Planning Experiments in the DALI Logic Programming Language*

Stefania Costantini Arianna Tocchio

Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito, I-67010 L'Aquila - Italy
{stefcost, tocchio}@di.univaq.it

Abstract. We discuss some features of the new logic programming language DALI for agents and multi-agent systems, also in connection to the issues raised in [12]. We focus in particular on the treatment of proactivity, which is based on the novel mechanism of the *internal events* and *goals*. As a case-study, we discuss the design and implementation of an agent capable to perform simple forms of planning. We demonstrate how it is possible in DALI to perform STRIPS-like planning without implementing a meta-interpreter. In fact a DALI agent, which is capable of complex proactive behavior, can build step-by-step her plan by proactively checking for goals and possible actions.

1 Introduction

The new logic programming language DALI [2], [4], [3] has been designed for modeling Agents and Multi-Agent systems in computational logic. Syntactically, DALI is close to the Horn clause language and to Prolog. In fact, DALI can be seen as a "Prolog for agents" in the sense that it is a general-purpose language, without prior commitment to a specific agent architecture. Rather, DALI provides a number of mechanisms that enhance the basic Horn-clause language to support the "agent-oriented" paradigm.

The definition of DALI has been meant to be a contribution to the understanding of what the agent-oriented paradigm may mean in computational logic. In fact, in the context of a purely logic semantics and of a resolution-based interpreter, some new features have been introduced: namely, events can be considered under different perspectives, and there is a careful treatment of proactivity and memory. In his new book [12], R. A. Kowalski discusses at length, based on significant examples, the principles and techniques an intelligent logical agent should be based upon. In this paper we will argue that DALI, although developed independently, is able to cope with many of the issues raised in [12].

DALI programs may contain a special kind of rules, reactive rules, aimed at interacting with an external environment. The environment is perceived in the form of

* We acknowledge the support by MIUR 40% project *Aggregate- and number-reasoning for computing: from decision algorithms to constraint programming with multisets, sets, and maps* and by the *Information Society Technologies programme of the European Commission, Future and Emerging Technologies* under the IST-2001-37004 WASP project.

external events, that can be exogenous events, observations, or messages from other agents. In response, a DALI agent can either perform actions or send messages. This is pretty usual in agent formalisms aimed at modeling reactive agents (see among the main approaches [10], [6], [7] [21], [20], [24]).

There are however in DALI some aspects that can hardly be found in the above-mentioned approaches. First, the same external event can be considered under different points of view: the event is first perceived, and the agent may reason about this perception; then a reaction can take place; finally, the event and the (possible) actions that have been performed are recorded as past events and past actions. The language has advanced proactive features, on which we particularly focus in this paper.

The new approach proposed by DALI is compared to other existing logic programming languages and agent architectures such as ConGolog, 3APL, IMPACT, METATEM, AgentSpeak in [4]. It is useful to remark that DALI is meant to be a general-purpose language, and thus does not commit to any specific agent architecture. Differently from other significant approaches, e.g., DESIRE [9], DALI agents do not have pre-defined submodules. Thus, different possible functionalities (problem-solving, cooperation, negotiation, etc.) and their interactions must be implemented specifically for the particular application. DALI is not directly related to the BDI approach, although its proactive mechanisms allow BDI agents to be implemented.

The declarative semantics of DALI, briefly summarized in Section 4, is an *evolutionary semantics*, where the meaning of a given DALI program P is defined in terms of a modified program P_s, where reactive and proactive rules are reinterpreted in terms of standard Horn Clauses. The agent receiving an event/making an action is formalized as a program transformation step. The evolutionary semantics consists of a sequence of logic programs, resulting from these subsequent transformations, together with the sequence of the models of these programs. Therefore, this makes it possible to reason about the "state "of an agent, without introducing explicitly such a notion, and to reason about the conclusions reached and the actions performed at a certain stage. Procedurally, the interpreter simulates the program transformation steps, and applies an extended resolution which is correct with respect to the model of the program at each stage.

The proactive capabilities of DALI agents, on which we concentrate in this paper, are based on considering (some distinguished) internal conclusions as events, called "internal events": this means, a DALI agent can "think" about some topic, the conclusions she takes can determine a behavior, and, finally, she is able to remember the conclusion, and what she did in reaction. Whatever the agent remembers is kept or "forgotten" according to suitable conditions (that can be set by directives). Then, a DALI agent is not a purely reactive agent based on condition-action rules: rather, it is a reactive, proactive and rational agent that performs inference within an evolving context.

An agent must be able to act in a goal-oriented way, to solve simple planning problems (regardless to optimality) and to perform tasks. To this aim, we have introduced a subclass of internal events, namely the class of "goals", that once invoked are attempted until they succeed, and then expire. For complex planning tasks however, from DALI rules it is possible to invoke an Answer Set Solver [23]. In fact, Answer Set Programming [17] [16] (based on the Answer Set Semantics of [13] [14]) is a new logic

programming paradigm particularly well-suited for planning. In particular, given (in a file) a knowledge base describing actions, constraints and the goal to be reached, the solver returns possible plans in the form of *Answer Sets*, each of them containing the composing steps of a single plan. The DALI agent can then choose among the Answer Sets according to her criteria.

To demonstrate the usefulness of the "internal event" and "goal" mechanisms, we consider as a case-study the implementation of STRIPS-like planning. We will show that it is possible to design and implement this kind of planning without defining a meta-interpreter like is done in [18] (Ch. 8, section on Planning as Resolution). Rather, each feasible action is managed by the agent's proactive behavior: the agent checks whether there is a goal requiring that action, sets up the possible subgoals, waits for the preconditions to be verified, performs the actions (or records the actions to be done if the plan is to be executed later), and finally arranges the postconditions.

The paper is organized as follows. In Section 2 we summarize how we have understood the discussion in [12]; in Section 3 the language syntax, main constructs and their use are illustrated; in Section 4 the evolutionary semantics is briefly recalled; in Section 5 we present the case-study, and finally in Section 6 we conclude.

