Preferring and Updating in Abductive Multi-Agent Systems

Pierangelo Dell'Acqua* Luís Moniz Pereira†

* Department of Science and Technology, Campus Norrköping,
Linköping University, Norrköping, Sweden
pier@itn.liu.se

† Centro de Inteligência Artificial - CENTRIA
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract. We present a logical framework and the declarative semantics of a multi-agent system in which each agent can communicate with and update other agents, can react to the environment, is able to prefer, whether beliefs or reactions, when several alternatives are possible, and is able to abduce hypotheses to explain observations. The knowledge state of an agent is represented by an updatable prioritized abductive logic program, in which priorities among rules can be expressed to allow the agent to prefer. We sketch two examples to illustrate how our approach functions, including how to prefer abducibles to tackle the problem of multiple hypotheses and how to perform the interplay between planning and acting.

We argue that the theory of the type of agents considered is a rich evolvable basis, and suitable for engineering configurable, dynamic, self-organizing and self-evolving agent societies.

1 Introduction

We present a logical formalization of a framework for multi-agent systems and we define its semantics. In this framework, we embed a flexible and powerful kind of agent. In fact, these agents are rational, reactive, abductive, able to prefer and they can update the knowledge base of other agents (including their own).

The knowledge state of each agent is represented by an an abductive logic program in which it is possible to express rules, integrity constraints, active rules and priorities among rules. This allows the agents to reason, to react to the environment, to prefer among several alternatives, to update both beliefs and reactions, and to abduce hypotheses to explain observations. We present a declarative semantics of this kind of agent.

These agents are then embedded into a multi-agent system in such a way that the only form of interaction among them is based on the notions of project and update. A project of the form $\alpha{:}C$ of an agent β denotes the intention of β of proposing to update the theory of an agent α with C. Correspondingly, an update of the form $\beta{\div}C$ in the theory of α denotes the intention of β to update the current theory of α with C. It is then up to α whether or not to accept that update. For example, if α trusts β and therefore α

is willing to accept it, then α has to update its theory with C. The new information may contradict what α believes and, if so, the new believed information will override what is currently believed by α. β can also propose an update to itself by issuing an internal project $\beta{:}C$.

The semantics of the multi-agent system provides a logical description of the agent interactions. The definition of the semantics depends also on the goal that each agent has to prove at a certain moment in time. In fact, by proving a goal the agent may abduce hypotheses that explain that goal, and in turn these hypotheses may trigger reactive rules, and so on. Hypotheses abduced in proving a goal G are not permanent knowledge, rather they only hold during the proof of G. To make them permanent knowledge, an agent can issue an internal project and update its own knowledge base with those hypotheses.

The engineering of our agent society is based exclusively on the notions of projects and updates to model the agent interactions since we aim at building autonomous and distributed agent systems. This form of interaction is powerful and flexible, and a number of communication protocols can be built based on it. However, the main contribution of the paper is to assemble all the ingredients of the agent architecture needed to engineer configurable, dynamic agent societies where the agents can self-organize themselves with respect to their goals, and self-evolve. In this way, the overall "emerging" structure will be flexible and dynamic: each agent will have its own representation of its organization which, furthermore, is updatable by preferring.

The remainder of the paper is structured as follows. Section 2 presents the logical framework. Section 3 introduces our conception of abductive agent and multi-agent system. The declarative semantics of abductive agents and multi-agent systems is presented in Sections 4 and 5, respectively. Two examples to illustrate how the approach functions, including how to prefer abducibles to tackle the problem of multiple hypotheses and how to perform the interplay between planning and acting, are shown in Sections 6 and 7. Finally, Section 8, Conclusion and Future Work, argues that the theory of the type of agents considered is a rich evolvable basis, and suitable for engineering configurable, dynamic, self-organizing and self-evolving agent societies, and discusses future work.

2 Logic Programming Framework

Typically, an agent can hold positive and negative information, and it can update its own knowledge with respect to the new incoming information. Thus the language of an agent should be expressive enough to represent both positive and negative information. In order to represent negative information in logic programs, we need a language that allows default negation $not\, A$ not only in premises of clauses but also in their heads[1]. We call such programs generalized logic programs. It is convenient to syntactically represent generalized logic programs as propositional Horn theories. In particular, we represent default negation $not\, A$ as a standard propositional variable.

[1] For further motivation and intuitive reading of logic programs with default negations in the heads see [2].

Propositional variables whose names do not begin with "*not*" and do not contain the symbols ":" and "÷" are called *objective atoms*. Propositional variables of the form $not\ A$ are called *default atoms*. Objective atoms and default atoms are generically called *atoms*.

Propositional variables of the form $\alpha{:}C$ (where C is defined below) are called *projects*. $\alpha{:}C$ denotes the intention (of some agent β) of proposing the updating the theory of agent α with C. Projects can be negated. A *negated project* of the form $not\ \alpha{:}C$ denotes the intention of the agent of not proposing the updating of the theory of agent α with C.

Propositional variables of the form $\beta \div C$ are called *updates*. $\beta \div C$ denotes an update that has been proposed by β of the current theory (of some agent α) with C. Updates can be negated. A *negated update* of the form $not\ \beta{\div}C$ in the theory of an agent α indicates that agent β does not have the intention to update the theory of agent α with C.

Definition 1. Let $\mathcal{K}$ be a set of propositional variables consisting of objective atoms and projects such that *false* $\notin \mathcal{K}$. The propositional language $\mathcal{L}_\mathcal{K}$ generated by $\mathcal{K}$ is the language which consists of the following set of propositional variables:

$$\begin{aligned}\mathcal{L}_\mathcal{K} \ = \ & \mathcal{K} \cup \{\mathit{false}\} \cup \ \{not\ A \mid \text{for every objective atom } A \in \mathcal{K}\} \\ & \cup \{not\ \alpha{:}C, \alpha{\div}C, not\ \alpha{\div}C \mid \text{for every project } \alpha{:}C \in \mathcal{K}\}.\end{aligned}$$

Definition 2 (Generalized rule). A generalized rule in the language $\mathcal{L}_\mathcal{K}$ is a rule of the form: $L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ $(n \geq 0)$, where L_0 (with $L_0 \neq \mathit{false}$) is an atom and every L_i $(1 \leq i \leq n)$ is an atom, an update or a negated update from $\mathcal{L}_\mathcal{K}$.

Note that, according to the above definition, only objective atoms and default atoms can occur in the head of generalized rules.

