A FIPA compliant Goal Delegation Protocol

♣Federico Bergenti, ♦Luis Miguel Botelho, ♣Giovanni Rimassa, ♣Matteo Somacher

♣AOT Lab - Dipartimento di Ingegneria dell'Informazione
Parco Area delle Scienze 181/A
43100 Parma, Italy
+39 0521 905708, +39 0521 905712
bergenti@ce.unipr.it, rimassa@ce.unipr.it,
somacher@ce.unipr.it,

♦Department of Computer Science of ISCTE
Av. Das Forças Armadas
Edifício ISCTE, 1600 Lisbon, Portugal
+351 21 7935000
Luis.Botelho@iscte.pt

ABSTRACT

This paper presents an interaction protocol, built on top of FIPA ACL, allowing an agent to delegate a goal to another agent, in the form of a proposition that the delegating agent intends its delegate to bring about. The proposed protocol addresses the concrete needs of a service that is to be deployed within the AgentCities network, but also helps to highlight some issues that are related to the FIPA ACL itself and its usage to build more complex agent interaction blocks.

Keywords

Interaction protocols, goal-delegation, BDI, FIPA-ACL semantics

1. INTRODUCTION

The AgentCities project aims at building an open, worldwide network [1] of agent based services, relying on FIPA compliant agent platforms. The participants to the various incarnations of AgentCities project believe that such a widespread and heterogeneous test bed is key to support the transition of Multi Agent Systems technologies from research labs to actual, deployed applications. The AgentCities effort is also quite interesting for the FIPA organization, because it will validate the whole set of the FIPA specifications (not just the FIPA ACL) on the widest scale so far.

Within the arena of distributed software infrastructures, FIPA promotes a landscape where applications are composed by *agents* receiving life support from *platforms*; FIPA tries to support both *agent-level* and *platform-level* interoperability through a comprehensive set of specifications. At the agent level, FIPA mainly deals with ACL, interaction protocols, message content and message ontology issues. Though the FIPA ACL is provided with a semantics formally rooted in multi-modal BDI logics, it is generally accepted that FIPA does not mandate a BDI architecture for agents, but only that observable agent behaviour can be interpreted within a BDI framework. Recognizing this suggests that a major feature of the FIPA infrastructure is the support for heterogeneous agent societies, where different members have different internal complexity. All of them will enjoy autonomy and sociality, but only a subset of them will really be gifted with an internal architecture providing reasoning capabilities.

Such a vision strives for *semantic scalability*, where software components of different internal complexity still exhibit a behaviour compliant with the FIPA ACL semantics; this becomes even more important when MAS technology tackles the new deployment scenarios arising from the convergence between the Internet and the wireless environments [3].

This paper proposes an interaction protocol to perform goal delegation between two agents, in the form of a proposition that the delegating agent wants the delegate agent to bring about. Section 2 explains the traits and usefulness of the goal delegation operation in the context of MAS, and clarifies the reasons for implementing goal delegation as an interaction protocol in the FIPA infrastructure environment. Section 3 describes the interaction protocol as a Finite State Machine decorated with semantic annotations, and shows its FIPA compliance and soundness. Lastly, section 4 puts the protocol in the practical context that caused its design in the first place: an Event Organizer service that is to be set up in the framework of the AgentCities project.

2. MOTIVATION AND REQUIREMENTS

Goal delegation arises quite naturally among cooperative, rational agents: every agent pursues its own goals, goal partitioning is a standard divide-and-conquer strategy, and in a collaborative environment there generally are enough hierarchy and trust relationships, so that an agent is likely to find some other one to delegate a sub goal to. When considered from an agent coordination perspective, goal delegation has two main facets:

- *Delegation of commitment*. This means that the delegate agent should embrace the intentions of the delegating agent, trying to achieve the goal as if it were one of its own. From the delegating agent point of view, this requires a kind of trust in the delegate good will: the delegating agent has to believe that the delegate is trustworthy and will honestly try to achieve the goal.
- *Delegation of strategy*. Delegating a declarative goal instead of an operational plan means that the delegating agent is interested only in the resulting outcome and not in the specific way the delegate achieves it. Thus, the delegating agent not only trusts the delegate good will, but also its skills. The delegating agent has to believe the delegate agent knows how to achieve the goal.

In [5] the authors analyse several aspects of trust in the perspective of the Information Society, taking into account both human and software agents, relating the theory of trust to computer security issues and stressing how computer mediated communication creates several new trust related issues. Our paper only deals with software agents, following a rather rigid and precise behaviour that relies on FIPA ACL semantics and the

proposed interaction protocol; however, a major aim of the AgentCities project is to insert such agents into the global Information Society, made by software, hardware and human participants. So, though the general considerations about trust at large don't directly affect the subject of this paper, they still remain in its conceptual landscape.

The two aforementioned facets of the goal delegation operation correspond to the *core trust* in *competence* and *disposition*, discussed in [7] as the basis for a trust relationship between two agents. In [5], the authors also observe that some additional mental attitudes are required in the delegating agent, in order for it to develop a delegation disposition toward its (about to become) delegate. These mental attitudes are the *dependence belief* and the *fulfilment belief*. *Dependence belief* amounts to believing that the goal achievement critically depends on the delegate agent, or at least that the goal can be achieved more efficiently by relying on the delegate agent. *Fulfilment belief* consists of believing that the goal will be achieved due to the delegate contribution.

The dependence belief is not directly addressed by our protocol, but stays implicit in the acquaintance structure of a specific application. For instance, performing a yellow pages search or running a service discovery protocol could result in the delegating agent getting a list of agents it can rely on for the task to delegate. This approach bases dependence belief on an existing service level infrastructure, by assuming that an agent that makes a service available through the infrastructure is indeed capable to provide the service, with an appropriate quality level. Such an assumption is equivalent to a shared trust that all the agents have towards the service level infrastructure (be it a directory service, a discovery protocol or whatever), following the approach of having a TTP (Trusted Third Party) mediating among conflicting stakeholders, which is quite common in computer security. Having the infrastructure act as a TTP requires it to be both reliable (it does not crash) and trustworthy (it does not cheat): in the AgentCities.RTD EU project a whole work package is devoted to the design and deployment of the AgentCities Network Architecture, which will have suitable reliability and trustworthiness features. Anyway, even if the current assumption holds for the AgentCities network, the authors acknowledge that a TTP is not available in every application scenario, but they leave the issue of global trust (i.e. including all the infrastructure) to discussion and future development.

