Preferring and Updating in Logic-Based Agents

Pierangelo Dell'Acqua[*] [†] Luís Moniz Pereira[†]

[*] Department of Science and Technology - ITN
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 update its knowledge state, of its own initiative or when it receives new incoming information, and it is able to prefer beliefs and reactions, when several alternatives are possible. The knowledge state of an agent is represented by an updatable prioritized logic program, in which priorities among rules can be expressed to allow the agent to prefer, and where the preference relation itself can be updated as well. An example is developed to illustrate how our approach works, including how preferring can enhance reactivity in agents. Finally, we discuss some Web applications where our approach can have a significant potential to contribute, and refer to the work of others.

1 Introduction

In a previous paper [12] we presented a combination of the Dynamic Logic Programming paradigm proposed by J. J. Alferes et al. [3, 17] and (a version of) KS-agents proposed by Kowalski and Sadri [14]. In the resulting framework, rational, reactive agents can dynamically change their own knowledge bases as well as their own goals. In particular, at every iteration of an observe-think-act cycle, the agent can make observations, learn new facts and new rules from the environment, and then it can update its knowledge accordingly. The agent can also receive a piece of information that contrasts with its knowledge. To solve eventual cases of contradiction within the theory of an agent, techniques of contradiction removal and preferences among several sources can be adopted [15]. The actions of an agent are modeled by means of updates, inspired by the approach in [5].

A semantic characterization of updates is given in [3] as a generalization of the stable model semantics of normal logic programs [13]. Such a semantics is generalized to the three-valued case in [5], which enable us to update programs under the well-founded semantics.

J. J. Alferes and L. M. Pereira [4] propose a logic programming framework that combines two distinct forms of reasoning: preferring and updating. In particular, they define a language capable of considering sequences of logic programs that result from the consecutive updates of an initial program, where it is possible to define a priority relation among the rules of all successive programs. Updating and preferring complement each other in the sense that updates create new models, while preferences allow us to select among pre-existing models. Moreover, within the framework, the priority relation can itself be updated. In [4] the authors also define a declarative semantics for such a language.

In this paper we set forth an extension of the language introduced in [12] for updating agents to include preferences among the rules of an agent. In the resulting framework agents can, not only update their own knowledge, for example on advice from some other agent, but can also express preferences about their own rules. Preferences will allow an agent to select among alternative models of its knowledge base. In our framework, preferences can also be updated, possibly on advice from others. We provide the declarative semantics of the multi-agent system, and we characterize the beliefs of an agent in terms of preferred stable models. Finally, we discuss some Web applications where our approach can have a significant potential to contribute. The main contributions of the paper are:

- We reformulate some previous results of ours and combine them into a multi-agent context by modifying, generalizing some of the definitions and by providing new ones.
- We provide the declarative semantics of the multi-agent system.
- We exhibit examples to show how our approach works, including how preferring can enhance reactivity in agents.

In the remaining of the paper, we will use the following example (taken from [4]) as a working example. This example, where rules as well as preferences change over time, shows the need to combine preferences and updates in agents, including the updating of preferences themselves.

Happy story. *(0) In the initial situation Maria is living and working everyday in the city. (1) Next, as Maria has received some money, Maria conjures up other, alternative but more costly, living scenarios, namely traveling, settling up on a mountain, or living by the beach. And, to go with these, also the attending preferences, but still in keeping with the work context, namely that the city is better for that purpose than any of the new scenarios, which are otherwise incomparable amongst themselves. (2) Consequently, Maria decides to quit working and go on vacation, supported by her increased wealth, and hence to define her vacation priorities. To wit, the mountain and the beach are each preferable to travel, which in turn gainsays the city. (3) Next, Maria realizes her preferences keep her all the while undecided between the mountain and the beach, and advised by a friend opts for the former.*

To keep the notation short, we follow some abbreviations: we let c stand for "living in the city", mt for "settling on a mountain", b for "living by the beach", t for "traveling", wk for "work", vac for "vacation", and mo for "possessing money".

2 Logic Programming Framework

This section introduces the language of an agent. Typically, an agent can hold positive and negative information. An agent, for example, looking at the sky, can observe that it is snowing. Then, after a while, when weather conditions change, the agent observes that it is not snowing anymore. Therefore, it has to update its own knowledge with respect to the new incoming information. This implies that the language of an agent should be expressive enough to represent both positive and negative information. Knowledge updating refers not only to facts, as above, but also to rules, say new legislation, which can also be overridden by newer rules which negate previous conclusions.

