The DALI Logic Programming Agent-Oriented Language*

Stefania Costantini and 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

1 The DALI language

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 new syntactic entities, i.e., predicates related to events and proactivity, are indicated with special postfixes (which are coped with by a pre-processor) so as to be immediately recognized while looking at a program.

The external events are syntactically indicated by the postfix *E*. When an event comes into the agent from its "external world", the agent can perceive it and decide to react. The reaction is defined by a reactive rule which has in its head that external event. The special token :>, used instead of : −, indicates that reactive rules performs forward reasoning. The agent remembers to have reacted by converting the external event into a *past event* (time-stamped). Operationally, if an incoming external event is recognized, i.e., corresponds to the head of a reactive rule, it is added into a list called EV and consumed according to the arrival order, unless priorities are specified.

The internal events define a kind of "individuality" of a DALI agent, making her proactive independently of the environment, of the user and of the other agents, and allowing her to manipulate and revise her knowledge. An internal event is syntactically indicated by the postfix *I*, and its description is composed of two rules. The first one contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction (in the second rule) may happen.

Internal events are automatically attempted with a default frequency customizable by means of directives in the initialization file. The user directives can tune several parameters: at which frequency the agent must attempt the internal events; how many times an agent must react to the internal event (forever, once, twice,...) and when (forever, when triggering conditions occur, ...); how long the event must be attempted (until some time, until some terminating conditions, forever).

* We acknowledge support by the *Information Society Technologies programme of the European Commission, Future and Emerging Technologies* under the IST-2001-37004 WASP project.

J.J. Alferes and J. Leite (Eds.): JELIA 2004, LNAI 3229, pp. 685–688, 2004.

When an agent perceives an event from the "external world", it does not necessarily react to it immediately: she has the possibility of reasoning about the event, before (or instead of) triggering a reaction. Reasoning also allows a proactive behavior. In this situation, the event is called present event and is indicated by the suffix *N*.

Actions are the agent's way of affecting her environment, possibly in reaction to an external or internal event. In DALI, actions (indicated with postfix *A*) may have or not preconditions: in the former case, the actions are defined by actions rules, in the latter case they are just action atoms. An action rule is just a plain rule, but in order to emphasize that it is related to an action, we have introduced the new token $:<$, thus adopting the syntax $action :< preconditions$. Similarly to external and internal events, actions are recorded as past actions.

Past events represent the agent's "memory", that makes her capable to perform future activities while having experience of previous events, and of her own previous conclusions. Past events, indicated by the suffix *P*, are kept for a certain default amount of time, that can be modified by the user through a suitable directive in the initialization file.

The DALI language has been equipped with a communication architecture consisting of three levels. The first level implements a FIPA-compliant [5] communication protocol and a filter on communication, i.e. a set of rules that decide whether or not to receive or send a message. The DALI communication filter is specified by means of meta-level rules defining the distinguished predicates *tell* and *told*. The second level includes a meta-reasoning layer, that tries to understand message contents, possibly based on ontologies and/or on forms of commonsense reasoning. The third level consists of the DALI interpreter.

The declarative and procedural semantics of DALI, is defined as an *evolutionary semantics*, so as to cope with the evolution of an agent corresponding to the perception of events [3]. The semantics has been generalized so as to include the communication architecture by resorting to the general framework RCL (Reflective Computational Logic) [1] based on the concept of reflection principle.

Following [7] and the references therein, the operational semantics of communication is defined [4] by means of a formal dialogue game framework that focuses on the rules of dialogue, regardless the meaning the agent may place on the locutions uttered. This means, we do not want to refer to the mental states of the participants.

2 The DALI Interpreter

The DALI interpreter has been implemented in Sicstus Prolog, and includes a FIPA-compliant communication library. The DALI interpreter is in principle able to interoperate with other FIPA-compliant platforms. Presently, we have implemented interoperability with JADE, which is one of the best-known non-logical middleware for agents (namely, it is written in java). DALI agents can be distributed on the web, as the implementation of the communication primitives is based on TCP/IP.

The interpreter is composed of three main modules: (i) the *DALI active_server* module, that manages the community of DALI agents; (ii) the *DALI active_user* module, that provides a user interface for the user to interact with the agents; (iii) the *active_dali*

module, that is automatically activated by the active_server whenever an agent is created (then, there are as many copies of the *active_dali* module running as the existing agents).