We use the following convention. Given a generalized rule r of the form $L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$, we use $head(r)$ to indicate L_0, $body(r)$ to indicate the conjunction $L_1 \wedge \ldots \wedge L_n$, $body_{pos}(r)$ to indicate the conjunction of all objective atoms and updates in $body(r)$, and $body_{neg}(r)$ to indicate the conjunction of all default atoms and negated updates in $body(r)$. Whenever L is of the form $not\ A$, $not\ L$ stands for the atom A.

Definition 3 (Integrity constraint). An integrity constraint in the language $\mathcal{L}_\mathcal{K}$ is a rule of the form: $\mathit{false} \leftarrow L_1 \wedge \ldots \wedge L_n \wedge Z_1 \wedge \ldots \wedge Z_m$ $(n \geq 0, m \geq 0)$, where every L_i $(1 \leq i \leq n)$ is an atom, an update or a negated update, and every Z_j $(1 \leq j \leq m)$ is a project or a negated project from $\mathcal{L}_\mathcal{K}$.

Integrity constraints are rules that enforce some condition over the state, and therefore always take the form of denials, without loss of generality, in a 2-valued semantics. Note that generalized rules are distinct from integrity constraints and should not be reduced to them. In fact, in generalized rules it is of crucial importance, when updating which atom occurs in the head.

Definition 4 (Query). A query Q in the language $\mathcal{L}_\mathcal{K}$ takes the form: $?\!-\ L_1 \wedge \ldots \wedge L_n$ $(n \geq 1)$, where every L_i $(1 \leq i \leq n)$ is an atom, an update or a negated update from $\mathcal{L}_\mathcal{K}$.

The following definition introduces rules that are evaluated bottom-up. To emphasize this aspect, we employ a different notation for them.

Definition 5 (Active rule). An active rule in the language $\mathcal{L}_{\mathcal{K}}$ is a rule of the form: $L_1 \wedge \ldots \wedge L_n \Rightarrow Z$ ($n \geq 0$), where every L_i ($1 \leq i \leq n$) is an atom, an update or a negated update, and Z is a project or a negated project from $\mathcal{L}_{\mathcal{K}}$.

We use the following convention: given an active rule r of the form $L_1 \wedge \ldots \wedge L_n \Rightarrow Z$, we use $head(r)$ to indicate Z, and $body(r)$ to indicate $L_1 \wedge \ldots \wedge L_n$.

Active rules are rules that can modify the current state, to produce a new state, when triggered. If the body $L_1 \wedge \ldots \wedge L_n$ of the active rule is satisfied, then the project (fluent) Z can be selected and executed. The head of an active rule must be a project that is either internal or external. An *internal project* operates on the state of the agent itself (self-update), e.g., if an agent gets an observation, then it updates its knowledge, or if some conditions are met, then it executes some goal. *External projects* instead are performed on the environment, e.g., when an agent wants to update the theory of another agent. A negated project that occurs in the head of an active rule denotes the intention (of some agent) not to perform that project at the current state.

We assume that for every project $\alpha{:}C$ in $\mathcal{K}$, C is either a generalized rule, an integrity constraint, an active rule or a query. Thus, a project can only take one of the following forms:

$$\alpha{:}(L_0 \leftarrow L_1 \wedge \ldots \wedge L_n) \qquad \alpha{:}(L_1 \wedge \ldots \wedge L_n \Rightarrow Z)$$
$$\alpha{:}(\mathit{false} \leftarrow L_1 \wedge \ldots \wedge L_n \wedge Z_1 \wedge \ldots \wedge Z_m) \qquad \alpha{:}(?{-}L_1 \wedge \ldots \wedge L_n)$$

Note that projects and negated projects can only occur in the head of active rules and in the body of integrity constraints.

Example 1. The integrity constraint $\mathit{false} \leftarrow A \wedge \beta{:}B$ in the theory of an agent α enforces the condition that α cannot perform a project $\beta{:}B$ when A holds. The active rule $A \wedge not \beta{\div}B \Rightarrow \beta{:}C$ in the theory of an agent α instructs it to perform project $\beta{:}C$ if A holds and agent β has not wanted to update the theory of α with B.

Let $<$ be a binary predicate symbol whose set of constants includes all the generalized rules in the language $\mathcal{L}_{\mathcal{K}}$.

Definition 6 (Priority rule). A priority rule in the language $\mathcal{L}_{\mathcal{K}}$ is a generalized rule defining the predicate symbol $<$.

It is assumed that the set of constants of $<$ does not include $<$ itself. $r_1 < r_2$ means that rule r_1 is preferred to rule r_2.

Definition 7 (Prioritized logic program). A prioritized logic program P in the language $\mathcal{L}_{\mathcal{K}}$ is a set of generalized rules (possibly, priority rules) and integrity constraints in the language $\mathcal{L}_{\mathcal{K}}$.

Definition 8 (Prioritized abductive logic program). A prioritized abductive logic program is a pair $(P, \mathcal{A})$, where P is a prioritized logic program and $\mathcal{A}$ is a set of atoms in the language $\mathcal{L}_{\mathcal{K}}$. The atoms in $\mathcal{A}$ are referred to as the abducibles.

The definition of prioritized abductive logic program generalizes the one given in [9] to allow for priority rules. The intuition is that the program P formalizes the theory of an agent, and the priority rules express preferences among rules in P, in a sense to be made concrete later. Abducibles can be thought of as hypotheses that can be used to extend the given prioritized logic program in order to provide an "explanation" for given queries. Explanations are required to "satisfy" the integrity constraints in P. Abducibles may also be defined in P by generalized rules as the result of a self-update which adopts an abducible as a fact.

3 Abductive Agents and Multi-Agent Systems

This section presents the conception of abductive agent and of multi-agent system. The initial knowledge of an agent is modeled by the notion of initial theory.

Definition 9 (Initial theory). The initial theory $\mathcal{T}$ of an agent α is a tuple $(P, \mathcal{A}, R)$, where $(P, \mathcal{A})$ is a prioritized abductive logic program and R is a set of active rules.

$(P, \mathcal{A})$ formalizes the initial knowledge state of the agent, and R characterizes its reactive behaviour. The knowledge of an agent can dynamically evolve when the agent receives new knowledge, albeit by self-updating rules, or when it abduces new hypotheses to explain observations. The new knowledge is represented in the form of an updating program, and the new hypotheses in the form of a (finite) set $\triangle \subseteq \mathcal{A}$ of abducibles, possibly negated.

Definition 10 (Updating program). Let $\mathcal{M}$ be a multi-agent system (defined below). An updating program U is a finite set of updates such that if an update $v{\div}C \in U$ then v is an agent of $\mathcal{M}$.

