Enhancing computational power: DALI child agents generation[*]

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. In this paper we introduce a novel feature of the DALI language: a DALI agent is now able to activate child agents and to feed them either with a goal to be reached or a result to be obtained. Each child agent is independent and can communicate with its father or with other agents. When the child finally reaches the given goal, it notifies the father. At any point, the latter may possibly decide to stop it. Then: (i) each child is aware of the identity of its father; (ii) each child will notify the father about its achievements; (iii) a child can be stopped by the father; (iv) the father may set a limited amount of time for children's activities completion. We introduce the mechanisms for children generation and the corresponding operational semantics, and then present an example.

1 Introduction

Intelligent agents by using their potentialities are able at least to some extent to overcome problems such as limited computational resources, non-deterministic environment, and insufficient knowledge. When a problem is not naturally multi-agent based, a sole agent is capable of solving it by taking enough computational resources and information about its environment.

In a lot of problem domains however, the context naturally requires several agents to take a role in problem-solving or, more generally, requires the adoption of a multi-agent strategy. A multi-agent system is composed of multiple interacting agents which are typically capable of cooperating to solve problems that are beyond the capabilities of any individual agent. Building a cooperation strategy is not easy: an agent, contrary to an object, can renounce to cooperate or, as emphasized in [5], can reveal itself an unreliable collaborator.

So, when an agent accepts the aid of another one, it implicitly assumes a certain risk degree on its future activity. Can an agent minimize this risk? In some cases the response to this query is 'yes'. Some kind of problems requiring a certain degree of computational power that a single agent cannot provide can be faced not by invoking the collaboration of external agents, but by generating child agents.

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

The difference is relevant: a child agent is reliable and cannot refuse to give assistance. In fact, the basic premise of coordination is that if an agent cannot solve an assigned problem using local resources/expertise, it will decompose the problem into sub-problems and try to find other willing agents with the necessary resources/expertise to solve these sub-problems. By using child agents, the sub-problems assignment is solved by a simple message exchange between father and children without adopting a contracting mechanism. Moreover, the possibility to assign complex tasks to one or more child agents allows the father to keep its energies for more strategic activities. In particular a father agent, by delegating a time-expensive jobs to a child, can maintain a high reactivity degree and respond timely to the changes in the environment. This is a not negligible detail. A limit of this approach is that a child agent cannot resolve tasks that require a knowledge degree that the father agent does not posses, unless the child acquires knowledge autonomously from other external sources.

According to the above considerations, we have introduced in the DALI framework the ability to generate children. An important motivation for this improvement has been the need for our agents to face not-trivial planning problems by means of the invocation of a performant planner, such as for instance an Answer Set solver [7]. The idea of Answer Set Programming [20] is to represent a given computational problem by means of a logic program whose answer sets correspond to solutions and then use an answer set solver, e.g., SMODELS or DLV, to find an answer set for this program. Answer Set Programming has proved to be a strong formalism for planning [12], and thus appears suitable for an integration with DALI. As a planning process can require a significant amount of time to find a solution, the possibility for an agent to assign this time-expensive activity to its children can constitute a real advantage.

Another motivation for generating children is, more generally, that of splitting an agent goal into subgoals to be delegated to children. This possibly with the aim of obtaining different results by means of different strategies, and then comparing the various alternatives and choosing the best ones. The father provides the child with all the information useful to find the solution and, optionally, with an amount of time within which to resolve the assigned problem.

In this paper, we present the details on the child generation capability of DALI agents while the current work to integrate DALI and Answer Set Programming will be presented in forthcoming papers. This paper is organized as follows: in Section 2 we introduce the main functionalities of the DALI language; in Section 3 we explain briefly the DALI communication architecture; in Section 4 we present the Operational Semantics of our language; Section 5 is reserved to outline the child generation mechanism of DALI agents, Section 6 presents the related operational semantics laws, Section 7 shows an example of application. Finally, we conclude this paper with some remarks and discussion of related work.

2 The DALI language

DALI [3] is an Active Logic Programming language designed in the line of [10] for executable specification of logical agents. The reactive and proactive behavior of the

DALI agent is triggered by several kinds of events: external events, internal, present and past events. All the events and actions are timestamped, so as to record when they occurred.

