal{ARM}$ and $\mathcal{ABM}$ levels), as illustrated in Figure 1.

We assume that agents can be grouped within *bunches*, which are constructed dynamically according to the requirements of interaction between agents, with the purpose, for instance, of finding the explanation for a given observation in a collaborative way.

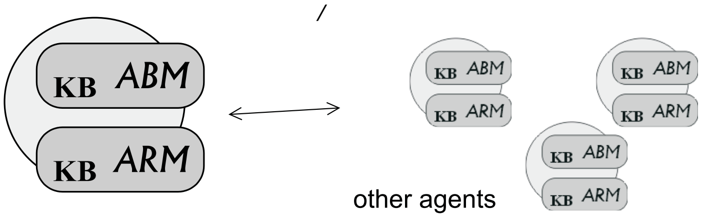

Figure 1. The structure of an *ALIAS* agent

Example 1 We introduce an example that we will use throughout the paper. Let us consider three agents, say a, b, and c. Let us suppose that a is missing a resource, and in particular a $nail$ that she needs in order to hang a picture to the wall. Then a might decide to request the nail from b. As we will see in the next section, the abduction mechanism can be used to implement agent dialogues, and in this case to produce a reply to the request. In fact, this request can be seen as the abductive explanation of a query, submitted by a to b, within b's own local knowledge base. b's reply will be enclosed in a set of possible hypotheses. It is up to a to decide whether such hypotheses are acceptable or not. Agent b, for instance, could try to sell a the nail for too much, asking 10 for a nail that – according to a's knowledge base – is worth at most 8.

It could also be the case that b implements the following policy in giving her resources: when asked the nail, b both tries and see if she has it and can give it, *and* asks for it from c. This could be seen as a competitive process, where b tries to return a suitable set of hypotheses, possibly choosing among several alternatives (the one produced by c and the other one produced by herself).

If we implement this example within ALIAS, the $\mathcal{ABM}$ module will contain all the behavior rules, such as *'if I need a nail then I will ask it to b'*, or *'if I am requested a nail then I will try to obtain it either from my set of resources or from c'*, and the $\mathcal{ARM}$ will contain the knowledge to be used locally, such as *'I am agent b'*, *'I have a nail'*, *'the cost of a nail is not greater than 8'*, and so on.
$\square$

3. Abduction in a Multi-Agent Environment

3.1. Preliminaries

The vocabulary of logic programs is defined in [39]. A logic program is a set of clauses

$\mathtt{A} :- \mathtt{L_1}, \ldots, \mathtt{L_n}$

where $\mathtt{A}$ is an atom, and each $\mathtt{L_i}$ is a literal, i.e. an atom $\mathtt{B}$ or its negation $\mathtt{not\ B}$. An *atom* is an n-ary predicate symbol followed by a list of n terms $p(t_1, \ldots, t_n)$, or `true`, or `false`. A *literal* is an atom or the negation of an atom. A *term* is a constant, a variable or a compound term. Also, in the following we will adopt, for both the abductive programs and the LAILA programs that we will introduce later, the Edinburgh notation for standard Prolog, where variable names start with upper case or '_', and constants start with lower case.

Note that `not` represents default negation or negation as failure. All variables in $\mathtt{A}, \mathtt{L_1}, \ldots, \mathtt{L_n}$ are assumed to be universally quantified from the outside. $\mathtt{A}$ is referred to as the *head* of the clause and $\mathtt{L_1}, \ldots, \mathtt{L_n}$ as its *body*.

An *abductive logic program* is a triple $\langle P, \mathcal{A}, IC \rangle$, where P is a logic program, possibly with undefined (open) predicates in clause bodies; $\mathcal{A}$ is a set of predicates, called *abducibles*, which, without loss of generality [25], can be assumed to be a subset of the undefined predicates in P; atoms in the abducible predicates (also called abducible atoms or hypotheses) can be used to extend P in order to "explain" given sentences, namely *queries* or *observations* or *goals*; IC is a set of sentences, referred to as *integrity constraints*, whose syntax depends on the abductive reasoning mechanism chosen to generate explanations.

We express the abductive knowledge base of ALIAS agents by means of an abductive logic program, and adopt the Kakas-Mancarella proof procedure of [26] as the abductive reasoning mechanism underlying the $\mathcal{ARM}$ of ALIAS agents. As a consequence, we will adopt *denial* integrity constraints à la Kakas-Mancarella, of the form

$: - \mathtt{L_1}, \ldots, \mathtt{L_n}$

where each $\mathtt{L_i}$ is a literal, at least one of the `Li` is an abducible atom and all variables in `L1`, ..., `Ln` are assumed to be universally quantified from the outside. [1]

Given an abductive program $\langle P, \mathcal{A}, \mathcal{IC} \rangle$ and a sentence G, the purpose of abduction is to find a (possibly minimal) set of abducible atoms Δ which together with P "entails" G and such that $P \cup \Delta$ "satisfies" IC. Various notions of "entailment" and "satisfaction" are possible, depending on the chosen semantics for the given logic program [25].

The following example illustrates these concepts.

Example 2 Let us consider the following logic program P, inspired by [43]:

```
grass_is_wet  :- rained_last_night
grass_is_wet  :- sprinkler_was_on
shoes_are_wet :- grass_is_wet
```

with integrity constraint IC:

```
:- rained_last_night, sprinkler_was_on.
```

Let the predicates `rained_last_night` and `sprinkler_was_on` be the only abducible predicates in $\mathcal{A}$. The observation $G =$ `shoes_are_wet` can be explained by two (minimal) sets of sentences, respectively:

$\Delta_1 = \{\mathtt{rained\_last\_night},\ \mathtt{not\ sprinkler\_was\_on}\}$, and
$\Delta_2 = \{\mathtt{sprinkler\_was\_on},\ \mathtt{not\ rained\_last\_night}\}$
$\square$

The Kakas-Mancarella proof procedure is an extension of the proof procedure for ordinary logic programming proposed by [16]. Both procedures extend the basic resolution mechanism adopted in SLD and SLDNF in ordinary logic programming by introducing the notion of *abductive* and *consistency* derivations. An abductive derivation computes a set of hypotheses "entailing" (with respect to the given logic program) a given goal/query/observation, starting from a given set of initial hypotheses. In order to compute such set and guarantee that the integrity constraints are "satisfied", an abducive derivation might require subsidiary consistency derivations, which in turn migh require abductive derivations to ensure integrity constraint "satisfaction". We will not provide any further details about this procedure here, the interested reader is referred to [27]. Note however that this procedure has

[1] Note however that in principle ALIAS is independent of the abductive reasoning mechanism.

been proven correct, for some specific notions of "entailment" and "satisfaction", with respect to the abductive semantics defined in [6], as well as the argumentation-semantics defined in [51].

3.2. Abductive reasoning in ALIAS

In the Multi-Agent ALIAS architecture, we assume that each agent is equipped with its own abductive logic program, but with respect to a common vocabulary and with respect to a common set of abducibles. Namely, given a set of agents in a bunch, $\{A_0, \ldots, A_n\}$, each A_i is equipped with a logic program $\langle P_i, \mathcal{A}, IC_i \rangle$. We assume the set of abducibles $\mathcal{A}$ to be a shared set as it represents in a way the common language that the agents in the same bunch use to communicate, as we will see later.

