WOA 2004

dagli Oggetti agli Agenti

Sistemi Complessi e Agenti Razionali

Torino (Italia), 30 novembre 2004 – 1 dicembre 2004

Atti delle giornate di lavoro a cura di:

Matteo Baldoni, Flavio De Paoli,
Alberto Martelli e Andrea Omicini

Organizzato da:

Associazione Italiana per l'Intelligenza Artificiale
Associazione Italiana Tecnologie Avanzate basate su concetti Orientati ad Oggetti

Con il patrocinio del Dipartimento di Informatica e dell'Universita' degli Studi di Torino

http://woa04.unito.it

woa04@di.unito.it

PREFAZIONE

Le tecnologie degli agenti stanno assumendo un ruolo centrale non solo nel settore dell'intelligenza artificiale, ma anche in settori più tradizionali dell'informatica quali l'ingegneria del software e i linguaggi di programmazione, dove il concetto di agente viene considerato una naturale estensione di quello di oggetto. L'importanza di queste tecniche è dimostrata anche in ambito industriale dall'interesse per il loro utilizzo nella realizzazione di strumenti e applicazioni in molteplici aree.

Il presente volume raccoglie gli atti della quinta edizione delle giornate di lavoro "dagli Oggetti agli Agenti" edizione 2004, dedicata al tema "Sistemi Complessi e Agenti Razionali". Le giornate di lavoro sono state organizzate dal gruppo di lavoro "Sistemi ad Agente e Multiagente" dell'Associazione Italiana per l'Intelligenza Artificiale (AI*IA) e l'Associazione Italiana Tecnologie Avanzate Basate su Concetti Orientati ad Agenti (TABOO) in collaborazione con il Dipartimento di Informatica dell'Università degli Studi di Torino, gruppo di lavoro "Logic Programming and Automated Reasoning". I venti articoli di questa collezione comprendono sei lavori che, avendo come primo autore uno studente, hanno partecipato al premio studenti, alla sua prima edizione ed indetto in occasione della prima miniscuola organizzata in occasione delle giornate di lavoro. Titolo della miniscuola "Agenti e Oggetti @ Work". L'evento si è svolto nei giorni 29 e 30 novembre e 1 dicembre 2004, presso il Dipartimento di Informatica dell'Università degli Studi di Torino, enti patrocinanti.

Un particolare ringraziamento va al comitato organizzatore locale, in particolare a Cristina Baroglio per la realizzazione del sito, allo staff tecnico del Dipartimento di Informatica e a Simone Donetti per il software OpenChair.

Torino, 4 novembre 2004.

Matteo Baldoni, Flavio De Paoli,
Alberto Martelli e Andrea Omicini

Gli atti delle giornate di lavoro sono pubblicati dalla Pitagora Editrice Bologna, ISBN 88-371-1533-4.

Comitato Scientifico Organizzatore

Matteo Baldoni	(Univ. di Torino)
Flavio De Paoli	(Univ. di Milano – Bicocca)
Alberto Martelli	(Univ. di Torino)
Andrea Omicini	(Univ. di Bologna – Cesena)

Comitato Organizzatore Locale

Matteo Baldoni	(Univ. di Torino)
Cristina Baroglio	(Univ. di Torino)
Alberto Martelli	(Univ. di Torino) – Presidente
Viviana Patti	(Univ. di Torino)

Comitato di Programma

Stefania Bandini	(Univ. di Milano – Bicocca)
Pietro Baroni	(Univ. di Brescia)
Carlo Bellettini	(Univ. di Milano)
Fabio Bellifemine	(TILab)
Federico Bergenti	(Univ. di Parma)
Enrico Blanzieri	(Univ. di Trento)
Paolo Bouquet	(Univ. di Trento e IRST)
Giacomo Cabri	(Univ. di Modena e Reggio Emilia)
Marco Cadoli	(Univ. di Roma "La Sapienza")
Giancarlo Cherchi	(Univ. di Cagliari)
Marco Colombetti	(Politecnico di Milano)
Francesco Donini	(Univ. della Tuscia – Viterbo)
Rino Falcone	(ISTC-CNR)
Letizia Leonardi	(Univ. di Modena e Reggio Emilia)
Marco Mamei	(Univ. di Modena e Reggio Emilia)
Sara Manzoni	(Univ. di Milano – Bicocca)
Viviana Mascardi	(Univ. di Genova)
Emanuela Merelli	(Univ. di Camerino)
Rebecca Montanari	(Univ. di Bologna)
Maria Teresa Pazienza	(Univ. di Roma – Tor Vergata)
Alessandro Ricci	(Univ. di Bologna – Cesena)
Giovanni Rimassa	(Univ. di Parma)
Corrado Santoro	(Univ. di Catania)
Carla Simone	(Univ. di Milano – Bicocca)
Eloisa Vargiu	(Univ. di Cagliari)
Mirko Viroli	(Univ. di Bologna – Cesena)
Giuseppe Vizzari	(Univ. di Milano – Bicocca)