The fulfilment belief, instead, is taken into account in our protocol, in that the goal delegation proper is decoupled from result notification. More clearly, when the goal is delegated, the delegating agent believes the goal will be achieved, but the delegate agent, after finding a plan and trying to execute it, tells the delegating agent whether the goal has been achieved or not, thus providing a chance for fulfilment belief revision.

The concept of goal delegation even goes beyond the specific domain of agent coordination, to enter the field of software engineering at large. The TROPOS methodology [12] performs an early requirement analysis phase that identifies and describes the various stakeholders in terms of their goals and the dependency among the different actors. When moving from requirement analysis to system architecture definition, actors can be mapped to software agents and goal delegation looks like a natural approach to implement the stakeholder dependencies identified during requirement engineering. TROPOS is meant to be a comprehensive software engineering methodology, covering the development process from early requirements engineering to detailed design and implementation; however, at its present level of development, TROPOS still concentrates more on the early development phases. Moreover, when considering AOSE methodologies, they are rather original in the analysis phase but become more and more similar to object-oriented ones along the development process phases; at present, agent-oriented design is very much like object-oriented design, and mostly stresses the role and organizational models with respect to the interaction aspects. We believe that valuable and reusable agent-oriented design components are to be found not only among structural role models, but also among behavioural conversation patterns. This paper presents one such component, originally motivated by a specific need but whose applicability is wide enough to be of general interest. Our goal delegation protocol acts within the current FIPA infrastructure and, in our opinion, can shed some light over the relationships among the various elements that compose the FIPA communication model, namely the FIPA ACL, FIPA content languages and ontologies, and interaction protocols.

FIPA agents are autonomous social software components whose external behaviour can be described with a BDI model. The semantic scalability promoted by FIPA suggests taking different approaches for different agent roles during design, depending on the sophistication and internal complexity needed for each role. Recognizing this suggests that a major feature of the FIPA infrastructure is the support for heterogeneous agent societies, where different members have different levels of internal complexity. All of them will enjoy autonomy and sociality, but only a subset will really be gifted with an internal architecture providing reasoning capabilities. In such a heterogeneous society, hierarchical collaboration can be achieved through either plan execution delegation or goal delegation. Delegating the execution is more likely to be used to coordinate the leaves of the hierarchy, probably made by the simplest agents wrapping physical actuators or reactive software servers. Delegating a goal will be surely used between reasoning capable agents, but also in all those cases where a looser coupling between the delegating and the delegate agent is desirable: the delegate agent could generate an utility function from the goal and set up a negotiation process, or a medium complexity agent could have a compiled-in set of plans to try out to achieve a pre-defined family of goals. Both the negotiating agent and the fixed-plans agent lack a full-fledged planning component, but they can still grant the loose coupling granted by goal delegation with respect to plan execution delegation.

While the plan execution delegation can obviously be implemented using FIPA ACL *request* communicative act and the *FIPA-Request* interaction protocol, there is no similar ready-made support for goal delegation.

In principle, a goal delegation design component can use any layer of the FIPA communication model: since we want our goal delegation component to be reusable across application domains, we avoid introducing ontological entities. Our goal delegation protocol is based on a FIPA ACL communicative act, named *achieve* after the KQML performative [11], but which is really a macro-act defined in term of the existing ones, so that the FIPA ACL semantics is left untouched. Moreover, from previous considerations stems that goal delegation is a complex, high-level conversation that involves much more than a single speech act;

therefore we define a complete interaction protocol to carry out goal delegation.

The protocol definition, given in Section 3, uses FIPA SL to define the *achieve* communicative act; this does not clash with our requirement of application domain independence, however, because the subset of FIPA SL we use is only the one required by [9] to specify the FIPA ACL semantics. So, any content language that can express the content of the primitive FIPA ACL communicative acts can replace FIPA SL in the definition of our protocol semantics.

During the past few years, several researchers [14], [6], [13] pointed out that the FIPA ACL semantics, being based on internal mental states of the communicating agents, was not really suited to drive interactions among independently developed agents, acting in open environments. This because the internal state of an agent, by definition, cannot be observed from outside. Instead, it was claimed that a more effective semantics for agent communication could be built around observable entities such as social commitments and agreements. One of the most common arguments to support a social semantics with respect to a mentalistic ones deals with protocol verification: if the communication semantics exploits observable properties, it is easier to design and build on-line compliance verifiers. Within the scope of this paper, the authors are more concerned with protocol design than with protocol verification. Therefore, they stay neutral with respect to the mentalistic vs. observable dilemma; the following section defines the goal delegation protocol using the mentalistic FIPA ACL semantics just because it is the official one. The authors are aware that FIPA set up a Semantics TC [10] to design a semantic framework taking into account social notions, and they believe that the ideas and techniques described in this paper could also be easily restated in a social semantics.

3. PROTOCOL DESCRIPTION

This section has four objectives: define the *achieve* performative, which can be used for goal delegation, design a goal delegation protocol, propose a framework for protocol analysis, and analyse the goal delegation protocol using the presented framework. The first subsection presents the *achieve* performative, which is defined in terms of the *request* and the *inform* performatives. <i, achieve(r, G)> is defined as the sender requesting the receiver to inform it that a plan to achieve G has been executed and G has been achieved. The formal semantics of the *achieve* performative is presented. The sub-section ends with the definition of the goal delegation protocol.

