Communication and Trust in the DALI Logic Programming Agent-Oriented Language*

Stefania Costantini   Arianna Tocchio   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. 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 implies being able 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 a novel communication architecture for the DALI agent-oriented logic programming language and proposes an example aimed at showing that the communication architecture is general enough as to model sophisticated concepts such as the level of trust. Trust is a type 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. Finally, we consider approaches that use a similar architecture and we outline the meaningful differences between the approaches.

1 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 ei-

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

ther in order to improve their knowledge, or to reach their goals, or to organize useful cooperation and coordination strategies.

However, the exchange of information between agents implies a certain degree of risk. In a global environment, entities meet and need to collaborate with other entities of which they have little or no information about reliability. In a traditional environment, decisions are usually delegated to a centralized authority. In the global computing environment, each single entity must take the decisions needed to behave autonomously in the absence of complete knowledge of the operating environment. [16] considers how the notion of trust has been developed to help the agents to deal with the partial information about the world. How can an agent be sure that the received information will not damage her internal state? Should she be always confident? Is it possible to introduce a certain level of trust in the communication? This is a relevant problem, coped with in the literature in different ways. For Josang in [12], trust is a belief that one entity has about another entity. He states that the reason behind trust is composed of many elements, like past experience, knowledge about the entity's nature, recommendations from other entities or some kind of faith. Yahalom et al. in [10] give an interesting classification of trust and develop an algorithm that use the concept of recommendation path. Denning [11] states that the word "trust" is a declaration made by an observer rather than an inherent property of the person, organization, or object observed and that we make assessments of trust based on our experiences in the world.

In our approach, we introduce the concept of trust by means of the filter level of DALI communication architecture. This layer by default verifies that a message respects the communication protocol, as well as some domain-independent coherence properties. Several other properties to be checked can be however additionally specified, by expanding the definition of the distinguished predicates *tell/told*. If the message does not pass the check, it is just deleted. We have experimented the capabilities of this filter by introducing the trust concept in the told rules and increasing/decreasing the trust value. We don't face deeply the trust problem because our present aim is to show how the filter can manage sophisticated communication forms, by changing the behavior of the agents.

We are aware that the approach to trust shown in this paper can be refined by introducing more specific algorithms like shown for instance in [13], where the authors introduce a particular trust evolution function that formalizes the dependency of trust on past experiences. Also Josang and Denning consider trust as a result of the experience and knowledge of the agent. In this paper, we emphasize the link between trust and knowledge coming from the direct observation of events in the world by introducing in the told rules the concept of the trust as a past event (what an agent remembers on the base of the past life). The past event/trust can be managed by using appropriate predicates within a DALI logic program. Then, each agent can change the value of this predicate estimating the behavior of the other agents. About the tell/told rules, a similar approach is adopted in a version of JADE: we will explain how our filter is more general and powerful due to expressivity of the logical structure and to the peculiar featuress of the DALI language.

Below we briefly recall the main features of DALI. Operationally, following [8] and the references therein, we provide the semantics of the language (including commu-

nication) by defining a formal dialogue game framework that focuses on the rules of dialogue, regardless of 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 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 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.

The paper is organized as follows. We start by shortly describing the main features of DALI in Section 2 and the communication architecture in Section 3. Then, we show a small part of the operational semantic of the global DALI interpreter in the Sections 4. In Section 5 we explain how the filter on the communication works and how the trust concept can be integrated in this structure. Finally, in section 6 we outline how different values of trust in a coordination system can change the behavior of the involved agents. We conclude this paper in Section 7 by outlining future directions of our research.

2 The DALI language

DALI [2] [14] 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 proactivities, are indicated with special postfixes (which are coped with by a preprocessor) so as to be immediately recognized while looking at a program.

2.1 External Events

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. E. g., the body of the reactive rule below specifies the reaction to the external event *bell_ringsE* that is in the head. In this case the agent performs an action, postfix *A*, that consists in opening the door.