2 How to be Artificially Intelligent, the Logical Way

This is the topic treated at length by R. A. Kowalski in his new book [12], which is aimed at understanding which principles an intelligent logical agent should be based upon. According to our understanding of the interesting and deep discussion reported there, there are some important features and functionalities that any approach to agents in computational logic should include, mainly the following:

- Being able of forward reasoning, for interacting with the external world: on the one hand, for going from perceptions to goals; on the other hand, for going from goals or candidate actions to actions.
- Making a distinction between high-level *maintenance goals* and *achievement goals* Maintenance goals constitute the "consciousness " of what the agent has to fulfill in order to stay alive, keep herself in an acceptable state, be able to perform her main tasks. Achievement goals are needed in order to reach maintenance goals, and can in turn leave to low-level subgoals. The step between a maintenance goal and the achievement goals that are needed in order to make it done is in principle a step of forward reasoning. Instead, achievement goals can be coped with by means of backward reasoning, unless they are low-level, and thus require a forward reasoning step for making actions that affect the world.
- Combining different *levels of consciousness*. An agent is computationally conscious when she is aware of what she is doing and why she is doing it, which means that her behaviour is described by means of an high-level program, which manipulates symbols that have meaningful interpretations in the environment. Equivalently, an agent is logically conscious when her behaviour is generated by reasoning with goals and beliefs. Consciousness must be suitably combined with lower-level input-output associations or condition-action rules, which can also be represented

as goals in logical form. Wishfully, it should be possible to compile and de-compile between high-level and low-level representations.
- Keeping track of time, both to timestamp externally observed events and to compare the current time with the deadlines of any internally derived future actions.
- Keeping memory of past observations, so as to be able to generate hypothetical beliefs, to explain the past and predict the future.
- Coping with a changing world, possibly by an approach focused on the occurrence of events and on the effect of events on local states of affairs, such as the Event Calculus [11].

In the rest of this paper we will argue that DALI, although developed independently, is able to cope with many of the issues raised in [12].

3 DALI

DALI is a logic programming agent-oriented language, aimed at a declarative specification of agents and multi-agent systems. While describing the main features of DALI, we will try to focus where these features find a convergence with the points raised in [12] that we have reported above.

A DALI program is syntactically very close to a traditional Horn-clause program. In fact, a Horn-clause program is a special case of a DALI program. Specific syntactic features have been introduced to deal with the agent-oriented capabilities of the language, and in particular to deal with events, actions and goals.

Having been designed for defining agents and multi-agent systems, DALI has been equipped with a communication architecture [5]. For the sake of interoperability, the DALI communication protocol is FIPA compliant, where FIPA (Foundation for Intelligent Physical Agents) is the most widely acknowledged standard for Agent Communication Languages. We have implemented the relevant FIPA primitives, plus others which we believe to be suitable in a logic setting. We have designed a meta-level where: on the one hand the user can specify, via two distinguished primitives tell/told, constraints on communication and/or a communication protocol; on the other hand, meta-rules can be defined for filtering and/or understanding messages via applying ontologies and forms of commonsense and case-based reasoning. These forms of meta-reasoning are automatically applied when needed by form of *reflection* [1].

3.1 Events

Let us consider an event arriving to the agent from its "external world", like for instance $bell\_ringsE$ (postfix E standing for "external"). From the agent's perspective, this event can be seen in different ways.

Initially, the agent has perceived the event, but she still has not reacted to it. The event is now seen as a present event $bell\_ringsN$ (postfix N standing for "now"). She can at this point reason about the event: for instance, she concludes that a visitor has arrived, and from this she realizes to be happy.

visitor_arrived :- *bell_ringsN*.
happy :- *visitor_arrived*.

Then, the reaction to the external event $bell\_ringsE$ consists in going to open the door. This is specified by the following *reactive rule*. The new token $:>$ used instead of :- emphasizes that this rule performs forward reasoning, and is activated by the occurrence of the event which is in the head.

bell_ringsE $:>$ *go_to_open*.

About opening the door, there are two possibilities: one is that the agent is dressed already, and thus can perform the action directly (*open_the_doorA*, postfix A standing for "action"). The other one is that the agent is not dressed, and thus she has to get dressed before going. The action *get_dressedA* has a defining rule. This is just a plain horn rule, but in order to emphasize that it has the role of specifying the preconditions of an action, the new token $:<$ is used instead of :- .

go_to_open :- *dressed*, *open_the_doorA*.
go_to_open :- *not dressed*,
get_dressedA, *open_the_doorA*.
get_dressed $:<$ *grab_clothes*.

DALI makes a distinction between low level *reaction* to the external events, and high-level *thinking* about these events. Since thinking and reacting are in principle different activities, we have introduced the two different *points of view* of the same event: as an external event to be reacted to, and as a present event to be conscious of. Then, when coping with external events DALI is able to combine, as advocated in [12], different "levels of consciousness": high-level reasoning performed on present events, that may lead the agent to revise or augment her beliefs; low-level forward reasoning for reaction.

DALI keeps track of time, since all events are timestamped. As we will see later, DALI also keeps track of events and actions that occurred in the past. The timestamp can be explicitly indicated when needed, and omitted when not needed. I.e., for any timestamped expression $Expr$, one can either write simply $Expr$, or $Expr : T$. External events and actions are used also for sending and receiving messages [5].

3.2 Proactivity in DALI

The basic mechanism for providing proactivity in DALI is that of the *internal events*. Namely, the mechanism is the following: an atom A is indicated to the interpreter as

an internal event by means of a suitable directive. If A succeeds, it is interpreted as an event, thus determining the corresponding reaction. By means of another directive, it is possible to tell the interpreter that A should be attempted from time to time: the directive also specifies the frequency for attempting A, and the terminating condition (when this condition becomes true, A will be not attempted any more).

Thus, internal events are events that do not come from the environment. Rather, they are predicates defined in the program, that allow the agent to introspect about the state of her own knowledge, and to undertake a behavior in consequence. This mechanism has many uses, and also provides a mean for gracefully integrating object-level and meta-level reasoning. It is also possible to define priorities among different internal events, and/or constraints stating for instance that a certain internal event is incompatible with another one. Internal events start to be attempted when the agent is activated, or upon a certain condition, and keep being attempted (at the specified frequency) until the terminating condition occurs. The syntax of a directive concerning an internal event p is the following:

try p [since *SCond*] [frequency f] [until *TCond*].