An updating program contains the updates that will be performed on the current knowledge state of the agent. To characterize the evolution of the knowledge of an agent we need to introduce the notion of sequence of updating programs. Let $S = \{0, 1, \ldots, m\}$ be a set of natural numbers. We call the elements $s \in S$ *states*. A *sequence of updating programs* $\mathcal{U} = \{U^s \mid s \in S$ and $s > 0\}$ is a set of updating programs U^s superscripted by the states $s \in S$.

Definition 11 (Agent α at state s). Let $s \in S$ be a state. An agent α at state s, written as Ψ^s_α, is a pair $(\mathcal{T}, \mathcal{U})$, where $\mathcal{T}$ is the initial theory of α and $\mathcal{U} = \{U^1, \ldots, U^s\}$ is a sequence of updating programs. If $s = 0$, then $\mathcal{U} = \{\}$.

An agent α at state 0 is defined by its initial theory and by an empty sequence of updating programs, that is $\Psi^0_\alpha = (\mathcal{T}, \{\})$. At state 1, α is defined by $(\mathcal{T}, \{U^1\})$, where U^1 is the updating program containing all the updates that α has received at state 1, either from other agents or as self-updates. In general, an agent α at state s is defined by $\Psi^s_\alpha = (\mathcal{T}, \{U^1, \ldots, U^s\})$, where each U^i is the updating program containing the updates that α has received at state i.

Definition 12 (Multi-agent system at state s). A multi-agent system $\mathcal{M} = \{\Psi^s_{\alpha_1}, \ldots, \Psi^s_{\alpha_n}\}$ at state s is a set of agents $\alpha_1, \ldots, \alpha_n$ each of which at state s.

Note that the definition of multi-agent system characterizes a static society of agents in the sense that it is not possible to add/remove agents from the system, and all the agents are at one common state. Distinct agents in $\mathcal{M}$ may have different sets $\mathcal{A}$ of abducibles.

Within logic programs we refer to agents by using the corresponding subscript. For instance, if we want to express the update of the theory of an agent Ψ_α with C, we write the project $\alpha{:}C$.

To begin with, the system starts at state 0 where each agent α is defined by $\Psi_\alpha^0 = (\mathcal{T}_\alpha, \{\})$. Suppose that at this state α abduces $\triangle = \{a, not\ b\}$ to provide an explanation for an observation made. Suppose also that at state 0 agent β wants to propose an update of the knowledge state of agent α with C by triggering the project $\alpha : C$. Then, at the next state α will receive the update $\beta \div C$ indicating that an update has been proposed by β. Thus, at state 1, α will be defined by $(\mathcal{T}_\alpha, \{U^1\})$, where $U^1 = \{\beta \div C\}$ if no other updates occur for α.

The hypotheses $\triangle$ abduced by α at state 0 are by default discarded at state 1. Thus, α normally does not have memory of what it has assumed. If we want instead to model an agent α that is able to enrich its experience by assuming hypotheses during the process of proving queries and explaining observations, and it is able to adopt previously assumed hypotheses, we can equip the theory of α with active rules of the form: $a \Rightarrow \alpha{:}a$, for whatever abducible $a \in \mathcal{A}$ desired. In this way, when α abduces a the project $\alpha{:}a$ will be triggered, and at the next state its theory will be updated with a itself.

4 Semantics of Abductive Agents

This section introduces the declarative semantics of abductive agents and of multi-agent systems. We need the following definitions. Let U be an updating program.

$$\Pi(U) = \{r \mid r \text{ is a generalized rule and } \alpha{\div}r \in U\}$$
$$\Omega(U) = \{a \mid a \text{ is an active rule and } \alpha{\div}a \in U\}$$

Definition 13 (Default assumptions). Let $(P, \mathcal{A})$ be a prioritized abductive logic program and M a model of P. Let $\triangle \subseteq \mathcal{A}$ be a set of abducibles. The set of default assumptions[2] is:

$$Default(P, \triangle, M) = \{not\ A \mid A \text{ is an object atom}, A \notin \triangle, not\ A \notin \triangle, \text{ and } \nexists r \in P \text{ such that } head(r) = A \text{ and } M \models body(r)\}.$$

The knowledge of an agent α is characterized at the start by its initial theory $\mathcal{T}$. Its knowledge can dynamically evolve when α receives new knowledge, via a sequence of updating programs $\mathcal{U} = \{U^1, \ldots, U^s\}$, or when α assumes new hypotheses to explain new observations, via a set of abducibles $\triangle$. Intuitively, the evolution of knowledge may be viewed as the result of, starting with $\mathcal{T}$, updating it with U^1, updating next with U^2, and so on. Similarly to the rationale of Dynamic Logic Programming (DLP) [2], the rules proposed via updates can be added to the knowledge state of α provided that α does not distrust the update, for instance if it goes against its presently assumed

[2] For simplicity, we assume positive abducibles false by default.

hypotheses and without bothering whether they conflict with previous knowledge of α itself. For example, if an agent α receives an update of the form $\beta \div C$ and α does not distrust it, then the rule C will be added to the knowledge state of α. Also similarly to DLP, the role of updating is to ensure that the rules contained in these newly added updates are in force, and that previous rules are still valid (by inertia) as far as possible, i.e., they are in force while they do not conflict with newly added rules, and they lose force if they so conflict and the new rules remain in force themselves. This rationale is at the basis of the notion of rejected rules.

Definition 14 (Rejected generalized rule). Let $(P, \mathcal{A})$ be a prioritized abductive logic program and M a model of P. Let $s \in S$ be a state of an agent, $\mathcal{U} = \{U^i \mid i \in S \text{ and } i > 0\}$ a sequence of updating programs and $\triangle \subseteq \mathcal{A}$ a set of abducibles. The set of rejected generalized rules at state s is:

$$
\begin{aligned}
&RejectGr(P, \mathcal{U}, \triangle, s, M) = \\
&\{r \in P \mid r \text{ is a generalized rule and } \exists\, \alpha \div r' \in U^i \text{ such that } 1 \leq i \leq s, \\
&\qquad head(r) = not\ head(r'), M \models body(r') \text{ and } M \models not\ distrust(\alpha \div r')\} \\
&\cup \{r \in P \mid r \text{ is a generalized rule and } \exists\, L \in \triangle \text{ such that } head(r) = not\ L\} \\
&\cup \{r \mid r \text{ is a generalized rule, } \exists\, \beta \div r \in U^i \text{ and } \exists\, \alpha \div r' \in U^j \text{ such that } i < j \leq s, \\
&\qquad head(r) = not\ head(r'), M \models body(r') \text{ and } M \models not\ distrust(\alpha \div r')\} \\
&\cup \{r \mid r \text{ is a generalized rule, } \exists\, \beta \div r \in U^i, 1 \leq i \leq s \text{ and } \exists\, L \in \triangle \text{ such that} \\
&\qquad head(r) = not\ L\}.
\end{aligned}
$$