Direttivo WOA

Giuliano Armano	(Univ. di Cagliari)
Antonio Corradi	(Univ. di Bologna)
Flavio De Paoli	(Univ. di Milano – Bicocca)
Andrea Omicini	(Univ. di Bologna – Cesena)
Agostino Poggi	(Univ. di Parma)
Franco Zambonelli	(Univ. di Modena e R. Emilia)

INDICE DEI LAVORI

Evaluating Trust Among Agents

GIACOMO CABRI, LUCA FERRARI, LETIZIA LEONARDI
Dipartimento di Ingegneria dell'Informazione – Università di Modena e Reggio Emilia
Via Vignolese, 905 – 41100 Modena – ITALY
Phone: +39-059-2056190 – Fax: +39-059-2056126
E-mail: {cabri.giacomo, ferrari.luca, leonardi.letizia}@unimo.it

Abstract – *Agent-based applications are more and more exploited in the development of distributed systems, with particular regard to the Internet ones. Even if the development of agent-based applications is not so difficult today – thanks to new paradigms and techniques – security problems are still present. In particular, it is important to deal with security of the data exchanged between agents at runtime. In fact, agents are social, and they interact with other agents in order to carry out specific tasks. Since interacting agents could be developed by different programmers, or provided by different third parties, there is the risk that the interacting counterpart could act maliciously with the received data. In this paper we propose an approach based on the concept of trust, which is more dynamic and adaptable than security, in order to evaluate if an interaction can be done or not.*

Keywords*: Agents, Roles, Interactions, Trust*

1. Introduction

In agent-based applications, interactions among agents are largely exploited in order to use services that they can provide. This situation leads to a continue cooperation between agents developed by different programmers and provided by different vendors, cooperation that often requires a data exchange. Often, the interaction with other agents is crucial for the success of the activities of an agent, so interactions must be carefully considered in agent-based applications.

In a static black and white world, an agent knows a priori whether interacting with another agent or not, while in a dynamic colored world many issues must be considered at runtime. The traditional approach based on security is no longer enough in a very dynamic, uncertain and unpredictable world such as the agents' one. As a first issue, a sure authentication may be not so easy to achieve in a wide environment such as the Internet. Second, the skill of the counterpart can be an important issue to decide whether to perform the interaction or not; an authenticated and secure agent could not provide the exact service needed, or it can provide the service not in the best way. In the evaluation of the skill, previous experiences can help in the decision. These considerations lead to a concept more flexible than security: *trust*. During an interaction between agents, it is important that each involved part can evaluate the trust that the interaction will have.

In this paper, we propose a preliminary study on an evaluation of trust level between two mobile agents, thanks to which agents will be able to start or reject an interaction with more confidence.

Our study is related to mobile agents, since they are an exploited technology in the development of distributed and Internet-based applications today. Furthermore, since interactions between agents are often exploited to carry out a task, and since there are good proposals that model interactions exploiting the concept of role [1, 4, 5, 6], our approach explicitly introduces the trust in the assumed role too.

The paper is organized as follows: section 2 introduces other concepts about trust, section 3 explains how our approach computes the trust level between agents, while section 4 details the Java implementation of our approach. Finally section 5 gives conclusions.

2. About Trust

The concept of trust applied to computer science is not new, and in fact we can find other studies in [2, 7, 11]. What these studies emphasize is that trust can be the compound result of trust assigned to different components, thus it is not possible to evaluate the global trust before having evaluated each component. Furthermore, trust depends on not immediately visible properties, and in particular it is based on the capability of predicting these properties and of trusting them. Finally, the trust level of an agent cannot be a fixed property of the single agent, but it depends also on the other agents with which it interacts (or have interacted).

