A Game-Theoretic Operational Semantics for the DALI Communication Architecture

Stefania Costantini Stefania Costantini Alessia Verticchio
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 present the communication architecture of the DALI Logic Programming Agent-Oriented language and we discuss its semantics. We have designed a meta-level where the user can specify, via the distinguished tell/told primitives, constraints on communication or even a new protocol. Moreover, the user can define meta-rules for filtering and/or understanding messages via applying ontologies and commonsense/case-based reasoning. Declaratively and procedurally, these forms of meta-reasoning are automatically applied by a form of implicit, logical reflection. Operationally, we define a transition system based on a dialog game syntax. Thus, our operational semantics provides a formal link between the dialog locutions and the DALI semantic mechanisms. We embed the DALI/FIPA locutions and protocol within a framework that filters and interprets messages, without resorting to the definition of "mental states" of the agent. The locutions we consider include the relevant FIPA-compliant primitives, plus others which we believe to be needed in a logic programming setting.**

I. Introduction

Interaction is an important aspect of Multi-agent systems: agents exchange messages, assertions, queries. This, depending on the context and on the application, can be either in order to improve their knowledge, or to reach their goals, or to organize useful cooperation and coordination strategies. In open systems the agents, though possibly based upon different technologies, must speak a common language so as to be able to interact.

However, beyond standard forms of communication, the agents should be capable of filtering and understanding message contents. A well-understood topic is that of interpreting the content by means of ontologies, that allow different terminologies to be coped with. In a logic language, the use of ontologies can be usefully integrated with forms of commonsense and case-based reasoning, that improve the "understanding" capabilities of an agent. A more subtle point is that an agent should also be able to enforce constraints on communication. This requires to accept or refuse or rate a message, based on various conditions like for instance the degree of trust in the sender. This also implies to be able to follow a communication protocol in "conversations". Since the degree of trust, the protocol, the ontology, and other factors, can vary with the context, or can be learned from previous experience, in a logic language agent should and might be able to perform meta-reasoning on communication, so as to interact flexibly with the "external world."

This paper presents the communication architecture of the DALI agent-oriented logic programming language [2] [3], and the operational semantics of this architecture. DALI is an enhanced logic language with fully logical semantics [4], that (on the line of the arguments proposed in [7]) integrates rationality and reactivity, where an agent is able of both backwards and forward reasoning, and has the capability to enforce "maintenance goals" that preserve her internal state, and "achievement goal" that pursue more specific objectives. An extended resolution and resolution procedure are provided, so that the DALI interpreter is able to answer queries like in the plain Horn-clause language, but is also able to cope with different kinds of events.

In this paper we also present the operational semantics of the communication architecture that we present. Actually, we have defined a full operational semantics for the DALI language, which has been a basis for implementing the DALI system and is being used for developing model-checking tools for verifying program properties. For providing the operational semantics of the DALI communication architecture, following [8] and the references therein, 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. This means that we formulate the semantics of communication locutions as steps of a dialogue game, without referring to the mental states of the participants. This because we believe that in an open environment agents may also be malicious, and falsely represent their mental states. However, the filter layer of the DALI communication architecture (discussed below) allows an agent to make public expression of its mental states, and other agents to reason both on this expression and on their own degree of belief, trust, etc. about it.

The DALI communication architecture specifies in a flexible way the rules of interaction among agents, where the various aspects are modeled in a declarative fashion, are adaptable to the user and application needs, and can be easily composed. DALI agents communicate via FIPA ACL [6], augmented with some primitives which are suitable for a logic language. As a first layer of the architecture, we have introduced a check level that filters the messages. This layer by default verifies that the message respects the communication protocol, as well

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

as some domain-independent coherence properties. The user can optionally add other checks, by expanding the definition of the distinguished predicates *tell/told*. Several properties can be checked, however in our opinion an important role of the filter layer is that of making it explicit which assumption an agent makes about the mental states of the other agents, their reliability, their skills, how much they can be trusted, etc. If a message does not pass the check, it is just deleted. As a second layer, meta-level reasoning is exploited so as to try to understand message contents by using ontologies, and forms of commonsense reasoning. The third layer is the DALI interpreter.

The declarative and procedural semantics (not treated here) are defined as an instance of the general framework RCL (Reflective Computational Logic) [1] based on the concept of reflection principle as a knowledge representation paradigm in a computational logic setting. Application of both the filter layer and the meta-reasoning layer are understood as application of suitable reflection principles, that we define in the following. RCL then provides a standard way of obtaining the declarative and procedural semantics, which can be gracefully integrated with the semantics of the basic DALI language [4].

The paper is organized as follows. We start by shortly describing the main features of DALI in Section II and the communication architecture in Section III. Then, we face the Operational semantics in Section IV. In order to make it clear the usefulness and usability of the proposed architecture, we present an example in Section V. Finally, we conclude with some concluding remarks.