According to the definition above, a generalized rule r either in P or proposed via an update in U^i is rejected at state s by a model M if there exists a newer generalized rule r' proposed via a subsequent update in U^j by any agent α, such that the head of r' is the complement of the head of r, the body of r' is true in M and the update is not distrusted. r can also be rejected if there exists an hypothesis $L \in \triangle$ that is the complement of the head of r. Mark that: (1) whenever an agent distrust an update proposed by another agent, the rule proposed is not in force; (2) at state 0 no rule is rejected, i.e., $RejectGr(P, \mathcal{U}, \triangle, 0, M) = \{\}$; (3) all non rejected rules persist by inertia; (4) *distrust/1* is a reserved predicate which can itself be updated; (5) it is not required of r' itself not to be rejected to allow it to reject r, simply because if r' is itself rejected then some subsequent rule has reinstated the conclusion of r and so the rejection of r is immaterial; this comment applies as well to the next definition. Generalized rules defining abducibles can also be rejected. Abducibles may have defining rules in the theory of an agent α either because communicated by other agents via updates, or because of a self-update, like in the case where there exists an active rule of the form $a \Rightarrow \alpha : a$ (with $a \in \mathcal{A}$). Thus, an agent may assume new hypotheses to motivate an observation, and later on discard those hypotheses because it turns out (by performing subsequent observations) to be not correct or because the world has evolved and those hypotheses became obsolete.

Active rules can also be rejected in a way similar to that of generalized rules.

Definition 15 (Rejected active rule). Let $(P, \mathcal{A}, R)$ be the initial theory of an agent and M a model of P. Let $s \in S$ be a state and $\mathcal{U} = \{U^i \mid i \in S \text{ and } i > 0\}$ a sequence of updating programs. The set of rejected active rules at state s is:

$$RejectAr(R,\mathcal{U},s,M) =$$
$$\{r \in R \mid \exists\, \alpha{\div}r' \in U^i \text{ such that } 1 \leq i \leq s, head(r) = not\ head(r'),$$
$$M \models body(r') \text{ and } M \models not\ distrust(\alpha{\div}r')\}$$
$$\cup\, \{r \mid \exists\, \beta{\div}r \in U^i, r \text{ is an active rule and } \exists\, \alpha{\div}r' \in U^j \text{ such that } i < j \leq s,$$
$$head(r) = not\ head(r'), M \models body(r') \text{ and } M \models not\ distrust(\alpha{\div}r')\}$$

As the head of an active rule is a project and not an atom, active rules can only be rejected by active rules. Rejecting an active rule r makes r not triggerable even if its body is true in the model. Thus, by rejecting active rules, we make the agent less reactive.

While updates allow us to deal with a dynamically evolving world, where rules change in time, preferences allow us to choose among various possible models of the world and among possible incompatible reactions. In [4], two criteria are established to remove unpreferred generalized rules in a program: removing unsupported generalized rules, and removing less preferred generalized rules defeated by the head of some more preferred one. Unsupported generalized rules are rules whose head is true in the model and whose body is defeated by the model.

Definition 16 (Unsupported generalized rule). Let P be a prioritized logic program and M a model of P. The set of unsupported generalized rules is:

$$Unsup(P,M) = \{r \in P \mid M \models head(r), M \models body_{pos}(r) \text{ and } M \not\models body_{neg}(r)\}.$$

Definition 17 (Unpreferred generalized rule). Let P be a prioritized logic program and M a model of P. $Unpref(P,M)$ is a set of unpreferred generalized rules of P and M iff:

$$Unpref(P,M) = \text{least}(Unsup(P,M) \cup \mathcal{X})$$

where $\mathcal{X} = \{\ r \in P \mid\ \exists r' \in (P - Unpref(P,M))$ such that:
$M \models r' < r,\ M \models body_{pos}(r')$ and $[\ not\ head(r') \in body_{neg}(r)$ or $(not\ head(r) \in body_{neg}(r'),\ M \models body(r))\ ]\ \}$.

In other words, a generalized rule is unpreferred if it is unsupported or defeated by a more preferred generalized rule (which is not itself unpreferred), or if it attacks (i.e., attempts to defeat) a more preferred generalized rule.

Active rules are rules that enjoy the property that they can be triggered when their body is true in the model. Triggering an active rule means to execute the project occurring in its head. For example, given an active rule $L_1 \wedge L_2 \Rightarrow \alpha : C$ and a model M, if $M \models L_1 \wedge L_2$, then the project $\alpha : C$ can be executed. Executing the project $\alpha : C$ denotes the intention of the agent of updating the knowledge state of agent α with C. The set of executable projects is the set containing the projects of all the active rules that are triggered and preferred by a model M.

Definition 18 (Executable project). Let R be a set of active rules and M a model. The set of executable projects is:

$$Project(R,M) = \{\alpha{:}C \mid \exists r \in R \text{ such that } head(r) = \alpha{:}C \text{ and } M \models body(r)\}.$$

The following definition introduces the notion of preferred abductive stable model of an agent α at a state s with set of hypotheses $\triangle$. Given the initial theory $\mathcal{T}$ of α, a sequence of updating programs $\mathcal{U}$ and the hypotheses $\triangle$ assumed at state s by α, a preferred abductive stable model of α at state s is a stable model of the program $\mathcal{X}$ that extends P to contain all the updates in $\mathcal{U}$, all the hypotheses in $\triangle$, and all those rules whose updates are not distrusted but that are neither rejected nor unpreferred. The preferred abductive stable model contains also the projects of all active rules (both in P or proposed via an update) that are triggered (i.e., all the executable projects).