The main difference between *security* and *trust* is that the latter is more subjective and context dependent. In fact, while security is typically set up before the execution of the application, allowing administrator to change rules during the application evolution, trust is decided by the application components themselves. In other words, while security is typically set up *externally* from the application, trust comes from the *inside* of the application, since it is evaluated by the running components themselves. Since trust comes within the application, without requiring external entities (e.g., administrators), this leads to a dynamic situation, where the application can take decisions considering the current environment.

Typically, what happens is that an agent starts an interaction with another agent only if the latter has a trust level greater than a threshold, which usually depends on the goal and the kind of task of the former agent.

It is important to note that building a system based on trust does not mean to simply apply one (or more) threshold to system parameters, since this would lead to a security-based application. To better explain this concept,

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

imagine that an agent trusts another agent if it owns at least the 80% of a common secret password. This could mean that the first agent trusts the second at the 80%. But at a deeper look, this does not represent a trust threshold, but a security threshold. In fact, in the above situation, it is like if the common password must be shorter than the complete one to allow agents to interact, which means that the security level is lower than the one required with the complete password. Even if the threshold can be changed during the application, the situation can be always reconsidered as a security issue.

From the above example it should be clear that evaluating trust does not mean to simply apply variable thresholds. Trust requires other control mechanisms, and, in particular, the capability to evaluate and change trust levels autonomously during the application evolution. Nevertheless, even being able to evaluate and adapt thresholds during the application does not suffice, and it is for this reason that trust needs also *history*. In fact, only evaluating the trust level over different time instants it is possible to get a very subjective value.

Trust should always be computed dependently on the target of the action the agent is doing, since it is not possible to evaluate trust just related to another agent without considering also the action to perform. This means that there could be different trust levels among the same agents, depending on the actions/interactions they are doing together. Starting from the above considerations, it should be clear that the trust computation should also have a fine grain, depending on the involved agents and interactions.

Furthermore it is important to note that trust should not be considered negatively, but positively. In other words, it is more important to understand which could be the positive consequences of granting trust to a partner, rather than the negative ones (or risks) due to a bad evaluation [7]. In this situation interactions will be promoted, and not rejected due to a not 100% trust level.

The following section shows the formula we propose to evaluate trust level between agents.

3. Computing the Trust Level

Since agents are computational entities, they cannot evaluate the trust as humans do (i.e., based on emotions, feelings, intuitions, instincts, and so on). In order to allow agents to evaluate the trust related to other agents or components in a computational way, we propose the following formula:

$$T_{ij} = \frac{(1-S)\cdot A\cdot c_A + S\cdot(c_S + c_I\cdot I) + H\cdot c_H + P\cdot c_P + R\cdot c_R}{(1-S)\cdot c_A + S\cdot(c_S + c_I) + c_H + c_P + c_R}$$

where T_{ij} represents the trust level of the agent *j* computed by the agent *i*. As detailed in section 1, the global trust can be evaluated only if all its components have been evaluated. In the above formula the terms c_x represent weights of the several parameters. They say how much the agent wants to consider a given parameter in the evaluation of the trust. The parameters are the following:

- *S* indicates if the agent is signed or not. The value can be only 0 or 1, depending on the presence of the signature(s) of the agent;
- *A* stands for the authentication of the agent and can embed both *credentials* and *code type*. The former could be, for example, passwords or secrets useful to authenticate the agent or its owner. The latter represents an introspection on the agent code in order to understand, for example, the base classes used to build it, or if it contains dangerous instructions, etc.;
- *I* represents the identity of the signer of the agent (if present);
- *H* represents the *history* of the interactions of the agent *j*. The history is important in order to evaluate in a more subjective way the trust level of *j* perceived by *i*, thus the agent *i* can understand if agent *j* has been a bad agent in the past or not;
- *P* stands for the previous host of the agent. Since this work has been done explicitly in the mobile agent context, we have decided to explicitly insert the previous host parameter in the formula;
- *R* represents the trust of the agent *j* feel by the role assumed by the agent *i*. It can be computed with a formula similar to the above one:

$$R = \frac{(1-S)\cdot A\cdot c_{AR} + S\cdot(c_{SR} + c_{IR}\cdot I) + H\cdot c_{HR} + P\cdot c_{PR}}{(1-S)\cdot c_{AR} + S\cdot(c_{SR} + c_{IR}) + c_{HR} + c_{PR}}$$

where all the terms and the weights have the same meaning described above, even if, as also indicated by the weight subscripts, they are related to the role and not to the agent itself.