II. The DALI language

DALI [2] [4] is an Active Logic Programming language designed 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 environment is perceived in the form of external events, that can be exogenous events, observations, or messages by other agents. In response, a DALI agent can perform actions, send messages, invoke goals. The reactive and proactive behavior of the DALI agent is triggered by several kinds of events: external events, internal, present and past events. It is important to notice that 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). The set of past events in a way constitutes the set of the new beliefs that the agent has collected from her interaction with the environment.

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 rule contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction, specified 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'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: 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 are kept for a certain default amount of time, that can be modified by the user through a suitable directive in the initialization file.

III. DALI Communication Architecture

A. The Architecture

The DALI communication architecture (Fig.1) consists of three levels. The first level implements the DALI/FIPA communication protocol and a filter on communication, i.e. a set of rules that decide whether or not receive or send a message. 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.

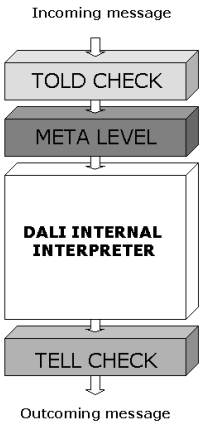

Fig. 1. The communication architecture of a DALI agent

The DALI/FIPA protocol consists of the main FIPA primitives, plus few new primitives which are peculiar of DALI.
In DALI, an out-coming message has the format:

$$message(Receiver, primitive(Content, Sender))$$

that the DALI interpreter converts it into an internal form, by automatically adding the missing FIPA parameters, and creating the structure:

$$\begin{aligned} message(\quad & receiver\_address, receiver\_name, \\ & sender\_address, sender\_name, \\ & language, ontology, \\ & primitive(Content, sender\_name)) \end{aligned}$$

Using this internal structure, an agent can include in the message the adopted ontology and the language. When a message is received, it is examined by a check layer composed of a structure which is adaptable to the context and modifiable by the user. This filter checks the content of the message, and verifies if the conditions for the reception are verified. If the conditions are false, this security level eliminates the supposedly wrong message. The DALI 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, which is of the form:

$$\begin{aligned} & told(Sender, primitive(Content)) : - \\ & \quad constraint_1, \ldots, constraint_n. \end{aligned}$$

where $constraint_i$ can be everything expressible either in Prolog or in DALI. 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.

Example: the proposal to perform an action is acceptable if the agent is specialized for the action and the Sender is reliable (this suggests that this model allows one to integrate into the filtering rules the concept the degree of trust).

$$\begin{aligned} told(\quad & Sender\_agent, propose(Action, Preconditions)) : - \\ & not(unreliableP(Sender\_agent)), \\ & specialized\_for(Action). \end{aligned}$$

Symmetrically to *told* rules, the messages that an agent sends are subjected to a check via *tell* rules. There is, however, an important difference: the user can choose which messages must be checked and which not. The choice is made by setting some parameters in the initialization file. The syntax of a *tell* rule is:

$$\begin{aligned} & tell(Receiver, Sender, primitive(Content)) : - \\ & \quad constraint_1, \ldots, constraint_n. \end{aligned}$$

For every message that is being sent, the interpreter automatically checks whether an applicable *tell* rule exists. If so, the message is actually sent only upon success of the goal $tell(Receiver, Sender, primitive(Content))$.

Example: the *tell* rule authorizes the agent to send the message with the primitive *inform* if the receiver is active in the environment and is presumably interested to the information.

$$\begin{aligned} tell(\quad & Agent\_To, Agent\_From, inform(Proposition)) : - \\ & active\_in\_the\_world(Agent\_To), \\ & specialized(Agent\_To, Specialization), \\ & related\_to(Specialization, Proposition). \end{aligned}$$

The FIPA/DALI communication protocol is implemented by means a piece of DALI code including suitable $tell/told$ rules. This code is contained in a separate file, *communication.txt*, that each DALI agent imports as a library, so that the communication protocol can be seen an "input parameter "of the agent. As mentioned, whenever an incoming message passes the $told$ check, its content $primitive(Content, Sender)$ is treated as an external event $primitive(Content, Sender)E$. If it corresponds to a DALI/FIPA locution, then it is managed by predefined reactive rules (included in *communication.txt*) that behave according to the protocol. If $primitive$ is the distinguished primitive $send\_message$, then $Content$ is interpreted as an external event $ContentE$ which is sent to the agent, in the sense that no predefined reactive rule is defined, and thus the agent has to react herself to this event.

Each DALI agent is also provided with a distinguished procedure called *meta*, which is automatically invoked by the interpreter in the attempt of understanding message contents. This procedure includes by default a number of rules for coping with domain-independent standard situations. The user can add other rules, thus possibly specifying domain-dependent commonsense reasoning strategies for interpreting messages, or implementing a learning strategy to be applied when all else fails.