$bell_ringsE :> open_the_doorA.$

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. Priorities are listed in a separate file of directives, where (as we will see) the user can "tune" the agent's behaviour under several respect. The advantage introducing a separate initialization file is that for modifying the directives there is no need to modify (or even to understand) the code.

2.2 Internal Events

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'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).

For instance, consider a situation where an agent prepares a soup that must cook on the fire for K minutes. The predicates with postfix *P* are past events, i.e., events or actions that happened before, and have been recorded. Then, the first rule says that the soup is ready if the agent previously turned on the fire, and K minutes have elapsed since when she put the pan on the stove. The goal *soup_ready* will be attempted from time to time, and will finally succeed when the cooking time will have elapsed. At that point, the agent has to react to this (by second rule) thus removing the pan and switching off the fire, which are two actions (postfix *A*).

$$
\begin{aligned}
&soup_ready :- \ turn_on_the_fireP, put_pan_on_the_stoveP : T,\\
&\qquad\qquad cooking_time(K), time_elapsed(T, K).\\
&soup_readyI :> take_off_pan_from_stoveA, turn_off_the_fireA.
\end{aligned}
$$

A suitable directive for this internal event can for instance state that it should be attempted every 60 seconds, starting from when *put_the_pan_on_the_stove* and *turn_on_the_fire* have become past events.

Similarly to external events, internal events which are true by first rule are inserted in a set IV in order to be reacted to (by their second rule). The interpreter, interleaving the different activities, extracts from this set the internal events and triggers the reaction (again according to priorities). A particular kind of internal event is the *goal*, postfix *G*, that stop being attempted as soon as it succeeds for the first time.

2.3 Present Events

When an agent perceives an event from the "external world", it doesn't 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*.

2.4 Actions

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.

2.5 Past events

Past events represent the agent's "memory", that makes her capable to perform its future activities while having experience of previous events, and of its own previous conclusions. As we have seen in the examples, past event are indicated by the postfix *P*. For instance, *alarm_clock_ ringsP* is an event to which the agent has reacted and which remains in the agent's memory. Each past event has a timestamp T indicating when the recorded event has happened. Memory of course is not unlimited, neither conceptually nor practically: it is possible to set, for each event, for how long it has to be kept in memory, or until which expiring condition. In the implementation, 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. Implicitly, if a second version of the same past event arrives, with a more recent timestamp, the older event is overridden, unless a directive indicates to keep a number of versions.

3 DALI Communication Architecture

The DALI communication architecture consists of four levels (see Fig.1). 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 to 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. The fourth level implements a filter for the outcoming messages.The DALI/FIPA protocol consists of the main FIPA primitives, plus few new primitives which are particular to DALI.
In DALI, an out-coming message has the form:

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

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

$$\begin{aligned} message(\ & receiver_address, receiver_name, sender_address, sender_name, \\ & language, ontology, content) \end{aligned}$$

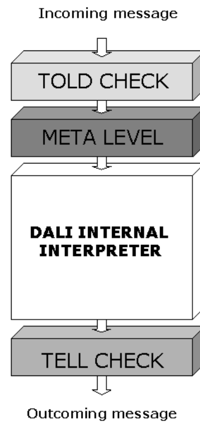

Fig. 1. The communication architecture of a DALI agent

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. Each DALI agent is also provided with a distinguished procedure called *meta*, which is automatically invoked by the interpreter in the attempt to understand 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. The internal interpreter determines the behavior of an agent on the grounds of the DALI logic program and the corresponding rules and events contained in it. Therefore, this level generates the autonomous, reactive, proactive and social abilities of a DALI software entity.

4 Operational semantic

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. The rules of the operational semantic show how the states

of an agent change according to 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 representing 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 defined as a triple. More precisely, this triple is composed of:

- the logic program P_{Ag};
- the internal state $IS_{Ag} \equiv < E, N, I, A, G, T, P >$;
- a particular attribute *Mode* describing what the interpreter is doing.

Therefore, the global state of a DALI agent can be written as:

$< Ag_x, S_{Ag_x} > \equiv < Ag_x, < P_{Ag}, IS_{Ag}, Mode_{Ag} >>$

The tuple $< E, N, I, A, G, T, P >$ is 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.