In order to perform abduction in a Multi-Agent environment we need to define an algorithm able of coping with several agents possibly working at explaining the same observations, and thus needing to compute the same set of hypotheses. Thus, if an agent assumes a new hypothesis h, all the agents belonging to the same bunch must check the consistency of h with their own integrity constraints. These checks could possibly raise new hypotheses, whose consistency within the bunch has to be recursively checked. Therefore, in ALIAS, the abductive explanation of a goal within a bunch of agents is a set of abduced hypotheses, agreed by all agents in the bunch.

In order to perform abduction in a multi-agent environment we need to introduce some mechanism to support agent bunches, local abduction and global consistency checks. As far as local abduction is concerned, we adopted the Kakas-Mancarella proof procedure. In ALIAS, we adopt a two-phase algorithm, which first collects the results of local[2] abduction processes running the Kakas-Mancarella proof procedure, and then, depending on the chosen coordination policy, checks their consistency, in order to find a Δ which is consistent with all the constraints present in the knowledge bases of the agents of the bunch. Thus, the first step involves a local abductive derivation, while the second one involves a consistency derivation in a group of agents. An operational semantics of the notions of consistency derivation in a multi-agent context is given in 4.2.

As we mentioned before, the abductive mechanism can be effectively adopted to implement agent dialogue. Let us consider a dialogue as a sequence of *dialogue moves*, i.e., requests, proposals, etc. that agents alternatively make to one another. For the sake of simplicity, we leave the time implicit. In a dialogue about obtaining a resource, let $\mathtt{req}(\mathtt{Sender}, \mathtt{Recipient}, \mathtt{Resource})$ and $\mathtt{sell}(\mathtt{Seller}, \mathtt{Buyer}, \mathtt{Item}, \mathtt{Price})$ be abducible predicates. The first one represents a request made by agent `Sender` and addressed to `Receiver`, and in particular a request to obtain a `Resource`; the second one represents an offer of an `Item` at a certain `Price`, the offer being made by agent `Seller` to agent `Buyer`. Now, we could use the integrity constraints as a tool to express how we want an agent to reply to a certain move, under certain conditions. Let us consider again Example 1. Let agent b's $\mathcal{ARM}$ contain the following integrity constraint:

```
% IC b:  offer a resource to a friend
:- req(X,b,R), have(R), friend(X), not sell(b,X,R,0)
```

Namely, if requested a resource by a friend, agent b will give it to her for free. By an operational point of view, if a requests a nail, the goal to be proven by b becomes $\mathtt{req}(\mathtt{a}, \mathtt{b}, \mathtt{nail})$. As this is abducible, it will be assumed

[2] In this context, *local* means *performed within the agent that is demonstrating the query*

true and checked against the integrity constraints. These, together with the facts `have(nail)` and `friend(a)`, will produce the abducible `sell(b,a,nail,0)` that is used to express that a resource is offered for free.

Let us consider now a's integrity constraint, used to decide whether a proposed price for a resource is to be accepted or not:

```
% IC a:  do not over-pay a resource
:- sell(_,a,R,P), max_price(R,MP), P > MP
```

In this case, if `sell(c,a,nail,10)` is a hypothesis made by an agent, say c, telling that she will sell a the nail for 10, and if in a's abductive program there is the fact `max_price(nail,8)`, such hypothesis will be rejected because it would produce an inconsistency within a.

4. The LAILA Language

The behavior of *ALIAS* agents is expressed by means of the *Language for AbductIve Logic Agents* (LAILA, for short). This language allows to model agent actions and interactions in a logic programming style. In particular we will focus on agent social behavior, and especially on how each agent can request proofs of goals to other agents in the system. For this purpose, we introduce two composition operators: the collaborative operator (&) and the competitive operator (;) that can be used by each agent to formulate and coordinate abductive queries to other (set of) agents. The language provides also a communication operator (>) that is used to submit queries to other agents, and a *down-reflection* ($\downarrow$) operator to trigger a local abductive derivation. In the remainder of this section, we will describe the syntax and the operational semantics of LAILA, then we will show an example and briefly discuss about an implementation of LAILA.

4.1. The Syntax of LAILA

Let us consider a system composed by multiple agents. Each agent encapsulates a LAILA program describing its behavior. Let $\mathcal{V}$ be the vocabulary of the language. We add to the standard logic programming vocabulary [39] the set $\{\ \&\ ,\ ;\ ,\ >\ ,\ \downarrow\ \}$ of operators, where:

- & is the *collaborative* coordination operator;
- ; is the *competitive* coordination operator;
- $>$ is the *communication* operator;
- $\downarrow$ is the *down-reflection* operator;

A *LAILA program* is a set of clauses. A *LAILA clause* (also, *L-Clause*) is defined as follows by a BNF grammar:

$$
\begin{array}{lll}
\textit{L-Clause} & ::= & \textit{Atom} \ :- \textit{L-Body}. \\
\textit{L-Body} & ::= & \textit{L-Expr}\ ,\ \textit{L-Body} \mid \textit{L-Expr} \\
\textit{L-Expr} & ::= & \textit{L-Fact}\ \&\ \textit{L-Expr} \mid \textit{L-Fact}\ ;\ \textit{L-Expr} \mid \textit{L-Fact} \\
\textit{L-Fact} & ::= & \downarrow \mathit{LiteralList} \mid \mathit{Agent} > \mathit{LiteralList} \mid \mathit{Literal} \mid (\ \textit{L-Body}\ ) \\
\mathit{LiteralList} & ::= & \mathit{Literal} \mid (\ \mathit{Literals}\ ) \\
\mathit{Literals} & ::= & \mathit{Literal}\ ,\ \mathit{Literals} \mid \mathit{Literal} \\
\mathit{Agent} & ::= & \mathit{Atom}
\end{array}
$$

We already defined in Section 3 the meaning of atoms, literals, and terms. *L-Body*, *L-Expr* and *L-Fact* respectively stand for *LAILA Body*, *LAILA Expression* and *LAILA Factor*. A computation can be started by a $Query$, defined as *L-Query* $::=$? *L-Body*. To clarify the sense of *L-Clauses*, let us consider, for instance, the following LAILA competitive query, formulated by agent A_0:

$$? \downarrow \texttt{G1} \ ; \ \texttt{A1} > \texttt{G2}$$

It means that A_0 must either perform a local abductive derivation for the goal G_1, *or* ask agent A_1 to demonstrate goal G_2. In case the local derivation fails the system will have to wait for A_1's answer, and vice versa. In case both fail, the whole query fails.

Let us consider, now, this collaborative query issued by agent A_0:

$$? \ \texttt{A1} > \texttt{G3} \ \& \ \texttt{A2} > \texttt{G4}$$

It means that agent A_0 asks agent A_1 to prove G_3 and A_2 to prove goal G_4; the result is obtained by merging in a unique consistent set of abducibles the answers respectively produced by A_1 and A_2.