It states that: p should be attempted at a frequency f; the attempts should start whenever the initiating condition *SCond* becomes true; the attempts should stop as soon as the terminating condition *TCond* becomes true. All fields are optional. If all of them are omitted, then p is attempted at a default frequency, as long as the agent stays alive.

Whenever p succeeds, it is interpreted as an event to which the agent may react, by means of a *reactive rule*:

$$pI :> R_1, \ldots, R_n.$$

The postfix I added to p in the head of the reactive rule stands for "internal", and the new connective $:>$ stands for *determines*. The rule reads: "if the internal event pI has happened, pI will determine a reaction that will consist in attempting $R_1, \ldots, R_n$". The reaction may involve making actions, or simply reasoning on the event.

Internal events are the DALI way of implementing *maintenance goals*. The relevant aspects of the agent's state are continuously kept under control. In fact, repeatedly attempting an internal event A means checking whether a condition that must be taken care of has become true. Success of A triggers a reaction: by means of a step of forward reasoning, the DALI agent goes from the internal event to whatever needs to be done in order to cope with it.

Frequency and priorities are related to the fact that there are conditions that are more critical then others, and or that evolve differently with time.

The reasons why A may fail in the first place and succeed later may be several. As a possible reason, the agent's internal state may change with time:

$$time\_to\_go\_home \text{ :- } time(T), T >= 17{:}00pm.$$
$$time\_to\_go\_homeI :> stop\_work, go\_to\_bus\_stopA, take\_busA.$$

Or, the agent's internal state may change with time, given her internal rules of functioning (below, she gets hungry after some time from last meal), which may imply setting achievement goals (*get_food*) and making actions (*eat_food*):

$$\begin{aligned} \mathit{hungry} &:\!\!- \mathit{time(T)}, \mathit{time\_last\_meal(T1)}, \mathit{finished\_energy(T,T1)}. \\ \mathit{hungryI} &:\!\!> \mathit{get\_food}, \mathit{eat\_foodA}. \end{aligned}$$

Notice that the reaction to an internal event corresponding to a maintenance goal resembles what in the BDI approach is called an "intention", i.e., the act of taking measures in order to reach a desirable state.

Another reason why an internal event may initially fail and then succeed is that the state of the external world changes with time. In the example below, a meal is ready as soon as the cooking time has elapsed. The definition uses timestamps: the agent knows when the soup was previously (postfix P) put on fire from the enclosed timestamp, and can thus estimate whether the cooking time has elapsed:

$$\begin{aligned} \mathit{soup\_ready} &:\!\!- \mathit{soup\_on\_fireP:T}, \\ &\quad\ \mathit{cooking\_time(soup,K)}, \mathit{time\_elapsed(T,K)}. \\ \mathit{soup\_readyI} &:\!\!> \mathit{take\_off\_pan\_from\_stoveA}, \mathit{turn\_off\_the\_fireA}. \end{aligned}$$

Or also, there may be new observations that make the internal event true:

$$\begin{aligned} \mathit{ready(cake)} &:\!\!- \mathit{in\_the\_oven(cake)}, \mathit{color(cake,golden)}, \mathit{smell(cake,good)}. \\ \mathit{readyI(cake)} &:\!\!> \mathit{take\_from\_oven(cake)}, \mathit{switch\_off\_the\_oven}, \mathit{eat(cake)}. \end{aligned}$$

Or, the reasoning about a present event may lead to a conclusion or to a new belief, that may trigger further activity. In a previous example, we have the conclusion *happy*, drawn from the present event *bell_rings* (but in general, this conclusion will be possibly drawn from several other conditions). It may be reasonable to consider "happiness" as a relevant aspect of the agent's state, and thus interpret predicate *happy* as an internal event, that causes a reaction (e.g., a smile) whenever true.

$$\begin{aligned} \mathit{visitor\_arrived} &:\!\!- \mathit{bell\_ringsN}. \\ \mathit{happy} &:\!\!- \mathit{visitor\_arrived}. \\ \mathit{happyI} &:\!\!> \mathit{smileA}. \end{aligned}$$

This shows that internal events not only can model maintenance goals, but can also model a kind of "consciusness" or "introspection" of the agent about her private state of affairs. When internal events are used in this way, the definition of the reaction specifies a sort of "individuality" of the agent, i.e., a kind of peculiar behaviour not strictly related to a need.

3.3 Past Events

The agent remembers events and actions, thus enriching her reasoning context. An event (either external or internal) that has happened in the past will be called *past event*, and written $bell\_ringsP$, $happyP$, etc., postfix P standing for "past". Similarly for an action that has been performed. It is also possible to indicate to the interpreter plain conclusions that should be recorded as *past conclusions* (which, from a declarative point of view, are just lemmas). Past events are time-stamped, i.e., they are actually stored in the form: $predP : timestamp$.

Then, past events can remind the agent of:

- An external event that has happened; in this case, the time-stamp refers to the moment in time when the agent has reacted to the event.
- An internal event that has taken place; also in this case, the time-stamp refers to reaction
- An action that has been performed; the time-stamp refers to when the action has been done.
- A conclusion that has been reached; the time-stamp records when.

It is important to notice that an agent cannot keep track of *every* event and action for an unlimited period of time, and that, sometimes, subsequent events/actions can make former ones no more valid. Then, we must equip an agent with the possibility to remember, but also to forget things.

According to the specific item of knowledge, the agent may want:

- To remember it forever.
- To forget it after a certain time.
- To forget it as as soon as subsequent knowledge makes it no more valid.

Moreover, if the recorded item concerns an event that may happen several time (i.e., $rain$) the agent may want:

- To remember all the occurrences.
- To remember only some of them, according to some conditions (the simplest one is a time-interval).
- To remember only the last occurrence.