Each transition 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 of all 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 few 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 semantic 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 another 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.

$R1: < Ag_1, < P, IS, received_message_x >> \stackrel{L_1}{\longrightarrow} < Ag_1, < P, IS, told_x >>$

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 does not affect the behavior of the software entity because the message is deleted. The sender agent is informed on the elimination.

$R2: < Ag_1, < P, IS, received_message_x >> \stackrel{not(L_1)}{\rightarrow} < Ag_1, < P, IS, wait >>$

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.

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

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

$R4: < Ag_1, < P, IS, told_x >> \stackrel{not(L_2), L_3}{\rightarrow} < Ag_1, < P, IS, apply_ontology_x >>$

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 of going in *wait mode* we can suppose that the agent will call a learning module but at this moment we have not implemented this functionality.

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

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

$R6: < Ag_1, < P, IS, apply_ontology_x >> \stackrel{L_2}{\rightarrow} < Ag_1, < P, IS, understood_x >>$

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

$R7: < Ag_1, < P, IS, apply_ontology_x >> \stackrel{not(L_2)}{\rightarrow} < Ag_1, < P, IS, wait >>$

A known message enters in the processing phase of the interpreter and it waits to be examined.

$R8: < Ag_1, < P, IS, understood_x >> \rightarrow < Ag_1, < P, IS, process_x >>$

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.

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

If the response is false, the message is immediately sent and the queue of the messages (actions) changes.
$R10: < Ag_1, < P, IS, send_x >> \stackrel{not(L_4), L_9}{\rightarrow} < Ag_1, < P, NIS, sent_x >>$

If the constraints of tell level are satisfied, the message is sent.
$R11: < Ag_1, < P, IS, tell_x >> \stackrel{L_5, L_9}{\rightarrow} < Ag_1, < P, NIS, sent_x >>$

A message sent by the agent Ag_1 is received by the agent Ag_2 that goes in *received message mode*.
$R12: < Ag_1, < P, IS, sent_x >> \rightarrow < Ag_2, < P, IS, received_message_x >>$

If the message does not overcome the tell level because the constraints are false, the agent returns to *wait mode*.
$R13: < Ag_1, < P, IS, tell_x >> \stackrel{not(L_5)}{\rightarrow} < Ag_1, < P, NIS, wait >>$

This last rule shows how, when a message is sent, the corresponding action becomes past event.
$R14: < Ag_1, < P, IS, sent_x >> \stackrel{L_{10}}{\rightarrow} < Ag_1, < P, IS, wait >>$

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

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

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.

$$< Ag_1, < P, IS, process_{send_message} >> \stackrel{L_{13}, L_9}{\rightarrow} < Ag_1, < P, NIS, send_{primitive} >>$$

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.

$$< Ag_1, < P, IS, process_{propose} >> \stackrel{L_{15}, L_{11}, L_9}{\rightarrow} < Ag_1, < P, NIS, send_{accept_proposal} >$$

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

$$< Ag_1, < P, IS, send_{accept_proposal} >> \stackrel{L_8,L_9}{\rightarrow} < Ag_1, < P, NIS, send_{inform} >$$

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.

$$< Ag_1, < P, IS, send_{accept_proposal} >> \stackrel{L_9}{\rightarrow} < Ag_1, < P, NIS, send_{failure} >$$

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

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

If the conditions in the $propose$ are unacceptable, the response can be only a $reject_proposal$.

5 The filter of the DALI architecture and the trust problem

5.1 The communication filter

In Multiagent Systems, the agents interact by exchanging messages in order to carry on useful cooperation or competition strategies. Interactions are needed to maintain the coordination between software entities, resolve conflicts and exchange informations to reach a goal. Coordination of software entities can be expressed in terms of coordination models and languages. In other words, a coordination model provides a framework in which the interaction of individual agents can be expressed and can be embodied in a (software) coordination architecture. In any real application, the cooperation between agents raises the problem of security. Real world applications, especially those working with public networks such as the Internet, must be carefully designed and developed, taking into consideration security issues.