The second subsection defines a framework for protocol analysis. This framework consists of defining the concept of protocol-state. Protocol state changes occur due to message sending/receiving. Each protocol state is the union of preceding state with the set of propositions that must be true if the sender of the state transition message complies with the message semantics and intends the message rational effects. Each protocol state is defined from the perspective of an external observer.

The third subsection analyses the defined goal-delegation protocol using the framework defined in the second subsection.

The fourth and last subsection presents an alternative design of the *achieve* performative and of the goal delegation protocol that fix a minor inconvenience of the design proposed in the first subsection.

The whole section uses the semantics of the FIPA ACL language as defined in [9] using the SL language. This option does not reflect a stance of the authors. The option was made because it is a well-known framework that is being used by the authors in the Agentcities project.

3.1 Goal delegation

If agent i has a goal G it wants to delegate to another agent r, then i may ask r to execute some plan of action whose execution r believes to result in a state of the world in which G is true. Without loss of generality, this section uses FIPA SL in order to keep the presentation more concrete.

In SL, the *Feasible* operator can be used to express the idea that it is possible to execute a given action resulting in the achievement of some state of the world. If an agent believes there is a plan of action ?p such that (Feasible ?p G), the agent believes ?p will bring about G. In SL, the *Done* operator can be used to express the idea that a certain action has been done. If an agent believes Done(A), it believes to be in a state of the world in which the action A has just been executed.

Given the semantics of the ACL inform performative, agent r can only send message <r, inform(i, P)> if r believes P to be true. If r informs i that a certain plan of action has just been executed, r must believe that the plan has actually been executed. The above elements are about all it takes to express goal-delegation messages. The delegating agent must request the delegate to inform it that some plan whose execution is believed to achieve the desired goal has been executed.

In dynamic and uncertain environments, the execution of a plan believed to bring about G does not ensure that G is actually achieved. Therefore, after the execution of the selected plan, the delegate agent must also check that the goal has actually been achieved. The complete message is

$$\langle i, request(r, \langle r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G)\rangle\rangle$$

That is, i requests r to inform it that some plan believed to achieve G has been performed and G has been achieved. According to the semantics of the *inform* performative, r will only send the inform message if it believes those conditions to hold. We propose to extend FIPA ACL with the new performative *achieve* defined as above

$$\langle i, achieve(r, G)\rangle \equiv \langle i, request(r, \langle r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G)\rangle)\rangle$$

In the remaining of this section, we analyse the feasibility preconditions and the rational effect of the *achieve* performative; and propose a protocol to be used for goal delegation. The analysis will rely on the proposed definition. Since the *achieve* performative is defined in terms of the *request* performative, its semantics will result of replacing the content of the request with

$$\langle r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G)\rangle.$$

In the FIPA Specifications [9], the semantics of the *request* message is defined by the following feasibility preconditions and rational effect:

FP of <i, request(r, A)>

– FP(A)[i\r]. The subset of the feasibility preconditions of action A that are mental attitudes of i;

– $B_i(Agent(r, A))$. The sender believes the receiver to be the agent of the requested action;

– $\neg B_i(I_r\ Done(A))$. The sender does not believe that the receiver already intends to perform the requested action otherwise there would be no point in requesting.

RE of <i, request(r, A)>

– $Done(A)$. The sender i can reasonably expect that the requested action will be done.

Replacing A by $<r, inform(i, \varphi)>$ in which

$\varphi \equiv \exists p(Feasible(p, G) \wedge Done(p)) \wedge G$

we obtain:

FP of $<i, achieve(r, G)> \equiv$ FP of $<i, request(r, <r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G))>)>$

– $FP(<r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G))>)[i\backslash r]$. The subset of the feasibility preconditions of $<r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G))>$ that are mental attitudes of i. The feasibility preconditions of the inform message are mental attitudes of the sender alone, which is the responder agent r. Therefore, this is the empty set;

– $B_i(Agent(r, <r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G))>))$. The sender believes the receiver to be the agent of the specified inform message;

– $\neg B_i(I_r\ Done(<r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G))>))$. The sender does not believe that the receiver already intends to send the specified inform message.

RE of $<i, achieve(r, G)> \equiv$ RE of $<i, request(r, <r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G))>)>$

– $Done(<r, inform(i, \exists p(Feasible(p, G) \wedge Done(p)) \wedge G))>)$. The sender i can reasonably expect that the *inform* communicative act will be done.

The above semantics of the *achieve* performative nearly fulfil all the requirements of protocol delegation as defined in section 2:

- The initiator believes that the responder is skilled enough to achieve the goal;
- The initiator believes the responder does not already intend to achieve the goal;
- The initiator does not care about the plan to be used to achieve the goal.

The first requirement can be shown to be implied by the *achieve* feasibility preconditions, because the initiator can only send the *achieve* message if it believes the responder to be the sender of the message informing that the plan has been executed and the goal has been achieved. Following the semantics of the *inform* performative, the responder can only send such a message if it believes to have actually achieved the desired goal. If we assume the responder is aware of the feasibility preconditions of the *inform* performative, the initiator can only believe the responder will be the sender of the message if it also believes the responder to be capable of achieving the goal.

The second requirement is not a consequence of the feasibility preconditions of the *achieve* performative. Actually, the initiator can't believe that the responder already has the intention of informing it that the goal has been achieved. But it is allowed to believe that the responder already intended to achieve the desired goal. This aspect will be the subject of section 3.4.