An external event is a particular stimulus perceived by the agent from the environment. In fact, if we define $S = \{s_1 : t_0, ..., s_n : t_k\}$ as the set of external stimuli s_k that the agent received from the world during the interval (t_0, t_k), where the set of "external events" E is a subset of S. In particular, we can define the set of external events as follows:

Definition 1 (Set of External Events). *We define the set of external events perceived by the agent from time t_1 to time t_n as a set $E = \{e_1 : t_1, ..., e_n : t_n\}$ where $E \subseteq S$.*

A single external event e_i is an atom indicated with a particular postfix in order to be distinguished from other DALI language events. More precisely:

Definition 2 (External Event). *An external event is syntactically indicated by postfix E and it is defined as:*
$ExtEvent ::= << Atom_E >> |seq << Atom_E >>$ *where an* Atom *is a predicate symbol applied to a sequence of* terms *and a* term *is either a constant or a variable or a function symbol applied in turn to a sequence of terms.*

External events allow an agent to react through a particular kind of rules, reactive rules, aimed at interacting with the external environment. 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.

Definition 3 (Reactive rule). *A reactive rule has the form:* $ExtEvent_E :> Body$ *or*
$ExtEvent_{1E}, ..., ExtEvent_{nE} :> Body$
where $Body ::= seq << Obj >>$ *and*
$Obj ::= << Action_A >> \mid << Goals_G >> \mid << Atom >> |...$

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 it proactive independently of the environment, of the user and of the other agents, and allowing it to manipulate and revise its knowledge. More precisely:

Definition 4 (Internal Event). *An internal event is syntactically indicated by postfix I:* $InternalEvent ::= << Atom_I >>$
The structure of an internal event is composed by 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:
$IntEvent : -Conditions$
$IntEvent_I :> Body$

where $Conditions ::= seq << Obj\_cond >>$ *and*
$Obj\_cond ::= << PastEvent_P >> \mid << Atom >> \mid << Belief >> \mid ...$
Moreover,
$Body ::= seq << Obj\_body >>$ *and*
$Obj\_body ::= << Action_A >> \mid << Goals_G >> \mid << Atom >> \mid ...$

Internal events are automatically attempted with a default frequency customizable by means of directives in the initialization file. The user's 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).

When an agent perceives an event from the "external world", it does not necessarily react to it immediately: it 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 formalized as follows:

Definition 5 (Present Event). *A present event is syntactically indicated by postfix* N*:*
$PresentEvent ::= << Atom_N >> \mid seq << Atom_N >>$
The syntax of a present event usage is:
$InternalEvent :- PresentEvent_N$
$InternalEvent_I :> Body$
where $Body ::= seq << Object >>$ *and*
$Object ::= << Action_A >> \mid << Goals_G >> \mid << Atom >> \mid ...$

Actions are the agent's way of affecting the environment, possibly in reaction to either an external or internal event. An action in DALI can be also a message sent by an agent to another one.

Definition 6 (Action). *An action is syntactically indicated by postfix* A*:*
$Action ::= << Atom_A >> \mid message_A << Atom, Atom >>$
Actions take place in the body of rules:
$Head :- Body$
where $Body ::= seq << Object >>$ *and*
$Object ::= << Action_A >> \mid << Goals_G >> \mid << Atom >> \mid ...$

In DALI, actions 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 following syntax:

Definition 7 (Action rule). *An action rule has the form:*
$Action :< Preconditions$
where $Preconditions ::= seq << Object >>$ *and*
$Object ::= << PastEvent_P >> \mid << Atom >> \mid << Belief >> \mid ...$

Similarly to external and internal events, actions are recorded as past actions.

A DALI agent is able to build a plan in order to reach an objective, by using internal events of a particular kind, called *planning goals*. A goal has postfix G, and like an internal event is defined by two rules. The first one is attempted when the goal is invoked and activates its subgoals, if any. The second one contains a reaction related to the reached subgoal. The relevant difference between an internal event and a planning goal is that while the former starts being attempted when the agent is born, the latter is attempted when invoked by a rule. A DALI agent is also able to verify if a goal was reached by using a special kind of atom with a postfix T. When the interpreter meets the construct $goal_T$, it checks if a past event $goal_P$ or a fact corresponding to this predicate exists.

