Computing and Informatics, Vol. 33, 2014, 1288–1311

SUPPORTING DYNAMICITY IN EMERGENCY RESPONSE APPLICATIONS

Ricard Fogues

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València, Valencia, Spain
e-mail: rilopez@dsic.upv.es

Jose M. Such

School of Computing and Communications
Lancaster University, Lancaster, UK
e-mail: j.such@lancaster.ac.uk

Juan M. Alberola, Agustin Espinosa, Ana Garcia-Fornes

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València, Valencia, Spain
e-mail: {jalberola, aespinos, agarcia}@dsic.upv.es

Abstract. Multiagent Systems are a promising paradigm for software development. It is feasible to model such systems with many components where each one can solve a specific problem. This division of responsibilities allows multiagent systems to work in dynamically changing environments. An example of an environment that is very changeable is related with emergencies management. Emergency management systems depend on the cooperation of all their components due to their specialization. In order to obtain this cooperation, the components need to interact with each other and adapt their interactions depending on their purpose and the system components they are interacting with. Also, new components may arrive on the scene, which must be informed about the interaction policies that original components are using. Although Multiagent Systems are suited to managing scenarios of this kind, their effectiveness depends on their capacity to dynamically modify and

adapt the protocols that control the interactions among agents in the system. In this paper, an infrastructure to support dynamically changing interaction protocols is presented.

Keywords: Dynamic agent interactions, interaction protocols, emergency management systems, multiagent systems

1 INTRODUCTION

As stated in [1], Multiagent Systems (MAS) allow the development of complex computational systems in changing environments. Agent-based technologies are involved in a wide range of domains, specially those that are identified as open and dynamic systems which agents can act upon on behalf of service owners, locating services, negotiating contracts and making pro-active runtime decisions while responding to changing circumstances. This view of systems requires the development and integration of infrastructures in which agents with different capabilities are able to collaborate. Each agent is specialized in one task, and none of the agents can accomplish all of the system's objectives on their own. Therefore, agents must collaborate and work together to accomplish these objectives.

Open systems must be able to dynamically adapt their structure and behaviour by means of adding, removing or substituting components of the system while it is running and without bringing it down [2, 3]. In a MAS, this means that new agents can enter the system and replace previous ones while the system is running, thereby improving the behaviour of the MAS. These new agents need to know how to communicate with the rest of the agents of the system in order to operate properly. The way that agents interact with each other is another component of the system, which can also be changed and improved. This flexibility requires adaptive technologies.

A common way for agents to manage interactions with each other is through the use of Interaction Protocols (IPs). IPs allow the specification of the agents' behaviour within a closed environment in which allowed interactions are predefined. As defined by the Foundation for Intelligent Physical Agents (FIPA)[1], IPs specify pre-agreed message exchange protocols. IPs define the set of rules that govern the interactions between participants [4]. However, in dynamic scenarios such as open systems, pre-specified IPs may not be enough.

In our previous work described in [5], a support software for managing interactions between agents in terms of dynamic IPs is proposed. In that work, we proposed an agent architecture that is oriented to conversations that follow IPs that can be dynamically modified, according to changes in both the environment and the system. This agent architecture has been integrated into the Magentix2 Agent

[1] `http://www.fipa.org/`

Platform (AP) [6]. In this paper, the proposal is extended so that IPs can be specified and exchanged among agents in order to agree upon the IPs to be used in the conversation as well as in order to communicate the modification of the IPs.

Dynamic IPs provide a high level of adaptability to the interactions between the agents of the system. Nevertheless, there is no way to coordinate IP modifications in the MAS. Thus, the inclusion of a mechanism to exchange IPs among agents became necessary. In dynamic systems, agents could suggest new (or modified) IPs to other agents when required by changes in the environment. Therefore, it is necessary for the agents to have a method to transmit an IP to another agent. Moreover, when an agent enters such a system, it may need to know which IPs are in use in the system in order to be able to communicate with the rest of the agents in the system. One way to learn these IPs could be to ask another agent for them. Developers also need a language that allows them to define IPs in a unified way that is easy for human beings to understand. In order to achieve all these aims, our work proposes a specification of IPs using the cpXML conversation description language and a mechanism to communicate and adopt new IPs.

In order to validate our proposal, we implement an emergency response system. As stated in [7], emergency response systems deal with very changeable environments and depend on the cooperation of all their components due to their specialization. To obtain this cooperation, the components need to interact with each other and adapt their interactions depending on their purpose and the system components they are interacting with. Additionally, new components may arrive on the scene, which must be informed about the interaction policies that the original components are using. Although multi-agent systems are suited to managing scenarios of this kind, their effectiveness depends on their capacity to dynamically modify and adapt the protocols that control the interactions among agents in the system.

The rest of the paper is organized as follows. Section 2 shows the agent architecture, which is oriented to conversations. Section 3 shows how the cpXML language can be used to communicate an interaction protocol to an agent. Section 4 presents an emergency management system that uses dynamic interaction protocols, and Section 5 explains how this system was implemented. Section 6 details previous works related to our proposal. Finally, Section 7 presents some concluding remarks.

2 CONVERSATIONAL AGENT ARCHITECTURE

Interactions among agents are usually represented by means of IPs. FIPA [8] defines IPs as patterns that ongoing interactions among agents often follow. In these cases, certain message sequences are expected, and, at any point in the interaction, other messages are expected to follow. According to the FIPA specifications, an IP is represented by two agent roles that define different behaviours: the *initiator* role corresponds to the agent that initiates the IP; the *participant* role corresponds to the agent (or agents) that participates in the IP. The sequence of messages regarding an IP is exchanged among agents that play these roles.

MAS are executed on an AP, which provides support for the development and execution of MAS. There are many APs developed by the agent community that provide different features and support [9, 10]. APs provide all the basic infrastructure required to create a MAS [11] and some of them bring support for IPs [15, 12, 14, 13]. Nevertheless, these APs only consider predefined IPs. Thus, IPs can neither be changed at execution time nor communicated to other agents.

In [5], we propose an agent architecture for implementing dynamic IPs that is modelled as finite state machines (FSM). In this work, each interaction between agents is called a conversation. The term "conversation" expresses every possible sequence and combination of messages that can be passed between two or more agents participating in a given agent system [16]. The agent architecture proposed in this work has been integrated in the Magentix2 AP. This architecture is composed by two main components: Conversation Factories (*CFactories*) and Conversation Processors (*CProcessors*). A *CFactory* represents an IP, and a *CProcessor* represents an instance of a *CFactory*. *CFactories* allow agents to maintain several conversations following a specific IP or different IPs simultaneously with other agents. Each of these conversations is managed by a different *CProcessor*.