Example: below are the default rules that apply the equivalences listed in an ontology, and possibly also exploit symmetry of binary predicates:

$meta($
$Initial\_term, Final\_term, Agent\_Sender) :-$
$clause(agent(Agent\_Receiver), \_),$
$functor(Initial\_term, Functor, Arity), Arity = 0,$
$((ontology(Agent\_Sender, Functor, Equivalent\_term);$
$ontology(Agent\_Sender, Equivalent\_term, Functor));$
$(ontology(Agent\_Receiver, Functor, Equivalent\_term);$
$ontology(Agent\_Receiver, Equivalent\_term, Functor))),$
$Final\_term = Equivalent\_term.$

$meta($
$Initial\_term, Final\_term, Agent\_Sender) :-$
$functor(Initial\_term, Functor, Arity), Arity = 2,$
$symmetric(Functor), Initial\_term = ..List,$
$delete(List, Functor, Result\_list),$
$reverse(Result\_list, Reversed\_list),$
$append([Functor], Reversed\_list, Final\_list),$
$Final\_term = ..Final\_list.$

Since the FIPA/DALI protocol is implemented by means of a piece of DALI code, and the link between the agent and the interpreter sending/receiving messages is modeled by the reflection principles specified above, the semantics of DALI communication is now complete. However, in the next section we propose an operational semantics that specifies in a language/independent fashion how the FIPA/DALI protocol works.

B. Related Approaches

The problem of a secure interaction between the agents is also treated in [9], [5]. However, [9] defines a system (Moses) with a global law for a group of agents, instead of a set of local laws for every single agent as in DALI. Moreover, in Moses there is a special agent, called *controller*, for every agent, while in DALI it is necessary to define a filter for each agent, defining constraints on the communication primitives. Our definition of tell/told rules is structurally different from the Moses approach: each law in Moses is defined as a prolog-like rule having in the body both the conditions that match with a control state of the object and some fixed actions that determine the behavior of the law. In DALI, the told/tell rules are the constraints on the communication and do not contain actions. The behavior (and in particular the actions) performed by an agent are determined by the logic program of the agent. Another difference is that the DALI filter rules can contain past events, thus creating a link between the present communication acts and the experience of the agent. A particularity of the Minsky law-governed system is that is possible to update on-line the laws [10]. In DALI, presently it is possible to change the rules locally by varying the name of the file that contains the tell/told rules but in the future we will improve our language by allowing an agent to modify even filter rules.

Santoro in [5] defines a framework for expressing agent interaction laws by means of a set of rules applied to each ACL message exchanged. Each rule has a prefixed structure composed by precondition, assignment and constraint where the precondition is a predicate on one or more fields of the message which triggers the execution of the assignment or the checking of the constraint. The constraint is a predicate which specifies how the message meeting the precondition has to be formed, and it is used to model the filtering function. The rules consider some specific fields of a message like the name of agents, the performative name, language, ontology, delivery mode and content. We think that the approach followed in DALI is only apparently similar. The Agent Communication Context (ACC) in JADE is applied only to outcoming messages, while in DALI we submit to the filter both the received messages and the sent messages. The structure of a DALI filter rule is different and more flexible: in ACC the rule specifies that if the preconditions are true, some fields of the message must be defined by the assignments in the body; in DALI, the body of a filter rule specifies only the constraints for the acceptance/sending of a message. Moreover, the constraints in DALI do not refer to specific fields. They can be procedures, past events, beliefs and whatever is expressible either in DALI or in Prolog. Therefore, even though both the approaches use the concept of communication filter, we think that there are notable differences also due to ability of Prolog to draw inferences and to reason in DALI with respect to java.

IV. Operational semantics

The operational semantics that we propose in this Section follows the approach of [8] (see also the references therein). 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. This means, we reformulate the semantics of FIPA locutions as steps of a dialogue game, without referring to the mental states of the participants. This approach has its origin in the philosophy of argumentation, while approaches based on assumptions about the mental states of participants build on speech-act theory. This because we believe that in an open environment agents may also be malicious, and falsely represent their mental states. However, as we have seen the filter layer of the DALI communication architecture allows an agent to make public expression of its mental states, and other agents to reason both on this expression and on their own degree of belief, trust, etc. about it.

The rules of the operational semantics show how the state of an agent changes according to the execution of the transition rules. We define each rule as a combination of states and laws. Each law links the rule to interpreter behavior and is based on the interpreter architecture.

We have three kinds of laws: those that model basic communication acts; those describing the filter levels; those that modify the internal state of the agent by adding items to the various sets of events. In order to make it clear how we express the formal link between the agent actual activity and the semantic mechanisms, we adopt some abbreviations:

- Ag_x to identify the name of the agent involved by the transition;
- S_{Ag_x} or NS_{Ag_x} to identify the state before and after the application of laws.
- L_x to identify the applied law.