In order to represent negative information in logic programs, we need more general logic programs that allow default negation *not* A not only in premises of their 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.

Propositional variables whose names do not begin with "*not*" and do not contain the symbols ":" and "$\div$" 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 of 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. Atoms, updates and negated updates are generically called *literals*.

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

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

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 $(L_0 \neq \mathit{false})$ is an atom and every L_i $(1 \leq i \leq n)$ is a literal from $\mathcal{L}_{\mathcal{K}}$.

[1] For further motivation and intuitive reading of logic programs with default negations in the heads see [3].

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 $false \leftarrow L_1 \wedge \ldots \wedge L_n \wedge Z_1 \wedge \ldots \wedge Z_m$ $(n \geq 0, m \geq 0)$, where every L_i $(1 \leq i \leq n)$ is a literal, 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 which atom occurs in the head.

Definition 4 (Generalized logic program). A generalized logic program P over the language $\mathcal{L}_\mathcal{K}$ is a set of generalized rules and integrity constraints in the language $\mathcal{L}_\mathcal{K}$.

Definition 5 (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 a literal 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 6 (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 a literal, 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, 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 operates on the state of other agents, 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.

Example 1. Suppose that the underlying theory of Maria (represented with m) contains the following active rules:

$$R = \left\{ \begin{array}{l} mo \Rightarrow m{:}not\ wk \\ b \Rightarrow m{:}goToBeach \\ mt \Rightarrow p{:}goToMountains \end{array} \right\}$$

The heads of the first two active rules are project internal to Maria. The first rule states that if Maria has money, then she wants to update her own theory with *not wk*. The head of the last active rule is an external project: if Maria wants to travel to the mountains, she proposes to go to the mountains to Pedro (represented with p).

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:

$$\begin{array}{ll} \alpha{:}(L_0 \leftarrow L_1 \wedge \ldots \wedge L_n) & \alpha{:}(L_1 \wedge \ldots \wedge L_n \Rightarrow Z) \\ \alpha{:}(false \leftarrow L_1 \wedge \ldots \wedge L_n \wedge Z_1 \wedge \ldots \wedge Z_m) & \alpha{:}(?{-}L_1 \wedge \ldots \wedge L_n) \end{array}$$

Note that projects and negated projects can only occur in the head of active rules and in the body of integrity constraints. For example, the integrity constraint $false \leftarrow A \wedge \beta{:}B$ in the theory of an agent α prevents it to 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 the project $\beta{:}C$ if A holds and agent β has not wanted to update the theory of α with B.

Example 2. Let Q be the following generalized logic program underlying the theory of Maria.

$$Q = \left\{ \begin{array}{ll} c \leftarrow not\ mt \wedge not\ b \wedge not\ t & (1) \\ wk & (2) \\ vac \leftarrow not\ wk & (3) \\ mt \leftarrow not\ c \wedge not\ b \wedge not\ t \wedge mo & (4) \\ b \leftarrow not\ c \wedge not\ mt \wedge not\ t \wedge mo & (5) \\ t \leftarrow not\ c \wedge not\ mt \wedge not\ b \wedge mo & (6) \end{array} \right\}$$

As Q has a unique 2-valued stable model $\{c, wk\}$, Maria decides to live in the city (c). Things change if we add the rule mo (indicating that Maria possesses money) to Q. In this case, Q has four stable models: $\{c, mo, wk\}$, $\{mt, mo, wk\}$, $\{b, mo, wk\}$ and $\{t, mo, wk\}$. Thus, Maria is now unable to decide where to live.

To incorporate the ability of preferring, we extend the agent language to make it possible to express preferences among the rules of the agent itself. Let $<$ be a binary predicate symbol whose set of constants includes all the generalized rules in the language $\mathcal{L}_\mathcal{K}$.

Definition 7 (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 8 (Prioritized logic program). A prioritized logic program P is a generalized logic program possibly containing 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. For simplicity, we adopt unique numbers for rules instead of the rules themselves in the priority relation $<$.

Example 3. Let Q be the generalized logic program of Example 2 and P the following prioritized logic program:

$$P = Q \cup \left\{ \begin{array}{ll} 1 < 4 \leftarrow wk & 4 < 6 \leftarrow vac \\ 1 < 5 \leftarrow wk & 5 < 6 \leftarrow vac \\ 1 < 6 \leftarrow wk & 6 < 1 \leftarrow vac \end{array} \right\}$$

The first three priority rules say that rule 1 is preferable to the rules 4, 5 and 6 if *wk* holds. If *vac* holds, then the rules 4 and 5 are both preferable to rule 6 which in turn is preferable to rule 1. The intuition is that as *wk* holds while *vac* does not, the first three priority rules are used to characterize the preferred model $\{c, wk\}$ of P. Thus, by reducing the number of models of a program, priority rules allow us to select among several alternative models.

