COMUNICAZIONE E LIVELLO DI FIDUCIA IN DALI, UN LINGUAGGIO LOGICO ORIENTATO AGLI AGENTI

COMMUNICATION AND TRUST IN DALI, AN AGENT-ORIENTED LOGIC PROGRAMMING LANGUAGE[1]

Stefania Costantini Arianna Tocchio Alessia Verticchio

SOMMARIO/*ABSTRACT*

L'interazione rappresenta un aspetto importante nei sistemi multi-agente: gli agenti si scambiano messaggi, asserzioni, query, ecc. A seconda del contesto applicativo, gli agenti si scambiano informazioni per allargare le loro conoscenze, per raggiungere i loro obiettivi o per organizzare strategie di cooperazione e di coordinamento. Realisticamente, la comunicazione deve basarsi su una quanche forma di valutazione dell'interlocutore come ad esempio la fiducia, intesa come forma di conoscenza sociale basata su vari fattori, fra cui le interazioni avvenute nel passato. In questo lavoro mostriamo come sia possibile modellare il concetto di "livello di fiducia" ed i cambiamenti che esso subisce nel tempo mediante l'architettura di comunicazione del linguaggio logico orientato agli agenti DALI, e proponiamo un esempio volto a mostrare l'efficacia dell'approccio.

Interaction is an important aspect of Multi-agent systems: agents exchange messages, assertions, queries, etc. This, depending on the application context, can be either in order to improve their knowledge or to reach their goals or to organize useful cooperation and coordination strategies. In real-world applications, communication should be based on some form of evaluation of the other party such as for instance trust, understood as a form of social knowledge based on various factors, among which past interactions. In this paper we demonstrate how to model the "level of trust" and its possible evolution in time by means of the communication architecture of the DALI agent-oriented logic programming language. We also propose an example aimed at showing the effectiveness of the approach.

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

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 either in order to improve their knowledge, or to reach their goals, or to organize useful cooperation and coordination strategies. However, the exchange of information implies a certain degree of risk. In a global environment, entities meet and need to interact with other entities of which they have little or no information about reliability. 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. The reader may refer to [10] for a discussion about trust as a notion that has been developed to help agents to deal with the partial information about the world. How can an agent be sure that the incoming information will not damage its internal state? Should it be always confident? Is it possible to introduce a certain level of reliability in communication? This is a relevant problem, coped with in the literature in different ways. For Josang in [14], trust is a belief that one entity has about another entity. He states that the motivation of 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 [2] give an interesting classification of trust and develop an algorithm based on a concept of recommendation path. Denning [8] argues 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 this paper we will not cope with the definition of trust, or with algorithms to compute trust. Rather, we focus on how to introduce the concept of "level of trust" in inter-agent communication in a flexible way. We mean that an agent must be able to set and update the level of trust according to its beliefs and experiences, and that the level

of trust should affect communication acts up to preventing them. In our approach, we introduce the concept of trust by means of the filter level of the 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 default definition. Basically, the same agent program becomes a different agent (when activated) if equipped with a different communication filter. We have experimented the capabilities of this filter by introducing the trust concept, and the possibility of dynamically increasing/decreasing the trust level. We demonstrate (also by means of a case-study) that the filter can manage sophisticated communication forms, and that changes in relevant parameters such as the level of trust actually affect the behavior of agents.

A real-world application of the approach outlined in this paper might use specific algorithms to compute the level of trust as defined 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 an agent. In this paper, we emphasize how trust can be affected by new knowledge coming from the direct observation of events in the world.

The paper is organized as follows. We start by shortly describing the main features of DALI in Section 2. Then we introduce the communication architecture and the filter in Section 3 where we also comment about related approaches. In Section 4 we show how the level of trust can be defined by means of the communication filter. Finally, in Section 5 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 6 by discussing future directions of this research.