In this context, an agent if not suitably self-defending can suffer from damages to its knowledge base or to its behavioral rules. This leads to the inability of the agent either because it has a wrong or devoid knowledge or because its rationality is affected. A DALI agent communicates with other software entities by using a single channel, the external event. Through this channel an agent receives messages and information, potentially very important for its survival and efficiency. It may happen that an agent sends to another one a message with a wrong content, intentionally or not, thus potentially bringing a serious damage. How can an agent recognize a correct message? And a wrong message? The filter adopted in DALI tries to answer, as far as possible, this question.

In Section 3 we shortly described the architecture of DALI language. Now our intention is to show more deeply how this filter works and how it is the practical result

of a research work initiated several years ago [15] by Costantini et al. In that paper, the authors introduced a representation of agents by means of theories and a communication among agents based on reflection within the metalogic programming paradigm and suggested a first idea about a communication filter based on predicates referring to the mental state of the agents. When a message is received, it is examined by a check level 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 respected. If the conditions are false, this security level eliminates the supposedly wrong message.

We have constrained the reception of messages by restricting the range of allowed utterances to the FIPA/DALI primitives, according to additional conditions defined by the user, or, in perspective, learned by the agent herself. For example, a filtering condition can be reliability of the sender agent. We specify the DALI filter by means of meta-level rules defining the distinguished predicates tell and told. These meta-rules are contained in a separate file, and can be changed without affecting or even knowing the DALI code. Then, communication in DALI is elaboration-tolerant with respect to both the protocol, and the filter. The filter that checks the message that the agent receives is specified by providing a definition for the distinguished predicate told. Whenever a message is received, with content part primitive(Content,Sender)) (that we have discussed before) the DALI interpreter automatically looks for a corresponding told rule, which is of the form:

$$told(Sender, primitive(Content)) : -constraint_1, \ldots, constraint_n.$$

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. Eventually, the agent will react to this event, by performing whatever is required by the message. Otherwise, the message is discarded. Semantically, this can be understood as implicit reflection up to the filter layer, followed by a reflection down to whatever activity the agent was doing, with or without accepting the message. For a detailed and general semantic account of this kind of reflection, the reader may refer to [**?**].

Below we propose a number of examples of filtering rules. Notice however that each agent can have her own set of filtering rules. Since she takes these rules from a separate file, she can vary her filtering criteria (by importing a different file) according to the context she is involved in.

The following rule constrain a software entity to accept a send_message primitive if she remembers (presumably from past experience) that the sender is reliable, and believes that the content is interesting. By using the primitive send_message an agent can invoke a reactive rule of a receiver agent.

$$\begin{aligned} &told(\ Sender_agent, send_message(External_event)) : - \\ &\quad not(unreliableP(Sender_agent)), interesting(External_event). \end{aligned}$$

In the next told rule we use the FIPA primitive confirm. An agent accepts a con-

firm if the Sender is reliable and the proposition is consistent with her knowledge base. The proposition is recorded as a past event and kept, according to the directive specified in this rule, 200 seconds.

$$told(\ Sender_agent, confirm(Proposition), 200) : - \\ not(unreliableP(Sender_agent)), consistent_with_knowledge_base(Proposition).$$

Finally, we can suppose that a proposal to do an action for an agent is acceptable if she is specialized for that action and the preconditions are acceptable.

$$told(\ Sender_agent, propose(Action, Preconditions)) : - \\ specialized_for(Action), acceptable(Preconditions).$$

The flexibility of the filter allows also to check if the communication protocol is respected from the incoming or outcoming messages. We show an example of this ability by using the propose and accept_proposal primitives. An agent in fact can receive an accept_proposal only in response to propose. The agent remembers as a past event (for 200 seconds) that she has accepted the proposal to perform an action. This information can be used by an internal event for further inferences.

$$told(\ Sender_agent, accept_proposal(Action, Conditions), \\ in_response_to(Message), 200) : - \\ not(unreliableP(Sender_agent)), functor(Message, F, _), F = propose.$$