In essence, there is a need to express forms of meta-information about the way in which the agent manages her knowledge base. Modifying this meta-information makes the behavior of the agent different. However, these aspects cannot be expressed in the agent logic program, which is a first-order Horn theory. Nor we want to hardwire them in the implementation.

Then, we have introduced the possibility of defining *directives*, that are by all means part of the specification of an agent, but are not part of the logic program. They are an input to the interpreter, and can be modified without altering (and even without looking at) the logic program, in order to "tune " the agent behavior.

Then, all the above-mentioned conditions can be specified via directives. Examples of directives are the following:

$keep$ *predP* $until$ *Time*.

where $PredP$ is removed at the time $Time$,

$keep$ *predP* $until$ *Condition*.

where $PredP$ is removed when $Condition$ becomed true,

$keep$ *predP* *forever*.

where $PredP$ is never removed (think as an example to the birth-date of people, as a kind of information that never expires).

As a default, just the last occurrence of $PredP$ is kept, with its time-stamp, thus overriding previous ones. A directive can however alter this behavior, and the agent can look for various versions (for the sake of simplicity, we do not detail this point here). In the agent program, when referring to $PredP$ the agent implicitly refers to the last version.

If the directives for keeping/removing past events/actions/conclusions are specified carefully, we can say that the set of the last versions of past events/actions/conclusions constitutes an implicit representation of the *frame axiom*. This because this set represents what has happened/has been concluded, and has not been affected yet by what has happened later in the agent evolution.

Past events, past conclusions and past actions, which constitute the "memory" of the agent, are an important part of the (evolving) context of an agent. Memories make the agent aware of what has happened, and allow her to make predictions about the future.

It is interesting to notice that DALI management of past events allows the programmer to easily define *Event Calculus* expressions. The Event Calculus (EC) has been proposed by Kowalski and Sergot [11] as a system for reasoning about time and actions in the framework of Logic Programming. The essential idea is to have terms, called *fluents*, which are names of time-dependent relations. Kowalski and Sergot write $holds(r(x, y), t)$ which is understood as "fluent $r(x, y)$ is true at time t".

Take for instance the default inertia law formulated in the event calculus as follows:

holds(f,t) $\leftarrow$ *happens(e)*,
initiates(e,f),
date(e,t_s),
t_s $<$ *t*,
not clipped(t_s,f,t)

where $clipped(t_s, f, t)$ is true when there is record of an event happening between t_s and t that terminates the validity of f. In other words, $holds(f, t)$ is derivable whenever in the interval between the initiation of the fluent and the time the query is about, no terminating events has happened.

In DALI, assuming that the program contains suitable assertion for $initiates$ as well as the definition of $clipped$, this law could be immediately reformulated as follows. We just reinterpret $Happens(e), date(e, t_s)$ as a lookup in the knowledge base of past events, where evp finds an event E with its timestamp T_s (where, in this case, T_s initiates fluent f):

holds(f,T) :- *evp(E,T_s)*,
initiates(E,T),
T_s $<$ *t*,
not Clipped(T_s,f,T)

The representation can be enhanced by defining $holds$ as an internal event. This means, the interpreter repeatedly attempts to prove $holds(f, T)$. Upon success, a reactive rule can state what to do in consequence of this conclusion. Then, $holds(f, T)$ will be recorded as a past event $holdsP(f, T)$, thus creating a temporal database where $holds$ atoms are kept, and possibly removed according to the associated directives.

3.4 Goals

A special kind of internal event is a *goal*. Differently from the other internal events, goals start being attempted either when encountered during the inference process, or when invoked by an external event. Each goal G will be automatically attempted until it succeeds, and then expires. Moreover, if multiple definitions of G are available, they are (as usual) applied one by one by backtracking, but success of one alternative prevents any further attempt. attempt. DALI goals are a way of implementing [12] *achievement goals*, that must be attempted whenever needed, and possibly decomposed into subgoals.

We have implemented goals (postfix G) on top of internal events, by exploiting a practically useful role of past conclusions: i.e., that of allowing one to eliminate subsequent alternatives of a predicate definition upon success of one of them. Assume that the user has designated predicate q as a conclusion to be recorded (it will be recorded with syntax qP). Then, she can state that only one successful alternative for q must be considered (if any), by means of the following definition:

q :- *not qP*, $\langle def_1 \rangle$.
...
q :- *not qP*, $\langle def_n \rangle$.

Coming back to goals, whenever $goalG$ becomes true, a reaction may be triggered, by means of an (optional) reactive rule:

$$goalGI :> R_1, \ldots, R_k$$

A slightly different postfix, namely GI, is used to distinguish the head of the reactive rule, so as to visually remark that this internal event is in particular a goal. After reaction, the goal is recorded as a past event $goalP$, so as the agent is aware that it has been achieved. If there is no reactive rule, the past event is recorded as soon as $goalG$ becomes true. This past event may in turn allow other internal events or goals to succeed, and so on. Then, a DALI agent is in constant evolution.

Goals can be used in a planning or problem-solving mechanism, for instance by employing the following schema.

RULE 1: goal prerequisites

$$\begin{aligned} goalG \ :- \ & condition_1, \ldots, condition_k(1) \\ & subgoalG_1, \ldots, subgoalG_n(2) \\ & subgoalP_1, \ldots, subgoalP_n(3) \end{aligned}$$

RULE 2: goal achievement

$$goalGI :> actionA_1, \ldots, actionA_m$$

where:

part (1) of Rule 1 verifies the preconditions of the goal;
part (2) of Rule 1 represents the invocation of the subgoals;
part (3) of Rule 1 verifies that previously invoked subgoals have been achieved (they have become past conclusions);
Rule 2 (optional) performs the actions which are needed to achieve the present goal, and to set its postconditions.

The reason why $goalG$ must be attempted repeatedly by Rule 1 is that, presumably, in the first place either some of the preconditions will not hold, or some of the subgoals will not succeed. The reason why part 3 of Rule 1 is needed is that each of the subgoals has the same structure as the overall goal. I.e., first its prerequisites have to succeed by Rule 1, and then it is actually achieved by the reaction in Rule 2 (if present), and finally becomes a past event. Then, by looking for past events part 3 checks that the subgoals have been properly achieved.