2 The DALI language

DALI [4] [6] [5] is an Active Logic Programming language designed for executable specification of logical agents [15]. A DALI program is a logic program, and in particular it is a Horn-clause program, both syntactically and procedurally (where it is based on an extended resolution). When activated however, a DALI program results in an agent which is capable of reactive and proactive behavior, triggered by several kinds of events, which are syntactically characterized explicitly: external events, internal, present and past events. A DALI program may contain a particular new kind of rules, reactive rules, aimed at coping with events. All the events and actions are time-stamped, so as to record when they occurred. Several aspects of the agent behavior can be tuned by suitable directives. These are aspects (e.g., how often to check for incoming messages) that do not affect the logical semantics [6] of the agent, but affect in a relevant way its run-time behavior. Directives are specified in a separate module, which is added to the agent program when the agent is initialized. Then, on the one hand directives can be modified without even knowing the agent program. On the other hand, the same agent program with different directives results in a different agent (e.g., apparently more quick, more lazy, eager to remember of ready to forget things, etc.). Directives are coped with in the operational semantics of the language [7]. DALI is implemented in prolog, and thus the implementation inherits prolog features.

2.1 External Events

External events are syntactically indicated by the postfix *E*. When an event coming from the "external world" reaches a DALI agent, the agent can perceive it and decide to react. The reaction is specified by a reactive rule which has in its head that external event. The special token $:>$, used instead of $:-$, indicates that reactive rules perform forward reasoning. For instance, 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). When 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 the file of directives.

2.2 Internal Events

Internal events define a kind of "individuality" of a DALI agent, making it proactive independently of the environment, of the user and of the other agents, and allowing it to manipulate and revise its knowledge. 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 an 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 it 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*).

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

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: it has the possibility of reasoning about the event, before (or instead of) triggering a reaction. Reasoning also allows a proactive behavior. In this situation, the event is called present event and is indicated by the suffix *N*.

2.4 Actions

Actions are the agent's way of affecting its 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 it 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*. Past events are kept in the memory of an agent 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 time-stamp, the older event is overridden, unless a directive indicates to maintain a number of versions. Past events record new information and pieces of knowledge that have been either acquired (as external events) or deduced (as internal events) by the agent during its "life". Then, in many application domains they can be understood as *beliefs*, and the set of past events can be understood as the current state of the world form the agent's point of view. Since however DALI is a general-purpose language, we do not explicitly commit to this view and terminology.

3 The Communication Architecture

3.1 Basic Architecture

The DALI communication architecture consists of the following levels. The first level consists of the DALI interpreter. The second 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 third level includes a meta-reasoning layer, that tries to understand message contents, possibly based on ontologies and/or on forms of commonsense reasoning. The DALI/FIPA protocol adopted in DALI consists of the main FIPA primitives, plus few new primitives which are peculiar of DALI. In DALI, an out-coming message is understood as a special kind of action, and has the form:

$$messageA(Receiver, Primitive(Content, Sender))$$

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

$$\begin{array}{l} message(receiver\_address, receiver\_name, \\ \qquad sender\_address, sender\_name, \\ \qquad language, ontology, content) \end{array}$$

3.2 The communication filter

In any concrete application, 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 DALI, when a message is received it is examined by the filter 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 or dangerous message. Otherwise, it is passed to the meta-reasoning layer that consists of a procedure *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 DALI communication filter is defined by means of meta-level rules defining the distinguished predicates *tell* and *told*. Actually, the FIPA/DALI communication protocol itself is implemented by means of a piece of DALI code consisting of default *tell/told* rules. This code is contained in a separate file that each DALI agent imports as an external module, so that the communication protocol can be seen an "input parameter" of an agent. The filter can be extended simply by adding new *tell/told* rules, so as to cope to the application domain at hand. The new rules can be added without affecting or even looking at the agent program. Therefore, communication in DALI is elaboration-tolerant with respect to both the protocol, and the filter. The same agent program, if equipped with a different filter, actually results in a different agent with different communication behavior, and different management of security and trust.

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:

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

where $constraint_i$ can be any condition. If such a rule is found, the interpreter attempts to prove *told(Sender, Primitive(Content))*. If this goal succeeds, then the message is accepted, and *Primitive(Content)* is added to the set of the external events incoming into the receiver agent. Otherwise, the message is discarded.

Symmetrically, the messages that an agent means to send 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 ones not. The choice is made by setting some parameters in the agent initialization file. 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.