4.2. The Semantics of LAILA

For the sake of simplicity, the following inference rules refer to the case of propositional LAILA programs. In the following, $\mathcal{U}$ is the universe of agents, and $A_0, \ldots, A_n$ denote agents in the system, i.e., $\{A_0, \ldots, A_n\} \subset \mathcal{U}$. The entailment symbol adopted is $\vdash_\delta^B$, whose superscript is the bunch B of agents involved in the computation, and whose subscript is the set of assumed hypotheses δ, both of which are output parameters of the derivation process. Given $A \vdash_\delta^B F$, F is the *formula* to prove (a *formula* is in general a *L-Body*), and A is the program used for the derivation. Such program is an agent's $\mathcal{ABM}$: for the sake of simplicity, and with abuse of notation, we will write the name of the agent instead of its $\mathcal{ABM}$. Therefore, $A \vdash_\delta^B F$ means that the formula F is proven within A's $\mathcal{ABM}$, producing as an output a set of hypotheses δ and a bunch B. Finally, we will adopt the following notation for the abductive and consistency derivations:

- $A \overset{abd}{\vdash}_\delta G$, where G is a set of literals, and $A \in \{A_0, \ldots, A_n\}$ is an agent enclosing in its $\mathcal{ARM}$ an abductive logic program $\langle P, \mathcal{A}, IC\rangle$ denotes the local abductive proof ("entailment") for the conjunction of all literals in G, performed by A within its $\mathcal{ARM}$.
- $B \overset{cons}{\vdash}_\delta \Delta$, where Δ is a set of abducibles, denotes the "consistency check" of the conjunction of all atoms in Δ, with respect to the integrity constraints of all agents in bunch B; in particular, let $S \subseteq \{1, \ldots, n\}$. Given

a bunch B of agents $\{A_i | i \in S\}$, each enclosing an abductive logic program $\langle P_i, \mathcal{A}, IC_i \rangle$, the declarative semantics of $B \overset{cons}{\vdash}_{\delta} \Delta$ is defined as follows:[3]

$$\begin{cases} \forall A_i \in B \quad P_i \cup \delta \text{ ``\textit{satisfies}'' } \bigcup_{j \in S} IC_j \\ \Delta \subseteq \delta \subseteq \mathcal{A} \end{cases}$$

We achieve in that a separation among the agents knowledge bases, ensuring the absence of conflicts between δ and each of their integrity constraints. Moreover, we would like to notice that, while in describing the declarative semantics we refer – for the sake of simplicity – to the union of the agent integrity constraints, the actual implementation of the system does not require at all that the agents share any knowledge in that respect (see the consistency check operational semantics further on for details). Knowledge bases and therefore integrity constraints are kept separate, as we would expect in a multi-agent setting.

We are now ready to describe the operational behavior of the system, by giving the inference rules of the LAILA operators.

Definition 1 Down reflection formula.

$$\frac{A \overset{abd}{\vdash}_{\delta} G}{A \vdash_{\delta}^{A} \downarrow G}$$

where A is an agent, $A \in \{A_0, ..., A_1\}$, and G a goal or *query* (i.e., a list of literals). The goal $\downarrow G$ triggers a local abductive derivation for G.

Definition 2 Communication formula.

$$\frac{Y \vdash_{\delta_1}^{B} G \ \wedge \ B \cup X \overset{cons}{\vdash}_{\delta} \delta_1}{X \vdash_{\delta}^{B \cup X} Y > G}$$

where X and Y are two distinct agents, $X, Y \in \{A_0, ..., A_1\}, X \neq Y$.

Therefore, the communication formula $Y > G$ used by X to ask Y to solve G requires a consistency check within the bunch composed by X and by those agents that participated in the derivation of G. Clearly, this bunch will at least include X and Y.

Definition 3 Competitive formula.

$$\frac{A \vdash_{\delta}^{B} f}{A \vdash_{\delta}^{B} f; F}$$

$$\frac{A \vdash_{\delta}^{B} F}{A \vdash_{\delta}^{B} f; F}$$

[3] As in Section 3, we are abstracting away from the notion of integrity constraint “satisfaction”. Different procedures/semantics will adopt different such notions.

where A is an agent, $A \in \{A_0, ..., A_1\}$, f is a *LAILA factor*, F is a *LAILA expression* and B is a bunch of agents, $B \subseteq \{A_0, ..., A_1\}$. The competitive formula results in a non-deterministic choice of one inference rule between the two listed above.[4]

Definition 4 Collaborative formula.

$$\frac{A \vdash_{\delta'}^{B'} f \ \wedge \ A \vdash_{\delta''}^{B''} F \ \wedge \ B' \cup B'' \overset{cons}{\vdash}_{\delta} \ \delta' \cup \delta''}{A \vdash_{\delta}^{B' \cup B''} \ f \& F}$$

where A is an agent, $A \in \{A_0, ..., A_1\}$, f is a *LAILA factor*, F is a *LAILA expression* and B, B' and B'' are three possibly distinct bunches of agents, $B, B', B'' \subseteq \{A_0, ..., A_1\}$.

The semantics of goal formulas is described by the following inference rules[5,6]:

Definition 5 Goal formula.

$$\frac{A \vdash_{\delta'}^{B'} h \ \wedge \ A \vdash_{\delta''}^{B''} G}{A \vdash_{\delta' \cup \delta''}^{B' \cup B''} \ h, G}$$

$$\frac{\exists\, h : -G \ \in abm(A) \ \wedge \ A \vdash_{\delta}^{B} G}{A \vdash_{\delta}^{B} \ h}$$

$$\frac{}{A \vdash_{\emptyset}^{A} \ \mathtt{true}}$$

where A is an agent, $A \in \{A_0, ..., A_1\}$, h is a literal, G is a *LAILA body* and B, B' and B'' are bunches of agents, $B, B', B'' \subseteq \{A_0, ..., A_1\}$. With $abm(A)$ we mean A's $\mathcal{ABM}$.

Finally, while we refer to [27] for an operational semantics for the abductive derivation, we describe the semantics of the consistency check through the following inference rules:

Definition 6 Consistency check.

$$\frac{\forall A_i \in B \ \ A_i \overset{abd}{\vdash}_{\delta_i} \delta \ \wedge \ B \overset{cons}{\vdash}_{\delta'} \ \bigcup_{A_i \in B} \delta_i \ \wedge \ \delta \subset \ \bigcup_{A_j \in B} \delta_j}{B \overset{cons}{\vdash}_{\delta'} \ \delta}$$

$$\frac{\forall A_i \in B \ \ A_i \overset{abd}{\vdash}_{\delta} \delta}{B \overset{cons}{\vdash}_{\delta} \ \delta}$$

[4] As it is, the competitive operator is equivalent to a disjunction. The reason why we included it explicitly in LAILA is that we intend to extend it in order to let the choice be deterministic, e.g. by the introduction of a (partial) ordering function mapping from the cartesian product $\mathcal{A}^2 \times \mathcal{U}^2$ to a real range of numbers $R \subseteq \mathbb{R}$. Such function, used to choose among the two possible inference rules, would necessarily be domain dependent and does not seem to be of much interest for the general case. Therefore, we omitted it here in order to let the competitive operator have a simpler notation.

[5] Note that here, with an abuse of notation, we will consider h, G to be equivalent to (h, G).

[6] Here, the sequential operator (the comma) does not require any consistency check between the two sets of abducibles δ_1 and δ_2. This is due to the fact that at the level of LAILA programs, there is no notion of integrity constraints, that are only part of the $\mathcal{ARM}$ abductive programs.

where A_i, A_j are agents, $A_i, A_j \in \{A_0, ..., A_1\}$, h is a literal, G is a *LAILA body* and B is a bunch of agents, $B \subseteq \{A_0, ..., A_1\}$.

Therefore, the consistency check of δ is first performed individually by every single agent via an abductive derivation, which could result in an enlargement of δ (in particular, this happens if exists a δ_i such that $\delta \subset \delta_i$). In case no agent raises new hypotheses, i.e., for all $A_i \in B$, $\delta \equiv \delta_i$, the consistency check terminates, otherwise the union of the δ_i has to be checked again within the bunch.