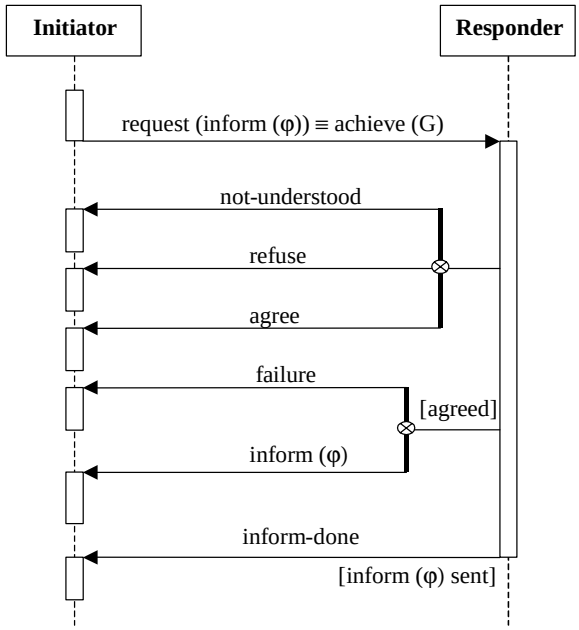

Figure 1 – FIPA Request Protocol for the goal delegation

The third requirement is captured by the proposition the initiator is requesting the responder to send

$$\exists p(Feasible(p, G) \wedge Done(p)) \wedge G$$

The existential quantifier in this proposition means that the plan to be executed will be any plan believed by the responder to achieve the desired goal. Therefore, the initiator does not care about the specific plan that is used. Examples of possible types of plans are:

- *Ask around, just in case*. Being lazy, r could ask its acquaintances if the goal is already achieved. Notice that this does indeed delegate the strategy but not the commitment. If anyone among the acquaintances of agent r answers positively, then the goal has been achieved, even if r doesn't know how.
- *Do it yourself*. r could find out a feasible plan for the goal, which hasn't been executed yet, and then execute it. This will of course achieve the goal.
- *Who's going to keep my promises*? r can further delegate the goal (both strategy and commitment), using the goal delegation protocol recursively. By induction on the nesting level, if there is a finite number of nested delegations that complete successfully, the goal will be achieved.

Since *achieve* has been defined in terms of the *request* message, we will analyse the FIPA-Request protocol as the basis for the goal delegation protocol. The FIPA-Request protocol is started by the initiator sending the *request* message to the responder. When the responder receives the *request* message, it has three alternatives. It may send a *not-understood* message; it may send a *refuse* message; or it may send an *agree* message. If the responder sends the *agree* message, it becomes committed to try to execute the requested action. When executing the requested action, the

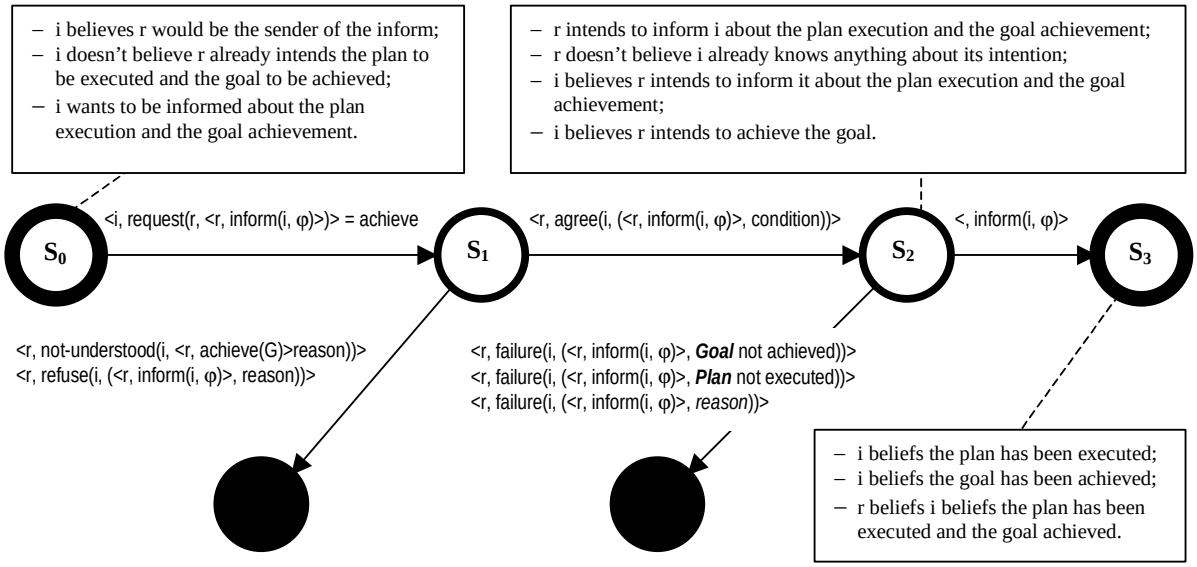

Figure 2 – The FIPA compliant goal delegation protocol

responder may send a *failure* message in case it fails to successfully execute the action; it may send an *inform-ref* message; and it may send an *inform-done* message. Given the above, in case of successful termination of the FIPA-Request protocol, the responder sends an *agree* message and then it sends an *inform-ref* or an *inform-done.*

Adapting the FIPA-Request for the goal delegation case, it would result in the protocol described in Figure 1.

Clearly, this protocol is not totally adequate for goal delegation. The first obvious inconvenience is that the *inform-done* in the last step of the successful protocol execution is not necessary because the responder would have already informed the initiator that the plan has been performed and the goal has been achieved. There is no point in informing the initiator that the requested *inform* message has already been sent. Less obvious is the content of the *failure* message in case something fails. There are three possible types of failure: (i) the responder may fail sending the inform message; (ii) the responder may fail to execute the plan; and (iii) the responder executed the plan but, due to unforeseen events or due to insufficient knowledge about the results of available actions, the plan failed to attain the desired result.

Considering the above three aspects we propose the following goal delegation protocol. Let G be the goal to be achieved, and let's define the proposition

$\varphi \equiv \exists ?plan\ (Feasible(?plan, G) \wedge Done(?plan)) \wedge G$

Notice that, although this looks like a higher order formula, it is not because, in each concrete case, G will be instantiated with a specific goal to be achieved. Therefore the formula is a proposition schema, not a higher order formula.