Definition 19 (Preferred abductive stable model of agent α at state s with hypotheses $\triangle$). Let $s \in S$ be a state. Let $\Psi^s_\alpha = (\mathcal{T}, \mathcal{U})$ be agent α at state s and M an interpretation such that $\mathit{false} \notin M$. Assume that $\mathcal{T} = (P, \mathcal{A}, R)$ and $\mathcal{U} = \{U^i \mid i \in S \text{ and } i > 0\}$. Let $\triangle \subseteq \mathcal{A}$ be a set of abducibles. M is a preferred abductive stable model of agent α at state s with hypotheses $\triangle$ iff:

− $\forall r_1, r_2$: if $(r_1 < r_2) \in M$, then $(r_2 < r_1) \notin M$

− $\forall r_1, r_2, r_3$: if $(r_1 < r_2) \in M$ and $(r_2 < r_3) \in M$, then $(r_1 < r_3) \in M$

− $M = \text{least}(\mathcal{X} \cup \mathit{Default}(\mathcal{Y}, \triangle, M) \cup \mathit{Project}(\mathcal{Z}, M))$, where:

$\mathcal{Y} = P \cup \bigcup_{1 \leq i \leq s} U^i \cup \triangle \cup \{r \mid r$ is a generalized rule or an integrity constraint and $\exists\, \alpha{\div}r \in \bigcup_{1 \leq i \leq s} U^i$ such that $M \models \mathit{not\ distrust}(\alpha{\div}r)\}$

$\mathcal{X} = \mathcal{Y} - \mathit{RejectGr}(P, \mathcal{U}, \triangle, s, M) - \mathit{Unpref}(\mathcal{Y} - \mathit{RejectGr}(P, \mathcal{U}, \triangle, s, M), M)$

$\mathcal{Z} = R \cup \{r \mid r$ is an active rule and $\exists\, \alpha{\div}r \in \bigcup_{1 \leq i \leq s} U^i\} - \mathit{RejectAr}(R, \mathcal{U}, s, M)$.

In general, at a certain state an agent α may have several preferred abductive stable models. Note that if $\triangle = \{\}$ and $s = 0$ or each updating program in $\mathcal{U}$ is empty, then the last condition of the definition above reduces to:

$$\mathcal{Y} = P \qquad \mathcal{X} = \mathcal{Y} - \mathit{Unpref}(\mathcal{Y}, M)$$
$$M = \text{least}(\mathcal{X} \cup \mathit{Default}(\mathcal{Y}, \{\}, M) \cup \mathit{Project}(R, M)).$$

This condition reduces further, if (in addition) there are no preference rules in P, to:

$$\mathcal{Y} = P \qquad \mathcal{X} = \mathcal{Y} - \mathit{Unsup}(\mathcal{Y}, M)$$
$$\begin{aligned} M &= \text{least}(\mathcal{X} \cup \mathit{Default}(\mathcal{Y}, \{\}, M) \cup \mathit{Project}(R, M)) \\ &= \text{least}(P \cup \mathit{Default}(\mathcal{Y}, \{\}, M) \cup \mathit{Project}(R, M)). \end{aligned}$$

When there are neither projects nor updates nor hypotheses (i.e., $\triangle = \{\}$), the semantics reduces to the Preferential semantics of Brewka and Eiter [6]. If updates are then introduced, the semantics generalizes to the Updates and Preferences semantics of Alferes and Pereira [4], which extends DLP with preferences. Our semantics takes the latter and complements it with abducibles, with mutual and self-updates by means of active rules and projects, plus queries, within a society of agents.

Definition 20 (Abductive explanation of agent α at state s for query Q). Let $s \in S$ be a state and $\Psi^s_\alpha = (\mathcal{T}, \mathcal{U})$ agent α at state s. Let Q be a query. An abductive explanation of agent α at state s for Q is any subset $\triangle$ of $\mathcal{A}$ such that there exists a preferred abductive stable model M of α at s with hypotheses $\triangle$ and $M \models Q$.

5 Semantics of Multi-Agent Systems

We can now present the semantics $\mathcal{S}$ of a multi-agent system $\mathcal{M}$ embedding abductive agents. The role of $\mathcal{S}$ is to characterize the relationship among the agents of $\mathcal{M}$. This is achieved by formalizing the relation between projects and updates. $\mathcal{S}$ is defined as the set of preferred abductive stable models of each agent of $\mathcal{M}$. It uses the following function. Let $\{M^{s-1}_{\alpha_1}, \ldots, M^{s-1}_{\alpha_n}\}$ be a set of the set of models of the agents $\alpha_1, \ldots, \alpha_n$ at state $s-1$.

$$\gamma(\{M^{s-1}_{\alpha_1}, \ldots, M^{s-1}_{\alpha_n}\}) = \{U^s_{\alpha_1}, \ldots, U^s_{\alpha_n}\}$$

where an update $(\alpha_j \div C) \in U^s_{\alpha_i}$ iff the project $\alpha_i{:}C$ belongs to all the models of α at state $s-1$, that is: $(\alpha_i{:}C) \in \bigcap M^{s-1}_{\alpha_j}$, for $1 \leq i, j \leq n$. The intuition is that α at state s can have several preferred abductive stable models, each of which may trigger distinct active rules (and therefore each model will contain distinct executable projects). According to γ the projects that will be executed are the selectable projects that occur in every model of α at state s. Note that when $j = i$ we have a self-update: the agent chooses to update its own theory.

Definition 21 (Semantics of $\mathcal{M}$). Let $\mathcal{M} = \{\Psi^0_{\alpha_1}, \ldots, \Psi^0_{\alpha_n}\}$ be an abductive multi-agent system at state 0. Let $s \in S$ be a state. The semantics $\mathcal{S}^s$ of $\mathcal{M}$ at state s is defined inductively as follows.

<u>Base Case</u> $(s = 0)$ That means that $\Psi^0_{\alpha_i} = (\mathcal{T}_{\alpha_i}, \{\})$, for every agent α_i (with $1 \leq i \leq n$). Suppose that $Q^0_{\alpha_i}$ is the query of α_i at state 0. Let $M^0_{\alpha_i}$ be the set of preferred abductive stable models of α_i at state 0 each of which with hypothesis any abductive explanation $\triangle^0_{\alpha_i}$ of α_i for $Q^0_{\alpha_i}$. Then, the semantics $\mathcal{S}^o$ of $\mathcal{M}$ is $\mathcal{S}^0 = \{M^0_{\alpha_1}, \ldots, M^0_{\alpha_n}\}$.

<u>Inductive step</u> $(s > 0)$ Let $\{\Psi^{s-1}_{\alpha_1}, \ldots, \Psi^{s-1}_{\alpha_n}\}$ be the multi-agent system $\mathcal{M}$ at state $s-1$ and $\mathcal{S}^{s-1} = \{M^{s-1}_{\alpha_1}, \ldots, M^{s-1}_{\alpha_n}\}$ its semantics. Suppose that $\Psi^{s-1}_{\alpha_i} = (\mathcal{T}_{\alpha_i}, \{U^1_{\alpha_i}, \ldots, U^{s-1}_{\alpha_i}\})$, for $1 \leq i \leq n$. Assume that $\gamma(\mathcal{S}^{s-1}) = \{U^s_{\alpha_1}, \ldots, U^s_{\alpha_n}\}$.