A *CFactory* defines an IP as a directed graph composed by nodes and arcs between nodes. Each node represents a state during the IP, and the arcs represent all possible transitions from one state to other states. A *CFactory* is in charge of creating *CProcessors* that follow the IP defined by the *CFactory* each time that the agent starts a new conversation that requires this IP. Each *CFactory* manages the incoming messages that follow the specific IP defined. If the message received belongs to an ongoing conversation associated to a *CProcessor*, the message is delivered to this *CProcessor*; otherwise, a new *CProcessor* is created to manage this conversation. Both *CProcessors* and *CFactories* can be modified dynamically. This allows the way that an agent interacts with other agents or the way an agent carries out an ongoing conversation to be changed.

A *CFactory* uses states and transitions to define the behaviour of each role of the IP. There are five types of states that can define an IP: *Begin*, *Final*, *Send*, *Receive*, and *Wait*. A transition between two states occurs depending on the type of the state where the conversation is at a given moment. Following, there is a list of the different states of the conversation according to the actions that trigger a transition from each state:

- *Begin*: A transition is triggered from this state when the conversation starts.
- *Receive*: A transition is triggered from this state when a message is received and managed.
- *Send*: A transition is triggered from this state when a message is sent by the agent.
- *Wait*: A transition is triggered from this state when a message is received. The difference between the transition from a *Wait* state and from a *Receive* state is that the Wait state waits for any message, and depending on the message, the

conversation travels to a specific *Receive* state. Then the *Receive* state processes the message and the conversation travels to the next state.

3 DESCRIBING AND EXCHANGING INTERACTION PROTOCOLS

In order for agents to be able to communicate the protocols that they are using in their conversations, we must specify IPs by using a conversation description language. In this work, we used cpXML [17] as the language for specifying IPs.

CpXML was created by IBM in 2002. It is an XML dialect for describing conversation policies (*CP*). A CP is a set of constraints on the schemas and sequencing of messages that may be sent in a conversation between two or more applications. A CP is a purely "passive" structure that describes a set of possible message sequences, without reference to which sequence is actually followed in any given conversation. In practice, the conversing applications need not follow the CP's constraints. That is, CPs are non-normative in the sense that no application is required to follow any given CP (or any CP at all, for that matter). CPs are intended to be used as common, conventional, standard protocols that are available for use by applications as an aid to interacting effectively. A single CP covers messages from all participants, which is written in a way that is independent of any particular participant (i.e., a CP is written from the point of view of a third party overhearing the conversation).

Other languages such as IOM/T [18] also allow the specification of IPs. IOM/T aims at a tight correspondence with AUML diagrams used in interaction design. The use of this language allows users to implement an interaction protocol in a unique place, without dispersing the codes of the interaction between the agents participating in it. In other words, the interaction protocol is only described in a single structure rather than being described twice (once for the initiator role and once for the participant role).

CpXML has been chosen for specifying the IPs because it allows a specification that is similar to FSM and the agent architecture proposed also models IPs as FSMs. Therefore, the algorithm to convert a cpXML specification into a *CFactory* is very intuitive. Another feature that has been considered choosing cpXML is that it is a XML dialect, therefore it is easy to parse and use in conjunction with Java.

3.1 Describing IPs with CPXML

CpXML specifications of IPs can be exchanged between agents in order to agree upon the IP they are going to use in the conversation and to communicate modifications in the IP. A cpXML document can be described as the script of a play. The script determines the set of characters to be played, the things each character has to say, and the order in which these things must be said. However, the script does not say anything about the actors. Thus, the interpretation that actors give to their characters, where or how the play is put on, what technology (if any) is used to reach the audience, etc. will be specific for each instance of the play.

A cpXML document defines the roles that identify the participants. The main content of the document is a set of states connected by transitions. One of the states is the initial one, and a subset of the states are final states. There are three types of transitions. Each one is triggered by a different event:

1. sending a message,
2. timeout, and
3. starting a subconversation.

This last type is a valuable feature since it allows the creation of complex CPs composed of several subconversations. Figure 1 shows the *FIPA Request* [19] definition. In this figure, the IP is represented with Agent UML, which is the common representation of IP used by FIPA. Figure 2 shows the same IP described with cpXML. As the figure shows, in cpXML, the conversation is represented as a whole. It is not represented from two different points of view, but rather from a unique third party's point of view. In the figure, "I" stands for Initiator and "P" for participant. "I $\rightarrow$ P" means that the Initiator sends a message to the Participant, and "P $\rightarrow$ I" means that the Participant sends a message to the Initiator. Each node in the graph represents a state in the conversation; for example, the first state "No request" represents the moment before the Initiator sends a message requesting something from the Participant. The conversation travels from one state to another when a message is sent; for example, in the first state of the conversation, the state would change whenever the Initiator sends the request message to the Participant.

There is a major difference between a cpXML specification and a *CFactory*. This difference is that in cpXML, an IP description covers messages from all the agents that take part in the conversation. On the contrary, two *CFactories* are needed to describe an IP (one for the *initiator* role and other for the *participant* role). Nevertheless, *CFactories* and cpXML are very similar since both can represent the IP as FSM. Therefore, it is easy to derive a *CFactory* from a cpXML. The algorithm for transforming a cpXML specification into a *CFactory* basically performs the following two actions:

- For each state of the cpXML specification that has associated transitions of sending messages, the algorithm creates a *Send* state if the agent plays the *initiator* role in the conversation. If the agent plays the *participant* role, the algorithm creates a *Wait* state and a *Receive* state and adds a transition between the two states. The *timeout* associated to the *Wait* state is the value that is specified in the timeout transition of the state in the cpXML specification. If there is not specified a timeout transition, then the *Wait* state associates an infinite timeout. Finally, each terminal state of the cpXML specification causes the creation of a *Final* state.
- Once all the states have been created, the algorithm adds transitions between them according to the transitions specified in the cpXML document.