We adopt the pair $< Ag_x, S_{Ag_x} >$ to indicate a link between the name of an agent and her state. The state of a DALI agent is defined as a triple: $S_{Ag_x} \equiv< P_{Ag}, IS_{Ag}, Mode_{Ag} >$

where P_{Ag} is the logic program, IS_{Ag} is the internal state and *Mode* is a particular attribute describing what the interpreter is doing. Hence, we can introduce the following equivalence:
$< Ag_x, S_{Ag_x} > \equiv < Ag_x, < P_{Ag}, IS_{Ag}, Mode_{Ag} >>$

The internal state of an agent is the tuple
$< E, N, I, A, G, T, P >$ composed by the sets of, respectively, external events, present events, internal events, actions,goals, test goals and past events.

Moreover, we denote by NP_{Ag} the logic program modified by the application of one or more laws and by NIS_{Ag} the internal state modified. We distinguish the internal state IS from the global state S because we want to consider separately the influence of the communication acts on the classes of events and actions within the agent. The semantic approach we describe in this paper is based on the framework of (labeled) *transition rules*. We apply them in order to describe the interactive behavior of the system. Each transition rule is described by two pairs and some laws. Starting from the first pair and by applying the current laws, we obtain the second pair where some parameters have changed (e.g., name, internal state or modality).

First, we introduce the general laws that modify the pairs. We start with the transitions about the incoming messages, by showing the behavior of the communication filter level. Next we show the semantic of meta-level and finally the communication primitives. For lack of space, we just consider some of them.

- **L0:** The **receive_message(.)** law:
  Locution: $receive\_message($
  $Ag_x, Ag_y, Ontology, Language, Primitive)$
  Preconditions: this law is applied when the agent Ag_x finds in the Tuple Space a message with her name.
  Meaning: the agent Ag_x receives a message from Ag_y(environment, other agents,...). For the sake of simplicity we consider the environment as an agent.
  Response: the interpreter takes the information about the language and the ontology and extracts the name of sender agent and the primitive contained in the initial message.
- **L1:** The **L1 told_check_true(.)** law:
  Locution:$told\_check\_true(Ag_y, Primitive)$
  Preconditions: the constraints of told rule about the name of the agent sender Ag_y and the primitive must be true for the primitive *told_check_true*.
  Meaning: the communication primitive is submitted to the check-level represented by the told rules.
  Response: depends on the constraints of told level. If the constraints are true the primitive can be processed by the next step.
- **L2 :** The **L2 understood(.)** law:
  Locution: $understood(Primitive)$
  Preconditions: in order to process the primitive the agent must understand the content of the message. If the primitive is *send_message*, the interpreter will check if the external event belongs to a set of external events of the agent. If the primitive is *propose*, the interpreter will verify if the requested action is contained in the logic program.
  Meaning: this law verifies if the agent understands the message.
  Response:the message enters processing phase in order to trigger a reaction, communicate a fact or propose an action.
- **L3 :** The **L3 apply_ontology(.)** law:
  Locution: $apply\_ontology(Primitive)$
  Preconditions: in order to apply the ontology the primitive must belong to set of locutions that invoke the meta-level(send_message,propose,execute_proc,query_ref,is_a_fact).
  Meaning: this law applies, when it's necessary, the ontologies to the incoming primitive in order to understand its content.
  Response:the message is understood by using the ontology of the agent and properties of the terms.
- **L4:** The **L4 send_message_with_tell(.)** law:
  Locution:$send\_msg\_with\_tell(Ag_x, Ag_y, Primitive)$
  Preconditions: the precondition for L4 is that the primitive belongs to set of locutions submitted to tell check.
  Meaning: the primitive can be submitted to the constraints in the body of tell rules.
  Response: the message will be sent to the tell level.
- **L5:** The **L5 tell_check(.)** law :
  Locution: $tell\_check(Ag_x, Ag_y, Primitive)$
  Preconditions: the constraints of tell rule about the name of the agent receiver Ag_x, the agent sender Ag_y and the primitive are true for L5.
  Meaning: the primitive is submitted to a check using the constraints in the tell rules.
  Response: the message will either be sent to addressee agent(L5).
- **Lk:** The **add_X(.)** law:
  Locution: $add\_X(.)$
  where
  $X \in \{internal\_event, external\_event, action,$
  $message, past\_event\}$
  Preconditions: the agent is processing X.
  Meaning: this law updates the state of the DALI agent adding an item to corresponding set to X.
  Response: the agent will reach a new state. The state S_{Ag} of the agent will change in the following way.
  k=6 and X=internal_event:
  $S_{Ag} =< P_{Ag}, < E, N, I, A, G, T, P >, Mode >$
  $NS_{Ag} =< P_{Ag}, < E, N, I_1, A, G, T, P >, Mode >$ where $I_1 = I \cup Internal\_event$.
  k=7and X=external_event:
  $S_{Ag} =< P_{Ag}, < E, N, I, A, G, T, P >, Mode >$
  $NS_{Ag} =< P_{Ag}, < E_1, N, I, A, G, T, P >, Mode >$ where $E_1 = E \cup external\_event$.
  k=8 and X=action:
  $S_{Ag} =< P_{Ag}, < E, N, I, A, G, T, P >, Mode >$
  $NS_{Ag} =< P_{Ag}, < E, N, I, A_1, G, T, P >, Mode >$ where $A_1 = A \cup Action$ or $A_1 = A \setminus Action$ if the communication primitive is $cancel$.
  k=9 and X=message:
  $S_{Ag} =< P_{Ag}, < E, N, I, A, G, T, P >, Mode >$
  $NS_{Ag} =< P_{Ag}, < E, N, I, A_1, G, T, P >, Mode >$ where A_1=A∪Message. In fact, a message is an action.
  k=10 and X=past_event:
  $S_{Ag} =< P_{Ag}, < E, N, I, A, G, T, P >, Mode >$
  $NS_{Ag} =< P_{Ag}, < E, N, I, A, G, T, P_1 >, Mode >$ where $P_1 = P \cup Past\_event$.
- **L11:** The **L11 check_cond_true(.)** law:
  Locution: $check\_cond\_true(Cond\_list)$
  Preconditions: The conditions of the $propose$ primitive are true.
  Meaning: this law checks the conditions inside the $propose$ primitive.
  Response: the proposed action will either be executed.
- **L12:** The **update_program(.)** law:
  Locution: $update\_program(Update)$
  Preconditions: No preconditions.
  Meaning: this law updates the DALI logic program by adding new knowledge.
  Response: the logic program will be updated.
- **Lk:** The **process**$_X$ law:
  Locution: $process_X(.)$
  where
  $X \in \{send\_message, execute\_proc, propose,$
  $accept\_proposal, reject\_proposal\}$