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 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 that each DALI agent imports as a library, so that the communication protocol can be seen an “input parameter ”of the agent. The syntax of a tell rule is:

$$tell(Receiver, Sender, primitive(Content)) : -constraint_1, \ldots, constraint_n$$

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 if the goal *tell(Receiver,Sender,,primitive(Content))* succeeds. For example, this 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. Via rules like this one we can considerably reduce useless exchange of messages.

$$tell(\ Agent_To, Agent_From, refuse(Something, Motivation)) : - \\ arg(1, Something, Primitive), functor(Primitive, F), \\ (F = is_a_fact; F = query_ref).$$

The problem of a secure interaction between the agents is also treated in [17, 19]. However, [17] 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 [18]. 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 [19] 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.

5.2 Introducing trust in the communication filter

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. We will now explain 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 type of social knowledge and encodes evaluations about which agents

can be taken as reliable sources of information or services. We focus on a practical issue: how the level of Trust influences communication and choices of the agents. We defined a trust as a DALI past event that the agent remembers forever. This event has a following structure:

$trustP(Agent_x, Agent_y, Trust_value)$

and it means that the $Agent_x$ trust in the $Agent_y$ with the value Trust_value. But, why did we choose to get together trust and past events? We thought that the trust of an agent toward another could depend on behavior of the second agent as time passed. A correct behavior will augment the trust value. In order to link the experience to the trust concept we used two kinds of DALI events: a past event to define the trust predicate and an internal event to combine the behavior of an agent to trust. We will show this concept by means of a simple example: we suppose that an $agent_x$ is composed by the following DALI logic program.

$askE(Y, Q) :> clause(agent(A), _), messageA(Y, agree(Q, A)).$

$trust_true(Y) : -askP(Y, Q), informP(agree(Q),$
$values(yes), Y), clause(know(Q), _).$
$trust_true(Y) : -askP(Y, Q), informP(agree(Q),$
$values(no), Y), not(clause(know(Q), _)).$
$trust_trueI(Y) : -increment_trustA(Y).$

$trust_false(Y) : -askP(Y, Q), informP(agree(Q), values(no), Y),$
$clause(know(Q), _).$
$trust_false(Y) : -askP(Y, Q), informP(agree(Q), values(yes), Y),$
$not(clause(know(Q), _)).$
$trust_falseI(Y) : -decrement_trustA(Y).$

$trust(_, Y, _) : -decrement_trustP(Y); increment_trustP(Y).$
$trustI(A, Y, V) : -choose(A, Y, V).$
$choose(A, Y, V) :> trustP(A, Y, V1), increment_trustP(Y), KisV1 + 1, V = K...$
$choose(A, Y, V) :> trustP(A, Y, V1), decrement_trustP(Y),$
$V1 > 0, KisV1 - 1, V = K...$

When this agent receives the external event $ask(agent_y, information)$, asks the $agent_y$ for the information. If the $agent_x$ knows the response (know(information)), she can check if the $agent_y$ has been honest. The first internal event $trust_true$ triggers the reaction and augments the trust of the $agent_x$ towards the $agent_y$ if the first agent remembers that she asked an information, she had the response and she verified the correctness of the response. The internal events $trust_false$ decrements, using the opposite policy, the trust. The last internal event $trust(Ag_x, Ag_y, Trust_value)$ increments/decrements the trust and becoming a past event after the reaction creating both the link with the experience of the $agent_x$ and the predicate see above that the agent can use in order to take decisions in the future. A this point, we introduced this trust past event in the body of tell/told rules specifying that a message can be sent or

received only if the trust value is greater that a fixed threshold:

$$told(Sender, send_message(_)) :- clause(agent(Ag), _),$$
$$trustP(Ag, Sender, N), N > threshold.$$

$$tell(Receiver, Ag, send_message(_)) : -trustP(Ag, Receiver, N_1), N_1 > threshold_1.$$

In this way, we have created a correlation between the communication and the experience that can protect an agent from intentional o reiterated damages. These rules contain the primitive $send_message$, but we can adopt similar rules for all others FIPA primitives. In order to improve our approach to trust, in the future we could introduce in the body of tell/told rule more sophisticated algorithms. Finally, the DALI language provides particular actions that manage the past events and increment/decrement the value of trust on the grounds of the expected behavior of the other agents involved in the coordination system. These actions are: $drop_past, add_past$ and set_past: $drop_past/add_past$ deletes/adds a past event while set_past sets the time of the memorization of a past event. Now we will show by means of an example how trust can influence the behavior of our agents.