The declarative semantics of the filter and meta-reasoning layers is based on the concepts introduced in [3] and [1]. Namely, invoking either *told/tell* or *meta* is understood as implicit *upward reflection* to the corresponding layer, followed by a *downward reflection* to whatever activity the agent was doing. For the operational semantics of the approach, the reader may refer to [7].

3.3 Related Work

The problem of reliable interaction among agents is treated for instance in [17] and [9]. The Moses system, defined in [17], provides a global filtering rule, or "law", for a group of agents, instead of local conditions for every single agent as in DALI. Moreover, in Moses there is a special agent, called *controller*, that monitors all the other agents. Each law in Moses is defined as a prolog-like rule whose body specifies: 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 constraints on the communication and not 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 an agent. A particularity of the Moses law-governed system is that is possible to update on-line the laws [18]. In DALI, presently it is possible to change the rules only statically, though a possible future improvement is to allow an agent to "tune" dynamically its own filter rules.

The Agent Communication Context (ACC), defined in [9] for the JADE multi-agent platform, allows agent interaction laws to be expressed by means of a set of rules applied to each 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. This approach is only apparently similar to the one adopted in DALI. In fact, it is applied only to out-coming messages, while in DALI we submit to the filter both the received messages and the sent messages. Also, the structure of a DALI filter rule is different and more flexible: an ACC 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 in DALI. Then, even though both approaches use the concept of communication filter, we believe that there are notable differences also due to ability of DALI to draw inferences and to reason, that can be hardly simulated in a java-based approach like JADE.

4 Trust in the communication filter

As we have seen, the filter layer of the DALI communication architecture allows an agent to reason explicitly about communication, and to decide whether to send or receive a message on the basis of several possible parameters, including its "mental state", its own definition of reliability and security and its own degree of belief, trust, etc. about the "external world", including the other agents. We will now we demonstrate the capabilities of the filter by proposing an approach aimed at modeling the "level of trust" and its possible evolution in time. Trust is intended here as a kind of social knowledge that encodes evaluations on whether other agents can be taken as reliable sources of information or services. We focus on a practical issue: how the *level of trust* may influence communication and choices of the agents. We define trust by means of a special predicate represented as a DALI past event, with the following form:

$$trustP(agent_x, agent_y, trust\_value)$$

It means that $agent_x$ trusts $agent_y$ with the degree indicated by $trust\_value$. The first argument (the agent $agent_x$ who trusts the others) is added for the sake of generality, so that agents may for instance exchange their own beliefs about trust. But, why model trust by using past events? Because trust of an agent towards another one may depend on their interactions and on the evolution of both agents in time. For instance, the repeated perception of a "desirable" behavior may presumably increase trust in the other agent. In order to link trust to past experience we use an internal event that updates the level of trust. This can be done by means of any kind of reasoning, and in particular by examining previous communication acts (that, as seen before, are themselves recorded as past events).

Below we demonstrate the approach by means of a simple example. Assume that $agent_x$ has received the information Q by another agent Y. $agent_x$ checks whether the information is correct or not, and modifies the trust level accordingly. Notice that the predicate $trust\_update$ is an internal event which is automatically attempted, thus checking available information, where success (information received and check performed) triggers the execution of the second (reactive) rule, thus determining the execution of an action that will result in the modification of the level of trust.

$$trust\_update(Y) : -receivedP(Y, Q), check(Q, R).$$
$$trust\_updateI(Y, R) :> modify\_trustA(Y, R).$$
$$modify\_trustA(Y, ok) : -increment\_trustA(Y).$$
$$modify\_trustA(Y, ko) : -decrement\_trustA(Y).$$

The internal event $trust(Ag_x, Ag_y, Trust\_value)$ makes the updates of the trust level effective. The meaning of the rules is that if an increment/decrement action has been done (the past event $decrement\_trustP(Y)$ or $increment\_trustP$ respectively is in the knowledge base) then the present trust value $trustP(A, Y, V)$ of the agent (who identifies itself by $myself(A)$) in the other agent Y is updated by a constant k. The reaction does nothing, but causes the new past event $trustP(A, Y, New\_V)$ to recorded, thus recording the updated level of trust.

$$trust(A, Y, New\_V) : -myself(A), trustP(A, Y, V), decrement\_trustP(Y), decrement(V, k, New\_V).$$
$$trust(A, Y, New\_V) : -myself(A), trustP(A, Y, V), increment\_trustP(Y), increment(V, k, New\_V).$$
$$trustI(A, Y, New\_V) :> myself(A).$$

This creates the link between the experience of $agent_x$ and its trust in the other agents, that $agent_x$ will be able to exploit in future decisions.