As priority rules are generalized rules, preferences can also be updated. For example, we can update the priority rules of the program P in the previous example by updating with the rule $4{<}5{\leftarrow}vac$. In this case, P will have a unique model also in case *vac* holds.

3 Agents and Multi-Agent Systems

This section presents the conception of 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 pair (P, R), where P is a prioritized logic program and R is a set of active rules.

P formalizes the initial knowledge state of the agent, and R characterizes its reactive behavior. The knowledge of an agent can dynamically evolve when the agent receives new knowledge, albeit by self-updating rules. The new knowledge is represented in the form of an updating program.

Definition 10 (Updating program). An updating program U is a finite set of updates.

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 a sequence of updating programs. A *sequence of updating programs* $\mathcal{U} = \{U^s \mid s \in S \text{ and } s > 0\}$ is a set of updating programs U^s superscripted by the set $S = \{0, 1, \ldots, m, \ldots\}$. We call the elements $s \in S$ *states*.

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 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 0, 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 $i-1$.

Definition 12 (Multi-agent system). 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.

To begin with, the system starts at state 0 where each agent is defined by $\Psi^0_\alpha = (\mathcal{T}_\alpha, \{\})$. Suppose that at state 0 an agent β wants to propose an update of the knowledge state of agent α with C by executing the project $\alpha : C$. Then, at the next common 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 $(\mathcal{T}_\alpha, \{U^1\})$, where $U^1 = \{\beta \div C\}$ if no other updates occur for α.

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 α with C, we write the project α:C.

4 Semantics of Agents and Multi-Agent Systems

To introduce the declarative semantics of agents and a multi-agent system, we need the following definitions.

Definition 13 (Default assumptions). Let P be a prioritized logic program and M an interpretation of P. The set of default assumptions is:

$$\mathit{Default}(P, M) = \{\mathit{not}\ \ A \mid \nexists r \in P \text{ such that } \mathit{head}(r) = A \text{ and } M \models \mathit{body}(r)\}.$$

To begin with, the knowledge of an agent α is characterized by its initial theory $\mathcal{T}$. Its knowledge can dynamically evolve when α receives new knowledge, via a sequence of updating programs $\mathcal{U} = \{U^1, \ldots, U^s\}$. Every updating program U^i in $\mathcal{U}$ contains the updates proposed at state $i-1$ to α by the agents, including itself. 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 [3], the rules proposed via updates can be added to the knowledge state of α provided that α does not distrust the update and without bothering whether they conflict with previous knowledge of α itself. For example, if 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 generalized rules.

Definition 14 (Rejected generalized rule). Let P be a prioritized logic program and M an interpretation 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 generalized rules at state s is:

$RejectGr(P, \mathcal{U}, s, M) =$

$\{r \in P \mid r \text{ and } r' \text{ are generalized rules and } \exists\, \alpha \div r' \in U^i \text{ such that } 1 \leq i \leq s,$
$head(r) = not\ \ head(r'), M \models body(r') \text{ and } M \models not\ \ distrust(\alpha \div r')\}$

$\cup\ \{r \mid r \text{ and } r' \text{ are generalized rules, } \exists\, \beta \div r \in U^i \text{ and } \exists\, \alpha \div r' \in U^j \text{ such that}$
$i < j \leq s, head(r) = not\ \ head(r'), M \models body(r') \text{ and } M \models not\ \ distrust(\alpha \div r')\}$

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 an interpretation 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. Mark that: (1) whenever an agent distrusts an update proposed by another agent, the rule proposed is not in force; (2) at state 0 no rule can be rejected, i.e., $RejectGr(P, \mathcal{U}, 0, M) = \{\}$; (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.

Example 4. Let $P = \{a \leftarrow b\}$, $\mathcal{U} = \{U^1\}$ with $U^1 = \{\alpha \div (not\, a \leftarrow c)\}$ and $M = \{c\}$. Then, the set of rejected rules $RejectGr(P, \mathcal{U}, 1, M) = \{a \leftarrow b\}$.