Preconditions: The agent calls the primitive X.
Meaning and Response: We must distinguish according to the primitives:
k=13 and X=$send\_message$: this law calls the external event contained in the primitive. As response the agent reacts to external event.
k=14 and X=$execute\_proc$:this law allows a procedure to be called within the logic program. As response the agent executes the body of the procedure.
k=15 and X=propose: If an agent receives a $propose$, she can choose to do the action specified in the primitive if she accepts the conditions contained in the request. The response can be either $accept\_proposal$ or $reject\_proposal$.
k=16 and X=$accept\_proposal$: An agent receives an $accept\_proposal$ if the response to a sent propose is yes. As response the agent asserts as a past event the acceptance received.
k=17 and X=reject_proposal: An agent receives a $reject\_proposal$ if the response to a sent proposal is no. In response, the agent asserts as a past event the refusal.

- **L18:** The **L18 action_rule_true(.)** law:
Locution: $action\_rule\_true(Action)$
Preconditions: The conditions of the action rule corresponding to the action are true.
Meaning: In a DALI program, an action rule defines the preconditions for an action.This law checks the conditions inside the action rule in the DALI logic program.
Response: the action will be executed.

We now present the operational semantics of the DALI communication. The following rules indicate how the laws applied to a pair determine, in a deterministic way, a new state and the corresponding behavior of the agent.

DALI communication is asynchronous: each agent communicates with other's one in such a way that she is not forced to halt its processes while the other entities produce a response. An agent in *wait* mode can receive a message taking it from the Tuple Space by using the law R0. The global state of the agent changes passing from the *wait* mode to *received_message* mode: the message is entered in the more external layer of the communication architecture.

$$R0: \ < Ag_1, < P, IS, wait >> \xrightarrow{L_0} \\ < Ag_1, < P, IS, received\_message_x >>$$

The L1 law determines the transition from the *received_message* mode to *told* mode because it can be accepted only if the corresponding told rule is true.

$$R1: \ < Ag_1, < P, IS, received\_message_x >> \xrightarrow{L_1} \\ < Ag_1, < P, IS, told_x >>$$

If the constraints in the told rule are false, the message cannot be processed. In this case, the agent returns in the wait mode and the message do not affect the behavior of the software entity because the message is deleted. The sender agent is informed about the elimination.

$$R2: \ < Ag_1, < P, IS, received\_message_x >> \overset{not(L_1)}{\rightarrow} \\ < Ag_1, < P, IS, wait >>$$

When a message overcomes the told layer, it must be understood by the agent in order to trigger, for example, a reaction. If the agent understands the communication act, the message will continue the way.

$$R3: \ < Ag_1, < P, IS, told_x >> \overset{L_2}{\rightarrow} \\ < Ag_1, < P, IS, understood_x >>$$

An unknown message forces the agent to use a meta-reasoning level, if the L3 law is true.