Past events represent the agent's "memory", that makes it capable to perform future activities while having experience of previous events, and of its own previous conclusions. Past events are kept for a certain default amount of time, that can be modified by the user through a suitable directive in the initialization file. A past event is formalized as follows:

Definition 8 (Past Event). *A past event is syntactically indicated by the postfix P:*
$PastEvent ::= << Atom_P >>$

3 DALI Communication Architecture

The DALI communication architecture consists of four levels. The first and last levels implement the DALI/FIPA communication protocol and a filter on communication, i.e. a set of rules that decide whether or not receive (*told* check level) or send a message (*tell* check level). The DALI communication filter is specified by means of meta-level rules defining the distinguished predicates *tell* and *told*. Whenever a message is received, with content part *primitive(Content,Sender)* the DALI interpreter automatically looks for a corresponding *told* rule. If such a rule is found, the interpreter attempts to prove $told(Sender, primitive(Content))$. If this goal succeeds, then the message is accepted, and $primitive(Content))$ is added to the set of the external events incoming into the receiver agent. Otherwise, the message is discarded. Symmetrically, the messages that an agent sends are subjected to a check via *tell* rules. 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.

4 Operational Semantics

The operational semantics of DALI system [4] is defined by adopting an approach which is a novelty in the agent world. The novelty in particular is that we use a formal dialogue game in order to define the *full* operational semantics of the DALI interpreter.

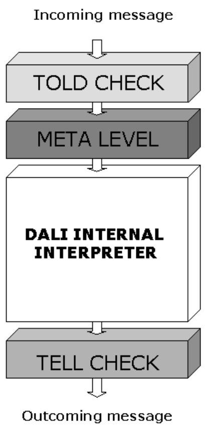

Fig. 1. DALI communication architecture

Recently, formal dialogue games, which have been studied in philosophy since the time of Aristotle, have found application as the basis for interaction protocols between autonomous agents [13] [14]. Dialogue games are formal interactions between two or more participants, in which participants "move" by uttering statements according to pre-defined rules.

Dialogue game protocols have been proposed for agent team formation, persuasion, negotiation over scarce resources, consumer purchase interactions and joint deliberation over a course of action is some situation ([11],[17],[18],[19]) but, to the best of our knowledge, they have not been used up to now to give a formal description of an agent language. In our formalization we assume that the DALI interpreter plays a game and thus makes "moves" not only towards other agents, but also towards itself. By adopting this approach we explain the behavior of each layer of the architecture and their interactions. We define a formal dialogue game framework that focuses on the rules of dialogue, regardless the meaning the agent may place on the locutions uttered. Dialogue games has been applied successfully in negotiation contexts because in these cases is possible to individuate easily players and moves.

The first question that we faced in order to formalize the operational semantics of DALI architecture has been in fact: which are the players and which moves can they make? We considered that the DALI architecture is composed by layers and each layer adopts a specific behavior. A layer can be viewed as a *dark box* whose behavior is determined only by moves of other correlated layers and by its policy. By adopting this view point, our players are the layers and moves are defined through laws and transitions rules.

A strategy for a player is a set of rules that describe exactly how that player should choose, depending on how the other player has chosen at earlier moves. The rules of the operational semantic show how the states of an agent change according to the application of the transition rules. We define a rule as a combination of states and laws. Each law links the rule to the interpreter behavior and is based on the DALI architecture. Our work demonstrates how solutions from game theory together with computing theories can be used to publicly specify rules and prove desirable properties for agent systems. In order to make it clear what we intend for state, law and transition rule, we adopt the following definitions.

Definition 9 (State of a DALI agent). *Let Ag_x be the name of a DALI agent. We define the internal state IS_{Ag_x} of a DALI agent as the tuple $< E, N, I, A, G, T, P >$ composed by its sets of events, actions and goals.*

Definition 10 (Law). *We define a law L_x as a framework composed by the following elements:*

– ***name:*** *the name of law;*
– ***locution:*** *the arguments that the law takes;*
– ***preconditions:*** *the preconditions to apply the law;*
– ***meaning:*** *the meaning of the law;*
– ***response:*** *the effects of the applied law;*

Definition 11 (Transition rule). *A transition rule is described by two pairs and some laws. If the transition is internal to the same agent, a transition rule corresponds to :*
$< Ag_x, < P, IS, Mode >> \xrightarrow{L_i,\ldots,L_j} < Ag_x, < NewP, NewIS, NewMode >>$
Starting from the first pair and by applying the current laws, we obtain the second pair where some parameters have changed. Each pair is defined as $< Ag_x, S_{Ag_x} >$, where Ag_x is the name of the agent and the operational state S_{Ag_x} is the triple $< P_{Ag_x}, IS_{Ag_x}, Mode_{Ag_x} >$. The first argument is the logic program (written in DALI) of the agent, the second one is the internal state, the third one is a particular attribute describing what the interpreter is doing. $NewP$, $NewIS$ and $NewMode$ indicate, respectively, P, IS and $Mode$ updated after applying $L_i, ..., L_j$ laws .