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

Definition 15 (Rejected active rule). Let (P, R) be the initial theory of an agent and M an interpretation 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:

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

$\cup\ \{r \mid r \text{ and } r' \text{ are active rules, } \exists\, \beta \div r \in U^i \text{ and } \exists\, \alpha \div r' \in U^j \text{ such that}$
$i < j \leq s, head(r) = not\ \ head(r'), M \models body(r') \text{ and } M \models not\ \ distrust(\alpha \div r')\}$

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 rules of a program: removing unsupported rules, and removing less preferred rules defeated by the head of some more preferred one. Unsupported 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 an interpretation of P. The set of unsupported generalized rules is:

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

Notice that $M \not\models body_{neg}(r)$ implies that there exists a default atom $not\ A$ in $body_{neg}(r)$ such that $M \models A$.

Definition 17 (Unpreferred generalized rule). Let P be a prioritized logic program and M an interpretation of P. The set of unpreferred generalized rules is:

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

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

Example 5. Let P be the following prioritized logic program:

$$P = \left\{ \begin{array}{ll} a \leftarrow not\ b & (1) \\ b \leftarrow not\ a & (2) \\ 1 < 2 & \end{array} \right\}$$

and $M = \{1 < 2, a\}$. Then, $Unpref(P, M) = \{2\}$. In fact, the rule (1) can be used to unprefer the rule (2) because (1) is preferred to (2) and the head of (1) occurs in the negative body of (2).

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 projects selected to be executed is the set containing the projects of all the active rules that are triggered by a model M.

Definition 18 (Selected project). Let (P, R) be the initial theory of an agent and M an interpretation of P. The set of selected projects is:

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

The following definition presents the notion of preferred stable model of an agent α. Given the initial theory $\mathcal{T} = (P, R)$ of α and the sequence of updating programs $\mathcal{U}$, a preferred 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 priority rules, and all those rules whose updates are not distrusted but that are neither rejected nor unpreferred. The preferred stable model contains also the projects of all the active rules (both in P or proposed via an update) that are selected.

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

1. $\forall r_1, r_2$: if $(r_1 < r_2) \in M$, then $(r_2 < r_1) \notin M$
2. $\forall r_1, r_2, r_3$: if $(r_1 < r_2) \in M$ and $(r_2 < r_3) \in M$, then $(r_1 < r_3) \in M$
3. $M = \text{least}(\mathcal{X} \cup \mathit{Default}(\mathcal{Y}, M) \cup \mathit{Project}(\mathcal{Z}, M))$, where:

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

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

$$\mathcal{Z} = R \cup \{r \mid r \text{ is an active rule and } \exists\, \alpha \div r \in \bigcup_{1 \le i \le s} U^i \text{ such that } M \not\models \mathit{distrust}(\alpha \div r)\} - \mathit{RejectAr}(R, \mathcal{U}, s, M)$$

The definition of preferred stable model semantics is based on the stable model semantics. The use of $\mathit{least}(\ldots)$ for generalized logic programs derives from the following two equivalent definitions [3]. A 2-valued consistent interpretation M is a stable model of a generalized logic program P:
- if M is the least model of the Horn theory $P \cup M^-$, i.e. $M = \mathit{least}(P \cup M^-)$;
- if $M = \{L \mid L \text{ is an atom and } P \cup M^- \vdash L\}$.

In general, an agent α may have several preferred stable models. Note that if $s = 0$ or each updating program in $\mathcal{U}$ is empty, then $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ reduce to:

$$\mathcal{Y} = P \qquad \mathcal{X} = \mathcal{Y} - \mathit{Unpref}(\mathcal{Y}, M) \qquad \mathcal{Z} = R.$$

If (in addition) there are no preference rules in P, $\mathcal{X}$ reduces further to:

$$\mathcal{X} = \mathcal{Y} - \mathit{Unsup}(\mathcal{Y}, M)$$

When there are neither projects nor updates, the semantics reduces to the Preferential semantics of Brewka and Eiter [9]. If updates are then introduced, the semantics generalizes to the Updates and Preferences semantics of Alferes and Pereira [4], which extends Dynamic Logic Programming with preferences. Our semantics takes the latter and complements it with mutual and self-updates by means of active rules and projects, plus queries, within a society of agents.