If the given goal is part of a problem-solving activity, or if it is part of a task, then the reaction may consist in directly making actions. In planning, the reaction may consist in updating the plan (by adding to it the actions that will have to be performed whenever the plan will be executed).

For convenience, a conjunction *goalG*,*goalP* that attempts a goal and waits for it to be achieved is denoted by the shorthand *goalD*, *D* standing for "done". Then, the above rules can be rewritten more shortly as:

RULE 1: goal prerequisites

$$goalG \ \text{:-} \ condition_1, \ldots, condition_k \\ subgoalD_1, \ldots, subgoalD_n.$$

RULE 2: goal achievement

$$goalGI \ :> \ actionA_1, \ldots, actionA_m$$

Also, it is possible to associate a timeout to the goal: by writing *goalD:T* we say that if the goal has not been achieved within the given time period T, then it fails.

Notice that the mechanism of DALI goals fulfills the structure advocated in [12] for achievement goals: there is a backward reasoning part in Rule 1, that possibly splits the goal into subgoals; there is (if needed) a forward reasoning part in Rule 2, for performing actions.

An easy improvement, demonstrated below, copes with situation where there are goals, and the agent may want to achieve as many of them as possible, regardless to the others.

$$many\_goals \ \text{:-} \ condition_1, \ldots, condition_k, goalsG. \\ goalsG \ \text{:-} \ goalD_1{:}T_1 :: \ldots :: goalD_n{:}T_n.$$

On the invocation of *goalsG*, the interpreter invokes all goals in the body of the rule. The body succeeds if at least one of them succeeds. This mechanism is composable, in the sense that any of the $goalG_i$'s can in turn be defined in this way.

Conceptually, there is a declarative rewriting of the above rule, taking profit of the fact that if there are alternative definitions for *goalG*, then the first successful alternative is taken. One should then specify as many rules as the possible combinations.

The examples that we propose in the ongoing for STRIPS-like planning are aimed at showing the power, generality and usability of DALI internal events and goals.

3.5 Coordinating Actions based on Context

A DALI agent builds her own context, as suggested in [12], by keeping track of the events that have happened in the past, and of the actions that she has performed. As discussed above, whenever an event (either internal or external) is reacted to, whenever an action subgoal succeeds (and then the action is performed), and whenever a distinguished conclusion is reached, this is recorded in the agent knowledge base.

Past events and past conclusions are indicated by the postfix P, and past actions by the postfix PA. The following rule for instance says that Susan is arriving, since we

know her to have left home.

is_arriving(susan) :- *left_homeP(susan)*.

The following example illustrates how to exploit past actions. We consider an agent who opens and closes a switch upon a condition. For the sake of simplicity we assume that no exogenous events influence the switch. The action of opening (resp. closing) the switch can be performed only if the switch is closed (resp. open). The agent knows that the switch is closed if she remembers to have closed it previously. The agent knows that the switch is open if she remembers to have opened it. Predicates $open$ and $close$ are internal events, that periodically check the opening/closing condition, and, whenever true, perform the action (if feasible). previously.

open :- *opening_cond*.
openI :> *open_switchA*.
open_switchA :< *switch_closed*.
switch_closed :- *close_switchPA*.
close :- *closing_cond*.
closeI :> *close_switchA*.
close_switchA :< *switch_open*.
switch_open :- *open_switchPA*.

In the example, the agent will remember to have opened the switch. However, as soon as she closes the switch this record becomes no longer valid and should be removed: the agent in this case is interested to remember only the last action of a sequence. As soon as the $until$ condition is fulfilled, i.e., the corresponding subgoal has been proved, the past action is removed. Then, the suitable directives for past actions will be in this case the following:

$keep$ *open_switchPA* $until$ *close_switchA*.

$keep$ *close_switchPA* $until$ *open_switchA*.

The following example illustrates the use of actions with preconditions. The agent emits an order for a product $Prod$ of which she needs a supply. The order can be done either by phone or by fax, in the latter case if a fax machine is available. We want to express that the order can be done either by phone or by fax, but not both, and we do that by exploiting past actions, and say that an action cannot take place if the other one has already been performed. Here, not is understood as default negation.

need_supplyE(Prod) :> *emit_order(Prod)*.

emit_order(Prod) :- *phone_order*A*(Prod)*,
not *fax_order*PA*(Prod)*.
emit_order(P) :- *fax_order*A*(Prod)*,
not *phone_order*PA*(Prod)*.

This can be reformulated in a more elaboration-tolerant way by the constraints:
:- *fax_order*A*(Prod)*, *phone_order*PA*(Prod)*
:- *fax_order*PA*(Prod)*, *phone_order*A*(Prod)*
thus eliminating negations from the body of the action rules.

4 Semantics

The DALI interpreter can answer user queries like the standard Prolog interpreter, but in general it manages a disjunction of goals. In fact, from time to time external and internal event will be added (as new disjuncts) to the current goal. The interpreter extracts the events from queues where they occur in the order in which they have been generated.

All the features of DALI that we have previously discussed are modeled in a declarative way. For a full definition of the semantics the reader may refer to [4]. We summarize the approach here, in order to make the reader understand how the examples actually work.

Some language features do not affect at all the logical nature of the language. In fact, attempting the goal corresponding to an internal event just means trying to prove something. Also, storing a past event just means storing a lemma.

Reaction and actions are modeled by suitably modifying the program. This means, inference is performed not in the given program, but in a modified version where language features are reformulated in terms of plain Horn clauses.

Reception of an event is modeled as a program transformation step. I.e., each event that arrives determines a new version of the program to be generated, and then we have a sequence of programs, starting from the initial one. In this way, we do not introduce a concept of state which is incompatible with a purely logic programming language. Rather, we prefer the concept of program (and model) evolution.

More precisely, we define the declarative semantics of a given DALI program P in terms of the declarative semantics of a modified program P_s, obtained from P by means of syntactic transformations that specify how the different classes of events/conclusions/actions are coped with. For the declarative semantics of P_s we take the Well-founded Model, that coincides with the the Least Herbrand Model if there is no negation in the program (see [19] for a discussion). In the following, for short we will just say "Model". It is important to notice that P_s is aimed at modeling the declarative semantics, which is computed by a bottom-up immediate-consequence operator. The declarative semantics will then correspond to the top-down procedural behavior of the interpreter.