A transition rule can also describe how an agent can influence an other one. In this case, we will have:
$< Ag_x, < P_{Ag_x}, IS_{Ag_x}, Mode_{Ag_x} >> \xrightarrow{L_i,\ldots,L_j} < Ag_y, < P_{Ag_y}, IS_{Ag_y}, Mode_{Ag_y} >>$
$wherex \neq y$

The operational semantics viewed with the eyes of game theory transforms TOLD filter into TOLD player, META level into META player, and so on until TELL filter that becomes TELL player. Also the DALI internal interpreter becomes a player that plays with the other structural player and with itself. What will we expect from these players? Their behavior is surely cooperative because only if all levels work together, a DALI agent will satisfy the user expectations. The players are not malicious because our game is innocent and does not involve any competition strategy. So, we expect each player to follow deterministically the laws and rules and produces a set of moves admissible. These moves will influence the other players and will determine the global game.

When does a player win? The game that an agent plays with itself and with the other agents is innocent, so we do not intend define rigorously the concept of winner.

Our winner is the player which play with success a specific game. More precisely, we intend, after defining the general operational semantics, to prove some relevant properties of DALI language. For us, each property that must be demonstrated is a particular game that a player must face through defined the laws and rules. A player wins if plays successfully a game/property proposed. Next sections will describe the ability of DALI agents to generate children.

5 Child generation capability

A DALI agent is able to activate child agents and to feed them either with a goal to be reached or a result to be obtained. Each child agent is independent and can communicate with its father or with other agents. When the child finally reaches the given goal, it notifies the father. At any point, the latter may possibly decide to stop it. This will mostly happen either after obtaining results, or when the time amount that the father means to allocate to the child's task has expired. Then: (i) each child is aware of the identity of its father; (ii) each child will notify the father about its achievements; (iii) a child can be stopped by the father; (iv) the father may set a limited amount of time for children's activities completion.

Apart from that, a child agent is a DALI one, equipped with its own knowledge base, directives and communication filter, and can in turn create children. This feature is relevant for DALI multi-agent system scalability. From a cognitive point of view, it allows the father for instance to: compute and then compare various alternative plans (or intentions in the BDI view); perform hypothetical reasoning; create its own local social setting in the form of a society of agents, each one with its role and commitment. The resulting architecture, useful to DALI agents to generate children, is divisible in three modules, each of which offers specific functionalities. The first module allows a father agent to create children, the second one establishes a connection between father and child, the third one determines the child life time.

5.1 Create children

This first module allows each DALI agent to activate, through a specific action, one or more children. The new generated agent can include, according to the fatherly will, either the knowledge base of the father or a different knowledge base KB specified at the generation moment. If the child incorporates the father logic program and knowledge, the action able to create it will be:

– $create_A(Num\_Children)$**,** where $Num\_Children$ specifies how many agents the father intends to generate.

The KB specification implies that the child agent will have the knowledge and logic program contained in the specified file:

– $create_A(NumFigli, KB)$**,** where $Num\_Children$ has the same meaning specified above and KB specifies the file name containing the knowledge base.

For instance, an agent $party$ who plays the role of a party organizer, can generate the following children:

$create_A(1, c : /kb/cake.txt, ontology\_1).$
$create_A(1, c : /kb/fizz.txt, ontology\_2).$

Children will be named by default $party\_child1$ and $party\_child2$ ($child1$ and $child2$ for the father). The files $cake.txt$ and $fizz.txt$ contain all the activation data (including knowledge bases and, optionally, ontologies) for the two agents.

After this generation process, a child agent will have all potentialities to be able itself to generate further children. In other words, each child agent can become a father one. A particular mechanism avoids child agents to be given the same name. The last step of this module is to check if the activation succeeds: to this aim, the father agent sends a specific message to each child.