6 An example

We consider a cooperation context where an ill agent asks her friends in order to find a competent specialist. When the agent has some particular symptoms, she calls a family doctor that recommends her to find a lung doctor. The patient, through a yellow pages agent, knows the names and the distance from her city of two specialists and asks the friends about them. The patient has a different degree of trust on her friends and each friend has a different degree of competence about the specialists. Moreover, the patient knows the ability of the friends about medical matters: a clerk will be less reliable than a nurse. In order to introduce this concept within the agent patient, we adopt some past event with the suffix skill:

$skillP(friend_nurse, S_1)$ and $skillP(friend_clerk, S_2)$

where $S_1 > S_2$. We suppose that the ill agent receives a message only if she has on the agent sender a trust value greater than a threshold 4:

$$told(Ag, send_message(_)) : -trustP(_, Ag, N), N > 4.$$

We can adopt a similar rule also for the outcoming messages. Now we face the trust problem and show more interesting DALI rules of the agents involved in this example. The cooperation activity begins when the agent Ag becomes ill, and communicates her symptoms to doctor. If those 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.

$consult_lung_doctorE(M) :> clause(agent(Ag), _), choose_if_trust(M, Ag).$
$choose_trust(_, Ag) :- \quad clause(i_know_lung_doctor(Sp),_{)},$
$\quad trustP(Ag, Sp, V_1), V_1 > 0, go_to_lung_doctorP(Sp).$
$choose_trust(M, Ag) :- messageA(yellow_page, send_message(search(M, Ag), Ag)).$

The yellow pages agent returns to patient, by using 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.

$take_ information_about(Sp) :- clause(lung_doctor(Sp), _).$
$take_ information_aboutI(Sp) :> clause(agent(Ag), _),$
$\quad messageA(friend1, send_message(what_about_competency(Sp, Ag), Ag)),$
$\quad messageA(friend2, send_message(what_about_competency(Sp, Ag), Ag)).$

Each friend, having the information $competent(lung_doctor_x, Value)$ about the ability of the specialists, sends an $inform$ containing the evaluation of the competence.

$what_about_competencyE(Sp, Ag) :> choose_competency(Sp, Ag).$
$choose_ competency(Sp, Ag) :- clause(competent(Sp, V), _),$
$\quad messageA(Ag, inform(lung_doctor_competency(Sp, V), friend_x)).$
$choose_ competency(Sp, Ag) :-$
$\quad messageA(Ag, inform(dont_know_competency(Sp), friend_x)).$

The patient is now aware of the specialist and friend's competency and has a value of trust $trustP(Ag_x, Friend_y, Trust_value)$ and a value of competence $skillP(Friend_y, Skill_value)$ in the medical matter on the friends consolidated through the time. Moreover, she knows the distance of the specialists from her house. By 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 she be cured? After some time, the patient will do an exam of 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 to a higher value:

$cured \quad (Sp, Friend) :- go_to_lung_doctorP(Sp), follow_adviceP(Friend),$
$\quad not(temperatureP), not(thorax_painP), not(coughP), not(out_of_breathP).$
$curedI \ (Sp, Friend) :> clause(agent(Ag), _), trustP(Ag, Friend, V), V_1 \ is \ V + 1,$
$\quad drop_pastA(trust(Ag, Friend, V)), add_pastA(trust(Ag, Friend, V1)),$
$\quad assert(i_know_lung_doctor(Sp)), set_pastA(trust(Ag, Friend, V), 100),$
$\quad add_pastA(trust(Ag, Sp, 1)), drop_pastA(go_to_lung_doctor(_)).$