Why there is the need of evaluating trust even of the assumed role? First of all it is important to recall the fact that roles are *external* components to agents, which are exploited by them during the execution of the application. The fact that roles are external entities, and the fact that they are usually tied to the local execution environment [3], means that agents have no warranties about the piece of code they are going to exploit. It is for this reason that the trust level must include also the trust about the role (if there is one).

It is important to note that *S* is the only one parameter that can assume a boolean value, depending on the presence of the signature(s), while the other can be between 0 and 1. The weights c_x are between 0 and 10[1], and this means that the final value of T_{ij} will be always less or equal to 1:

$$S \in \{0,1\}; A, I, H, P, R \in [0,1]; c_x \in [0,10] \Rightarrow T_{ij} \in [0,1]$$

Please note that, as shown in the first formula, when the agent is not signed (i.e., *S*=0) the identity term *I* is not considered in the computation of the trust level, while when the agent is signed (*S*=1), the term of the authentication *A* is not considered. In fact, since the agent is signed, there is no need to authenticate it, but the signature(s) can be used as authentication as well.

A very important term in the first formula is *S*, which represents the history of the actions of the agent *j*, thus the agent *i* can try to understand if the opponent has been fair or not. The term *S* is not trivial to calculate, due to the fact that is not always simple to keep a track of the history of past actions of each agent, depending on the platform

[1] Please note that the trust level T_{ij} will always be normalized, independently from the range of values the weights c_x belongs to. The choice of the latter range depends on the level of granularity required by the computation.

implementation. What an agent can do, in the case that the platform does not provide support for the history of actions, is to keep a private track of actions/interactions with a specific agent, in order to be able to further evaluate the term *S* when needed. For example, an agent can progressively compute *S* with the following simple count:

$$S = \frac{succesfully\_done\_transactions\_with\_agent\_X}{totally\_done\_transactions\_with\_agent\_X}$$

while the weight c_s should become greater as the successful transactions are recent or not.

All the capitalized terms in the first and second formulas represent parameters that are fixed for all agents, but what makes the formulas subjective is the use of weights c_x. In fact, while an agent should not change values of the parameters, it can change values of weights, in order to adapt the computation of the trust level to its execution environment.

3.1. Considerations about the Formula

It is important to note that the formula can be used to compute trust even if not all terms are available. For example, in some implementations the history (H) could not be present, thus agents have to compute the trust level with a "partial" formula. Of course, the use of an uncomplete formula will produce less reliable results, since the space of possible result values is shrinked.

Another important thing to note is that, even if "trust" is not the same of "security", as already written, the formula is partially based on a set of security terms (like A and I), since we believe that first of all trustness should imply also security (but not vice versa).

Finally, please note that the choice of weights is in charge of the agent (and its developer), since the agent must evaluate by itself how much important are the information about the different interacting entity.

4. A Java Implementation

In order to ease the use of the first formula we have implemented a set of Java classes that Java agents can exploit. This section gives a presentation of this set of classes as first, and then briefly shows an application developed using IBM Robocode, which exploits the above classes.

4.1. Java Classes to Compute the Trust Level

All the classes are contained in a single package, `it.unimo.brain.trust`. It is important to note that, in order to grant a high flexibility in the computation of the trust level, almost all classes are abstract. Nevertheless, in order to give developers a library ready to use, we provided a subpackage, called `impl`, which contains default implementations of the main abstract classes.

Since the T_{ij} is a sum of terms, each one composed of other sums or multiplications of a weight and a capitalized term, we introduced the base class *Term* (see Figure 1), which represents the result of a capitalized term and its weight. In this way it is quite simple to compute the whole formula, since developers have just to add each term, while the terms will compute themselves transparently.

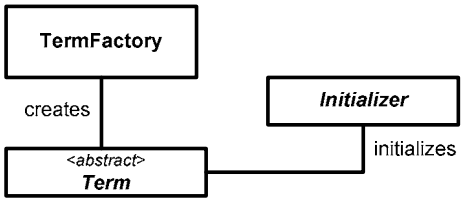


Figure 1 Main classes of the Java implementation.