We assume that events which have happened are recorded as facts. We have to formalize the fact that a reactive rule is allowed to be applied only if the corresponding

event has happened. We reach our aim by adding, for each event atom $p(Args)E$, the event atom itself in the body of its own reactive rule. The meaning is that this rule can be applied by the immediate-consequence operator only if $p(Args)E$ is available as a fact. Precisely, we transform each reactive rule for external events:

$p(Args)E \;\; :> \;\; R_1, \ldots, R_q.$

into the standard rule:

$p(Args)E \;\; :- \;\; p(Args)E, R_1, \ldots, R_q.$

In a similar way we specify that the reactive rule corresponding to an internal event $q(Args)I$ is allowed to be applied only if the subgoal $q(Args)$ has been proved.

Then, we have to declaratively model actions, without or with an action rule. An action is performed as soon as its preconditions are true *and* it is invoked in the body of a rule, such as:

$B \;\; :< \;\; D_1, \ldots, D_h, aA_1, \ldots, aA_k. \quad h \geq 1, k \geq 1$

where the aA_i's are actions and the D_j's are not actions. Then, for every action atom aA, with action rule

$aA \;\; :- \;\; C_1, \ldots, C_s. \quad s \geq 1$

we modify this rule into:

$aA \;\; :- \;\; D_1, \ldots, D_h, C_1, \ldots, C_s.$

If aA has no defining clause, we instead add clause:

$aA \;\; :- \;\; D_1, \ldots, D_h.$

We repeat this for every rule in which aA is invoked.

In order to obtain the *evolutionary* declarative semantics of P, we explicitly associate to P_s the list of the external events that we assume to have arrived up to a certain point, in the order in which they are supposed to have been received. We let $P_0 = \langle P_s, [] \rangle$ to indicate that initially no event has happened.

Later on, we have $P_n = \langle Prog_n, Event\_list_n \rangle$, where $Event\_list_n$ is the list of the n events that have happened, and $Prog_n$ is the current program, that has been obtained from P_s step by step by means of a *transition function* Σ. In particular, Σ specifies that, at the n-th step, the current external event E_n (the first one in the event list) is added to the program as a fact. E_n is also added as a present event. Instead, the previous event E_{n-1} is removed as an external and present event, and is added as a past event.

Formally we have:

$\Sigma(P_{n-1}, E_n) = \langle \Sigma_P(P_{n-1}, E_n), [E_n | Event\_list_{n-1}] \rangle$

where

$\Sigma_P(P_0, E_1) = \Sigma_P(\langle P_s, [] \rangle, E_1) = P_s \cup E_1 \cup E_{1N}$

$\Sigma_P(\langle Prog_{n-1}, [E_{n-1}|T] \rangle, E_n) =$

$\{\{Prog_{n-1} \cup E_n \cup E_{nN} \cup E_{n-1P}\} \setminus E_{n-1N}\} \setminus E_{n-1}$

It is possible to extend Σ_P so as to deal with internal events, add as facts past actions and conclusions, and remove the past events that have expired.

Definition 1. *Let P_s be a DALI program, and $L = [E_n, \ldots, E_1]$ be a list of events. Let $P_0 = \langle P_s, [] \rangle$ and $P_i = \Sigma(P_{i-1}, E_i)$ (we say that event E_i* determines *the transition*

from P_{i-1} to P_i). The list $\mathcal{P}(P_s, L) = [P_0, \ldots, P_n]$ is the program evolution *of P_s with respect to L.*

Notice that $P_i = \langle Prog_i, [E_i, \ldots, E_1] \rangle$, where $Prog_i$ is the program as it has been transformed after the ith application of Σ.

Definition 2. *Let P_s be a DALI program, L be a list of events, and PL be the* program evolution *of P_s with respect to L. Let M_i be the Model of $Prog_i$. Then, the sequence $\mathcal{M}(P_s, L) = [M_0, \ldots, M_n]$ is the* model evolution *of P_s with respect to L, and M_i the* instant model at step i .

The evolutionary semantics of an agent represents the history of the events received by the agents, and of the effect they have produced on it, without introducing a concept of a "state". It is easy to see that, given event list $[E_n, \ldots, E_1]$, DALI resolution simulates standard SLD-Resolution on $Prog_n$.

Definition 3. *Let P_s be a DALI program, L be a list of events. The* evolutionary semantics *$\mathcal{E}_{P_s}$ of P_s with respect to L is the couple $\langle \mathcal{P}(P_s, L), \mathcal{M}(P_s, L) \rangle$.*

The behaviour of DALI interpreter has been modeled and checked with respect to the evoltionary semantics by using the Murϕ model checker [8].

5 A sample application: STRIPS-like planning

In this section we show that the DALI language allows one to define an agent that is able to perform planning (or problem-solving) in a STRIPS-like fashion, without implementing a metainterpreter.

For the sake of simplicity, the planning capabilities that we consider are really basic, e.g., we do not consider here the famous STRIPS anomaly, and we do not have any pretense of optimality.

We consider the sample task of putting on socks and shoes. Of course, the agent should put her shoes on her socks, and she should put both socks and both shoes on.

We suppose that some other agent sends a message to ask our agent to wear the shoes. This message is an external event, which is the head of a reactive rule: the body of the rule specifies the reaction, which in this case consists in invoking the goal put_your_shoesG.

goE :> *put_your_shoesG*.

This goal will be attempted repeatedly, until it will be achieved.

It is important to recall the mechanism of DALI goals:

– For a goal g to be achieved, first of all the predicate gG must become true, by means of a rule gG :- $Conds$, where $Conds$ specify preconditions and subgoals.

– For a goal g to be achieved, as soon as gG becomes true the (optional) reactive rule $gGI :> PostAndActions$ is activated, that performs the actions and/or sets the postconditions related to the goal.
– as soon as a goal gG is achieved (or, in short, we say that gG *succeeds*, even though this involves the above two steps), it is recorded as a past event, in the form gP.
– the conjunction gG, gP that invokes a goal and waits for it to be achieved is denoted by gD.