Finally, we can introduce the definition of a successful top-down derivation.

Definition 7 Successful top-down derivation.

Let A be an agent and F a *LAILA body*. A *successful top-down derivation* for F in A's $\mathcal{ABM}$, with δ a set of abduced hypotheses and B the bunch of agents dynamically involved in it, can be traced in terms of a (possibly infinite) tree such that:

- The root node is labeled by $A \vdash_{\delta}^{B} F$, where A is an agent;
- The internal nodes are derived by using, backwards, the inference rules defined above;
- All the leaves are labeled by the empty formula, or represent a successful abductive/consistency derivation.

For instance, we have seen how a collaborative formula $A \vdash_{\delta}^{B} \; f \; \& \; F$ results in a ramification of the tree to three branches (sub-trees), one for demonstrating *f*, another one for F and a last one for the consistency check. Similarly, the communication formula produces two sub-trees, while the down reflection and the competitive formulas produce only one branch, and so on.

4.3. Let us hang the picture: an example of use of LAILA operators

Let us consider once more Example 1. We can now explicit the content of the three agents' $\mathcal{ABM}$ and $\mathcal{ARM}$[7], and see how agent a can obtain a nail and have the picture hung (see Figure 2). The meaning of the predicates and their arguments is the following: $\mathtt{achieve(Goal)}$ is the initial query that the agent is expected to trigger in order to achieve her $\mathtt{Goal}$ (in this case, the goal is to have the picture hung); $\mathtt{plan(Goal, Plan)}$ returns a $\mathtt{Plan}$ (sequence of actions) to achieve a given $\mathtt{Goal}$; $\mathtt{missing(Plan, Missing)}$ returns for a given $\mathtt{Plan}$ the set of resources that are required to execute the plan but that the agent does not hold; $\mathtt{max\_price(Item, Price)}$ represents the maximum price that an agent is willing to pay for a given $\mathtt{Item}$; $\mathtt{cost(Item, Price)}$ is the price that a seller agent is going to ask for a given $\mathtt{Item}$; $\mathtt{get(Buyer, Seller, Item)}$ represents a request of $\mathtt{Buyer}$ to obtain from $\mathtt{Seller}$ a certain $\mathtt{Item}$; $\mathtt{req/3}$ and $\mathtt{sell/4}$ are dialogue moves that may become part of the abduced set of hypotheses, as we have seen in Section 3, and in particular $\mathtt{req/3}$ represents a goal for the abductive reasoning module.

Now, let us suppose as we were saying that agent a runs the $\mathcal{ABM}$ query $?\,\mathtt{achieve(hung(picture))}$, and that a plan is given for such goal, that requires a missing resource (the nail). Following her LAILA program, at a certain point a will ask b to prove the goal $\mathtt{get(a, b, nail)}$. As a consequence, b will competitively issue a local abductive derivation for $\mathtt{req(a, b, nail)}$ and ask c to solve the goal $\mathtt{get(b, c, nail)}$. The set of hypotheses returned by her local $\mathcal{ARM}$, say δ_b, is $\delta_b = \{\mathtt{req(a, b, nail)}, \mathtt{sell(b, a, nail, 0)}\}$, that is also consistent with a's knowledge base. The set of hypotheses returned by c is $\{\mathtt{req(a, b, nail)}, \mathtt{req(b, c, nail)}, \mathtt{sell(c, b, nail, 5)}\}$,

[7] We are giving here only those bits of information that are strictly needed in order to follow the example.

a) $\mathcal{ABM}$

```
achieve(G) :-
    ↓ plan(G,P),
    ↓ missing(P,M),
   obtain(M).
obtain([R|Tail]) :-
    ↓ friend(X),
   X > get(a,X,R) &
   obtain(Tail).
obtain([]) ← true.
```

$\mathcal{ARM}$

```
friend(b).
have(picture).
have(hammer).
max_price(nail,8).
:-max_price(R,M),
   sell(_,a,R,M1),
   M1 > M.
```

b) $\mathcal{ABM}$

```
get(X,b,R) :-
    ↓ req(X,b,R) ;
   c > get(b,c,R).
```

$\mathcal{ARM}$

```
friend(a).
have(nail).
:-req(X,b,R), have(R),
   friend(X),
   not sell(b,X,R,0).
:-req(X,Y,R), Y ≠ b,
   sell(Y,b,R,P),
   P1 is P * 1.2,
   not sell(b,X,R,P1).
```

c) $\mathcal{ABM}$

```
get(X,c,R) :-
    ↓ req(X,c,R).
```

$\mathcal{ARM}$

```
have(nail).
cost(nail,5).
:-req(X,c,R), have(R),
   cost(R,P),
   not sell(c,X,R,P).
```

Figure 2. The agent knowledge bases

that in order to be consistent with b's integrity constraints will have to be enlarged and becomes $\delta'_c = \{\mathtt{req}(\mathtt{a},\mathtt{b},\mathtt{nail}), \mathtt{req}(\mathtt{b},\mathtt{c},\mathtt{nail}), \mathtt{sell}(\mathtt{c},\mathtt{b},\mathtt{nail},5), \mathtt{sell}(\mathtt{b},\mathtt{a},\mathtt{nail},6)\}$. Both δ_b and δ'_c are acceptable by a. As a conclusion, a will either accept b's nail for free or pay 6 for c's nail. As we already said before, LAILA does not specify which one will be returned by the competitive operator, leaving this choice open. This is the case where the ordering function mentioned in Definition 3 (Competitive formula) could be useful, setting a preference in the order in which b will return the δs to a.

4.4. An implementation of ALIAS

The current (prototypical) version of *ALIAS* has been implemented on the top of Jinni [50], a logic programming language extended with Linda-like primitives for concurrent programing. Although we are aware that a more customized, lower level implementation should be more efficient than the current one, we chose Jinni mainly for its high-level and declarative features that allowed us to achieve a rapid prototyping of the system. Unlike previous releases [7,2], both the reasoning and the coordination issues introduced by *ALIAS* are solved by means of logic clauses, leading to the development of a fully declarative design. *ALIAS* is platform-independent, since Jinni is written in Java.

As shown in Figure 3, an agent in *ALIAS* is implemented by two meta-interpreters, the first one supporting the execution of *LAILA programs*, and the second one representing the Abductive Reasoning Module. In this