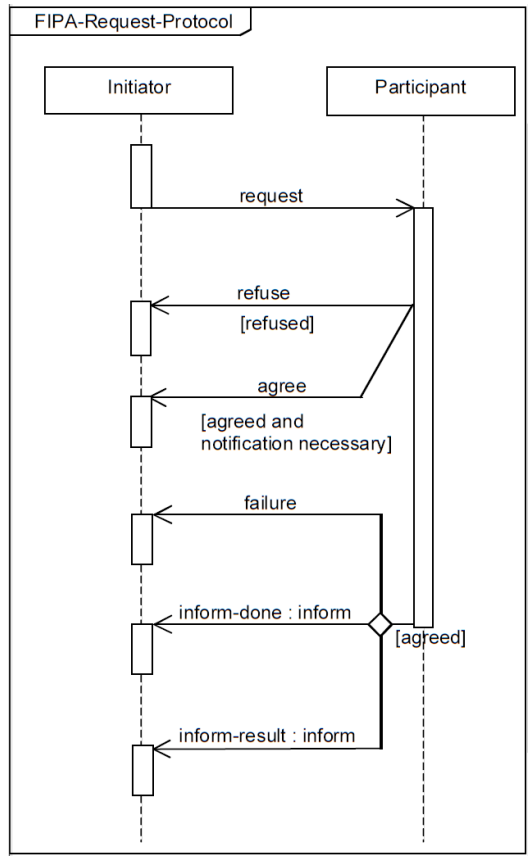

Figure 1. FIPA Request Interaction Protocol described in AUML

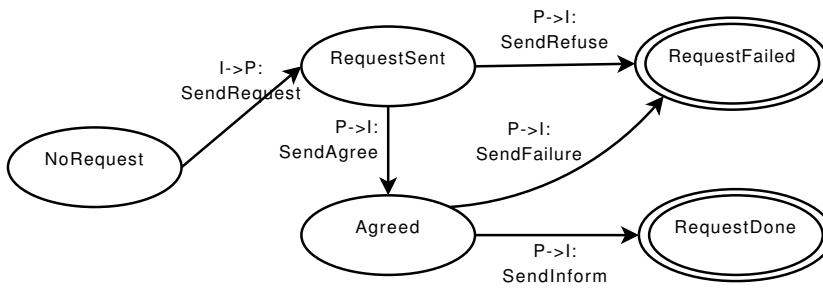

Figure 2. FIPA Request Interaction Protocol described with cpXML

Figure 3 shows the *FIPA Request* IP as a *CFactory* for the initiator role[2]. This *CFactory* has been constructed from the cpXML specification using the algorithm described above.

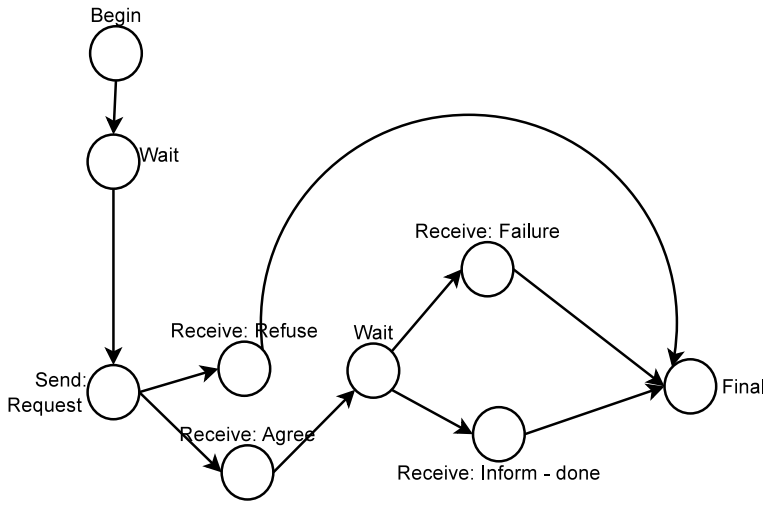

Figure 3. FIPA Request Interaction Protocol as a *CFactory*

3.2 IP Exchange Using cpXML

Agents can exchange IPs by using messages. There are two possible scenarios for IP exchange. On the one hand, an agent may need to communicate with another agent, but it does not know how to do it. Therefore, the first agent has to ask the second one to tell it a valid IP in order to have an interaction. On the other hand, an agent may propose another agent an IP for their conversation.

When agent *Alice* wants to communicate with another agent *Bob* using a specific IP, agent *Alice* sends a *propose* message to *Bob*. The message contains the cpXML specification of the IP that agent *Alice* wants to use in the conversation as well as the role that agent *Bob* will play in this conversation. Then, agent *Bob* can transform the cpXML specification into a *CFactory* and send an *accept* message to agent *Alice*, or it can decline the proposal and answer with a *refuse* message. Figure 4 shows how this *IP Proposal* method works. Agent *Bob* just entered the system, and agent *Alice* wants to communicate with the new agent. Therefore, agent *Alice* sends a *propose* message containing the cpXML specification of an IP. Agent *Bob* accepts the proposal, converts the cpXML specification into a *CFactory*

[2] The *CFactory* for the participant role is not included in the figure for the sake of clarity

and answers back to agent *Alice* with an *accept* message. From this moment on, these agents can start new conversations following the proposed IP.

If an agent needs to communicate with another agent but it does not know what IP to use, instead of proposing an IP to the other agent, it can ask for an IP description. In this case, agent *Bob*, who is willing to communicate, sends a *request* message soliciting the IP to agent *Alice*. Then, agent *Alice* can answer with an *inform* message containing the IP or with a *refuse* message; this means that agent Alice does not want to interact with agent *Bob*. This method of *IP Request* is shown in Figure 5. Agent *Bob* wants to communicate with agent *Alice* but agent *Bob* does not know which IP it can use. Hence, agent *Bob* requests a valid IP to agent *Alice*. When agent *Bob* receives the valid IP, *Bob* converts it into a *CFactory*, and, from this moment on, agent *Bob* can communicate with agent *Alice* using that IP.

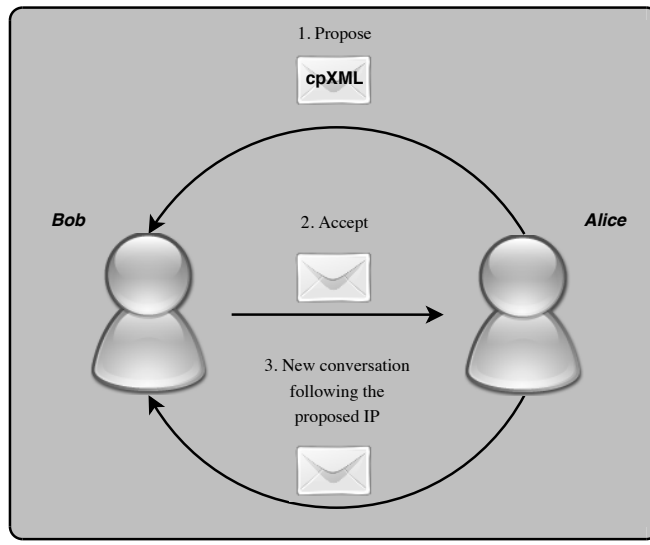

Figure 4. *IP Proposal* method for IP exchange

4 AGENT INTERACTION IN EMERGENCY SCENARIOS

In this section, we explain how an emergency management scenario can be developed using the proposed support for IPs. In this scenario, different elements with unique capabilities need to cooperate with each other in order to achieve their objectives. These elements also have to be capable of coordinating themselves quickly due to the emergency context they have to work in. In cases like riots, fire, or traffic accidents, a police patrol may need to call on the services of a number of specialists who can provide crucial assistance. In real life, it is more likely for the coordination of these elements to be carried out by a central coordinating element; this element