This explains the structure of the rule below:

put_your_shoesG :- *put_right_shoeD*, *put_left_shoeD*.

In particular, it is required that the agent puts both the right and left shoe on. This means, *put_your_shoesG* will become true as soon as both of its subgoals will have been achieved. In practice, after the invocation of the subgoals, the overall goal is suspended until the subgoals become past events.

In the meantime, the subgoals *put_right_shoeG* and *put_left_shoeG* will be attempted.

put_right_shoeG :- *have_right_shoe*, *put_right_sockD*.

This rule verifies a precondition, i.e., that of having the shoe to put on. Then it attempts the subgoal *put_right_sockG* and waits for its success, i.e., waits for the subgoal to become a past event. The rule for the subgoal is:

put_right_sockG :- *have_right_sock*.

This rule doesnt invoke subgoals, but it just checks the precondition, i.e., to have the right sock. Upon success, the corresponding reactive rule is triggered:

put_right_sockGI :> *right_sock_on*.

Now we have two possibilities: in a problem-solving activity, we will have the rule:

right_sock_on :- *wear_right_sockA*.

that actually executes the action of wearing the sock.

In a planning activity, we will have instead the rule:

right_sock_on :- *update_plan(wear_right_sock)*.

that adds to the plan that is being built the step of wearing the sock. In any case, the goal *put_right_sockG* has been achieved, and will be now recorded as past event, and thus *put_right_sockD* becomes true. Consequently, also *put_right_shoeG* becomes true, thus triggering the reactive rule:

put_right_shoeGI :> *right_shoe_on*.

After having made (or recorded) the action of wearing the shoe, *put_right_shoeP* will become true, thus obtaining *put_right_shoeD*.

Analogously, the agent will eventually record the past event *put_left_shoeP*, thus obtaining *put_left_shoeD*. Since the subgoals of wearing the right and the left shoe are unrelated, no order is enforced on their execution. Only, the overall goal becomes true whenever both of them have been achieved.

At this point, the reactive rule related to the overall goal will be activated:

put_your_shoesGI :> *message(tell_shoes_onA)*.

which means that the goal has succeeded, and in particular the agent declares to have the shoes on.

The planning mechanism that we have outlined consists of a descendant process that invokes the subgoals, and of an ascending process that executes (or records) the corresponding actions.

This methodology allows an agent to construct plans dynamically. In fact, a change of the context, i.e., new information received from the outside, can determine success of subgoals that could not succeed before.

A future direction of this experimental activity is that of writing a meta-planner with general meta-definitions for root, intermediate and leaf goals. This meta-planner would accept a list of goals, with the specification of their kind, and for each of them the list of preconditions, subgoals and actions.

Below we show an example of use of conditional goal rules, where
gG :- $Conds, gD_1 :: , \ldots , , :: gD_n$
means, as previously discussed, that if $Conds$ are true, then the body of the rule succeeds provided that at least one of the gD_i's succeeds (though the interpreter invokes them all, and tries to achieve as many as possible). The point is that the failure of non-critical partial objectives does not determine the failure of the overall goal. The example consists in an agent that has to go to the supermarket in order to buy milk and bananas, and to the hardware shop in order to buy a drill. He tries to go both to the supermarket and to the hardware shop. However, if one of them is closed he just goes to the other

one. In each shop, he tries to buy what he needs, without making a tragedy if something is missing. There is a failure (and then the agent is disappointed) only if either both shops are closed, or all items are missing.

$$
\begin{aligned}
\mathit{buy} &\ \texttt{:-}\ \mathit{buy\_allD}.\\
\mathit{disappointed} &\ \texttt{:-}\ \mathit{not\ buy}.\\
\mathit{buyI} &\ :\!>\ \mathit{go\_homeA}.\\
\mathit{disappointedI} &\ :\!>\ \mathit{some\_reaction}.\\
\mathit{buy\_allG} &\ \texttt{:-}\ \mathit{buy\_at\_supermarketD} :: \mathit{buy\_at\_hardware\_shopD}.\\
\mathit{buy\_at\_supermarketG} &\ \texttt{:-}\ \mathit{supermarket\_open},\\
&\qquad \mathit{buy\_bananasD} :: \mathit{buy\_milkD}.\\
\mathit{buy\_at\_hardware\_shopG} &\ \texttt{:-}\ \mathit{hardware\_shop\_open},\\
&\qquad \mathit{buy\_drillG}.
\end{aligned}
$$

6 Conclusions

We have presented how to implement a naive version of STRIPS-like planning in DALI, mainly by using the mechanism of internal events and goals. However, the ability of DALI agents to behave in a "sensible" way comes from the fact that DALI agents have several classes of events, that are coped with and recorded in suitable ways, so as to form a context in which the agent performs her reasoning. In fact, we have argued that DALI fulfills many of the points raised in [12] about which features any logical formalism aimed at defining intelligent agents should possess.

A simple form of knowledge update and "belief revision" is provided by the conditional storing of past events, past conclusions and past actions. They constitute the "memory" of the agent, and are an important part of her evolving context: memories make the agent aware of what has happened, and allow her to make predictions about the future. The ability of specifying how long and under which conditions memories should be kept allows the agent behavior to be specified in a more sophisticated and flexible way. For the sake of flexibility and of conceptual clarity, the directives that cope with the knowledge base of the agent memories are distinct from the agent logic program. In the future however, more sophisticated belief revision strategies will be integrated into the formalism.

DALI is fully implemented in Sicstus Prolog [22]. The implementation, together with a set of examples, is available at the URL http://gentile.dm.univaq.it/dali/dali.htm.

7 Acknowledgments

Many thanks to Stefano Gentile, who joined the DALI project, cooperates to the implementation of DALI, has designed the language web site, and has supported the authors

in many ways. We also gratefully acknowledge Prof. Eugenio Omodeo for useful discussions and for his support to this research.