Example 6. Let $\Psi_m^0 = (\mathcal{T}_m, \{\})$ be the theory of Maria at state 0 with $\mathcal{T}_m = (P, R)$, where P is is the prioritized logic program of Example 3 and R is the set of active rules of Example 1. Then, $M1 = \{c, wk, 1{<}4, 1{<}5, 1{<}6\}$[2] is the unique preferred stable model of agent α at state 0. In fact, we have that:

$$\mathcal{Y} = P \qquad \mathcal{X} = \mathcal{Y} - \mathit{Unsup}(\mathcal{Y}, M1) \qquad \mathcal{Z} = R$$
$$\mathit{Default}(\mathcal{Y}, M1) = \{not\ \ mt, not\ \ b, not\ \ t, not\ \ vac, not\ \ mo\}$$
$$\mathit{Unsup}(\mathcal{Y}, M1) = \mathit{Project}(\mathcal{Z}, M1) = \{\}$$

Thus, it holds that $M1 = \mathit{least}(\mathcal{Y} \cup \mathit{Default}(\mathcal{Y}, M1))$.

Example 7. Let $\Psi_m^1 = (\mathcal{T}_m, \{U^1\})$ be the theory of Maria at state 1, where $\mathcal{T}_m$ is the initial theory of Maria given in Example 6 and $U^1 = \{l \div mo\}$ stating that Maria has received the information from the lottery (represented with l) that she has received some money. Then, $M2 = \{c, wk, l \div mo, mo, 1{<}4, 1{<}5, 1{<}6, m{:}not\ \ wk\}$ is the unique preferred stable model of Maria at state 1. In fact, we have that:

$$\mathcal{Y} = P \cup \{l \div mo, mo\} \qquad \mathcal{X} = \mathcal{Y} - \mathit{Unpref}(\mathcal{Y}, M2) \qquad \mathcal{Z} = R$$
$$\mathit{Default}(\mathcal{Y}, M2) = \{not\ \ mt, not\ \ b, not\ \ t, not\ \ vac\}$$
$$\mathit{RejectGr}(P, \{U^1\}, 1, M2) = \mathit{RejectAr}(P, \{U^1\}, 1, M2) = \{\}$$
$$\mathit{Unsup}(\mathcal{Y}, M2) = \{\}$$
$$\mathit{Unpref}(\mathcal{Y}, M2) = \{4, 5, 6\}$$
$$\mathit{Project}(\mathcal{Z}, M2) = \{m{:}not\ \ wk\}$$

Thus, it holds that $M2 = \mathit{least}(\mathcal{X} \cup \mathit{Default}(\mathcal{Y}, M2))$.

Not always do the priority rules allow us to select exactly one model. This may happen when the priority rules do not specify a complete linear order among the alternative choices as shown by the following example.

Example 8. Let $\Psi_m^2 = (\mathcal{T}_m, \{U^1, U^2\})$ be the theory of Maria at state 2, where $\mathcal{T}_m$ and U^1 are the same as in Example 7 and $U^2 = \{m \div not\ \ wk\}$. Then, at state 2 Maria has two preferred stable models:

$$M3 = \{vac, mt, l \div mo, m \div not\ \ wk, mo, 4{<}6, 5{<}6, 6{<}1, p{:}goToMountains\}$$
$$M4 = \{vac, b, l \div mo, m \div not\ \ wk, mo, 4{<}6, 5{<}6, 6{<}1, m{:}goToBeach\}$$

[2] Note that we do not write default atoms in models. Thus, a model $\{a, not\ b\}$ is written as $\{a\}$.

$M3$, for example, is a preferred stable model since we have that:

$\mathcal{Y} = P \cup \{l \div mo, m \div not\ wk, mo, not\ wk\}$ $\quad \mathcal{X} = \mathcal{Y} - \{1,2,6\}$ $\quad \mathcal{Z} = R$

$Default(\mathcal{Y}, M3) = \{not\ c, not\ b, not\ t\}$

$RejectGr(P, \{U^1, U^2\}, 2, M3) = \{2\}$

$RejectAr(P, \{U^1, U^2\}, 2, M3) = \{\}$

$Unsup(\mathcal{Y} - \{2\}, M3) = \{\}$

$Unpref(\mathcal{Y} - \{2\}, M3) = \{1, 6\}$

$Project(\mathcal{Z}, M3) = \{p{:}goToMountains\}$

Thus, it holds that $M3 = least(\mathcal{X} \cup Default(\mathcal{Y}, M3) \cup Project(\mathcal{Z}, M3))$.

If the priority rules do specify a complete linear order among the alternative choices then there will exist a unique preferred stable model. Consider the following situation where Maria receives the update $p \div (4{<}5 \leftarrow vac)$ from her friend Pedro.