Then, the multi-agent system $\mathcal{M}$ at state s is $\{\Psi^s_{\alpha_1}, \ldots, \Psi^s_{\alpha_n}\}$ with $\Psi^s_{\alpha_i} = (\mathcal{T}_{\alpha_i}, \{U^1_{\alpha_i}, \ldots, U^{s-1}_{\alpha_i}, U^s_{\alpha_i}\})$. Let $Q^s_{\alpha_i}$ be the query of α_i at state s, and $M^s_{\alpha_i}$ the set of preferred abductive stable models of α_i at state s each of which with hypothesis any abductive explanation $\triangle^s_{\alpha_i}$ for $Q^s_{\alpha_i}$, for $1 \leq i \leq n$. Then, $\mathcal{S}^s = \{M^s_{\alpha_1}, \ldots, M^s_{\alpha_n}\}$.

The semantics of the multi-agent system depends on the queries of the agents. In fact, the models of an agent at a certain state depend on the abducibles assumed at that state which in turn depend on the queries of the agent.

6 Preferring Abducibles

In our framework we defined the priority relation over generalized rules. A possible question is: Can we also express preferences over abducibles ? Being able to do so will allow us to perform for example *exploratory data analysis* which aims at suggesting a pattern for further inquiry, and contributes to the conceptual and qualitative understanding of a phenomenon.

Preferences over alternative abducibles can be coded into cycles over negation, and preferring a rule will break the cycle in favour of one abducible or another. Let $\lhd$ be a binary predicate symbol whose set of constants includes all the abducibles in $\mathcal{A}$. $a \lhd b$ means that abducible a is preferred to abducible b.

Assume that a unexpected phenomenon, x, is observed by an agent α, and that α has three possible hypotheses (abducibles), a, b and c, that are capable of explaining it. In exploratory data analysis, after observing some new facts, we abduce explanations and explore them to check predicted values against observations. Though there may be more than one convincing explanation, we abduce only the more plausible ones. Suppose explanations a and b are both more plausible than (and therefore preferable to) explanation c. This can be expressed as:

$$P = \left\{ \begin{array}{l} x \leftarrow a \\ x \leftarrow b \\ x \leftarrow c \end{array} \right\} \text{ and the preferences as: } \left\{ \begin{array}{l} a \lhd c \\ b \lhd c \\ \end{array} \right\}.$$

The program above can be represented by a prioritized abductive logic program $(P', \mathcal{A})$, where $\mathcal{A} = \{a, b, c, a^*, b^*, c^*\}$ and

$$P' = P \cup \left\{ \begin{array}{ll} a \leftarrow a^* \wedge not\, b \wedge not\, c & (1) \\ b \leftarrow b^* \wedge not\, a \wedge not\, c & (2) \\ c \leftarrow c^* \wedge not\, a \wedge not\, b & (3) \\ 1 < 3 & \\ 2 < 3 & \end{array} \right\} \cup \left\{ \begin{array}{l} a^+ \leftarrow not\, a^* \wedge not\, b^+ \wedge not\, c^+ \\ b^+ \leftarrow not\, b^* \wedge not\, a^+ \wedge not\, c^+ \\ c^+ \leftarrow not\, c^* \wedge not\, a^+ \wedge not\, b^+ \\ not\, a \leftarrow a^+ \\ not\, b \leftarrow b^+ \\ not\, c \leftarrow c^+ \end{array} \right\}.$$

The first three rules (called (1), (2) and (3)) code the alternative abducibles a, b and c into cycles over negation, and the following two priority rules state that the first two rules (i.e., rules (1) and (2)) are preferable to the third rule (i.e., rule (3)). Since it is possible in our framework to abduce also negative abducibles, it is needed to add the rules stating that the negative abducibles $not\ a$, $not\ b$ and $not\ c$ are alternative abducibles. This is achieved by the last six rules by introducing the new atoms a^+, b^+ and c^+.

For simplicity, assume that the initial theory of α is $\mathcal{T} = (P', \mathcal{A}, \{\})$ and that there are no updates in the sequence $\mathcal{U}$ of updating programs of α. Then, α has two preferred abductive stable models to explain its observation x: $M_1 = \{a, x, 1{<}3, 2{<}3, a^*, b^+, c^+\}$ and $M_2 = \{b, x, 1{<}3, 2{<}3, b^*, a^+, c^+\}$. By removing the priority atoms, the abducibles superscripted by * and the atoms superscripted by $^+$ from M_1 and M_2, we obtain the intended models of the initial program P, where abducibles a and b are both preferred over c. This technique can be generalized to prefer among sets of abducibles.

In this example, we have only a partial priority theory over abducibles. Thus, we cannot select exactly one abducible (i.e., one model), as it were the case had we a complete priority relation over all abducibles in $\mathcal{A}$. To prefer between a and b, α can perform some experiment *exp* to obtain confirmation (by observing the environment) about the most plausible hypothesis. For example, by using the following rules (where

e plays the role of the environment):

$$\left\{\begin{array}{l} \mathit{choose} \leftarrow a \\ \mathit{choose} \leftarrow b \end{array}\right\} \text{ together with } \left\{\begin{array}{l} a \Rightarrow \alpha : \mathit{chosen} \\ b \Rightarrow \alpha : \mathit{chosen} \\ \mathit{choose} \Rightarrow \alpha : (\mathit{not\,chosen} \Rightarrow e : \mathit{exp}) \end{array}\right\}.$$

Suppose that agent α has two hypotheses, a and b, that are capable to explain the observed phenomena x, and that α must discover the correct one. α chooses an hypothesis if a or b or both holds:

$$\begin{array}{l} \mathit{choose} \leftarrow a \\ \mathit{choose} \leftarrow b. \end{array}$$

With this knowledge, agent α has still two preferred abductive stable models: $M_1 = \{a, x, \mathit{choose}, \ldots\}$ and $M_2 = \{b, x, \mathit{choose}, \ldots\}$. As *choose* holds in both models, the last active rule is triggerable. When triggered, it will add (at the next state) the active rule $\mathit{not\,chosen} \Rightarrow e : \mathit{exp}$ to the theory of α, and if $\mathit{not\,chosen}$ holds, α will perform the experiment *exp*. The first two active rules are needed to prevent α by performing *exp* when α has chosen one of the abducibles.

7 Acting and Planning

A typical use of abduction is in generating plans in a planning problem (cf. [3, 10, 13]). Suppose the situation where some information of the planning problem is missing or changes during the planning process. To tackle this situation we must be able to make a (possibly partial) plan, to start executing it, and in case something goes wrong, to detect the anomaly and to replan. This entails the ability to detect such cases and to replan in order to achieve the original goal.