At this point, we are able to introduce the trust past event in the body of *tell/told* rules. For instance, the rules below specify that a message can be sent/received only if the trust value in the other agent is greater that a fixed threshold:

$$told(Sender, Primitive(Content)) : - myself(Ag), trustP(Ag, Sender, N), N > threshold.$$

$$tell(Receiver, Ag, Primitive(Content)) : - trustP(Ag, Receiver, N_1), N_1 > threshold_1.$$

In this way, we have stated a correlation between communication and experience that can protect an agent from communication acts that might result in a risk of damage. We can adapt the above rules so as to apply them either to some or to all communication primitives.

The DALI language provides distinguished actions to manage past events, that can be used to 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. In the next section we will show by means of an example how trust can influence the behavior of agents.

5 An example

Consider a cooperation context where an ill agent asks its friends in order to identify a competent specialist. Initially, the agent has some particular symptoms and asks a family doctor, that recommends it to consult a lung doctor. The patient, through a yellow pages agent, becomes aware of the names and of the distance from its city of two specialists and asks some friends about them. The patient has a different degree of trust in its friends and each friend has a different degree of competence about the specialists. Moreover, the patient has its beliefs about the ability of the friends about medical matters: a clerk will be presumably less reliable than a nurse. In order to introduce this concept within the agent patient, we use a past event related to a predicate *skill*:

$skillP(friend\_nurse, S_1)$. and
$skillP(friend\_clerk, S_2)$.

where $S_1 > S_2$. We set the filter so that the ill agent receives a message only if the trust in the sender agent has a value greater than a threshold (e.g., 4):

$told(Ag, send\_message(\_)) : -trustP(\_, Ag, N), N > 4.$

We can adopt a similar rule also for the out-coming messages. The cooperation activity begins when agent Ag becomes ill, and communicates its symptoms to a doctor. If those symptoms are serious, the doctor advises the patient to find a competent lung doctor. If the agent knows a specialist Sp and has a positive trust value V_1 on it, it goes to this lung doctor, else it asks a yellow page agent.

$choose\_trust(Sp, Ag) : -$
$\quad myself(Ag),$
$\quad i\_know\_lung\_doctor(Sp),$
$\quad trustP(Ag, Sp, V),$
$\quad V > 0, go\_to\_lung\_doctorP(Sp).$
$choose\_trust(Sp, Ag) : -$
$\quad myself(Ag),$
$\quad messageA(yellow\_page,$
$\quad\quad send\_message(search(Sp, Ag), Ag)).$

The yellow pages agent returns to the patient, by using the $inform$ primitive, a list of lung doctors. Now the patient must decide which lung doctor is more competent and reliable. How can it choose? It asks its friends for help.

$take\_information\_about(Sp) : -$
$\quad lung\_doctor(Sp).$
$take\_information\_aboutI(Sp) :>$
$\quad myself(Ag),$
$\quad messageA(friend1, send\_message($
$\quad\quad what\_about\_competency(Sp, Ag), Ag)),$
$\quad messageA(friend2, send\_message($
$\quad\quad what\_about\_competency(Sp, Ag), Ag)).$

Each friend will receive from Ag the message. If the message passes their communication filter, its content will result in the external event $what\_about\_competencyE(Sp, Ag)$. If the friend has the information $competent(lung\_doctor_x, Value)$ about the ability of the specialists, it will send back an $inform$ containing the evaluation of the competence.