Example 9. Let $\Psi_m^3 = (\mathcal{T}_m, \{U^1, U^2, U^3\})$ be the theory of Maria at state 3, where $\mathcal{T}_m$, U^1 and U^2 are the same as in Example 8 and $U^3 = \{p \div (4{<}5 \leftarrow vac)\}$. As the set of unpreferred rules at state 3 is $\{1, 5, 6\}$, Maria has a unique preferred stable model:

$$M5 = \{\, vac, mt, l \div mo, m \div not\ wk, p \div (4{<}5 \leftarrow vac), mo, 4{<}5, 4{<}6, 5{<}6, 6{<}1, \\ p{:}goToMountains \,\}$$

We can now present the semantics $\mathcal{S}$ of a multi-agent system $\mathcal{M}$. 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 stable models of each agent of $\mathcal{M}$. It uses the function γ defined below. Let $\{M_{\alpha_1}^{s-1}, \ldots, M_{\alpha_n}^{s-1}\}$ be a set of the set of preferred stable models of the agents $\alpha_1, \ldots, \alpha_n$ at state $s-1$.

$$\gamma(\{M_{\alpha_1}^{s-1}, \ldots, M_{\alpha_n}^{s-1}\}) = \{U_{\alpha_1}^s, \ldots, U_{\alpha_n}^s\}$$

where an update $(\alpha_j \div C) \in U_{\alpha_i}^s$ iff the project $\alpha_i{:}C$ belongs to all the models of α_j at state $s-1$, that is: $(\alpha_i{:}C) \in \bigcap M_{\alpha_j}^{s-1}$, for $1 \leq i, j \leq n$. The intuition is that α_j at state s can have several preferred stable models, each of which may trigger distinct active rules (and therefore each model will contain distinct selected projects). According to γ the projects that will be executed are the selected projects that occur in every model of α_j at state s. Thus, the definition of γ characterizes a cautious behavior for agents.[3] Note that when $j = i$ we have an internal update: the agent chooses to update its own theory.

[3] An alternative definition for γ would be to execute all the projects that occur in any model of α_j at state s: brave behavior.

Definition 20 (Semantics of $\mathcal{M}$). Let $\mathcal{M} = \{\Psi^0_{\alpha_1}, \ldots, \Psi^0_{\alpha_n}\}$ be a 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.

Base Case ($s = 0$) That means that $\Psi^0_{\alpha_i} = (\mathcal{T}_{\alpha_i}, \{\})$, for every agent α_i at state 0, with $1 \leq i \leq n$. The semantics $\mathcal{S}^o$ of $\mathcal{M}$ at state 0 is $\mathcal{S}^0 = \{M^0_{\alpha_1}, \ldots, M^0_{\alpha_n}\}$, where every $M^0_{\alpha_i}$ is the set of preferred stable models of the agent α_i at state 0.

Inductive step ($s > 0$) 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 $1 \leq i \leq n$. If $\gamma(\mathcal{S}^{s-1}) = \{U^s_{\alpha_1}, \ldots, U^s_{\alpha_n}\}$, 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 $M^s_{\alpha_i}$ be the set of preferred stable models of an agent α_i at state s, for $1 \leq i \leq n$. Then, $\mathcal{S}^s = \{M^s_{\alpha_1}, \ldots, M^s_{\alpha_n}\}$.

Note that the interaction among the agents of the multi-agent system $\mathcal{M}$ is defined synchronously: at each state of the system all the agents have to synchronize and execute their projects. In [11] we elaborate over this work and set forth a logic based non-synchronous multi-agent system, and provide its semantics.

5 Happy Story

This section describes the example "happy story" of Section 1 through a multi-agent system $\mathcal{M}$. The system $\mathcal{M}$ has three agents: Maria (m), Pedro (p) and the lottery (l). To keep the example simple, we illustrate how the system works mainly from the perspective of Maria. At state 0, $\mathcal{M}$ and its semantics $\mathcal{S}$ are:

$$\mathcal{M}^0 = \{\Psi^0_m, \Psi^0_p, \Psi^0_l\} \qquad \mathcal{S}^0 = \{M^0_m, M^0_p, M^0_l\}$$

where $\Psi^0_m = (\mathcal{T}_m, \{\})$ is the same as in Example 6. As the unique preferred stable model of Maria at state 1 is $M1$ (cf. Example 6), that is $M^0_m = \{M1\}$, and $M1 \models c$, Maria decides to live in the city. Suppose that Maria receives the information from the lottery that she has received some money, that is, $l \div mo$:

$$\gamma(\{M^0_m, M^0_p, M^0_l\}) = \{U^1_m, U^1_p, U^1_l\} = \{\,\{l \div mo\}, \{\}, \{\}\,\}$$