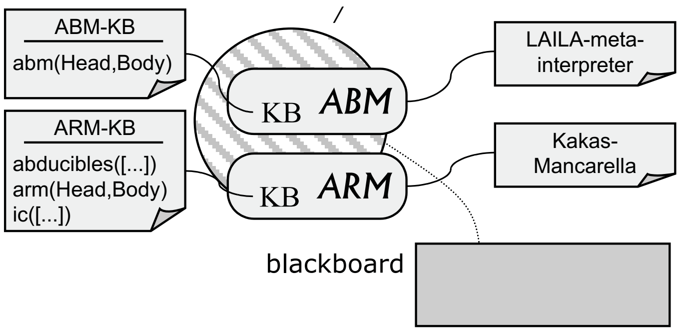

Figure 3. Prolog implementation of *ALIAS* agents' modules

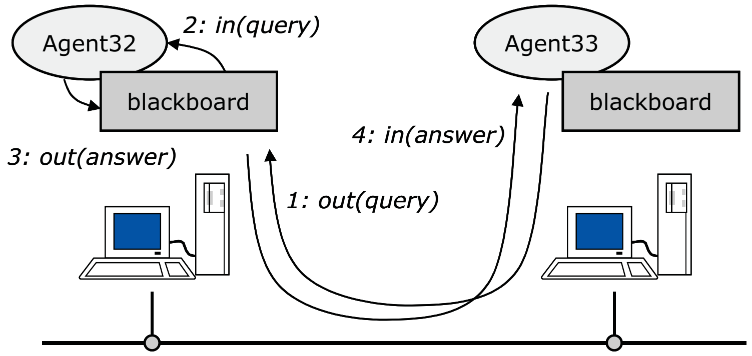

Figure 4. Networked *ALIAS* agents

implementation we adopted the Kakas-Mancarella proof-procedure for propositional abductive logic programs, but we plan to extend it to first order, non-ground abductive logic programs, using the ACLP framework [1] which also supports constraint handling as in constraint logic programming. The knowledge bases are composed of an abductive logic program for the $\mathcal{ARM}$, whose clauses are expressed in the form $\mathtt{arm}(\mathtt{Head}, \mathtt{Body})$, integrity constraints in the form $\mathtt{ic}(\mathtt{List})$ and abducibles as $\mathtt{abducibles}(\mathtt{List})$, and a *LAILA program*, composed of clauses in the form $\mathtt{abm}(\mathtt{Head}, \mathtt{Body})$.

ALIAS agents can run on different networked machines. Agents communicate through Linda-like primitives, by posting and consuming tuples in a shared blackboard space. To this purpose, each agent provides a blackboard as a communication space (see Figure 4), where it fetches the queries coming from the other agents, such as, for instance, the request for the demonstration of a given *LAILA query*.

The blackboard is also used to update the status of the goal demonstration. In particular, as the demonstration of a query is started, a special atom called *coordination tuple* is written on the blackboard by the agent that starts the demonstration. When an agent has some update about the query to publish, either because it produced a new answer or because it wants to change the status of the computation (e.g. it wants the computation to terminate), it consumes the tuple (through an 'in' operation), updates it and then posts the updated tuple in the blackboard again (through an 'out' operation). This mechanism ensures conflict avoidance among the threads. At a higher level, the *LAILA* meta-interpreter is a direct implementation of the operational semantics given.

5. Abductive dialogue and negotiation

In [47], building on [3], the authors introduce a framework where negotiation is carried out by means of a distributed abduction process. Here, the agents try to achieve their goals by means of plans that may require some resources that they miss. In that case, the negotiation process aims at obtaining such resources. The authors represent the beliefs of agents as logic programs and integrity constraints, as in abductive logic programming, with the dialogue performatives treated as abducibles. The integrity constraints play the role of dialogue/negotiation protocols. In this section we first will give a brief description of the basic elements that are needed to build a dialogue, then we will show how ALIAS can effectively support the collaborative/competitive negotiation framework.

5.1. A two-agent dialogue framework

The dialogue framework introduced in [47] is composed of two agents sharing a *language* and a *protocol*, and having each a *knowledge representation*, a *reasoning mechanism*, and a *planner* capable of producing plans (i.e., sets of actions) for a given goal. The approach is totally independent of the planner.

- The *language* is a set of performatives, e.g., `request`, used to request a resource, such as an information, or a physical tool, `accept` and `refuse`, used to reply affirmatively or negatively to a request or proposal, and finally `challenge`, `justify`, and `promise`, used for proper negotiation, allowing the agents to share more information than just that attached to a request and a reply. An example of dialogue, adapted from [42], is the following, where agent a wants to hang a picture on the wall and plans to do it by using a nail and a hammer. Unfortunately, a does not have a nail and needs to negotiate with b to try to obtain it.

  dialogue 1
  $\mathtt{tell}(\mathtt{a}, \mathtt{b}, \mathtt{request}(\mathtt{give}(\mathtt{nail})), 1)$
  $\mathtt{tell}(\mathtt{b}, \mathtt{a}, \mathtt{refuse}(\mathtt{request}(\mathtt{give}(\mathtt{nail}))), 2)$
  $\mathtt{tell}(\mathtt{a}, \mathtt{b}, \mathtt{challenge}(\mathtt{refuse}(\mathtt{request}(\mathtt{give}(\mathtt{nail})))), 3)$
  $\mathtt{tell}(\mathtt{b}, \mathtt{a}, \mathtt{justify}(\mathtt{refuse}(\mathtt{request}(\mathtt{give}(\mathtt{nail}))), \{\mathtt{not\,have}(\mathtt{nail})\}), 4)$

  In this example, a uses the `challenge` performative to ask the reason why b refuses to give the nail: agent b says that she does not have it. The dialogue could go on with other moves, depending on the agent abductive logic programs that produce them, as we will see next.
- The *protocol* used for negotiation is a set of constraints that must not be violated, and that describe the dialogue, the possible moves, and its termination.
- The *agent knowledge* is represented by the tuple $< \mathcal{B}, \mathcal{R}, \mathcal{I}, \mathcal{D}, \mathcal{G} >$, containing respectively $\mathcal{B}$*eliefs*, $\mathcal{R}$*esources*, $\mathcal{I}$*ntentions*, past $\mathcal{D}$*ialogue* moves, and of course $\mathcal{G}$*oal(s)*. An intention is a plan intended to be used to achieve a goal.
- The *reasoning* mechanism is that of *abduction*, and is used to generate a dialogue move starting from a knowledge base, including the past dialogue. The abductive logic program used is contained in the agent knowledge base, namely in $\mathcal{B}$. The proof-procedure adopted is not necessarily the same for both agents, but for the sake of clarity we will write the ICs as condition-action rules, as in the case of the if-and-only-if (IFF) abductive

proof-procedure [20]. This will put a stress on the dialogue move to be uttered (the 'action') in case certain 'conditions' (including a previous move from the counterpart) hold.

To start up, let us suppose that all agents are provided with the same set of integrity constraints. Let us write only one of them as an example:

```
% IC 1:  accept a request
```

$i\_am(X) \wedge tell(Agent, X, request(give(R))) \wedge have(R) \wedge \sim need(R)$
$\Rightarrow tell(X, Agent, accept(request(give(R))))$

This rule models the behavior of a collaborative agent, that gives a resource that she has and does not need, in case she is request to give it. It is possible to rewrite such constraint and express it as a denial, as following:

$:- \mathtt{i\_am(X)},\ \mathtt{tell(Agent, X, request(give(R)))},\ \mathtt{have(R)},\ \mathtt{not\ need(R)},$
$\quad \mathtt{not\ tell(X, Agent, accept(request(give(R))))}$

Of course, more integrity constraints are needed to produce the example of dialogue 1. Also, we will not define here the exact semantics of predicates such as `need` and `have`, whose rough meaning we will consider intuitive. Those interested can refer to [47] for details.

Before moving on to the next part, we would like to stress a couple of points. First of all, the negotiation process, as in [47], is driven by the need for a specific resource for a specific plan. Second, the dialogue is intended to be between (only) two agents. Bearing such considerations in mind, let us see how this framework could be used in a multi-agent environment, where agents could ask for multiple resources and try different plans, if provided with an appropriate support for collaboration and competition.

5.2. Negotiating ALIAS agents

The problems addressed by Multi-Agent Systems extend in many respects the situation where two agents negotiate to make a plan feasible, involving in general more than two agents, possibly provided with more than one plan for each goal, while the resources needed for each plan are not necessarily all owned by the same agent. Moreover, it is reasonable not to assume that the sets of integrity constraints modeling the dialogue are the same for all the agents[8], this last possibility leading to the risk of non-terminating dialogues, and other unpleasant effects.