Before it can be used, a term must be initialized, that means it must be able to compute the right value. For this reason we provided an interface, *Initializer*, which has been specialized for each term in order to load the right values. For example, in the case of the computation of *H*, the initializer must contain a table of known host and the values for the trust for each of them. To make all the formulas more flexible, we provided also a factory class that gives the current implementation of each term. The agent is just in charge of calling the method *getTerm* of the factory with the constant that identifies the term. The following piece of code shows an example of the initialization and use of the *S* term:

```
// get a new initializer for F
Initializer init = new SignatureInitializer();
// get the Term from the factory
Term S = TermFactory.getTerm(S_TERM);
// initialize the term
S.initialize(init,agent.getClass().getName());
. . .
// use the term
float weight = ...;
float val_S = S.getValue(weight);
```

It is important to note that the initialization does not provide the weight used in the formula, and this is to obtain a more dynamic system. In fact, since weights adapt the formula to the current context, and since they must be personalized for each agent, they must be provided at the moment of the computation, i.e., when the *getValue* method is called.

4.2. Exploiting the Formula in Robocode

In order to prove the usability of the first formula and of its Java implementation, we have tested it in an application developed using the IBM Robocode game platform [8, 9, 10]. Robocode is a Java platform used to implement simple Java games (see Figure 2), where developers can program robots (represented as tanks) that battle each other.

We have chosen this particular platform for two main reasons. First of all the scenario is very similar to the one of agents, and in fact, robots are free to move and interact each other, in a cooperative or competitive way. Furthermore, robots can cheat and can be cheated, and in this situation the computation of trust gain more importance. The second reason is that the use of Robocode gives developers a concrete visible evolution of the application, that means it is possible to understand how robots trust each other simply watching the battle. This is useful in particular in didactic experiences.

Figure 2 The Robocode battle-of-trust.

In the developed application, there is a particular robot that evaluates the trust levels between itself and the other robots, killing those it does not trust. This leads to a situation where only trusted robots survive.

Of course, in this simulation a few parameters of the first formula have just been set to the default values, since they do not have a specific meaning. For example, since the robots execute in the same host, the *H* parameter has been set to a value depending on the team they belong, in order to simulate the provenience from different hosts.

Conclusions

In this paper we proposed a preliminary study for trust evaluation in agent interactions. Unlike other approaches, ours explicitly takes care of mobility and of the exploitation of roles in interactions.

Our approach is based on a formula, which allows agents to compute the trust level as composed of different components that include the history of previous interactions, in order to allow a complete evaluation. Thanks to the use of weights in the formula, which can be adapted for the current context, the formula is suitable for different situations and agents. This allowed us to develop a set of Java classes which can be exploited to compute the formula value (i.e., the trust level) in Java agent applications. We applied the formula also to other scenarios, similar to those of agents, in order to demonstrate that is quite general and can be easily adapted to different applications.

Future work includes a better evaluation of each component of the formula, in order to understand if they are complete or must be extended. Furthermore, a standardization of the computation of the history will help in the computation of trust.

Acknowledgments: Work supported by the Italian MIUR and CNR within the project "IS-MANET, Infrastructures for Mobile ad-hoc networks", and by the MIUR within the project "Trust and law in the Information Society. Fostering and protecting trust in the market, in the institutions and in the technological infrastructure".

References

[1] D. Baumer, D. Riehle, W. Siberski, M. Wulf, "The Role Object Patterm", Pattern Languages of Programming conference, 1997, Monticello, Illinois, USA

[2] P. A. Buhler, M. N. Huhns, "Trust and Persistence", IEEE Internet Computing, April 2001, p. 85-87

[3] G. Cabri, L. Ferrari, L. Leonardi, "The Role Agent Pattern: a Developers Guideline", in Proceedings of the 2003 IEEE International Conference on System, Man and Cybernetics, 5-8 October 2003, Washington D.C., U.S.A.

[4] G. Cabri, L. Leonardi, F. Zambonelli, "Separation of Concerns in Agent Applications by Roles", in Proceedings of the 2nd International Workshop on Aspect Oriented Programming for Distributed Computing Systems (AOPDCS 2002), at the International Conference on Distributed Computing Systems (ICDCS 2002), Wien, July 2002

[5] G. Cabri, L. Leonardi, F. Zambonelli, "Modeling Role-based Interactions for Agents", The Workshop on Agent-oriented methodologies, at the 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2002), Seattle, Washington, USA, November 2002

[6] G. Cabri, L. Leonardi, F. Zambonelli, "Implementing Role-based Interactions for Internet Agents", 2003 International Symposium on Applications and the Internet, Orlando (USA), January 2003.