$$R4: \ < Ag_1, < P, IS, told_x >> \overset{not(L_2),L_3}{\rightarrow} \\ < Ag_1, < P, IS, apply\_ontology_x >>$$

The meta-reasoning level can help the agent to understand the content of a message. But only some primitives can use this possibility and apply the ontology. Instead going in *wait mode* we can suppose that the agent will call a learning module but up to now we do not have implemented this functionality.

$$R5: \ < Ag_1, < P, IS, told_x >> \overset{not(L_2),not(L_3)}{\rightarrow} \\ < Ag_1, < P, IS, wait >>$$

After the application of the ontology, if the agent understands the message, she goes in the *understood mode*.

$$R6: \ < Ag_1, < P, IS, apply\_ontology_x >> \overset{L_2}{\rightarrow} \\ < Ag_1, < P, IS, understood_x >>$$

If the L2 law is false, the message cannot be understood and the agent goes in *wait mode*.

$$R7: \ < Ag_1, < P, IS, apply\_ontology_x >> \overset{not(L_2)}{\rightarrow} \\ < Ag_1, < P, IS, wait >>$$

A known message enters in the processing phase and the internal state of the agent changes because an item can be added to internal queues of events and actions. The logic program can change because we can add some facts using the confirm primitive.

$$R8: \ < Ag_1, < P, IS, understood_x >> \overset{L_6,L_7,L_8,L_9}{\rightarrow} \\ < Ag_1, < NP, NIS, process_x >>$$

When an agent sends a message, the L4 law verifies that it must be submitted to tell level. In this rule we suppose that the response is true.

$$R9: \ < Ag_1, < P, IS, send_x >> \overset{L_4}{\rightarrow} \\ < Ag_1, < P, IS, tell_x >>$$

If the response is false, the message is immediately sent and the queue of the messages(actions) changes.

$$R10: \ < Ag_1, < P, IS, send_x >> \overset{not(L_4),L_9}{\rightarrow} \\ < Ag_1, < P, NIS, sent_x >>$$

If the constraints of tell level are satisfied, the message is sent.

$$R11: \ < Ag_1, < P, IS, tell_x >> \overset{L_5,L_9}{\rightarrow} \\ < Ag_1, < P, NIS, sent_x >>$$

A message sent by the agent Ag_1 is received by the agent Ag_2 that goes in *received message mode*.

$$R12: \ < Ag_1, < P, IS, tell_x >> \overset{L_5}{\rightarrow} \\ < Ag_2, < P, IS, received\_message_x >>$$

If the message do not overcome the tell level because the constraints are false, the agent returns in *wait mode*.

$$R13: \ < Ag_1, < P, IS, tell_x >> \overset{not(L_5)}{\rightarrow} \\ < Ag_1, < P, NIS, wait >>$$

This last rule shows how, when a message is sent, the corresponding action becomes past event.

$$R14: \ < Ag_1, < P, IS, sent_x >> \overset{L_{10}}{\rightarrow} \\ < Ag_1, < P, IS, wait >>$$

The DALI primitive send_message: by using this locution a DALI agent is able to send an external event to the receiver.

$$< Ag_1, < P, IS, process_{send\_message} >> \overset{\wedge_{i=6,7,8,10,12} L_i}{\rightarrow} \\ < Ag_1, < NP, NIS, wait >>$$

According to the specific reactive rule, several sets of events can change. In fact, in the body of rule we can find actions and/or goals. Since the external event will become a past event, the sets of external and past events must be updated. After processing the reactive rule the interpreter goes in *wait* mode.

$$< Ag_1, < P, IS, process_{send\_message} >> \overset{L_{13}, L_9}{\rightarrow} \\ < Ag_1, < P, NIS, send_{primitive} >>$$

In the body of rule there could be some messages that the agent must send.

The FIPA primitive propose: this primitive represents the action of submitting a proposal to perform a certain action, given certain preconditions.

$$< Ag_1, < P, IS, process_{propose} >> \overset{L_{15}, L_{11}, L_9}{\rightarrow} \\ < Ag_1, < P, NIS, send_{accept\_proposal} >$$

This transition forces an agent receiving the *propose* primitive to answer with $accept\_proposal$ if the conditions included in the propose act are acceptable.

$$< Ag_1, < P, IS, send_{accept\_proposal} >> \overset{L_8, L_9}{\rightarrow} \\ < Ag_1, < P, NIS, send_{inform} >$$

When an agent accepts the proposal, then she performs the action. In this case the internal state of agent changes by adding the action. Finally, the agent communicates to the proposer that the action has been done.

$$< Ag_1, < P, IS, send_{accept\_proposal} >> \overset{L_9}{\rightarrow} \\ < Ag_1, < P, NIS, send_{failure} >$$

If the action cannot be executed, then the agent sends a failure primitive to the proposer.

$$< Ag_1, < P, IS, process_{propose} >> \overset{L_{15}, not(L_{11}), L_9}{\rightarrow} \\ < Ag_1, < P, NIS, send_{reject\_proposal} >>$$

If the conditions in the $propose$ are unacceptable, the response can be only a $reject\_proposal$.