If she is still ill, she decreases the trust value on the friend that has recommended the lung doctor:

$no_ cured(Sp) :- go_to_lung_doctorP(Sp), temperatureP.$
$no_ cured(Sp) :- go_to_lung_doctorP(Sp), thorax_painP.$
$no_ cured(Sp) :- go_to_lung_doctorP(Sp), coughP.$
$no_ cured(Sp) :- go_to_lung_doctorP(Sp), out_of_breathP.$
$no_ curedI(_) :> clause(agent(Ag), _), follow_adviceP(Am),$
$trustP(Ag, Am, V), V >= 1, V_1\ is\ V - 1,$
$drop_pastA(trust(Ag, Am, V)), set_pastA(trust(Ag, Am, V1), 1000),$
$add_pastA(trust(Ag, Am, V1)), drop_pastA(go_to_lung_doctor(_)).$

The decrement of the trust value of a friend can affect the check level of communication 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 friend that the incoming message has been eliminated, by using an inform primitive:

$send_ message_to(friend,$
$inform(send_message(what_about_competency(lung_doctor, patient), patient),$
$motivation(refused_message), patient), italian, [])$

where $send_message(what_about_competency(lung_doctor, patient), patient)$ is the eliminated message with the motivation $motivation(refused_message)$.
In our system, trust can change dynamically, so that it's possible that an agent, excluded from the communication because she has a too low value of trust, increases this value by making some actions or by asking other agents to plead her case.

7 Conclusion

In this paper we have faced the trust problem with a simple approach, using cooperating DALI agents and some parameters such as trust and competence which change dynamically. We have also shown how the filter level can work, eliminating the messages that could damage the agents. In the future, we intend to study and implement more realistic algorithms: in particular, we mean to take advantage of some related results of game theory. We also mean to improve the DALI filter level, by introducing forms of meta-reasoning also in the body of tell/told rules.

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. S. Costantini, A. Tocchio and A. Verticchio, *Semantic of the DALI Logic Programming Agent-Oriented Language*, submitted to *Logics in Artificial Intelligence, Proc. of the 9th Europ. Conf., JELIA 2004*.
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. Yuh-Jong Hu, *Some thoughts on agent trust and delegation*, Proceedings of the fifth international conference on Autonomous agents,2001.
10. Yahalom, R. Klein, B. Beth, T. Sch. of Bus, *Trust relationships in secure systems-a distributed authentication perspective* Admin., Hebrew Univ., Jerusalem; This paper appears in: Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on.
11. Dorothy E. Denning, *A new paradigm for trusted systems*, Proceedings on the 1992-1993 workshop on New security paradigms, Little Compton, Rhode Island, United States.
12. Audun Josang, *The right type of trust for distributed systems*, Proceedings of the 1996 workshop on New security paradigms,1996,Lake Arrowhead, California, United States.
13. Catholijn M. Jonker and Jan Treur, *Formal Analysis of Models for the Dynamics of Trust Based on Experiences*, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World,1999,Springer-Verlag.
14. 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.
15. S. Costantini, P. Dell'Acqua and G. A. Lanzarone, *Reflective Agents in Metalogic Programming*, Meta-Programming in Logic (Meta92), A. Pettorossi (ed.), LNCS 649, pp. 135-147, 1992.
16. Colin English, Sotirios Terzis, and Waleed Wagealla, *Engineering Trust Based Collaborations in a Global Computing Environment*, Trust Management: Second International Conference, iTrust 2004, Oxford, UK, March 29 - April 1, 2004. Proceedings, Springer-Verlag Heidelberg.
17. Naftaly H. Minsky and Victoria Ungureanu, *Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems*, ACM Trans. Softw. Eng. Methodol.,2000,ACM Press.
18. Naftaly H. Minsky *The Imposition of Protocols Over Open Distributed Systems*, IEEE Trans. Softw. Eng.,1991,IEEE Press.
19. A. Di Stefano and C. Santoro *Integrating Agent Communication Contexts in JADE*, Telecom Italia Journal EXP, Sept. 2003.