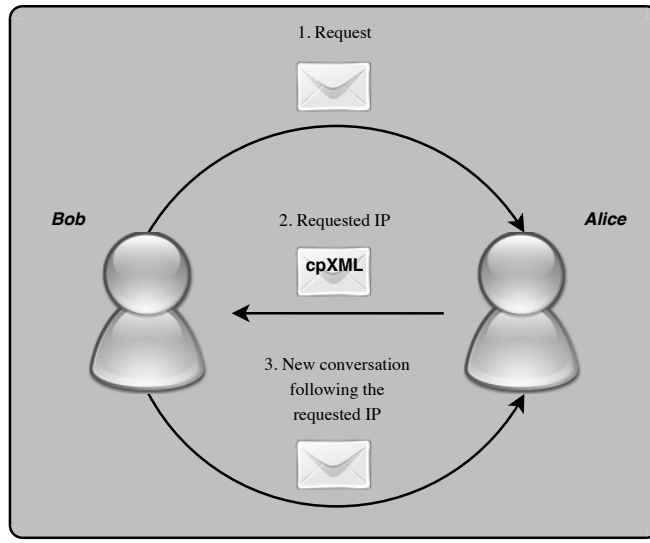

Figure 5. *IP Request* method for IP exchange

proxies the messages calling for a service that comes from the requester to the specialists. Specialists who offer their services are not always in the same geographic location or may not be able to attend to all emergencies at all times. It is more likely for them to dynamically change their availability and geographic location. Therefore, enviromental emergency managment systems must attempt to manage these dynamic situations. As stated in [7], a primary challenge in responding to both natural and man-made disasters is communication. Therefore, this scenario is the perfect situation in which to test our interaction support software. Since emergency scenarios are complex systems where unforseen events may change the conditions of the system, the elements that make up the system have to adapt to these changes. Communication as a main component has to be prepared to evolve and adapt to any modification in the system. Meissner et al. [20] state that flexibility is necessary for emergency management systems to work properly. They must be designed for fast adaptation to modifications of the organizational structure due to situation changes. Meissner defines three kinds of actors in the system with regard to their degree of mobility:

- Stationary actors: Government organizations like police headquarters.
- Semi-mobile actors: mobile command posts.
- Mobile actors: frontline personnel like firefighters or paramedics.

The scenario in this paper includes actors with different degrees of mobility. Therefore, it must deal with this dynamism at the communication level by offering flexible and adaptative IPs.

This paper analyzes a situation in which a police patrol is patrolling a city and the roads around it. There are many situations that the police patrol can manage by itself, such as robbery or small disturbances. However, there are other situations that a single police patrol is not able to control. If there is a car accident, the police patrol cannot attend to the injuried people. This requires specialized medical personnel, firefighting professionals, and equipment. As new specialist teams become available to attend to an emergency, they are added to the set of potential participants. From an open multiagent point of view, any former agent of the system may request a service offered by a new agent, as long as this agent is provided with the appropriate interaction protocols.

Each actor participating in the scenario has a mobile device with a software agent that acts as his/her representative. Every agent knows the geographic position, availability, and other important data of the personnel it represents. The communication between the people who participate in the scenario is always carried out by their agents. For example, if a firefighter requires assistance, he will specify to his agent what kind of assistance is needed and the agent will do the rest (i.e. check availability of the assistant personnel, communicate with the person in charge, etc.). The agents help by keeping actors' communication easy and precise. The agent abstraction also allows us to model an emergency management scenario as a MAS.

Let us suppose a scenario where a police patrol has witnessed a serious traffic accident in which some people have been badly injured and a fire has started. The police patrol has an idea of the kind of assistance that is needed, mainly medical assistance and the fire department. The police patrol contacts police central using the agent in their mobile device to request assistance. In this interaction, the police patrol plays the initiatior role and police central the participant one. The initiator communicates with police central in order to know which services are available and to call on them. Figure 6 shows a global view of the system. The figure shows how the police patrol sends proxied messages to police central, and how its agent central locates specialist agents that are capable of attending to the request. Once they are found, the police central agent informs the police patrol and tells them that their request can be attended to and starts to communicate with the located specialist agents. Depending on the request that the police patrol has made, the police central agent will use a specific interaction protocol with the specialist agents. In summary, the police central agent acts as an intermediary that recruits specialist agents depending on the needs of the police patrol that has assessed the situation.

There are three main components in this scenario:

1. police patrol, which acts as initiator;
2. police central, which acts as participant, and
3. specialist agents, which act as target agents.

The interaction shown in Figure 6 corresponds to the *FIPA Recruiting* IP specification [21]. The messages sent to the participant agent by the initiator will be

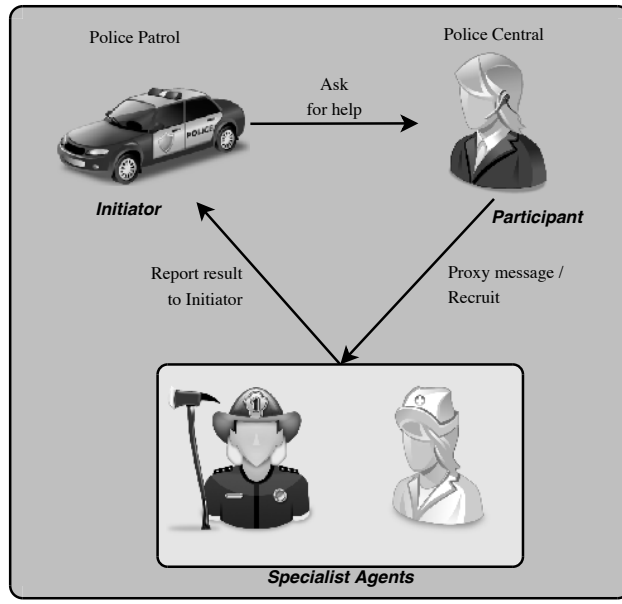

Figure 6. Global vision of emergy management system

proxied to the target agents. Then the target agents will report the action result to the initiator. The proxy communication is composed of subprotocols between the participant and the target agents. This complicated scenario with so many elements involved is a good context to demonstrate how agent-based systems that implement their interaction with the proposed software can respond effectively to changing conditions.