Then, at state 1 $\mathcal{M}$ is:

$$\mathcal{M}^1 = \{\Psi^1_m, \Psi^1_p, \Psi^1_l\}$$

with $\Psi^1_m = (\mathcal{T}_m, \{U^1_m\})$, where $U^1_m = \{l \div mo\}$. As Maria does not distrust agent l, at state 1 she believes mo and therefore she proposes the internal update m:*not wk* by triggering the first active rule in R. In fact, the theory of Maria has a unique preferred stable model, $M^1_m = \{M2\}$ (cf. Example 7) and $M2 \models m$:*not wk*. Suppose that both p and l at state 1 do not have any executable project. Then, we have that:

$$\gamma(\{M^1_m, M^1_p, M^1_l\}) = \{U^2_m, U^2_p, U^2_l\} = \{\,\{m \div not\ wk\}, \{\}, \{\}\,\}$$

Thus, at state 2 the theory of Maria is $\Psi_m^2 = (\mathcal{T}_m, \{U_m^1, U_m^2\})$ with $U_m^2 = \{m \div not\ wk\}$ whose two preferred stable models are $M_m^2 = \{M3, M4\}$ (cf. Example 8). Consequently, Maria is unable to take any action at all (cautious behavior). Now suppose that Maria is advised by her friend Pedro to prefer mountains to beaches for vacation:

$$\gamma(\{M_m^2, M_p^2, M_l^2\}) = \{U_m^3, U_p^3, U_l^3\} = \{\ \{p \div (4{<}5 \leftarrow vac)\}, \{\}, \{\}\ \}$$

At state 3 we have that $\Psi_m^3 = (\mathcal{T}_m, \{U_m^1, U_m^2, U_m^3\})$ with $U_m^3 = \{p \div (4{<}5 \leftarrow vac)\}$ whose unique preferred stable models is $M_m^3 = \{M5\}$ (cf. Example 9). Thus, Maria can finally go to the mountains as $M5 \models p{:}goToMountains$.

6 Related Work and Concluding Remarks

The use of computational logic for modeling multi-agent systems has been widely investigated (e.g., see [1] and [20] for a roadmap). One approach close to our is the agent-based architecture proposed by Kowalski and Sadri [14] which aims at reconciling rationality and reactivity. Agents are logic programs that continuously perform the "observe-think-act" cycle. The thinking component consists in explaining the observations, generating actions in response to observations, and planning to achieve its goals. The reacting component is defined via a proof-procedure which exploits integrity constraints.

Another approach close to our is DALI multi-agent system proposed by Costantini [10]. DALI is a language and environment for developing logic agents and multi-agent systems. DALI agents are rational agents that are able of reactive and proactive behavior. These abilities rely and are implemented on the notion of events.

The IMPACT system [6] represents the beliefs of an agent by a logic-based program and integrity constraints. Agents are equipped with an action base describing the actions they can perform. Rules in the program generalize condition-action rules by including deontic modalities to indicate for instance that actions are permitted and forbidden. Integrity constraints, as in our approach, specify situations that cannot arise and actions that cannot be performed concurrently. Alternative actions can be executed in reaction to messages.

The BDI approach [19] is a logic-based formalism to represent agents. In this approach, an agent is characterized by its *beliefs*, *desires* (i.e., objectives it aims at), and *intentions* (i.e., plans it commits to). Beliefs, desires and intentions are represented via modal operators with a possible world semantics.

Another logic-based formalism proposed for representing agents is Agent0 [21]. In this approach, an agent is characterized by its *beliefs* and *commitments*. Commitments are production rules that can refer to non-consecutive states of the world in which the agent operates. Both the BDI and the Agent0 approach use logic as a tool for representing agents, but rely upon a non-logic-based execution model. This causes a wide gap between theory and practice in these approaches [20].

We have presented a logical framework of a multi-agent system in which each agent can communicate with and update other agents and is able to prefer beliefs and reactions when several alternatives are possible. Applications in which our agent technology can have a significant potential to contribute, and that we are considering, are Web applications, e.g., information integration and web-site management. To this aim we have designed an agent architecture [16] whose implementation is under way. 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 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. [18]).

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.).