The protocol works as follows (see also Figure 2):

1. <i, request(r, <r, inform(i, φ)>)>
2. Action Alternatives
   (a) <r, not-understood(i, (<i, request(r, <r, inform(i, φ)>>, reason for not understanding))>
   (b) <r, refuse(i, (<r, inform(i, φ)>, Reason for refusing))>
   (c) <r, agree(i, (<r, inform(i, φ)>, Condition of action execution))>
3. [agreed] Action Alternatives
   (a) <r, failure(i, (<r, inform(i, φ)>, Reason for the failure of the inform))>
   (b) <r, failure(i, (<r, inform(i, φ)>, ***Plan*** was not completely executed))>
   (c) <r, failure(i, (<r, inform(i, φ)>, ***Goal*** has not been achieved))>
   (d) <r, inform(i, φ)>

Some details of the above specification are worth noting. The protocol specification is richer than AUML diagrams [2] currently used in the FIPA specifications, because it specifies parts of the contents of some of the involved messages. Symbols ***Plan*** and ***Goal*** appearing in messages 3(b) and 3(c) will be instantiated with concrete plan and goal expression, at the time the messages are actually sent. This specification should be part of the protocol description. The conversation identifiers in all of the possible messages must be the same. It is the responsibility of the protocol initiator to create that identifier. This specification should also be part of the formal protocol description. Finally, each set of alternative courses of action is available only at certain junctures, that is, in certain protocol states. For example, alternative actions 3(a) to 3(d) are available to the agent only if the agent has agreed to perform the requested action. It is necessary to explicitly and formally specify protocol state changes [8].

In the following subsections we present a framework for protocol analysis and we analyse the proposed goal delegation protocol.

3.2 Protocol Analysis

This section provides a framework that may be used to analyse interaction protocols with respect to the set of propositions that should be true in each protocol state. This proposal lays down the basis for a protocol verification system, which could be built in a Court Agent that could be developed in agent societies.

The main ideas behind our protocol analysis methodology are *compliance* and *intentional action*. We assume that when an agent sends a message (i) it does so intentionally, and (ii) it is desirable that it complies with the message semantics. It results from the above assumptions that, when a message is observed, the message feasibility preconditions should hold (because the sender should comply with the message semantics) and the sender intended the message rational effects (because it sent the message intentionally). For instance, when agent i receives message <r, inform(i, P)>, it may assume that

$B_i P \wedge \neg B_i(Bif_i P \vee Uif_i P)$ (*inform* feasibility preconditions)

and

$I_r B_i P$ (the agent intends the rational effects of the message).

Given the above reasons, and acknowledging the fact that protocol state changes reflect message sending/receiving, we may attach to each protocol state, a set of propositions that should be true from a normative point of view. The state that results of a state transition from state S due to message <i, M> is the union of state S with the feasibility preconditions of M and I(i, RE(M)), in which RE(M) is the set of rational effects of M, and $I(i, \Delta)=\{I_i(p): p \in \Delta\}$ represents the fact that the sender intends all the propositions in Δ.

$S_l=S_k \cup FP(i, j, M_{l.k}) \cup I(i, RE(i, j, M_{l.k}))$, in which S_l and S_k are protocol states, $M_{l.k}$ is the message that resulted in the protocol state transition from state S_l to state S_k, $RE(i, j, M_{l.k})$ is the set of Rational Effects of message $M_{l.k}$, indexed to the sender i and the receiver j, and $FP(i, j, M_{l.k})$ is the set of Feasibility Preconditions of message $M_{l.k}$ indexed to sender i and receiver j. All protocols have an initial empty state, the state before the initiating message is sent.

In the following sections, we analyse the case of successful execution of the goal delegation protocol, as defined in section 3.1, using the concept of protocol-state just presented.

3.3 Goal delegation analysis

3.3.1 Step 1: Protocol initiation

Before the protocol is initiated, the protocol is in the initial state (S_0), which is the empty set. The protocol initiator (agent i) sends message <i, request(r, <r, inform(i,φ)>)>, resulting in a protocol state transition to state S_1. According to the definitions presented in subsections 3.1 and 3.2, S_1 is composed by the *achieve* feasibility preconditions and the intention of its rational effects.

$S_1=\{B_i(Agent(r, <r, inform(i, \varphi)>), \neg B_i(I_r\ Done(<r, inform(i, \varphi)>)), I_i Done(<r, inform(i, \varphi)>)\}$

That is, the observer is entitled to conclude that (i) the initiator believes that the responder will be the agent of the inform message; (ii) the initiator does not believe that the responder already has the intention of having informed the initiator that the plan has been executed and the goal has been achieved; and (iii) the initiator wants the responder to inform it that the plan has been executed and the goal achieved.

3.3.2 Step 2: the responder agrees

In the second step, the responder agrees to inform the initiator that the plan has been executed and the goal has been achieved. This message results in a new state transition to state S_2. S_2 is the union of S_1 with the feasibility preconditions of the *agree* message and the intention of its rational effects. The feasibility preconditions and the rational effects of the *agree* message are those specified in [9].

$S_2=S_1 \cup \{B_r I_r\ Done(<r, inform(i, \varphi)>, \phi), \neg B_r(Bif_i\ I_r\ Done(<r, inform(i, \varphi)>, \phi) \vee Uif_i\ I_r\ Done(<r, inform(i, \varphi)>, \phi)), I_r B_i\ I_r\ Done(<r, inform(i, \varphi)>, \phi)\}$

in which ϕ is the condition under which the inform message will be sent.

The observer of the *agree* message is now entitled to have additional beliefs. The responder believes it has the intention to inform the initiator that the plan has been executed and the goal has been achieved. The responder does not believe the initiator already knows anything about its intention. The responder intends the initiator to believe it has the intention of informing it of the success of the goal delegation process.

In order to check the soundness of the designed protocol, it could be determined if each protocol state is consistent. S_2 is obviously consistent since the beliefs and intentions ascribed to each participant are not contradictory.