Figures 7 and 8 show the interaction protocol used by the police patrol and police central. First, the police patrol agent sends a message to the participant. This message specifies which service is needed and other parametres, such as the number of agents or what type of agent is needed. Then the participant answers the police patrol with an *agree* message if it wants to act as intermediary; otherwise, it sends a *refuse* message. If it agrees, then it will check for specialist agents that fit the specifications of the police patrol's message. If police central locates one or more agents it will start a subprotocol with each located specialist agent in order to recruit it. If police central cannot find any specialist agent, then it will send a failure message to the police patrol. Finally, police central sends a message to the police patrol informing whether or not the proxy action was succesful. Even if specialist agents interact with police central during the subprotocol execution, they report the result of the subprotocol to the police patrol.

The subprotocol that the central police agent and specialist agents use to communicate with each other is the appropriate one to manage the request that comes from the police patrol. This shows how useful the flexibility of our conversation

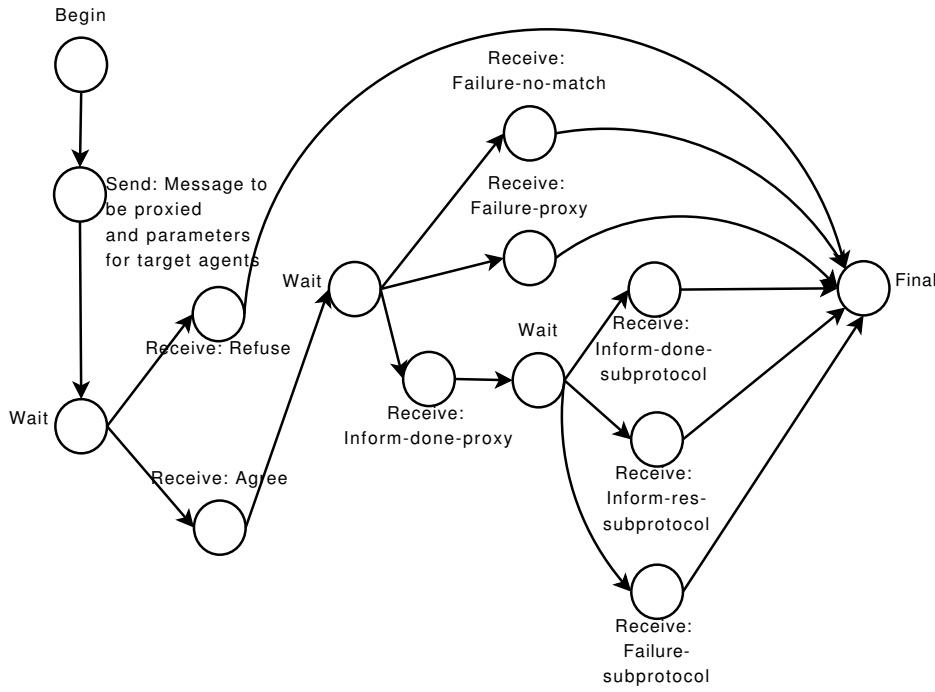

Figure 7. Recruiting interaction protocol for the initiator role

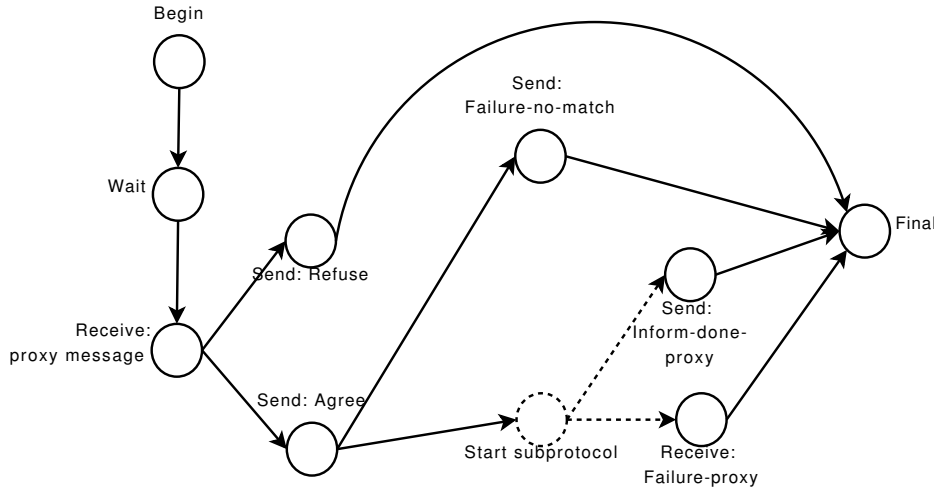

Figure 8. Recruiting interaction protocol for the participant role

support to join protocols by means of subprotocols is in a scenario like this. The subprotocol used by police central and specialist agents is not shown here due to its diversity. Depending on the message sent by the police patrol, the police central has to start an appropriate subprotocol. For example, one of the protocols used could be the *FIPA Request* IP shown in the previous section. Each subprotocol could be arranged between the police central and a specialist agent when it enters the system or could be specified by the system in advance and communicated to each new specialist agent.

Let us assume that now the scenario gets more complex. The accident involved many cars and is blocking a highway with a lot of traffic. Also, some of the injured people need medical helicopter assistance. The police patrol requests more police back-up in order to control and divert the traffic to other secondary roads. The helicopter aid cannot be requested by the police patrol due to their inability to diagnose injuries. Therefore, medical personnel have to request helicopter support and act as initiatior in the recruiting protocol in the same way the police patrol does. It is hard for police central to cope with all this complexity. Therefore, there is a risk of police central delaying petition handling. Henceforth, police central concludes that it is necessary to change the protocols governing the MAS' interactions. Once police central has decided which protocols better fit the new situation, it sends a message with the appropriate protocol described in cpXML to the agents. As explained in the previous section, the message is a proposal for changing the IP used between the specialist or police patrol agent and police central. An agent can accept or refuse the proposed change. If the agent accepts, then it will convert the cpXML specification into a *CFactory* and replace the previous IP used to communicate with police central by the new one[3], otherwise police central may propose another IP to the agent or may not consider the agent for future interactions. With this mechanism, the interactions in the MAS can adapt themselves to any circumstance and improve the communication among agents.

5 IMPLEMENTATION

This section first describes the implementation of the emergency scenario by using of the emergency scenario by using the agent architecture provided by Magentix2. It then explains how IPs are exchanged at the implementation level.