V. An example of application of the DALI communication filter

We will now demonstrate how the filter level works by means of an example, that demonstrates how this filter is powerful enough to express sophisticated concepts such as 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: how the level of Trust influences communication and choices of the agents.

We consider as a case-study a cooperation context where an ill agent asks her friends to find out a competent specialist. When the agent has some particular symptoms, she calls a family doctor that recommends her to consult a lung doctor. The patient, through a yellow pages agent, becomes aware of the names and of the distance from her city of the two specialists, and asks her friends about them. The patient has a different degree of trust on her friends and each friend has a different degree of competence on the specialists. Moreover, the patient is aware of the ability of the friends about medical matters: a clerk will be less reliable than a nurse. In this context we experiment the communication check level joining the potentiality of tell/told rules and the trust concept. We suppose that the ill agent receives a message only if she has a level of trust on the sender agent greater than a fixed threshold:

$$told(Ag, send\_message(\_)) : - \\ trustP(\_, Ag, N), N > threshold.$$

We can adopt a similar rule also for the out-coming messages. Now we discuss the trust problem by showing the more interesting DALI rules defining the agents involved in this example. The cooperation activity begins when the agent Ag becomes ill and communicates her symptoms to doctor. If these symptoms are serious, the doctor advises the patient to find out a competent lung doctor M. If the agent knows a specialist Sp and has a positive trust value V_1 on her, she goes to lung doctor, else asks a yellow page agent.

$$consult\_lung\_doctorE(M) :> \\ clause(agent(Ag), \_), \\ choose\_if\_trust(M, Ag).$$

$$choose\_trust(\_, Ag) : - \\ clause(i\_know\_lung\_doctor(Sp),_) , \\ trustP(Ag, Sp, V_1), V_1 > 0, \\ go\_to\_lung\_doctorP(Sp).$$

$$choose\_trust(M, Ag) : - \\ messageA(yellow\_page, \\ send\_message(search(M, Ag), Ag)).$$

The yellow page agent returns to patient, by means of the inform primitive, a list of the lung doctors. Now the patient must decide which lung doctor is more competent and reliable. How can she choose? She asks her friends for help.

$$take\_information\_about(Sp) : - \\ clause(lung\_doctor(Sp), \_).$$

$$take\_information\_aboutI(Sp) :> \\ clause(agent(Ag), \_), \\ messageA(friend1, \\ send\_message(what\_about\_competency(Sp, Ag), Ag)), \\ messageA(friend2, \\ send\_message(what\_about\_competency(Sp, Ag), Ag)).$$

Each friend, having the information $competent(lung\_doctor_x, Value)$ about the ability of the specialists, sends an inform containing the evaluation of the competency.

$$what\_about\_competencyE(Sp, Ag) :> \\ choose\_competency(Sp, Ag).$$

$choose\_competency(Sp, Ag) : -$
$clause(competent(Sp, V), \_),$
$\quad messageA(Ag,$
$\quad inform(lung\_doctor\_competency(Sp, V), friend_x)).$
$choose\_competency(Sp, Ag) : -$
$\quad messageA(Ag,$
$\quad inform(dont\_know\_competency(Sp), friend_x)).$

The patient is now aware of the specialist and friend's competency and has a value of trust on the friends consolidated through the time. Moreover she knows the distance of the specialists from her house. Using a simple rule that joins those parameters, she assigns a value to each advice: $specialist\_evaluation(lung\_doctor_x, friend_y, Value)$.

The ill agent will choice the lung doctor in the advice having the greater $Value$ and will go to the specialist: $follow\_adviceA(Friend), go\_to\_lung\_doctorA(Sp)$.

Will he be cured? After some time the patient will reconsider her health. If she does not have any symptom (temperature, thorax pain, cough, out of breath), she increases the trust on the friend that has recommended the lung doctor and sets the trust on that specialist a smallest value:

$cured(Sp, Friend) : -$
$\quad go\_to\_lung\_doctorP(Sp),$
$\quad follow\_adviceP(Friend),$
$\quad not(temperatureP),$
$\quad not(thorax\_painP),$
$\quad not(coughP),$
$\quad not(out\_of\_breathP).$

$curedI(Sp, Friend) :>$
$\quad clause(agent(Ag), \_),$
$\quad trustP(Ag, Friend, V), V_1 \ is \ V + 1,$
$\quad drop\_pastA(trust(Ag, Friend, V)),$
$\quad add\_pastA(trust(Ag, Friend, V1)),$
$\quad assert(i\_know\_lung\_doctor(Sp)),$
$\quad set\_pastA(trust(Ag, Friend, V), 100),$
$\quad add\_pastA(trust(Ag, Sp, 1)),$
$\quad drop\_pastA(go\_to\_lung\_doctor(\_)).$