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

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

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 in time. Moreover, it knows the distance of the specialists from its house. By using a simple rule that joins those parameters, it assigns an esteem to each advice:

$specialist\_evaluation(lung\_doctor_x, friend_y, Value).$

The ill agent will choose the lung doctor by accepting the advice having the greater $Value$ and will go to that specialist. Will it be cured? After some time, the patient will revise the state of its health. If it shows no symptom (temperature,thorax pain, cough, out of breath), it increases the trust for the friend that has recommended the lung doctor and sets the trust on that specialist to a high value v:

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

$curedI(Sp, Friend) :>$
$\quad myself(Ag), increment\_trustA(Friend),$
$\quad assert(i\_know\_lung\_doctor(Sp)),$
$\quad add\_pastA(trust(Ag, Sp, v)),$
$\quad drop\_pastA(go\_to\_lung\_doctor(\_)).$

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

$not\_cured(Sp, Am) : -$
$\quad go\_to\_lung\_doctorP(Sp),$
$\quad follow\_adviceP(Am), symptoms.$

$not\_curedI(\_, Am) :>$
$\quad decrement\_trustA(Am),$
$\quad drop\_pastA(go\_to\_lung\_doctor(\_)).$

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 trust in that agent is less than the 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:

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

where

$send\_message(what\_about\_competency($
$\quad\quad lung\_doctor, patient), patient)$

is the eliminated message, with motivation $motivation(refused\_message)$.

As in our approach trust can change dynamically, it is however possible that an agent, excluded from the communication because it has a low value of trust, increases this value by making some "desirable" actions or by asking other agents to plead its case.

6 Conclusion

In this paper we have presented a simple approach for modeling trust-based cooperation in communicating DALI agents. This by introducing parameters such as trust and competence which change dynamically. We have also shown how the filter layer of DALI agents works, and makes an agent able to eliminate presumably useless or potentially dangerous messages. 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 communication filter, 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, *J. of Logic and Computation*, Vol. 10, N. 6, December 2000, Oxford University Press, UK.

[2] T. Beth, B. Klein and R. Yahalom. *Trust relationships in secure systems-a distributed authentication perspective Proceedings on the IEEE Computer Society Symposium on Research in Security and Privacy*, 1993.

[3] S. Costantini, P. Dell'Acqua and G. A. Lanzarone, Reflective Agents in Metalogic Programming, A. Pettorossi (ed.), *Meta-Programming in Logic (Meta92)*, LNCS 649, Springer-Verlag, Berlin, pp. 135-147, 1992.

[4] 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.

[5] 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.

[6] 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.

[7] S. Costantini, A. Tocchio and A. Verticchio. A Game-Theoretic Operational Semantics for the DALI Communication Architecture, In: *Proceedings of WOA'04, held in Turin, November 29 – December 1, 2004*.

[8] D. E. Denning, A new paradigm for trusted systems, *Proceedings on the 1992-1993 workshop on New security paradigms*, Little Compton, Rhode Island, United States, 1993.

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

[10] C. English, S. Terzis and W. Wagealla. Engineering Trust Based Collaborations in a Global Computing Environment, *Trust Management: Second International Conference, iTrust 2004*, held in Oxford, UK, March 29 - April 1, Springer-verlag, Berlin, 2004.

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

[12] H. Yuh-Jong Hu. Some thoughts on agent trust and delegation, *Proceedings of the fifth international conference on Autonomous agents*, 2001.

[13] C. M. Jonker and J. 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*, Springer-Verlag, Berlin 1999.

[14] A. Josang, The right type of trust for distributed systems, *Proceedings of the 1996 workshop on New security paradigms*, held in Lake Arrowhead, California, 1996.

[15] 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.

[16] P. Mc Burney, R. M. Van Eijk, S. Parsons, L. Amgoud. A Dialogue Game Protocol for Agent Purchase Negotiations, *J. of Autonomous Agents and Multi-Agent Systems*, Vol. 7 No. 3, November 2003.

[17] N. H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems, *ACM Trans. on Software Engineering Methodologies*, ACM Press, 2000.

[18] N. H. Minsky. The Imposition of Protocols Over Open Distributed Systems, *IEEE Trans. on Software Engineering*, IEEE Press, 1991.