5.1 Implementation of Conversational Agents

The agent architecture shown in Section 2 focuses all the actions that the agent performs on the conversations in which it participates. The *initiator* agent has

[3] It is assumed that agents in the system are benevolent and will follow IP specifications in every interaction.

an associated *initiator CFactory*. This *CFactory* would have an associated *CProcessor* specifying the protocol shown in Figure 7. The *participant* agent has an associated *participant CFactory* that would create *CProcessors* following the protocol shown in Figure 8 and as many *initiator CFactories* as different subprotocols it would need. Some source code of the different agents that make up the MAS for emergency management is shown below. These examples show how conversational agents work and how they are programmed, thereby providing, a global view of the MAS implementation.

Before showing the code of any agent of the MAS, the protocol templates must be explained. These templates have been created to make the use of protocols specified by the FIPA standard easier (for example *FIPA Recruiting* or *FIPA Request*). Protocol templates are offered in the same API used by conversational agents, and they can be used to create new *CFactories*. These *CFactories* are adaptations made by the programmer of interaction protocols defined in the FIPA standard. The templates allow the agent programmer to focus on the actions the agent has to perform in each conversation state. It is not necessary for the programmer to consider the different states of the conversation and the transitions between them. For example, when programming the agent that plays the participant role in the *FIPA Recruiting* IP, the programmer has to define how the agent selects specialist agents to recruit depending on the message sent by the initiator role. However, the programmer does not need to define all the possible transitions from that state. This is already done by the interaction template. Interaction templates make programming with *CProcessors* and *CFactories* easier, but they do not limit the dynamic possibilities of our proposal (i.e., any *CFactory* programmed using an interaction template can be modified during execution time). In this specific example, all the agents start using FIPA standard protocols, but as the emergency situation evolves, the IPs also change and the *CFactories* created from the templates are modified or simply replaced.

Now, we show the skeleton of the police central agent source code. The code shows that the agent inherits from the CAgent class. This class refers to a *Conversational Agent*. Thus, every agent inheriting from this class can manage conversations by means of *CProcessors* and *CFactories*. The *CFactory* that manages the *FIPA Recruiting* conversation is defined in the lines of code implemented below. This *CFactory* inherits from the *FIPA_Recruiting* template. Through this inheritance, the programmer only needs to implement a few methods (more can be defined, but the template offers default behaviours in some states that can be modified if needed). The methods defined are:

- `doReceiveProxy`: In this method, the agent manages the initial message sent by the initiator and decides whether he accepts the proxy action.
- `doLocateAgents`: In this method, depending on the message sent by the initiator, the agent locates the collection of specialist agents it has to recruit.

```
1  public class PoliceCentral extends CAgent{
2    ...
3    class myFIPARecruitingParticipant extends
        FIPA_RECRUITING_Participant{
4      @Override
5      protected ArrayList<AgentID> doLocateAgents(CProcessor
          myProcessor, ACLMessage proxyMessage) {
6        ...
7      }
8      @Override
9      protected String doReceiveProxy(CProcessor myProcessor,
          ACLMessage msg) {
10       ...
11     }
12   ...
13     protected void execution(CProcessor firstProcessor,
          ACLMessage welcomeMessage) {
14       MessageFilter filter = new MessageFilter("protocol =
            fipa-recruiting");
15       CFactory recruiting = new myFIPARecruitingParticipant()
            .newFactory("TALK", filter, null, 1, this, 0);
16       this.addFactoryAsParticipant(recruiting);
17     }
18   }
19 }
```

The method `execution` is the main method of the agent. In order for *CFactory* to be able to create *CProcessors* for *FIPA Recruiting* conversations, the agent creates a new *CFactory* from the template. This is done in line 15. It is necessary to pass some arguments to the *CFactory* creation, like the name of the *CFactory*, or the filter; this specifies that only messages with the parameter protocol set to `fipa-recruiting` will start new conversation, etc..

5.2 IP Exchange between Conversational Agents

Conversational agents can exchange IPs using messages. As explained in Section 3, an agent can request an IP or it can propose the use of a specific IP to another agent. Each of these methods of exchanging an IP constitutes an IP itself. These IPs are known beforehand by all conversational agents, and the management of conversations following any of these IPs is transparent to the agent. Our agent architecture offers two methods to exchange IPs: `proposeIP(AgentID id, cpXML IP)` and `requestIP(AgentID id)`. Each method starts an appropriate conversation depending on the agent that is requesting or proposing an IP. Once a new IP is accepted (if it was proposed) or sent (if it was requested), it is automatically added as a new CFactory to the agent.

Each subprotocol that police central may start with any specialist agent has to be defined as a cpXML. The police central agent and each specialist agent have to agree beforehand as to when to use each subprotocol and with whom. In our MAS, when a new specialist agent is available, it announces its availability to police central. Then, police central answers by proposing a protocol that will be used when police central needs to recruit this specialist agent. If the proposed protocol is accepted, that specific protocol is associated with that specific agent; otherwise, the police central agent can propose another protocol or just not consider the new specialist agent as recruitable. The police central makes its proposal by sending a message to the specialist agent. This message has a cpXML specification of a CP as content. The message also specifies which role the specialist agent will play in a conversation following the protocol. Below, we show how to define the FIPA Request protocol in cpXML. This cpXML is one of the protocols that the police central agent proposes to the specialist agents as they enter the system.

```
<conversationpolicy>
<name>FipaRequest</name>
   <roles>
      <role>Initiator</role>
      <role>Participant</role>
   </roles>

   <initialstate>NoRequest</initialstate>

   <state StateId="NoRequest">
      <SendMessageTransition TransitionName="SendRequest">
         <Target>RequestSent</Target>
         <Sender>Initiator</Sender>
         <Event>SendMessage</Event>
         <Message>
            <Encoding>xml-document</Encoding>
            <Schema>Request</Schema>
         </Message>
      </SendMessageTransition>
   </State>

   <state StateId="RequestSent">
      <SendMessageTransition TransitionName="SendRefuse">
         <Target>RequestFailed</Target>
         <Sender>Participant</Sender>
         <Event>SendMessage</Event>
         <Message>
            <Encoding>xml-document</Encoding>
            <Schema>Refuse</Schema>
```

```
30          </Message>
31       </SendMessageTransition>
32
33       <SendMessageTransition TransitionName="SendAgree">
34          <Target>Agreed</Target>
35          <Sender>Participant</Sender>
36          <Event>SendMessage</Event>
37          <Message>
38             <Encoding>xml-document</Encoding>
39             <Schema>Agree</Schema>
40          </Message>
41       </SendMessageTransition>
42    </State>
43
44    <State StateId="Agreed">
45       <SendMessageTransition TransitionName="SendFailure">
46          <Target>RequestFailed</Target>
47          <Sender>Participant</Sender>
48          <Event>SendMessage</Event>
49          <Message>
50             <Encoding>xml-document</Encoding>
51             <Schema>Failure</Schema>
52          </Message>
53       </SendMessageTransition>
54
55       <SendMessageTransition TransitionName="SendInform">
56          <Target>RequestDone</Target>
57          <Sender>Participant</Sender>
58          <Event>SendMessage</Event>
59          <Message>
60             <Encoding>xml-document</Encoding>
61             <Schema>Inform</Schema>
62          </Message>
63       </SendMessageTransition>
64    </State>
65
66   <State StateId="RequestDone">
67         <return>Done</return>
68   </State>
69
70   <State StateId="RequestFailed">
71         <return>Fail</return>
72   </State>
73
74 </ConversationPolicy>
```

The implementation of the police patrol is quite similar to the implementation of police central. Obviously, the *CFactory* that is added to the agent is not the same. In this case, the police patrol uses the template for the initiator role of the FIPA Recruiting protocol. When this template is used, the most important method is the `setProxyMessage` method. In this method, the agent has to specify the message that will be proxied and to which specialist agents.