The actions $drop\_past, add\_past$ and set_past are typical commands of DALI language useful to manage the past events: $drop\_past$/add_past deletes/adds a past event while set_past sets the time of the memorization of a past event. If she is still ill, she decreases the trust value on the friend that has recommended the lung doctor:

$no\_cured(Sp) : -$
$\quad go\_to\_lung\_doctorP(Sp), temperatureP.$
$no\_cured(Sp) : -$
$\quad go\_to\_lung\_doctorP(Sp),$
$\quad thorax\_painP.$
$no\_cured(Sp) : -$
$\quad go\_to\_lung\_doctorP(Sp), coughP.$
$no\_cured(Sp) : -$
$\quad go\_to\_lung\_doctorP(Sp),$
$\quad out\_of\_breathP.$

$no\_curedI(\_) :>$
$\quad clause(agent(Ag), \_),$
$\quad follow\_adviceP(Am),$
$\quad trustP(Ag, Am, V), V >= 1, V_1 \ is \ V - 1,$
$\quad drop\_pastA(trust(Ag, Am, V)),$
$\quad set\_pastA(trust(Ag, Am, V1), 1000),$
$\quad add\_pastA(trust(Ag, Am, V1)),$
$\quad drop\_pastA(go\_to\_lung\_doctor(\_)).$

The decrement of the trust value of a friend can affect the check level of communication, thus preventing the sending/receiving of a message to/from that friend. This happens if the trust on the agent is less than the trust's threshold specified in the body of a told/tell rule. In this case, the patient communicates to the friend that the incoming message has been eliminated by using an inform primitive:

$send\_message\_to(friend,$
$\quad inform(send\_message$
$\quad (what\_about\_competency($
$\quad lung\_doctor, patient), patient),$
$\quad motivation(refused\_message), patient), italian, [])$

where $send\_message(what\_about\_competency($ $lung\_doctor, patient), patient)$ is the eliminated message, with the motivation $motivation(refused\_message)$.
In our system, the level of trust can change dynamically. In this way it is possible that an agent is excluded from the communication because of a too low value of trust, and she is readmitted later since the value increases, due either to her subsequent actions or to other agents pleading her case.

We face the problem of trust with a simple approach, where cooperating DALI agent adopt some parameters such as trust and competency, and update then dynamically. In the future, we intend to explore this area by adopting more formal approaches to model these concepts.

VI. Concluding Remarks

In this paper we have described an operational semantics of communication for the DALI language which is not based on assumptions on mental states of agents, which in real interaction can be in general uncertain or unknown. Instead, we consider each locution as a move of a game, to which the other agents may respond with other moves, according to a protocol. Each locution of course provided information, and thus influences the state of the receiving agent. This kind of formalization is made possible as the DALI language provides a communication architecture (of course coped with in the semantics) that provides a filter layer where an agent can explicitly describe her own mental attitudes and the assumptions she mades about the other agents. We have shown the usability of the architecture by means of an example. A future direction of this research is that of experimenting formal models of cooperation and trust.

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, editors, *Proc. of 2nd International Workshop on component-based Software Development in Computational Logic (COCL'99)*, PLI'99, (held in Paris, France, September 1999), Available on-line, URL
http://www.di.unipi.it/
brogi/ResearchActivity/COCL99/proceedings/index.html.

[3] 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.

[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*, (held in Cosenza, Italy, September 2002), LNAI 2424, Springer-Verlag, Berlin, 2002.

[5] A. Di Stefano and C. Santoro *Integrating Agent Communication Contexts in JADE*, Telecom Italia Journal EXP, Sept. 2003.

[6] FIPA. *Communicative Act Library Specification*, Technical Report XC00037H, Foundation for Intelligent Physical Agents, 10 August 2001.

[7] 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.

[8] 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.

[9] N. H. Minsky and V. Ungureanu, *Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems*, ACM Trans. Softw. Eng. Methodol.,2000,ACM Press.

[10] N. H. Minsky *The Imposition of Protocols Over Open Distributed Systems*, IEEE Trans. Softw. Eng.,1991,IEEE Press.

[11] J. M. Serrano, S. Ossowski. *An Organisational Approach to the Design of Interaction Protocols*, In: *Lecture Notes in Computer Science, Communications in Multiagent Systems: Agent Communication Languages and Conversation Policies*, LNCS 2650, Springer-Verlag, Berlin, 2003.

[12] E.C. Van der Hoeve, M. Dastani, F. Dignum, J.-J. Meyer, *3APL Platform*, In: *Proc. of the The 15th Belgian-Dutch Conference on Artificial Intelligence(BNAIC2003)*, held in Nijmegen, The Netherlands, 2003.