Future work will be devoted to investigate possible generalizations of the multi-agent system and to make it dynamic in a way that the agents can enter and leave the system. Currently, we are working on an implementation of our agent framework. In [2] we have developed a syntactical transformation for updating and preferring reasoning that is at the basis for a proof procedure, and we have proved the properties of the transformation. The size of the transformed program in the worst case is quadratic on the size of the original program. The agent framework is implemented as follows: its logical parts (e.g., logical reasoning, preferring, updating, etc.) are implemented in XSB Prolog [8], while its non-logical parts (e.g., agent communication, user interface, etc.) are implemented in Java. We then use InterProlog [7] to interface Java and the XSB system.

Acknowledgements

This work was partially supported by POCTI project 40958 "FLUX - FleXible Logical Updates". Dell'Acqua acknowledges a Research Grant by the *Stiftelsen Lars Hiertas Minne*.

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. Invited paper in The Asso-

ciation for Logic Programming Newsletter, 14(3), August 2001. Available at http://centria.di.fct.unl.pt/∼lmp.
2. J. J. Alferes, P. Dell'Acqua, and L. M. Pereira. A compilation of updates plus preferences. To appear in: *Logic in Artificial Intelligence* (Jelia'02), LNAI, 2002.
3. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The J. of Logic Programming*, 45(1-3):43–70, 2000. A short version titled *Dynamic Logic Programming* appeared in A. Cohn and L. Schubert (eds.), *KR'98*, Morgan Kaufmann.
4. J. J. Alferes and L. M. Pereira. Updates plus preferences. In M. O. Aciego, I. P. de Guzmán, G. Brewka, and L. M. Pereira (eds.), *Logics in AI, Procs. JELIA'00*, LNAI 1919, pp. 345–360, Berlin, 2000. Springer.
5. J. J. Alferes, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, and P. Quaresma. Preliminary exploration on actions as updates. Proc. of Joint Conf. on Declarative Programming (APPIA-GULP-PRODE'99).
6. IMPACT system. Available at http://www.cs.umd.edu/projects/impact.
7. InterProlog. Available at http://www.declarativa.com/InterProlog/default.htm.
8. XSB-Prolog. Available at http://xsb.sourceforge.net.
9. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109:297–356, 1999.
10. S. Costantini. Towards active logic programming. Electronic Proc. of COCL'99, Second Int. Workshop on Component-Based Software Development in Computational Logic, 1999.
11. P. Dell'Acqua, U. Nilsson, and L. M. Pereira. A logic based asynchronous multi-agent system. Computational Logic in Multi-Agent Systems (CLIMA02). To appear in Electronic Notes in Theoretical Computer Science (ENTCS).
12. P. Dell'Acqua and L. M. Pereira. Enabling agents to update their knowledge and to prefer. In P. Brazdil and A. Jorge (eds.), *Progress in Artificial Intelligence, 10th Portuguese Int. Conf. on Artificial Intelligence (EPIA'01)*, LNAI 2258, pp. 183–190. Springer-Verlag, 2001.
13. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen (eds.), *ICLP'88*, pp. 1070–1080. MIT Press, 1988.
14. R. A. Kowalski and F. Sadri. Towards a unified agent architecture that combines rationality with reactivity. In D. Pedreschi and C. Zaniolo (eds.), *Logic in Databases, Intl. Workshop LID'96*, LNCS 1154, pp. 137–149. Springer, 1996.
15. E. Lamma, F. Riguzzi, and L. M. Pereira. Strategies in combined learning via logic programs. *Machine Learning*, 38(1/2):63–87, 2000.
16. J. A. Leite, J. J. Alferes, and L. M. Pereira. Minerva - A Dynamic Logic Programming Agent Architecture. In J. J. Meyer and M. Tambe (eds.), Pre-Procs. of the 8th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL'01), pp. 133-145, Seattle, USA, 2001.
17. J. A. Leite and L. M. Pereira. Iterated logic program updates. In J. Jaffar, editor, *Proc. Joint Intl. Conf. Symp. on Logic Programming 1998*, pp. 265–278, Cambridge, Mass., 1998. MIT Press.
18. A. Y. Levy and D. S. Weld. Intelligent internet systems. *Artificial Intelligence*, 118:1–14, 2000.
19. A. S. Rao and M. P. Georgeff. Modelling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall (eds.), *Proc. of Knowledge Representation and Reasoning (KR&R-92)*, pp. 473–484. Morgan Kaufmann, 1991.
20. F. Sadri and F. Toni. Computational logic and multiagent systems: a roadmap, August 2001. Available at http://www2.ags.uni-sb.de/net/Forum/Supportdocs.html.
21. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.