It is not necessary to define any protocol during the implementation of the specialist agents. These agent will receive the specification of the protocol that they will use to communicate with police central when they enter the MAS. This learning capability is the same that all the agents in the MAS use when the emergency situation conditions change and the agents need to adapt to them. The previous section showed how the emergency situation can change the interaction of the agents. Police is in charge of determining which protocols must be changed. To accomplish this, police central will send a *propose* message to every agent in the system. The content of these messages will be a new and appropriate protocol for the new situation.

6 RELATED WORK

Several APs provide support for specifying IPs. As an example, Jade [12] and Madkit [14] support the execution of IPs proposed by FIPA. However, they do not provide support for developing their own IPs. Zeus [13] also provides support for specifying their own IPs by means of the *Protoz* environment. It allows the specification of the roles involved in an IP by means of FSM. The transitions associated to the FSM are triggered by incoming messages (from other agents or from internal procedures). Agentbuilder [15] also provides tools for specifying IPs. It provides a protocol editor tool that allows the specification of IPs by means of FSM.

The major drawback of these supports for IPs is that they do not allow the modification of the IPs while the system is running. Therefore, applications that involve runtime changes in the IP specification are not able to be developed. Furthermore, since IPs are specified before execution, agents are not able to negotiate the IP associated to their conversation.

With regard to works that deal with dynamic IPs, Artikis et al. [22, 23] present a framework for specifying open systems from the perspective of organizations instead of individual agents. They represent open MAS as normative systems by specifying what is permitted, prohibited, and obligatory. In this framework, the specification of IPs is carried out at design time but can be modified at execution time since the rules that govern the protocol may change. However, this approach restricts the range of applications to normative systems. We propose a support at the AP level so that the users can apply their own system model according to the requirements of the specific applications. In fact, Artikis' framework could be implemented using our work in this paper.

Walton et al. [24] propose a method for defining interaction protocols during runtime. Therefore, agents are able to interact in systems where the interaction

protocol may be unknown beforehand. They provide a language for defining IPs that will be created during conversations by the participants. Although this proposal focuses on large and open MAS, it is not integrated in any AP. This severely limits its practical application. Our proposal provides a support for dynamic IPs at the AP level. Therefore, it supports the management of not only single messages but also of entire conversations.

The support for design and execution of IPs that we presented in [5] did not use a structured language to specify IPs. By using a language for specifying the IP, agents are able to exchange the IP they are going to use in their conversation or in the modification of a previous IP. Besides, by representing the IP in two different places (in the initiator and in the participant agents), it is difficult for a user to interpret the IP. To correct these problems, a conversation description language and two different methods for exchanging IPs have been added to our proposal. The conversation description languages aim to fill the gap between conversation definition and implementation.

Our proposal provides support to IPs modelled as FSM, but other approaches model IPs through the use of different methods. One of such methods is the use of Enhanced Dooley Graphs (EDG) for agent modeling [25], which is a technique based on Dooley graphs [26]. This proposal is focused on basing the design of new agents on the analysis of existing agents and their interactions. This analysis finds patterns that can guide subsequent implementation of agents. However, this proposal has not been implemented. Another method is Nowostawski's proposal [27], which is based on modeling IPs as coloured Petri nets (CPN). He also provides an implementation at the agent level, which is integrated in the AP Opal.

7 CONCLUSIONS

In this paper, a support based on the specification and execution of dynamic IPs has been presented. Although some other approximations have been presented, few have been implemented or can support dynamic protocols. The support presented in our approach can even implement some of these works.

This support can be used to create a MAS where the interaction between the agents is non-trivial and the IPs directing these interactions have to adapt to a dynamic changing environment. This support greatly helps to improve applications where service-providing agents can enter and leave the system during its execution. These new agents need to know which interaction protocols are in use in the system in order to offer their services and to request other agents' services. The conversational agent architecture can be used to program the agents that make up a MAS with these characteristics. Moreover, the capacity of agent architecture to maintain several interactions following the same protocol improves the MAS performance.

CpXML has been added to our previous proposal. This addition has increased the functionality of our software. CpXML language can be used to communicate

IPs among the agents of the MAS. CpXML is also used to centralize the definition of IPs and make them easier to manage and understand by humans.

We have also shown an implementation of an emergency management system using the support. An emergency situation is very unpredictable and can change easily; hence, systems that manage these situations have to be flexible and adaptive. The agent architecture is offered in an API integrated in Magentix2, which is already available. Using conversational agents and their functionality, our system is able to cope with dynamically changing situations such as an emergency. The API also offers protocol templates that help programmers to easily modify IPs that are widely used without losing any adaptive characteristic.

In the future, we plan to implement a visual tool for designing interaction protocols. This tool should facilitate the use of cpXML to define CP. Once a CP has been defined, it would be converted into Java code. This code could then be used to implement agents. This tool could also be embedded in a CASE utility to facilitate the design and implementation of MAS. Another objective for the future is to extend our interaction support as a part of an interaction-guided agent model. In this model, concurrent interactions with other agents will be considered in the reasoning process of the agent.

Acknowledgements

This work has been partially supported by CONSOLIDER-INGENIO 2010 under grant CSD2007-00022, and project TIN2008-04446.

REFERENCES

[1] Luck, M.—McBurney, P.—Shehory, O.—Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink, 2005.

[2] Valetto, G.—Kaiser, G.E.—Kc, G.S.: A Mobile Agent Approach to Process-Based Dynamic Adaptation of Complex Software Systems. Proceedings of the 8th European Workshop on Software Process Technology (EWSPT '01), Springer-Verlag, London, UK 2001, pp. 102–116. ISBN 3-540-42264-1, http://dl.acm.org/citation.cfm?id=646199.681826.