3.3.3 Step 3: success

In the third step, the responder agent informs the initiator that it has successfully executed the plan believed to achieve the delegated goal and the goal has been achieved. This message produces another protocol-state transition resulting in state S_3. Given the semantics of the inform message, as defined in [9], the new state will be defined as follows

$S_3=S_2 \cup \{B_r\varphi\ ,\ \neg B_r(Bif_i\varphi \vee Uif_i\varphi), I_r B_i\varphi\}$

in which

$\varphi \equiv \exists plan(Feasible(plan, G) \wedge Done(plan)) \wedge G$

$S_3 = \{B_i(Agent(r, <r, inform(i, \varphi)), \neg B_i(I_r\ Done(<r, inform(i, \varphi)>)), I_i Done(<r, inform(i, \varphi)>), B_r I_r\ Done(<r, inform(i, \varphi)>, \phi), \neg B_r\ (Bif_i\ I_r\ Done(<r, inform(i, \varphi)>, \phi) \vee Uif_i\ I_r\ Done(<r, inform(i, \varphi)>, \phi)), I_r B_i\ I_r\ Done(<r, inform(i, \varphi)>, \phi), B_r\varphi\ ,\ \neg B_r(Bif_i\varphi \vee Urf_i\varphi), I_r B_i\varphi\}$

Among other things, the observer of this state will know that the responder believes there is a plan that results in the delegated goal becoming achieved; it also believes that plan has been executed; and it also believes the goal to have been achieved. By virtue of being the receiver of the message that caused this last state transition, the protocol initiator is an observer of the last protocol state (S_3). Therefore, the initiator concludes the responder believes to have achieved the desired goal. That is, in case of successful termination, the goal delegation protocol fulfils the purpose of its design.

Using a similar analysis, it could easily be shown that the protocol also works appropriately in the other termination conditions. From the point of view of protocol soundness, it can also be seen that S_3 does not contain contradictions. This is a good criterion to assume the protocol to be well formed.

As can be seen, the last state of the protocol clearly shows that it is legitimate to assume that the initiator knows the plan has already been executed and the goal has been achieved. Therefore, as previously argued (see section 3.1), the inform-done message that would be generally necessary in the request protocol is not needed in the goal delegation protocol.

3.4 Alternative design

As argued in section 3.1, the proposed definition of the *achieve* performative does not fulfil all requirements for goal delegation. Specifically, it does not follow from the semantics of the performative that the protocol initiator does not believe the responder to already have the intention to achieve the desired goal. The proposed definition can only ensure that the responder agent (the delegate) does not already intend to inform the initiator that the goal has been achieved. Although this is not a very important drawback, it would be desirable if it could e fixed.

The referred problem arises because SL, the language used to express the semantics of the performative, is not rich enough to overcome that difficulty. This subsection proposes to extend SL with a new action operator that enables overcoming the mentioned problem. The new operator, execute, has also been proposed in [4].

Execute is a general-purpose action operator used to express the action of executing a given action description passed as an argument. Using execute, the protocol initiator can ask the responder to execute any plan that achieves the desired goal, instead of asking the responder to inform it that the plan has been executed. Using this design, all goal delegation requirements will be met, and the goal delegation protocol will more closely mirror the request protocol.

We start analysing the way of expressing the action of executing a plan that achieves the goal. Feasible(p, G) means that p can be executed and achieves G. Any(p, Feasible(p, G)) refers a plan (anyone) that can achieve G. Execute(Any(p, Feasible(p, G))) is the action of executing the plan referred by Any(p, Feasible(p, G)), that is a plan that achieves the desired goal.

Given the above elements, the achieve performative could have the alternative definition

$$\langle i, achieve(r, G)\rangle \equiv \langle i, request(r, \langle r, execute(any(p, Feasible(p, G)))\rangle\rangle$$

that is characterized by:

FP of <i, achieve(r, G)>

- FP execute(any(p, Feasible(p, G))) [i\r]. The subset of the feasibility preconditions of <r, execute(any(p, Feasible(p, G)))> that are mental attitudes of i;
- B_i(Agent(r, <r, execute(any(p, Feasible(p, G)))>)). The sender believes the receiver to be the agent of the action of executing the plan;
- $\neg B_i(I_r$ Done(<r, execute(any(p, Feasible(p, G)))>)). The sender does not believe that the receiver already intends execute a plan that achieves the goal.

RE of <i, achieve(r, G)>

- Done(<r, execute(any(p, Feasible(p, G)))>). The sender i can reasonably expect that a plan that achieves the goal will be done.

This alternative definition fulfils all the goal delegation requirements presented in section 2:

- The initiator believes that the responder is skilled enough to achieve the goal;
- The initiator believes the responder does not already intend to achieve the goal;
- The initiator does not care about the plan to be used to achieve the goal.

The first requirement can be shown to be implied by the *achieve* feasibility preconditions, because the initiator can only send the *achieve* message if it believes the responder to be the agent of the action of executing the plan believed to achieve the goal. Therefore it must believe the responder can do it.

The second requirement is exactly the second feasibility precondition of the *achieve* performative.

The third requirement is captured by the action the initiator is requesting the responder to perform: any plan that is believed to achieve the goal.

This alternative definition has a consequence that must be handled. The initiator does not ask the responder to inform it that the plan has been executed and the goal has been achieved. This will be handled at the protocol level, not at the performative level. The new protocol definition is defined below, in which $\psi \equiv$ any(p, Feasible(p, G)):

1. <i, request(r, <r, execute(ψ)>)>
2. Action Alternatives
   (a) <r, not-understood(i, (<i, request(r, <r, execute(ψ)>>, Reason for not understanding))>
   (b) <r, refuse(i, (<r, execute(ψ)>, Reason for refusing))>
   (c) <r, agree(i, (<r, execute(ψ)>, Condition of action execution))>
3. [agreed] Action Alternatives
   (a) <r, failure(i, (<r, execute(ψ)>, ***Plan*** was not completely executed))>
   (b) <r, failure(i, (<r, execute(ψ)>, ***Goal*** has not been achieved))>
   (c) <r, inform(i, Done(ψ))>

The new protocol design is simpler because it has less alternatives in step 3. Besides, it is more closely related to the request protocol. This protocol specifies two cases of failure messages.