This approach to planning may involve also the ability to generate revisable plans, that is, plans that are generated in stages. After having executed the actions of each stage, we test whether the performed actions have been successful in order to move to the next stage. If some action failed, then we must revise our plan. A successful plan is a revisable plan that has been completely executed such that the obtained final state satisfies the original goal.

To express revisable plans we must be able to express consecutive actions. We represent actions as abducibles. Let $\diamond$ be a binary predicate symbol whose set of constants includes all the abducibles in $\mathcal{A}$. $a \diamond b$ means that action a must be executed before action b.

Typically, for every action a there will be one or more active rules where the action occurs in the body. For example, $a \wedge L_1 \wedge \ldots \wedge L_k \Rightarrow Z$ represents an action whose preconditions are $L_1 \wedge \ldots \wedge L_k$ and whose effect is the project Z.

Suppose that in order to solve a plan p we must execute two actions, a and b, sequentially, a before b. This can be expressed as:

$$\left\{\begin{array}{l} p \leftarrow a \diamond b \\ a \wedge \mathit{precA} \Rightarrow Z_1 \\ b \wedge \mathit{precB} \Rightarrow Z_2 \end{array}\right\}.$$

The program above can be represented by a prioritized abductive logic program $(P, \mathcal{A})$, where $P = \{p \leftarrow a \wedge seq(a, b) \wedge seq(b, true)\}$ and $\mathcal{A} = \{a, b\}$, together with the active rules:

$$R = \left\{ \begin{array}{l} a \wedge precA \wedge seq(a, G) \Rightarrow Z_1 \wedge \alpha{:}(?{-}G) \\ b \wedge precB \wedge seq(b, G) \Rightarrow Z_2 \wedge \alpha{:}(?{-}G) \end{array} \right\}.$$

Suppose now that we want to test whether the action a has been successful before executing b, that is, $p \leftarrow a \diamond test \diamond b$. In case a has been successful, we execute b, otherwise a is executed again. Let *propA* be the property that must hold whenever a has been successful. This can be formalized as: $\mathcal{A} = \{a, b, test\}$, $P = \{p \leftarrow a, seq(a, test), seq(test, b), seq(b, true)\}$ together with the active rules:

$$R = \left\{ \begin{array}{l} a \wedge precA \wedge seq(a, G) \Rightarrow Z_1 \wedge \alpha{:}(?{-}G) \\ test \wedge not\, propA \Rightarrow \alpha{:}(?{-}a) \\ test \wedge propA \wedge seq(test, G) \Rightarrow \alpha{:}(?{-}G) \\ b \wedge precB \wedge seq(b, G) \Rightarrow Z_2 \wedge \alpha{:}(?{-}G) \end{array} \right\}.$$

8 Conclusion and Future Work

We have presented a logical framework of a multi-agent system in which each agent can communicate with and update other agents, can react to the environment, is able to prefer and to abduce hypotheses in order to explain observations made, which in turn may trigger reactions.

Applications in which our agent technology can have a significant potential to contribute, and that we are considering, are internet applications, e.g., information integration and web-site management. To this aim we have designed an agent architecture [14] and we are going to start implementing it. The problem of information integration consists in how to integrate data from multiple heterogeneous sources. The goal of a data integration system is to provide a uniform interface to a several data sources. An example of information integration is the task of providing information about movies from data sources on the web (for example, cf. [15]).

The goal of web site management applications is the flexible construction and modification of web sites. Web sites contain and integrate several pieces of data that are linked together into a navigational structure. One possible approach in these applications is to declaratively represent web sites, that is, to logically represent the kind of data and the structural aspects of web sites. One advantage would be the ability to automatically reconstruct and to enforce integrity constraints on web sites. The underlying idea is that such sites restructure themselves depending on the usage patterns, and can adapt themselves according to the profile of individual users. If we represent declaratively a web site, then we can reason about the structure of the site, for example, to see if the integrity constraints hold any time the site is updated (e.g., no dangling pointers, an employee's homepage should point to the department's homepage, etc.).

We believe that the theory of the agents considered is a rich evolvable basis, and suitable for engineering configurable, dynamic, self-organizing and self-evolving agent societies. In [12, p. 289] Jennings argues that open, networked systems are characterized by the fact that there is no simple controlling organization and that there is constant change. Thus, these systems are collections of independently developed software entities that are interacting with each other. Jennings claims that the computational model of these systems places several requirements:

(i) the individual entities need to be able to act to achieve specified objectives (i.e., they must be active and autonomous);
(ii) the individual entities need to be reactive and proactive;
(iii) the computational entities need to be capable of interacting with entities that were not foreseen at design time (i.e., they must be able to engage in flexible interactions);
(iv) any organizational relationships that do exist must be reflected in the behaviour and actions of the agents (i.e., the organizational relationships must be explicitly represented).

Our agents have those abilities. Regarding to (iv) we can represent dynamic organizational structures in the theory of agents that can self-organize themselves and self-evolve via the ability of preferring and updating.

Davidsson [8] categorises agent societies based on the following properties:

- *openness*, i.e., the possibility for agents to join the society without any restriction;
- *flexibility*, i.e., the degree to which agents are restricted in their behaviour by the society;
- *stability*, i.e., predictability of the consequences of actions, and
- *trustfulness*, i.e., the extent to which the owners of the agents trust the society.

Davidsson argues that whereas open societies support openness and flexibility, closed societies support stability and trustfulness. In several systems, e.g., in systems characterized as information eco-systems, there is a need for societies that support all these aspects. Thus, he investigates other kinds of agent societies among which *semi-open societies*, where any agent can join the society provided that it follows some well-specified restrictions.

The agents that we have presented can be used for example to animate semi-open societies. In fact, to do so, the agents must exhibit several rational abilities, e.g., they must be able to decide which society to join according to their own goals, to decide whether or not to accept the restrictions imposed to enter into the society, and eventually to negotiate them. Thus, to increase the degree of trustfulness and stability in open societies we need to empower the agent theories with rational abilities to make the agents able to react to events unforeseen at design time.

Another framework where our agents can be of use is the one proposed by Vercouter [16]. He proposes a distributed approach to handle openness in multi-agent systems. Typically, the openness of a system is ensured by an entity, called the middle-agent, that supports the addition and removal of agents from the system, and knows the capabilities of all the agents. The middle-agent handles also the connection problem: when an agent looks for another agent that provides a service it needs. The middle-agent (centralized) approaches to the problem of openness have a number of limits, and to overcome them Vercouter investigates a distributed approach (i.e., without the middle-agent technique) based on the notion of *presentation/recommendation protocol*. Each agent can have its own representation of other agents, and can detect its possibilities of cooperation. When an agent joins a coalition, the agent presents itself to some other agents of the coalition and informs them of its capabilities. During this phase, the agent can ask them for recommendations of other agents that provide services that it needs, and start a presentation protocol with them. This is clearly another framework where several rational abilities are needed in agents, for example, to reason about, to evaluate and to prefer which agents are relevant for its purposes.