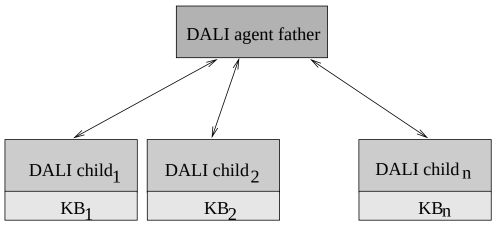

Fig. 2. DALI agent and children

5.2 Connect module

This module provides two functionalities: the first one establishes the connection between father and child agents; the second one allows the father to assign a subgoal to its child that, when it reaches its task, advises the father on its success. As soon as the child agent becomes active, it receives by the father the message: $born(Father\_name)$. Its child keeps in its memory the father name and sends to it the message: $hello\_dad(Child\_name)$ Starting from the moment in which this happens, two agents can communicate between them. When the father reaches the internal conclusion that it is necessary to assign a goal to a child, it sends one of the following messages:

- $solve\_goal(Goal, Ev, Time)$*:* the child has a time limit to resolve its task. The Ev parameter is necessary because the father must trigger specific reactive rules (in the child program) to activate the resolution process;
- $solve\_goal(Goal, Ev)$*:* the child agent does not have a fixed amount of time to return the solution to the father;

The child, as soon as its goal is reached, tells the father through a *confirm* message.

5.3 Lifetime module

This third module kills the child agent when its allocated time has expired. DALI child agents have a specific internal event that checks from time to time if the current agent elapsed time has exceeded the value specified at the generation act. In this case, not only the agent is killed but also its data are erased.

6 Operational semantics of children generation

In this Section we show the operational semantics rules that cope with children generation. In particular, the laws are L19-L24 in the context of the 119 overall transition rules [21].

- **L19: initialize_child(.)** law:
  Locution: $initialize\_child(Logic\_program/KB, Ontology)$
  Preconditions: The agent reaches the conclusion (by an internal event) that it needs a child.
  Meaning: This law allows an agent to generate a child agent. If either Logic_program or Ontology are empty, the generated child will inherit the parameters of the father, else it takes the specified value.
  Response: The agent has a child agent.

- **L20:** The **active_child** law:
  Locution: $active\_child$
  Preconditions: The child agent has been initialized.
  Meaning: This law activates a child agent. After the activation, the child agent enters the "wait" mode and is ready to receive communication acts from the father. Father and child can communicate by using the usual DALI primitives.
  Response: The child agent is active.

- **L21:** The **expired_time_child** law:
  Locution: $expired\_time\_child$
  Preconditions: The time assigned from the father to child is expired.
  Meaning: This law checks the time assigned to the child agent.
  Response: The father informs the child that the time is finished and asks for the results.

- **L22:** The **obtain_result** law:
  Locution: $obtain\_result$
  Preconditions: The time assigned to the child has expired.
  Meaning: The child agent has reached the requested result and it sends it to the father.
  Response: The father obtains the result.

- **L23:** The **not_obtain_result** law:
  Locution: not_obtain_result
  Preconditions: The time assigned to child has expired.
  Meaning: The child agent has not achieved the requested result.
  Response: The father does not obtain the result.

- **L24:** The **kill_child** law:
  Locution: $kill\_child$
  Preconditions: The child agent terminates its job.
  Meaning: The father resets the internal state of the agent and removes it from the environment.
  Response: The child is dead.

7 An example: organizing a party

In this section we show an example in which an agent, having had a promotion, organizes a party in order to offer a cake and a fizz bottle to its friends. To this aim, it identifies two subgoals: to prepare the cake and to buy the bottle. Then, it creates two children in order to assign them the two tasks. We suppose that the internal event triggering the party organization is $organize\_party$:

$$
\begin{aligned}
&organize\_party :- promotion_P.\\
&organize\_party_I :>\\
&\qquad\qquad child\_name(F1, 1),\\
&\qquad\qquad child\_name(F2, 2),\\
&\qquad\qquad message_A(F1, confirm(\\
&\qquad\qquad\qquad solve\_goal(cake\_ready, cake), user)),\\
&\qquad\qquad message_A(F2, confirm(\\
&\qquad\qquad\qquad solve\_goal(fizz\_ready, fizz, 120000), user)).
\end{aligned}
$$

where the $child\_name/2$ predicate is useful to obtain the child agents names. Via the messages $solve\_goal$, the children receive the goals assignment. When the father agent receives the communications from the children that their tasks have been accomplished, it starts the party.

$$
\begin{aligned}
&start\_party :- cake\_ready_P, fizz\_ready_P.\\
&start\_party_I :> write('The\ party\ is\ starting...'), invite\_everyone_A.
\end{aligned}
$$