Although this alternative definition of the goal delegation protocol is better than the one proposed in section 3.1, it relies upon an extension of the SL language. Therefore, in the case study described in the next section we assume the initial definition.

4. CASE STUDY: AGENTCITIES EVENT ORGANIZER SEVICE

The Agentcities event organizer fulfils service compositions using the services, provided by the Agentcities network, needed to set up a social event. It shows that agents offer dynamic and flexible solutions for supply chains, especially to deal with unexpected events and chain reorganization. In the reference scenario, a conference chair attempts to develop a schedule for her conference and to book the venues and services that she requires, e.g., hotel, restaurant and amusement events. She delegates to the event organizer the work, monitoring the progress of arrangements. The event organizer service is available in the Parma Agentcities node [1].

The main actors involved in the event organizer are:

- the user, i.e., the conference chair;

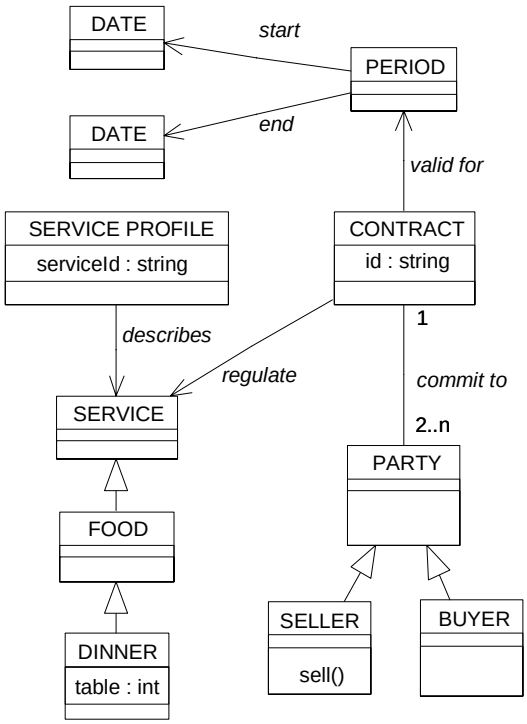

Figure 4 – Event Organizer Ontology

- the event organizer agent, i.e., the agent that tries to achieve the global goal that the user submitted;
- the solvers, i.e., the skilled agents that search the needed services and negotiate the contracts for buying them with the service provider agents;
- the service provider agents.

The Figure 3 shows some of the ontology classes used in the agent interactions described in the following subsections. These classes are part of the complete Event Organiser ontology.

The process starts when the chair decides to organize the conference and requests the event organizer agent to set up a set of needed services, fixing some constraints and a priority for each service. It finishes when all mandatory services are bought or reserved. These interactions are governed thanks to the FIPA-Request protocol for the goal delegation case proposed in section 3.1, where the event organizer plays the role of the initiator and the solver plays the one of the responder. Due to some limitations in the FIPA-ACL semantics, some interaction rules are implicitly defined in the agent code, e.g., the deadlines that the solver has to respect for the plan execution.

The following step can be iterated until the conference is fully organized.

4.1 Conversation 1: Goal Delegation

This conversation is carried out between the event organizer and the solver.

4.1.1 Protocol initiation

The chair fixes through a Web page the finite set of services that she wants to buy for the conference and a finite set of associated constraints. These parameters are translated in a global goal assigned to the event organizer, e.g., “make it so that all the 20 attendees have a dinner together and rooms booked for five nights in nearby hotels”.

For the sake of simplicity we assume that the idea of “constraints” or “service priority” will not be exchanged among the agents. Only the event organizer agent knows about the full set of required constraints and the priorities of the services. This eases the problem solving process because the event organizer agent centralizes the validation of constraints without delegating it to solvers.

Then, the event organizer decomposes its given global goal into sub-goals, each of which is proposed, with the following performative, to one particular problem solving agent (so-called solver), based on its functional capabilities to achieve the goal assigned.

```
(request
  :sender   (agent-identifier : name EventOrganizerAgent)
  :receiver (agent-identifier : name RestaurantSolverAgent)
  :content
    (inform
      :sender   (agent-identifier : name RestaurantSolverAgent)
      :receiver (agent-identifier : name EventOrganizerAgent)
      :content  (exists ?plan
                   (and (and (feasible (?plan γ))(done ?plan)) γ))
      :language fipa-sl
      :protocol goal-delegation
      :ontology Conference_Organizer_Ontology1.0
      :conversation-id goal02
    )
  :language fipa-sl
  :protocol goal-delegation
  :ontology Conference_Organizer_Ontology1.0
  :conversation-id goal01
)
where
γ =(exists ?cID
    (and
     (regulate (Contract: id ?cID) (Dinner :table 10))
     (commit-to(Contract: id ?cID) (Buyer  :AID EventOrganizerAgent))))
```

The solvers are either newly created by the event organizer as instances of functional agent classes or have been spawned in the past and therefore already exist. In our scenario a sub-goal corresponds to the search of suitable contracts for the services asked by the chair, without considering the cross-services constraints, e.g., the solver searching for restaurants does not consider that the restaurant cannot be too far away from the hotel, only the event organizer agent deals with such a constraint.

4.1.2 The solver agrees

The solver agrees to achieve the assigned sub-goals and builds a plan.

4.1.3 The solver executes the plan

Each solver uses the search infrastructure services offered by the Agentcities network architecture to find suitable service providers.

The solver chooses the providers that fit its tasks best. This can be done through a direct interaction or through a market place. Once a suitable service provider is found, the solver negotiates with it to reach a preliminary agreement for a contract that regulates the requested service.

4.1.4 The solver informs the event organizer about the contract

The solver informs the event organizer the sub-goal is achieved and it knows about some contracts.

4.2 Converstion 2: Contract Retrieval

This conversation is carried out between the event organizer and the solver.