Within the proposed multi-agent system framework we can represent groups, teams, coalitions of agents implicitly based on the internal mental states of the members. It is advocated, especially in open multi-agent systems (cf. [5, 12, 17]), that there is a need to make the organisational elements as well as the formalisation of the agent interactions of a multi-agent system externally visible rather than being embedded in the mental state of each agent, i.e., it is needed to explicitly represent the organisational structure and the agent interactions. For example, we may formalize a group structure as an abstract description of a group that identifies all the roles and interactions that can occur within a group. This will allow us to precisely define concepts like permission, obligation, responsibility, social laws, requirements and roles of the society, etc. We are investigating how to explicitly represent organisational structures in our framework, and how to animate them with agents in a way that each agent will automatically have a view (perhaps partial) of the organisational structure and the externally visible events. We believe that this can be achieved through the concept of *organisational reflection*[3]. In fact, we may embed the theory of agents with various degrees of reflection abilities to make an agent able to introspect various aspects of the organisation where it belongs. We may also imagine a more complex organisational structure composed of several nested organisational substructures each of which with its own specific rules and with more general rules inherited by the parent structure. In this way, an agent may only have a partial view of the entire organisational structure. Thus, the members of a society will have information of it (e.g., they will know its rules, requirements and laws), and they will have more information when reasoning about other agents' actions. In addition, the agents will be able to reason upon organisational structures and eventually try to modify them. In this way organisational structures will not be rigid, but flexible and can evolve with the agents' intervention.

Organisational structures and rules are needed to provide control over the member agents with respect to their actions and interactions.

Castelfranchi [7] argues that social order is a major problem in multi-agent systems. The idea of a total control and a technical prevention against chaos, conflicts and deception in computers is unrealistic and even self-defeating in some case, like in the building trust. He states that there is some illusion in Computer Science about solving this problem by rigid formalization and rules, constraining infra-structures, security device, etc. and there is skepticism or irritation towards more soft and social approaches, that leave more room to spontaneous emergence, or to decentralized control, or to normative "stuff" which is not externally imposed but internally managed by the agents. He claims that the most effective solution to this problem is social modelling and that

[3] This term was first introduced by J. Ferber and O. Gutknecht [11], but it is used here with a different meaning.

it should leave some flexibility and try to deal with emergent and spontaneous form of organizations (that is, decentralized and autonomous social control). The problem with this approach is that of modeling the feedback from the global results to the local/individual layer.

To solve this problem, we need two ingredients (still missing in our agent theory): the ability of introspection and meta-reasoning. In fact, in order to be able to dynamically change the organization, structure of the multi-agent system, agents must be aware (even if partially) of the structure and must be able to introspect about it. By using meta-reasoning the agent can evaluate it, obtain feedback from it and eventually try to modify it via preferences and updates in a rational way. Our ongoing work and future projects is discussed in [1].

In conclusion, our claim is that in order to have dynamic, flexible agent societies we need to have suitable agent theories, otherwise the structure modeling the agent society will be rigid in the sense that it will not be modifiable by the agents themselves.

Acknowledgements

L. M. Pereira acknowledges the support of PRAXIS project MENTAL and POCTI project FLUX.

References

1. J. J. Alferes, P. Dell'Acqua, E. Lamma, J. A. Leite, L. M. Pereira, and F. Riguzzi. A logic based approach to multi-agent systems. ALP Newsletter, August 2001. Available at http://centria.di.fct.unl.pt/∼lmp.
2. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinski, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *J. Logic Programming*, 45(1-3):43–70, 2000.
3. J. J. Alferes, J. A. Leite, L. M. Pereira, and P. Quaresma. Planning as abductive updating. In D. Kitchin, editor, *Procs. of the AISB'00 Symposium on AI Planning and Intelligent Agents*, pages 1–8, Birmingham, England, 2000. AISB.
4. J. J. Alferes and L. M. Pereira. Updates plus preferences. In M. O. Aciego, I. P. de Guzmn, G. Brewka, and L. M. Pereira, editors, *Logics in AI, Procs. JELIA'00*, LNAI 1919, pages 345–360, Berlin, 2000. Springer.
5. A. Artikis and G. Pitt. A formal model of open agent societies. Proc. of Autonomous Agents, 2001.
6. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109:297–356, 1999.
7. C. Castelfranchi. Engineering Social Order. In Andrea Omicini, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agents World. 1st Int. Workshop ESAW 2000. Revised Papers*, LNAI 1972, pages 1–18, Berlin, 2000. Springer-Verlag.
8. P. Davidsson. Categories of artificial societies. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *ESAW 2001*, pages 1–9, 2001. Available at: http://lia.deis.unibo.it/confs/ESAW01/.
9. P. Dell'Acqua and L. M. Pereira. Updating agents. In S. Rochefort, F. Sadri and F. Toni (eds.), Procs. of the ICLP'99 Workshop on Multi-Agent Systems in Logic (MASL'99), 1999.
10. M. Denecker and A. Kakas, editors. *Special Issue on Abductive Logic Programming*, volume 44(1-3). J. Logic Programming, 2000.

11. J. Ferber and O. Gutnecht. A meta-model for the analysis and design of organisations in multi-agent systems. In *Int. Conf. on Multi-Agent Systems (ICMAS-98)*, pages 128–135. IEEE Computer Society, 1998.
12. N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.
13. A. C. Kakas, R. S Miller, and F. Toni. Planning with incomplete information. In C. Baral and M. Truszczynski, editors, *Proc. NMR'2000, 8th Int. Workshop on Non-Monotonic Reasoning, Spec. Session on Representing Actions and Planning*, 2000.
14. J. A. Leite, J. J. Alferes, and L. M. Pereira. Minerva - A Dynamic Logic Programming Agent Architecture. To appear in: ATAL01 - 8th Int. Workshop on Agent Theories, Architectures, and Languages, 2001.
15. A. Y. Levy and D. S. Weld. Intelligent internet systems. *Artificial Intelligence*, 118:1–14, 2000.
16. L. Vercouter. A distributed approach to design open multi-agent systems. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *ESAW 2001*, pages 24–37, 2001. Available at: http://lia.deis.unibo.it/confs/ESAW01/.
17. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, LNCS 1957, pages 127–141, Berlin, 2001. Springer-Verlag.