After the generation, the child agents tell the user about their birth by printing:

Hello World..... My name is party_child1
My father is party

while the father *party*, verified the success of the generation process, writes:

My son is party_child1
My son is party_child2

Once started, children will react to an event of the form $solve\_goal(G)$ coming from their father. In this case, for instance, the father will be able to ask children to prepare a cake and drinks respectively, by means of the messages:

$message_A(child1, confirm(solve\_goal(cake\_ready)).$
$message_A(child1, confirm(solve\_goal(buy\_drinks)).$

The father will be notified by the children when the goal will have been reached, and made aware of results. Notice that the second child has a time limit to give a solution. Below we show the logic programs of two children.

The agent *party_child1* The knowledge base of this agent consists in the *cake.txt* file and contains the following rules:

$cake_E :> preparing\_cake_G.$
$preparing\_cake : -haveFlour_P.$
$preparing\_cake_I :> cake\_ready_A.$

The agent triggers the goal $preparing\_cake_G$ while the *connect module* starts to verify if the assigned time is expired. In order to reach its goal, the agent is in need of flour. If the agent receives the flour, it prepares the cake and informs its father:

make(cake_ready)
send_message_to(party_child1,send_message(cake,party_child1))
send_message_to(party_child1,agree(cake_ready,party_child1))
send_message_to(party_child1,
*inform(agree(cake_ready),**values(yes)**,party_child1))*
Reached Goal: cake_ready
send_message_to(party,confirm(cake_ready,party_child1)).

The agent *party_child2* This agent has the following logic program:

$fizz_E :> buy\_fizz_G.$
$buy\_fizz : -haveMoney_P.$
$buy\_fizz_I :> fizz\_ready_A.$

In order to reach its goal, this child must have sufficient money. In this case, it buys the bottle and advices its father. The last exchanged messages are:

Reached Goal: fizz_ready
send_message_to(party,confirm(fizz_ready,party_child2))

The party is starting After receiving the messages indicating that the subgoals have been reached, the father agent starts the party:

The party is starting...
make(invite_everyone)

8 Conclusions and Related Work

We conclude this discussion with some considerations on DALI agents generation capabilities. The father agent is not required to know the contents of children KB except concerning external events that trigger children activities. Each child is under every respect a DALI agent that can interact with the other entities in the environment and can increase its knowledge independently of the father. The latter can only kill the child when it is no more useful.

While the children work, the father can continue its activity without losing contact with the environment. The father can also assign to children intermediate sub-goals and reorganize the obtained results. Each child can create its own children, thus increasing the computational power of the system. Finally, the time limit allows a system to spare computational resources. The child generation capability that we have presented is the starting point to improve DALI agents computational power: in the future we will make it possible for the father agent to specialize its children by making them import specific library modules. Also, this mechanism can be a useful features in the context of more general coordination frameworks and strategies.

In fact, the DALI communication architecture together with the children generation mechanism constitute a basic support for cooperation that DALI provides. The communication architecture neatly separates an agent's core behavior from the agent's behavior related to communication. The same DALI agent program equipped with a different communication architecture actually results in a different agent, as its relationship with its environment is different, and affects its internal state in a different way. Sub-agents can be employed so as to perform in a distributed fashion different specific tasks. These features combined together allow significant forms of social knowledge to be represented and reasoned about, and to evolve in time based on the agent's beliefs, experience, and interactions with other agents [4].

Many current multi-agent teamwork coordination strategies are based on theoretical frameworks such as [2], [8], [9], and typically involve the recognition of agent mental states, possibly by relying on the BDI ("Belief, Desire, Intentions") model [1]: and agent's *beliefs* correspond to information the agent has about the world, which may be incomplete and incorrect; an agent's *desires* intuitively correspond to its objectives, or to the tasks allocated to it; as an agent will not, in general, be able to achieve all its desires, the desires upon which the agent commits are *intentions* that the agent will try to achieve. These coordination approaches, and also those mainly based on communication, are limited whenever communication is unreliable, or information on the source incomplete.

Mediated interaction and environment-based coordination focus on cognitive and social theories and explicitly take into account the role of the environment in coordination, such as [2], [8], [15]. In [16], it is emphasized that any real conceptual and engineering framework for this approach should: (i) do not rely on simple reactivity only; (ii) not restrict to solution tailored to specific coordination problems; (iii) provide methodologies and infrastructures to make the framework effective.