[3] Dignum, V.—Dignum, F.—Sonenberg, L.: Towards Dynamic Reorganization of Agent Societies. Proceedings of Workshop on Coordination in Emergent Agent Societies, 2004, pp. 22–27.

[4] Li, N.—Tarus, H.—Irvine, J. M.—Moessner, K.: A Communication Middleware for Ubiquitous Multimedia Adaptation Services. Computing and Informatics, Vol. 29, 2010, No. 4, pp. 628–646.

[5] Fogués, R. L.—Alberola, J. M.—Such, J. M.—Espinosa, A.—Garcia-Fornes, A.: Towards Dynamic Agent Interaction Support in Open Multiagent Sys-

tems. Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence, IOS Press, 2010, pp. 89–98.

[6] Magentix2: `http://users.dsic.upv.es/grupos/ia/sma/tools/magentix2/index.php`.

[7] Manoj, B. S.—Baker, A. H.: Communication Challenges in Emergency Response. Communications of the ACM, Vol. 50, 2007, No. 3, pp. 51–53.

[8] FIPA: FIPA Interaction Protocol Library Specification. 2002

[9] Alberola, J. M.—Such, J. M.—Garcia-Fornes, A.—Espinosa, A.—Botti, V.: A Performance Evaluation of Three Multiagent Platforms. Artificial Intelligence Review, Vol. 34, 2010, No. 2, pp. 145–176.

[10] Such, J.M.—Alberola, J. M.—Barella, A.—Garcia-Fornes, A.: A Secure Group-Oriented Framework for Intelligent Virtual Environments. Computing and Informatics, Vol. 30, 2011, No. 6, pp. 1225–1246.

[11] Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley and Sons 2002.

[12] Bellifemine, F.—Poggi, A.—Rimassa, G.: Jade: A fipa2000 Compliant Agent Development Environment. Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS '01), ACM Press, New York 2001, pp. 216–217.

[13] Nwana, H. S.—Ndumu, D. T.—Lee, L. C.—Collis, J. C.—Re, I. I.: Zeus: A Tool-Kit for Building Distributed Multi-Agent Systems. Applied Artificial Intelligence Journal, Vol. 13, 1999, pp. 129–186.

[14] Gutknecht, O.—Ferber, J.: MadKit: A Generic Multi-Agent Platform. Proceedings of the Fourth International Conference on Autonomous Agents, ACM, 2000, pp. 78–79.

[15] Agentbuilder: Acronymics, Inc. `http://www.agentbuilder.com`.

[16] McGinnis, J.—Robertson, D.: Dynamic and Distributed Interaction Protocols. Proceedings of the AISB 2004 Convention, 2004, pp. 45–54.

[17] Hanson, J. E.—Nandi, P.—Levine, D. W.: Conversation-Enabled Web Services for Agents and E-Business. Proceedings of the International Conference on Internet Computing (IC-02), CSREA Press, 2002, pp. 791–796.

[18] Takuo, D.—Tahara, Y.—Honiden, S.: IOM/T: An Interaction Description Language for Multi-Agent Systems. Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, 2005, pp. 778–785.

[19] FIPA: FIPA Request Interaction Protocol Specification. 2002.

[20] Meissner, A.—Luckenbach, T.—Risse, T.—Kirste, T.—Kirchner, H.: Design Challenges for an Integrated Disaster Management Communication and Information System. The First IEEE Workshop on Disaster Recovery Networks (DIREN 2002), New York City, USA, June 24, 2002.

[21] FIPA: FIPA Recruiting Interaction Protocol Specification. 2002.

[22] Artikis, A.: Dynamic Protocols for Open Agent Systems. Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), 2009, pp. 97–104.

[23] Artikis, A.—Sergot, M.: Executable Specification of Open Multi-Agent Systems. Logic Journal of the IGPL, Vol. 18, 2010, No. 1, pp. 31–65.

[24] Walton, C.: Dialogue Protocols for Multi-Agent Systems. Informatics Research Report EDI-INF-RR-0183, University of Edinburgh, 2003, 12 pp.

[25] Parunak, H. V. D.: Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis. Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS '96), 1996, pp. 275–282.

[26] Dooley, R. A.: Repartee as a Graph. Appendix B in R. E. Longacre: An Anatomy of Speech Notions, Peter de Ridder, Lisse, Holland, 1976, pp. 348–358.

[27] Nowostawski, M.—Purvis, M.—Cranefield, S.: A Layered Approach for Modelling Agent Conversations. Proceedings of the Second International Workshop on Infrastructure for Agents, MAS and Scalable MAS, the Fifth International Conference on Autonomous Agents, Montreal, Canada, 2001, pp. 163–170.

Ricard Fogues is currently working at Universitat Politècnica de València towards the Ph. D. degree in computer science and holds a research grant supported by the European Union. He received the B. Sc. from Universitat Jaume I, Castello, Spain and M. Sc. degree in artificial intelligence, pattern recognition and digital image from Universitat Politècnica de València, Valencia, Spain, in 2007 and 2010, respectively. His research interests include privacy, access control models, and self presentation, and relationship management on social media.

Jose M. Such is Lecturer (Assistant Professor) in the School of Computing and Communications at Lancaster University (UK) since 2012. In 2011 he was awarded by Ph. D. degree in computer science from Universitat Politècnica de València (Spain), where he was research fellow. His main research interests are on the intersection between artificial intelligence and cyber security, and in particular, intelligent/automated approaches to privacy, identity management, access control models, trust and reputation. He is also interested in human factors in cyber security and machine learning applied to cyber security.

Juan M. Alberola is a postdoc researcher at the Departament de Sistemes Informàtics i Computació of the Universitat Politècnica de València. He received his Ph. D. in computer sience in 2013. His research interests include agent organizations, adaptation, multiagent systems, artificial intelligence application in educational environments, teamwork and coalition formation, case-based-reasoning, prediction markets, and electronic markets.

Agustín Espinosa is Lecturer at the Departament de Sistemes Informàtics i Computació of the Universitat Politècnica de València and a researcher in the GTI-IA research group of the Universitat Politècnica de València. His research interests include multiagent systems, agent architectures, agent platforms, agent frameworks, and real-time agents. He received his Ph. D. in computer science from the Universitat Politècnica de València, Spain in 2003.

Ana Garcia-Fornes holds a position of Associate Professor of computer science at the Universitat Politècnica de València since 1999, where she has taught since 1986. She is co-founder of the GTI-IA research group and Director of the Area of Research Programs and Initiatives at the UPV. Her research interests include knowledge based systems, multi-agent systems (negotiation, privacy, platforms, adaptation), agreement technologies, real-time artificial intelligence and real-time systems scheduling.