The event organizer believes that the solver has negotiated at least one contract to purchase the assigned service. It starts a FIPA-Query protocol, where it plays the role of initiator and the solver plays the role of responder, to get such a contract. The solver gives its best proposal back to the event organizer for a subsequent use.

```
(query-ref
  :sender   (agent-identifier : name EventOrganizerAgent)
  :receiver (agent-identifier : name RestaurantSolverAgent)
  :content
    (any ?cID
      (and
        (regulate (Contract: id ?cID) (Dinner :table 10))
        (commit-to(Contract: id ?cID) (Buyer  :AID EventOrganizerAgent)))
  :language fipa-sl
  :protocol fipa-query
  :ontology Conference_Organizer_Ontology1.0
  :conversation-id getcontract01
)

(inform
  :sender   (agent-identifier : name RestaurantSolverAgent)
  :receiver (agent-identifier : name EventOrganizerAgent)
  :content  (=
              (any ?cID
                (and
                  (regulate (Contract: id ?cID) (Dinner :table 10))
                  (commit-to(Contract: id ?cID) (Buyer  :AID
                   EventOrganizerAgent))
                )
              cID001)
  :language fipa-sl
  :protocol fipa-query
  :ontology Conference_Organizer_Ontology1.0
  :conversation-id getcontract02
)
```

4.3 Conversation 3: Services Acceptance

This conversation is carried out between the event organizer and the chair.

Once each instance of the protocol with the solvers ended, the event organizer agent has enough information to build the global plan satisfying the chair's requirements. To do so, it first composes the proposals received from the solvers and validates the cross-service constraints. If a consistent solution is found, it is proposed to the chair for a final acceptance.

Now, the event organizer agent informs the chair about the contracts she has to sign for achieving the global goal. If no consistent solution is found, the event organizer agent iterates the previous steps until an acceptable solution is found or until the chair decides to change some constraint.

The iteration consists of assigning new sub-goals to the solvers exploiting the knowledge about which cross-service constraints has not been satisfied. For example, if the process failed because the restaurant and the hotel were too far from each other, the new sub-goal will be "operate so that the attendees have dinner in a restaurant within 1 Kilometre from the hotel and give me back a new suitable contract for that".

4.4 Conversation 4: Services Purchase

This conversation is carried out between the event organizer and the service provider agents.

Once the chair accepted the proposed solution, the event organizer agent starts a FIPA-Request protocol with the service provider agents in order to buy the service directly from them.

```
(request
  :sender   (agent-identifier : name EventOrganizerAgent)
  :receiver (agent-identifier : name ServiceProviderAgent)
  :content  (sell cID001(Buyer :AID EventOrganizerAgent))
  :language fipa-sl
  :protocol fipa-request
  :ontology Conference_Organizer_Ontology1.0
  :conversation-id getcontract01
)
```

5. CONCLUSION

In this paper we proposed a FIPA compliant protocol to perform goal delegation between two agents. The motivation of this work starts from a real need, i.e. to build an application for the Agentcities.RTD project where agents delegated to other skilled agents their goals We approached the problem with the idea to only use what FIPA provides.

We proposed a framework for protocol analysis and we used it to validate our goal delegation protocol that uses the FIPA ACL semantics as it is. We argued that the protocol still have a minor drawback and we proposed a new SL operator *execute* that allows to fulfil all the requirements for goal delegation pointed out in the first part of the paper.

Finally we described the concrete application that was realized thanks to the effort of this work.

6. ACKNOWLEDGMENTS

The research described in this paper is partly supported by the EC project Agentcities.RTD, reference IST-2000-28385. The opinions expressed in this paper are those of the authors and are not necessarily those of the Agentcities.RTD partners.

7. REFERENCES

[1] Agentcities.RTD, reference IST-2000-28385, http://www.agentcities.net

[2] Bauer B., Müller J.P., Odell J.R.E, Agent UML: A Formalism for Specifying Multiagent Interaction, In Paolo Ciancarini and Michael Wooldridge (eds) Agent-Oriented Software Engineering (Berlin 2001), Springer, 91-103.

[3] Bergenti F., Burg B., Caire G., Poggi A. Deploying FIPA-compliant systems on handheld devices. IEEE Internet Computing, Volume 5 Issue 4 (July-Aug. 2001), 20-25.

[4] Botelho L. M., Antunes N., Ebrahim M., Ramos P. Greeks and Trojans Together, Submitted paper, 2002.

[5] Castelfranchi C., Falcone R. Socio-Cognitive Theory of Trust. http://alfebiite.ee.ic.ac.uk/docs/papers/D1/ab-d1-cas+fal-soccog.pdf.

[6] Colombetti M. A Commitment-Based Approach to Agent Speech Acts and Conversations. In Proc. Workshop on Agent Languages and Communication Policies, 4th International Conference on Autonomous Agents (Barcelona 2000), 21-29.

[7] Castelfranchi C., Pedone R. A Review on Trust in Information Technology. http://alfebiite.ee.ic.ac.uk/docs/papers/D1/ab-d1-cas+ped-trust.pdf.

[8] Freire J., Botelho L. M. Executing explicitly represented protocols. Submitted paper, 2002.

[9] FIPA spec. XC00037H. FIPA Communicative Act Library Specification. http://www.fipa.org/specs/fipa00037/.

[10] FIPA TC Semantics Call for Information, http://www.fipa.org/docs/output/f-out-00099/ f-out-00099.pdf.

[11] Finin T., Labrou Y. KQML as an agent communication language. In J.M. Bradshaw (ed.), Software Agents, MIT Press, (Cambridge, MA, 1997), 291-316.

[12] Giunchiglia F., Mylopoulos J., Perini A.. The Tropos Development Methodology: Processes, Models and Diagrams. Submitted at AAMAS 2002.

[13] Pitt J., Kamara L., Artikis A.. Interaction Patterns and Observable Commitments in a Multi-Agent Trading Scenario. http://alfebiite.ee.ic.ac.uk/docs/papers/D1/ab-d1-pitkamart-ipoc.pdf.

[14] Singh M. P. Agent Communication languages: Rethinking the principles. IEEE Computer, 31 (12) (1998), 40-47.