The DALI/FIPA communication protocol consists of the main FIPA primitives, plus few new primitives which are peculiar of DALI. The code implementing the FIPA primitives is contained in the file *communication_fipa.txt*, imported by agents as a library. The DALI/FIPA communication protocol is implemented by means a piece of DALI code including suitable $tell/told$ rules. Whenever a message is received, with content part *primitive(Content,Sender)* the DALI interpreter automatically looks for a corresponding *told* rule that specifies whether the message should be accepted. Symmetrically, whenever a message should be sent, with content part *primitive(Content,Sender)* the DALI interpreter automatically looks for a corresponding *tell* rule that specifies whether the message can be actually issued. The DALI code defining the DALI/FIPA protocol is contained in a separate predefined file, *communication.txt*, imported by agents as a library. In this way, both the communication primitives and the communication protocol can be seen as "input parameters "of the agent. It is important to notice that the file *communication.txt* can modified by the user by adding new rules to the default ones. Typically, a user will add new application-dependent $tell/told$ rules to the file *communication.txt*. Possibly however, both library files can be replaced by different ones, thus specifying a different communication protocol.

Each DALI agent must be generated by specifying the following parameters. (a) The name of the file that contains the DALI logic program (a .txt file). (b) The name of the agent. (c) The ontology the agent adopts (a .txt file); (d) The language (e.g., Italian or English etc.) used in the communication acts; (e) The name of the file containing the communication constraints, a .txt file; as mentioned, a predefined standard version *communication.txt* is provided. (f) The name of the communication library, a .txt file; as mentioned, a standard version *communication_fipa.txt* is provided. (g) The skills that the agent means to make explicit to the community of DALI agents (e.g., profession, hobbies, etc.). Below is an example of activation of an agent.

```
agent('demo/program/prog',gino,'demo/pippo_ontology.txt',italian,
        ['demo/communication'],['demo/communication_fipa'],[tourist]).
```

From the program file, say *prog.txt*, a pre-processing stage extracts three files. (1) The file *prog.ple*, that contains a list of the special tokens occurring in the agent program, denoting internal and external events, goals, actions, etc. (2) The file *prog.plf*, that contains a list of user-defined directives, that the DALI environment provides for *tuning* the behavior of an agent: the user can decide for instance: the priority among events; how long to keep memory of past events, and/or upon which conditions they must be removed; the starting and terminating conditions for attempting internal events, and the frequency. (3) The file *prog.pl* which contains the code of the agent; it must be noticed that all variables are reified, so as to guarantee safe communication and reliable meta-reasoning capabilities.

3 Case Studies

We are able of course to show many simple examples, aimed at illustrating the basic language features. More complex case studies can however be demonstrated.

To demonstrate the usefulness of the "internal event" and "goal" mechanisms, we have considered as a case-study the implementation of STRIPS-like planning. We can show that it is possible in DALI to design and implement this kind of planning without defining a meta-interpreter. 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.

We can generalize this example to dynamic planning, where an agent is able to recover from unwanted or unexpected situations by suitably modifying its plan.

To explain how the filter level works, we have implemented and experimented a case-study that demonstrates how this filter is powerful enough to express sophisticated concepts such as expressing and updating the *level of trust*. Trust is a kind of social knowledge and encodes evaluations about which agents can be taken as reliable sources of information or services. We focus on a practical issues: namely, how the level of Trust influences communication and choices of the agents.

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. Many references about DALI and PowerPoint presentations can be found at the URLs: http://costantini.di.univaq.it/pubbls_stefi.htm and http://costantini.di.univaq.it/AI2.htm.
3. 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*, (held in Cosenza, Italy, September 2002), LNAI 2424, Springer-Verlag, Berlin, 2002.
4. S. Costantini, A. Tocchio and A. Verticchio, *Semantic of the DALI Logic Programming Agent-Oriented Language*, submitted.
5. FIPA. *Communicative Act Library Specification*, Technical Report XC00037H, Foundation for Intelligent Physical Agents, 10 August 2001.
6. 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.
7. P. Mc Burney, R. M. Van Eijk, S. Parsons, L. Amgoud, *A Dialogue Game Protocol for Agent Purchase Negotiations*, J. Autonomous Agents and Multi-Agent Systems Vol. 7 No. 3, November 2003.