In this setting, for a given goal to achieve, an agent will have to manage multiple dialogues, either in 'competition', if referring to alternative plans, or in 'collaboration', if aimed at obtaining all the resources needed to make a certain plan feasible. If we want to frame the dialogue in an argumentative abduction based system, it is clear that we need a mechanism to express and manage collaboration and competition between autonomous abductive reasoning systems. Here, consistency enforcement can be needed in order to guarantee that certain dialogue rules are not violated, and that some desired properties still hold, despite the possible difference in the agent dialogue rules and constraints. In such scenario, ALIAS comes in hand since it provides collaboration and competition patterns within a solid formal theory, that can be used to prove if such properties will hold or not.

[8] The assumption that all the agent constraints are the same, would also lead to computationally hard problems in terms of verification, and seems quite unlikely to be easily realizable.

Example 1, that we have been considering throughout the paper, can be considered a first rough sketch of such application. Let us consider again a's knowledge base, as reported in Section 4. The predicate `obtain` was in charge of trying to get all the missing resources, needed for a certain plan. To that purpose, several communication formulas of the kind $\mathtt{X} > \mathtt{get}(\mathtt{a}, \mathtt{X}, \mathtt{R})$ were issued in collaboration, since all the resources (and not only part of them) must be obtained in order to make a plan executable.

The case of alternative multiple plans can be easily implemented via a competitive operator, with a predicate `pick_one(PlanList)` defined as follows:

```
pick_one([Plan|Tail]) :-
(
    ↓missing(Plan, Missing),
    obtain(Missing)
) ;
(
    non_empty(Tail),
    pick_one(Tail)
).
```

The role of consistency checking has already been exemplified in the case of the communication formula, when a asks for a nail but refuses to spend more than 8 to buy it. But there is another, more general, use of the constraints that becomes important in an open, Multi-Agent, environment: that is, the implementation of the dialogue protocol and the verification that it is not being violated.

In the first part of this section we assumed that two agents that are having a dialogue share the same set of integrity constraints. This is because in that way we are able to prove, for instance, that a certain kind of dialogue will terminate [47]. When an agent is put in an open context, this assumption cannot be made any more. Each agents must check within herself that, no mind if the other agents have or not the same set of constraints, at least they produce compatible hypotheses with respect to their own programs, without violating the dialogue protocol. For instance, one of the properties of a dialogue as described in [47] is that of termination in a finite number of steps. This is possible because there are some constraints that rule the dialogue (e.g., it is not possible to tell the same thing twice at different times). This kind of constraints can be easily checked within a bunch of agents, that will accept the other agents' moves, i.e., abducibles, as soon as they do not violate the protocol.

6. Related Work

The abductive coordination protocol introduced in Section 3 is grounded on a query-based proof procedure originally defined by Eshghi and Kowalski [16] that recovers negation as default through abduction, and further extended by Kakas and Mancarella in [27] in order to manipulate arbitrary abducibles and integrity constraints. In [6] this proof procedure has been further extended to consider a second kind of negation, and proved sound and weakly complete with respect to a three-valued abductive semantics. In [7] the authors describe the implementation of an extension to the Kakas-Mancarella, namely the DAA (Distributed Abduction Algorithm), to the multi-agent case. The DAA works on statically defined bunches of agents and does not allow to combine and coordinate them in different ways, as LAILA does instead.

Other relevant literature exists on proof procedure for extended and abductive logic programs. Satoh and Iwayama [48] proposed a query-based proof procedure which improves Kakas and Mancarella's one, and addresses completeness with generalized stable model semantics [26], but at the price of a more complex integrity checking. Another proof procedure has been proposed by Denecker and De Schreye in [13,14], by extending the SLDNF resolution to the case of abduction. Other work has been done within the ACLP project [24,1] in order to extend the Kakas-Mancarella proof procedure towards the treatment of non-ground abductive goals. The system, ACLP, combines abductive reasoning and constraint solving by integrating the framework of Abductive Logic Programming (ALP) with that of Constraint Logic Programming (CLP). The ACLP framework has been implemented on top of the ECL^{i}PS^e language [15] as a meta-interpreter using explicitly its low-level constraint solver that handles constraints over finite domains.

A parallel implementation of the proof procedure described in [27] has been proposed by Kakas and Papadopoulos in [28]. Various forms of parallelism (OR and dependent AND parallelism) are exploited to parallelize deductive logic programming in the framework of abductive logic programming. This parallel model for abduction is then mapped into the Andorra model. However, the focus of [28] is on a parallel implementation of a single abductive logic programming computation rather than on the multi-agent case.

A notion of abduction for the family of Concurrent Constraint (cc) languages was proposed by Codognet and Saraswat in [11]. The key idea of the cc framework is to use constraints (and a global set of constraints called the constraint *store*) to synchronize concurrent logic computations. Multiple agents run concurrently and synchronize by adding a new constraint to the *store* (*tell* operation) or checking whether a given constraint is entailed by the *store* (*ask* operation). An asking agent may block and suspend in information in the *store* is insufficient to determine whether the constraint is entailed or not. In (cc) computations this corresponds to a situation where only partial information in the *store* is available. In [11], thus, abduction is used to generate some hypothetical constraint that would lead some *ask* operations to be satisfied and therefore resume some agent computations. With respect to our work, they consider constraints instead of atoms as hypotheses, and the consistency ckeck is delayed, during the computation, till some other agents will add to the store a possibly inconsistent constraint.

Relevant work concerning logic agents is contained in [31,32,34], where the authors analyze differences between rational and reactive architectures. Rational agents have both beliefs and goals. A reactive agent, instead, has neither an explicit knowledge base, nor an explicit representation of goals. In [32], the authors propose a uniform agent architecture that aims to capture both rationality and reactivity. In a rational agent, goals are represented explicitly and knowledge is represented as goal reduction rules of kind *conclusion if condition*. Furthermore, condition-action rules, typical of reactive agents, can be interpreted as integrity constraints. The two kinds of behaviour are then unified in a single logic-based architecture which is grounded on a proof procedure where integrity constraints and queries are treated identically [36]. This proof procedure is general, and has been shown to unify abductive logic programming, constraint logic programming and semantic query optimization.

According to [31,32], our logic agents are to be considered rational agents, since their main capability is to answer queries (i.e., goals) from the environment (i.e., other arguing agents) or from high-level goals. Rather than on agent's behaviour, though, the emphasis in our work is on the coordination of abductive reasoning inside a multi-agent system. In this respect, our work can be considered complementary to [31,32].

Related work in the area of negotiation is huge, and difficult to synthesize in a few lines; a good introduction to it is in [45].

The issue of automated negotiation using an argumentation based approach has been discussed in a number of works. In [49,41], the authors adopt a modal logic approach and focus on the mental states of agents that move towards an agreement, and on the way to persuade a counterpart in order to foster cooperation. The execution is intertwined with the dialogue, allowing the agents to express threats, promises of rewards, etc. and to update in consequence of that a mental model of the other partners. Such approach, focused on persuasion, leads to different results, being somewhat more descriptive, not aimed at determining specific properties of the negotiation protocol.