References

1. J. Barklund, S. Costantini, P. Dell'Acqua e G. A. Lanzarone, *Reflection Principles in Computational Logic*, Journal of Logic and Computation, Vol. 10, N. 6, December 2000, Oxford University Press, UK.
2. S. Costantini, *Towards active logic programming*, In A. Brogi and P. Hill, (eds.), *Proc. of 2nd International Works. on Component-based Software Development in Computational Logic (COCL'99)*, PLI'99, Paris, France, September 1999, http://www.di.unipi.it/ brogi/ ResearchActivity/COCL99/ proceedings/index.html.
3. S. Costantini, S. Gentile and A. Tocchio, *DALI home page:* http://gentile.dm.univaq.it/dali/dali.htm.
4. S. Costantini and A. Tocchio, *A Logic Programming Language for Multi-agent Systems*, In S. Flesca, S. Greco, N. Leone, G. Ianni (eds.), *Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002*, Cosenza, Italy, September 2002, LNAI 2424: Springer-Verlag, Berlin, 2002.
5. S. Costantini, A. Tocchio **(submitted)**. *Communication in the DALI Agent-Oriented Logic Programming Language.* submitted to ICLP'04, International Conference on Logic Programming.
6. P. Dell'Acqua, F. Sadri, and F. Toni, *Communicating agents*, In *Proc. International Works. on Multi-Agent Systems in Logic Progr., in conjunction with ICLP'99*, Las Cruces, New Mexico, 1999.
7. M. Fisher, *A survey of concurrent METATEM – the language and its applications*, In *Proc. of First International Conf. on Temporal Logic (ICTL)*, LNCS 827, Springer Verlag, Berlin, 1994.
8. B. Intrigila, I. Melatti, A. Tocchio, *Model-checking DALI with Murϕ*, Tech. Rep., Univ. of L'Aquila, 2004.
9. C. M. Jonker, R. A. Lam and J. Treur, *A Reusable Multi-Agent Architecture for Active Intelligent Websites. Journal of Applied Intelligence*, vol. 15, 2001, pp. 7-24.
10. R. A. Kowalski and F. Sadri, *Towards a unified agent architecture that combines rationality with reactivity*, In *Proc. International Works. on Logic in Databases*, LNCS 1154,Springer-Verlag, Berlin, 1996.
11. R. A. Kowalski and M. A. Sergot, *A logic-based calculus of events*, New Generation Computing 4, 1986.
12. R. A. Kowalski, *How to be Artificially Intelligent - the Logical Way*, Draft, revised February 2004, Available on line, URL http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html.
13. M. Gelfond and V. Lifschitz, *The Stable Model Semantics for Logic Programming,* In: R. Kowalski and K. Bowen (eds.), Logic Programming: Proc. of 5th International Conference and Symposium, The MIT Press, 1988.
14. M. Gelfond and V. Lifschitz, *Classical Negation in Logic Programming and Disjunctive Databases,* New Generation Computing 9, 1991: 365–385.
15. *How to be Artificially Intelligent the Logical Way,* book drafta (revised February 2004), Available on-line at the URL: http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html

16. V. Lifschitz, *Answer Set Planning,* in: D. De Schreye (ed.) Proc. of the 1999 International Conference on Logic Programming (invited talk), The MIT Press, 1999: 23–37.
17. W. Marek and M. Truszczyński, *Stable Models and an Alternative Logic Programming Paradigm,* In: The Logic Programming Paradigm: a 25-Year Perspective, Springer-Verlag, Berlin, 1999: 375–398.
18. D. Poole, A. Mackworth, R. Goebel, *Computational Intelligence*: ISBN 0-19-510270-3, Oxford University Press, New York, 1998.
19. H. Przymusinska and T. C. Przymusinski, *Semantic Issues in Deductive Databases and Logic Programs*, R.B. Banerji (ed.) Formal Techniques in Artificial Intelligence, a Sourcebook: Elsevier Sc. Publ. B.V. (North Holland), 1990.
20. A. S. Rao, *AgentSpeak(L): BDI Agents speak out in a logical computable language*, In W. Van De Velde and J. W. Perram, editors, *Agents Breaking Away: Proc. of the Seventh European Works. on Modelling Autonomous Agents in a Multi-Agent World*, LNAI: Springer Verlag, Berlin, 1996.
21. A. S. Rao and M. P. Georgeff, *Modeling rational agents within a BDI-architecture*, In R. Fikes and E. Sandewall (eds.), Proc. of Knowledge Representation and Reasoning (KR&R-91): Morgan Kaufmann Publishers: San Mateo, CA, April 1991.
22. SICStus home page: *http://www.sics.se/sicstus/*.
23. Web location of the most known ASP solvers:
aspps: *http://www.cs.uky.edu/ai/aspps/*
CCalc: *http://www.cs.utexas.edu/users/tag/cc/*
Cmodels: *http://www.cs.utexas.edu/users/tag/cmodels.html*
DLV: *http://www.dbai.tuwien.ac.at/proj/dlv/*
NoMoRe: *http://www.cs.uni-potsdam.de/ linke/nomore/*
SMODELS: *http://www.tcs.hut.fi/Software/smodels/*
24. V. S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Özcan, and R. Ross, *Heterogenous Active Agents*: The MIT Press, 2000.

We discuss some features of the new logic programming language DALI for agents and multi-agent systems, also in connection to the issues raised in [12]. We focus in particular on the treatment of...Â Costantini S., Tocchio A. (2004) Planning Experiments in the DALI Logic Programming Language. In: Dix J., Leite J. (eds) Computational Logic in Multi-Agent Systems. CLIMA 2004. GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together. Sign up. An auction program written in the logic programming language DALI. 3 commits. 1 branch. 0 releases. Fetching contributors. Branch: master. New pull request. DALI [3], [2] is an Active Logic Programming Language designed in the line of [6] for executable specification of logical agents. A DALI agent is a logic program that contains a particular kind of rules, reactive rules, aimed at interacting with an external environment. The reactive and proactive behavior of a DALI agent is triggered by several kinds of events: external, internal, present and past events. All the events and actions are timestamped, so as to record when they occurred. The newâ€¦