The support for coordination provided by the DALI language, simple as it is on the one hand addresses some the problems that arise in BDI-based and communication-based approach, due to its powerful communication filter. On the other hand, DALI addresses issues (i)-(iii) above as it is a general-purpose language with powerful proactive features, has a precise declarative and operational semantics, and is fully implemented. A future aim of this research is to further extend and refine DALI support to coordination, and to put it at work in complex application domain such as for instance peer-to-peer negotiation.

References

1. M. E. Bratman, D. J. Israel and M. E. Pollack. Plans and Resource-bounded Practical Reasoning, *Computational Intelligence*, vol. 4, 1988, 349–355.
2. P. Cohen and H. Levesque. Teamwork, Nous, Special Issue on Cognitive Science and AI, vol. 25, no. 4, 1991, 487–512.
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, LNAI 2424, Springer-Verlag, 2002.
4. S. Costantini, A. Tocchio and A. Verticchio. A Game-Theoretic Operational Semantics for the DALI Communication Architecture, Proc. of WOA04, 2004.
5. S. Costantini, A. Tocchio and A. Verticchio. Communication and Trust in the DALI Logic Programming Agent-Oriented Language, In: M. Cadoli, M. Milano and A. Omicini (eds.), Italian Conference on Intelligent Systems AI*IA'04, 2004.
6. M. d'Inverno and M. Luck. Engineering AgentSpeak(L): A Formal Computational Model, Journal of Logic and Computation 8(3), 1998, 233–260.
7. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming, In Proceedings of the Fifth Joint International Conference and Symposium. The MIT Press, 1988, 1070–1080.
8. B.J. Grosz and S. Kraus. Collaborative Plans for Complex Group Action, Artificial Intelligence, 86(2), 1996, 269–357.
9. D. Kinny, M. Ljungberg, A. Rao, E. Sonenberg, G. Tidhar and E. Werner. Planned Team Activity, In: Artificial Social Systems, 4th Europ. Worksh. on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'92), Selected Papers, S.Martino al Cimino, Italy, 1994, 227–256.
10. 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.
11. K. Larson and T. Sandholm. An alternating offers bargaining model for computationally limited agents, In: First International Conference on Autonomous Agents and Multiagent Systems (AAMAS'02), 135–142.
12. V. Lifschitz. Answer Set Programming and Plan Generation Artif. Intelligence 138 (1–2), Elsevier Science Publishers, 2002, 39–54.
13. P. McBurney and S. Parsons. Dialogue Games Protocols for Agent Purchase Negotiations, In: M.-P. Huget (ed.), Communication in Multi-Agent Systems: Agent Communication Languages and Conversation Policies, LNAI 2650, Springer-Verlag, 2001.

14. P. McBurney, R. M. Van Eijk, S. Parsons and L. Amgoud, A Dialogue Game Protocol for Agent Purchase Negotiations, Autonomous Agents and Multi-Agent Systems 7(3), Kluwer Academic Publishers, 2003, 235–273.
15. B.A. Nardi. Context and Consciousness: Activity Theory and Human-Computer Interaction, MIT Press, 1996.
16. A. Omicini, A. Ricci, M. Viroli,C. Castelfranchi,L. Tummolini. Coordination Artifacts: Environment-based Coordination for Intelligent Agents, In: *Proc. of the Third Int. Joint Conf. on Autonomous Agents and Multiagent Systems(AAMAS'04),* ACM Press, 2004.
17. David C. Parkes. Optimal Auction Design for Agents with hard Valuation Problems In: Agent-Mediated Electronic Commerce Workshop at the International Joint Conference on Artificial Intelligence, Stockholm, 1999.
18. T. Sandholm. Unenforced e-commerce Transactions, IEEE Internet Computing 1(6) (November-December 1997), 47-54.
19. T. Sandholm, S. Suri, A. Gilpin and D. Levine. CABOB: A Fast Optimal Algorithm for Combinatorial Auctions, In: Proc. IJCAI-0l, Seattle, WA, 2001, 1102–1108.
20. Web location of the most known ASP solvers.
    Cmodels: *http://www.cs.utexas.edu/users/yuliya/*
    Aspps: *http://www.cs.uky.edu/ai/aspps/*
    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/*
21. A. Tocchio. Multi-Agent sistems in computational logic. Ph.D. Thesis (draft).