We have already described how our work relates to [3] in Section 5. In [42] the authors adopt a formal system of argumentation in order to get the support necessary to build agents which negotiate to reach an agreement. Their approach is also modal logic based, and is more descriptive than procedural. In order to develop an agent architecture which has a clear link between its specification and its implementation, they make use of multi-context systems [23]. Our work can also be considered a particular instance of multi-context systems, where $\mathcal{ABM}$ and $\mathcal{ARM}$ are two *units*, each having its own *logics* and *theories*, and where the LAILA *down-reflection* operator is a *bridge rule* between one agent's $\mathcal{ABM}$ and $\mathcal{ARM}$, and the *communication* operator is the *bridge rule* between the $\mathcal{ABM}$s of two different agent.

An approach for expressing communication for abductive logic agents is proposed in [12], where the primitives *tell* and *ask* are treated as abducibles, similarly to what we do in ALIAS, and framed in an "observe-think-act" cycle [33]. The main difference between this approach and ours is that we provide operators to ensure the consistency of a set of hypotheses with respect to the integrity constraints of a set of agents, while in [12] the abductive reasoning process is individually performed. Also, we opted for a strong separation between a communication / coordination meta-language, and the abductive knowledge base contained in the $\mathcal{ARM}$.

The issue of combining agents into groups has been discussed in [5], where the authors present the GroupLog coordination language for collective agent based systems. GroupLog defines extensions to the Extended Horn Clause language (EHC), that are supported at two levels: L_1 defines agents as program units and L_2 defines groups of agents. Agents are structured in classes. While in L_1 it is possible to define the agent interface to other agents, in L_2 it is possible to coordinate the behavior of a group of agents of the same class. To that extent, a group of agents is considered itself a meta-agent, allowing the modular composition of the group notion. In ALIAS, coordination within a group of agents is not achieved through the notion of group but is built in the semantics of the consistency operators: group coordination in ALIAS is needed to ensure the consistency of a given set of abduced hypotheses and is transparent to the agent, while the behavior of the individual agents can be expressed in the $\mathcal{ABM}$.

Other work on the representation of complex actions is that of the language CONGOLOG [37], a programming language whose formal semantics is based on the situation calculus. CONGOLOG supports the specification of agent behaviors at a very high level of abstraction. While CONGOLOG has been thought to overcome the lack of formal semantics of the KQML definition [9], it does not aim at separating communication issues from agent reasoning. Our agent architecture, instead, with a separation introduced between a reasoning module, the $\mathcal{ARM}$, and a communication/behavior module, the $\mathcal{ABM}$, allows for a higher level programming style. It is less general, in that it considers a specific class of agents (abductive logic agents). However, we are currently working to extend and generalize the system towards other forms of reasoning.

Among the other agent languages, we would cite Agent-0 [46] and Concurrent MetateM [19], where the focus is on the specification of the agent's internal behavior: while the former has single communication acts built-in, the latter does not model interactions between agents at all. More recently, much work towards standardization has been done with KQML [17], a widely used specification of a language for inter-agent communication, and

FIPA ACL [18].

Relevant work about the combination of multiple knowledge bases is that of [4], where the authors define the concept of combining the knowledge present in a set of knowledge bases, and present algorithms to maximally combine them, so that their combination is consistent with the integrity constraints associated with the knowledge base sets. In [29] Konieczny investigates the logical properties of knowledge base combination operators proposed in the literature, and discusses the difference between merging knowledge bases and combining them. BReLS [38] is another system for the knowledge bases integration; the authors distinguish among three different approaches to the problem: belief revision, merging and update, and present a framework that integrates them.

In general, work on knowledge merging or combining is based on the definition of ordering and preference operators, aimed at defining a maximal amount of knowledge that can be put together without violating some integrity constraints. Our point is the other way round, as we aim at keeping separate the agent knowledge bases. In a way, we approach the issue of cooperation rather than integration, in that we adopt the agent model in order to allow different and possibly contradicting knowledge bases to coexist and to participate to some common reasoning activity, without preferring one over the one. In ALIAS, agent coordination is query-dependent, and it is left up to the querying agent to decide the way to combine the knowledge coming from its partners, whether in a competitive or collaborative way.

7. Conclusions and Future Work

In this paper we presented ALIAS, an agent architecture based on abductive logic agents, and we focused on coordination between ALIAS agents. To this end we provided an infrastructure where agents are grouped in bunches and can interact following different patterns. The infrastructure is given by the LAILA language, by means of which the programmer can write abductive multi-agent applications in a declarative way, and independently of the particular abductive mechanism that each agent may adopt. The underlying use of abductive reasoning makes the system particularly suited for solving problems in domains where knowledge is incomplete, as it is the case of negotiation between agents. In this setting, reasonable models and protocols should assume that agents lack knowledge about other agents.

Building on a framework designed for two-agent abduction-based negotiation [47], we have shown how ALIAS allows to add interesting new features, providing a suitable support for multiple negotiation processes. In particular, given an agent, a goal, and n alternative plans to achieve it, the agent will adopt a *collaborative* interaction pattern in order to collect all the missing resources for the execution of a chosen plan and make it feasible, and the agent will adopt a *competitive* pattern to try to select one feasible plan, out of the n alternative plans, by exploring them concurrently.

We implemented ALIAS on top of Jinni [50], a logic language extended with Linda-like primitives, thus following a fully declarative approach. Although in this version of ALIAS we implemented the Kakas-Mancarella abductive proof procedure within each agent, the system architecture is independent of any concrete algorithm. For instance, we tested a simple two-agent negotiation dialogue with a different algorithm (namely, the IFF proof-procedure [20]).

There are a number of issues that we intend to address in future work. We want to extend the implementation of ALIAS to first-order non ground abductive programs; in doing that we intend to merge in ALIAS the ACLP framework [1]. Also, we intend to explore various negotiation protocols, each described by a different abductive logic program. In this respect, we plan to analyze the properties of the system, e.g., termination, convergence, etc., when putting together different agent varieties, i.e., abductive logic programs that implement different negotiation policies. Moreover, the language used in the current agent dialogues is a set of point-to-point performatives: we intend to generalize the negotiation framework by introducing broadcast/multicast performatives. A possible application domain of this extension could be that of automated auctions. Finally, while abduction is a form of reasoning that fits well into the intelligent multi-agent systems requirements, we feel that ALIAS could be further improved, and its scope extended, by introducing other forms of inference, such as, for instance, inductive reasoning. In a negotiation setting, this could lead to a dynamic improvement of the agents' logic programs and of their interaction patterns, based on past negotiation experiences.

References

[1] *ACLP: Abductive Constraint Logic Programming*. http://www.cs.ucy.ac.cy/aclp/.

[2] *ALIAS: Abductive LogIc Agent System*. http://www.lia.deis.unibo.it/Software/ALIAS/.

[3] L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue and negotiation. In Horn, W., ed., *Proc. ECAI00, 14th European Conference on Artificial Intelligence*, Berlin, Germany, 2000. IOS Press.

[4] C. Baral, S. Kraus, and J. Minker. Combining Multiple Knowledge Bases. In *IEEE Trans. Knowledge and Data Engineering*, 3(2), pages 208–220, June 1991.

[5] F. Barbosa, J. C. Cunha. A Coordination Language for Collective Agent Based Systems: GroupLog. In *Proc. ACM SAC'00*, Villa Olmo, Como, Italy, 2000.

[6] A. Brogi, E. Lamma, P. Mancarella, and P. Mello. A Unifying View for Logic Programming with Non-Monotonic Reasoning. In *Theoretical Computer Science*, Vol. 184, 1–49, North Holland, 1997.

[7] A. Ciampolini, E. Lamma, P. Mello and P. Torroni. An Implementation for Abductive Logic Agents. *Proceedings AI*IA99*, Pitagora Editore, Bologna, Italy, 1999.

[8] K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, eds., *Logic and Databases*. Plenum Press, New York, 1978.

[9] P. R. Chen and H. J. Levesque. Communicative actions for artificial agents. In V. Lesser and L. Gasser, eds., *Proc. 1st ICMAS*, San Francisco, CA, 1995. AAAI Press/MIT Press.

[10] P. T. Cox and T. Pietrzykowski. Causes for events: Their computation and applications. In *Proc. CADE-86*, 608, 1986.

[11] P. Codognet and V. Saraswat. Abduction in Concurrent Constraint Programming. INRIA-Roquencourt Technical Report, March 1992.

[12] P. Dell'Acqua, F. Sadri, and F. Toni. Communicating Agents. In *Proc. ICLP Workshop on Multi-Agent Systems*, Las Cruces, NM, 1999.

[13] M. Denecker and D. De Schreye SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, Elsevier, 1998.

[14] M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In *Proc. International Logic Programming Symposium ILPS93*, pages 147–163. The MIT Press, 1993.

[15] ECL^{i}PS^e User Manual Release 3.3. ECRC, 1992.

[16] K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proc. 6th International Conference on Logic Programming*, 234. MIT Press, 1989.

[17] T. Finin, Y. Labrou and J. Mayfield, KQML as and agent communication language. Invited chapter in Jeff Bradshaw (Ed.), *Software Agents*, MIT Press, Cambridge, 1997.

[18] *FIPA (Foundation for Intelligent Physical Agents)*. http://www.fipa.org/.

[19] M. Fisher, A survey of Concurrent METATEM – the language and its applications. In *Proc. 1st Int. Conf. Temporal Logic*. LNAI, pages 480–505. Springer-Verlag 1994.

[20] Fung, T. H., Kowalski, R. A. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 1997.

[21] D. Gelernter, *Generative Communication in Linda*. *ACM Toplas*, 7(1):80–112, 1985.

[22] D. Gelernter and N. Carriero, Coordination languages and their significance. In *Communications of the ACM*, 35(2), pages 97–107, 1992.

[23] F. Giunchiglia and L. Serafini, Multilanguage hierarchical logics (or: How we can do without modal logics). *Artificial Intelligence*, 65, pages 29–70, 1994.

[24] A. C. Kakas, ACLP: integrating abduction and constraint solving. *Proc. NMR'00*, Breckenridge, CO, 2000.

[25] A. C. Kakas, R. A. Kowalski, and F. Toni, The role of abduction in logic programming. *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, pages 235-324, D.M. Gabbay, C.J. Hogger and J.A. Robinson eds., Oxford University Press (1998)

[26] A. C. Kakas and P. Mancarella, Generalized stable models: a semantics for abduction. In *Proc. 9th European Conference on Artificial Intelligence*. Pitman Pub., 1990.

[27] A. C. Kakas and P. Mancarella, On the relation between Truth Maintenance and Abduction. In *Proc. PRICAI90*, 1990.

[28] A. C. Kakas, and G. A. Papadopoulos, Parallel Abduction in Logic Programming. In *Proc.Int. Symposium on Paralle Symbolic Computation PASCO94*. pages 214-224, World Scientific Pub., 1994.

[29] S. Konieczny, On the Difference between Merging Knowledge Bases and Combining them. In *Proc. KR'00* Breckenridge, CO, 2000.

[30] R. A. Kowalski, Problems and promises of computational logic. In *Proc. Symposium on Computational Logic*, 1-36. Springer-Verlag, Nov. 1990.

[31] R.A. Kowalski, Using metalogic to reconcile reactive with rational agents. In *Meta-Logics and Logic Programming*, K. Apt and F. Turini (eds.), MIT Press, 1995.

[32] R.A. Kowalski, and F. Sadri, Towards a unified agent architecture that combines rationality with reactivity. In *Proc. International Workshop on Logic in Databases*, San Miniato, Italy, Springer-Verlag, LNCS 1154, 1996.

[33] R.A. Kowalski, and F. Sadri, From Logic Programming to Multi-Agent Systems. In *Annals of Mathematics and Artificial Intelligence*, 1999.

[34] R.A. Kowalski, F. Sadri, and F. Toni, An agent architecture that combines backward and forward reasoning. In *Proc. CADE-15 Workshop on Strategies in Automated Deduction*, B. Gramlich and F. Pfenning eds., pages 49-56, 1998.

[35] R.A. Kowalski, F. Sadri, and F. Toni, Logic-based Multi-agent Systems. In *Compulog Newsletters*, December 1998. `http://www.cs.ucy.ac.cy/compulog/dec98update/mainframe.htm`

[36] R.A. Kowalski, and F. Toni, and G. Wetzel, Towards a declarative and efficient glass-box clp language. In *Proc. of Logic Programming Workshop WLP94*, N. Fuchs and G. Gottlob (eds.), 1994.

[37] Y. Lesprance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl, Foundations of a Logical Approach to Agent Programming. In *Intelligent Agents Volume II – Proc. ATAL'95*, M. Wooldridge, J. P. Mller, M. Tambe (eds.), Springer-Verlag, LNAI, 1996.

[38] P. Liberatore and M. Shaerf, BReLS: A System for the Integration od Knowledge Bases. In *Proc. KR'00* Breckenridge, CO, 2000.

[39] J. W. Lloyd. Foundations of Logic Programming. Springer, 1987.

[40] N. R. Jennings and M. J. Wooldridge, eds., *Agent Technology*. Springer-Verlag, 1998.

[41] Kraus, S., Sycara, K., and Evenchik, A. Reaching agreements through argumentation; a logical model and implementation. In *Artificial Intelligence* 104, 1–69. Elsevier, 1998.

[42] S. Parsons, C. Sierra, and N. R. Jennings, Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3), 261-292, 1998.

[43] J. Pearl, Embracing causality in formal reasoning. *Proc. National Conference on Artificial Intelligence*, Seattle, WA, pages 369-373, 1987.

[44] D. L. Poole, A logical framework for default reasoning. *Artificial Intelligence*, 36:27. Elsevier, 1988.

[45] Rosenschein, J. S., and Zlotkin, G. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers.* MIT Press, Cambridge, Massachusetts, 1994

[46] Y. Shoham, Agent-oriented programming. In *Artificial Intelligence Computing*, 60(1)51–92. Elsevier, 1993.

[47] F. Sadri, F. Toni, and P. Torroni. Logic agents, dialogues and negotiation: an abductive approach. In Proc. AISB'01 Convention, York, UK, 2001.

[48] K. Satoh, and N. Iwayama, A Query Evaluation Method for Abductive Logic Programming. In K. Apt, editor, *Proc. Int. Joint Conf. and Symp. on Logic Programming*, pages 671–685. The MIT Press, 1992.

[49] Sycara, K. P. Argumentation: Planning Other Agents' Plans. In *Proc. IJCAI 1989*, pag. 517–523

[50] P. Tarau, Jinni: Intelligent Mobile Agent Programming at the Intersection of Java and Prolog. *Proc. PAAM*, London, 1999.

[51] F. Toni, A semantics for the Kakas-Mancarella procedure for abductive logic programming. In *Proc. GULP'95*, M. Alpuente and M. I. Sessa, eds., pages 231-242 (1995).

[52] M. Wooldridge and N. R. Jennings, Intelligent agents: theory and practice. In *The Knowledge Engineering Review*, 10(2)115–152, 1995.