[7] R. Falcone, O. Shehory, "Tutorial 12 – Trust delegation and autonomy: foundations for virtual societies", Autonomous Agents & Multiagent Systems (AAMAS2002), Bologna, Italy, July 2002

[8] IBM Robocode Ufficial Site http://robocode.alphaworks.ibm.com/home/home.html

[9] IBM Robocode - API Documentation http://robocode.alphaworks.ibm.com/docs/robocode/index.html

[10] Sing Li, "Rock 'em, sock 'em Robocode!", paper available on line at http://www-106.ibm.com/developerworks/java/library/j-robocode/

[11] J. Seigneur, S. Farrell, C. Jensen, E. Gray, Y. Chen, "End-to-end Trust Starts with Recognition", Proceedings of the First International Conference on Security in Pervasive Computing, Boppard, Germany, March 2003

Customer Information Sharing between E-commerce Applications

Liliana Ardissono, Marco Botta
Luca Di Costa, Giovanna Petrone
Dipartimento di Informatica
Università di Torino
Corso Svizzera 185, Torino, Italy
Email: {liliana, botta, giovanna}@di.unito.it

Fabio Bellifemine, Angelo Difino
Barbara Negro
Telecom Italia Lab
Multimedia Division
Via Reiss Romoli, 274 - Torino, Italy
Email: {Fabio.Bellifemine, Angelo.Difino, Barbara.Negro}@tilab.com

***Abstract*— The management of one-to-one business interaction is challenged by the latency in the acquisition of information about the individual customer's preferences. Although sharing this type of information would empower service providers to personalize the interaction with new customers since the first connection, this idea can be hardly applied in real cases if the service provider cannot protect the data it has acquired from competitors and select the trusted parties from which it wants to receive information.**

As a solution, we propose a framework supporting the controlled sharing of customer information between e-commerce applications. Our framework includes two main components: 1) a Trust Management System (running off-line with respect to the information sharing service), which enables the service provider administrator to specify restrictions on the service providers to be considered as trusted parties; 2) a User Modeling Agent, which manages the propagation of customer data between service providers, given their trust relationships. The User Modeling Agent also takes care of combining the customer information provided by the trusted parties in order to generate an overall view of the customer preferences.

I. Introduction

Various techniques have been applied in Web-based stores and electronic catalogs to personalize the recommendation of products; see [1], [2], [3], [4]. For instance, collaborative filtering [5] steers the recommendation of goods by analyzing the similarities in the purchase histories of different people. Moreover, content-based filtering (e.g., see [6]) recommends goods having properties that the individual customer preferred in the past. In all cases, the customer's behavior has to be observed for some time in order to acquire a user model describing her preferences. Thus, a delay occurs before the service provider application personalizes the interaction in an effective way.

Indeed, the preference acquisition process can be speeded up if the service providers exchange their customer information with one another. For instance, if two book sellers trust each other, they might share the user models describing their customers in order to increase the knowledge about the common customers and to extend the set of visitors they can handle as known ones. In Business to Customer e-commerce, several service providers already exploit their own user modeling systems to analyze clickstream data and locally manage their customers' profiles. For these providers, the main purpose of sharing customer information with other (trusted) parties is that of acquiring information about unknown customers (first time visitors) or recently acquired customers, whose profiles are not yet complete.

In this paper, we propose a framework supporting a controlled propagation of customer information among e-commerce applications. The framework includes a Trust Management System that enables the administrators of individual service providers to specify their trust relationships with other providers and to examine the set of service providers eligible for information sharing, possibly modifying it by adding and removing individual service providers. Moreover, the framework includes a User Modeling Agent that coordinates the exchange of customer information according to the network of declared trust relationships: when a service provider requests information about a customer, the User Modeling Agent merges the information provided by the trusted parties into a user model ready to be exploited for personalization purposes.

From the viewpoint of trust management, our framework enables the service provider administrator to select partners for information sharing both at the individual level and at the class level (on the basis of their features). More generally, our framework has the following advantages:

- Service providers are supported in the information sharing by a trusted third party (the User Modeling Agent).
- Service providers do not need to modify the core of their applications when they register for information sharing. In fact, each application may continue to exploit its own personalization system: the application may personalize the interaction with an individual customer by exploiting its local user model, the model provided by the User Modeling Agent, or it may integrate the two models.
- A service provider not equipped with its own user modeling system may question the central User Modeling Agent when it needs information about a customer and exploit the returned information for personalization purposes.

In this paper, we will focus on the Trust Management System, which provides the basis for the customer information propagation, and we will only sketch the main aspects of the User

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

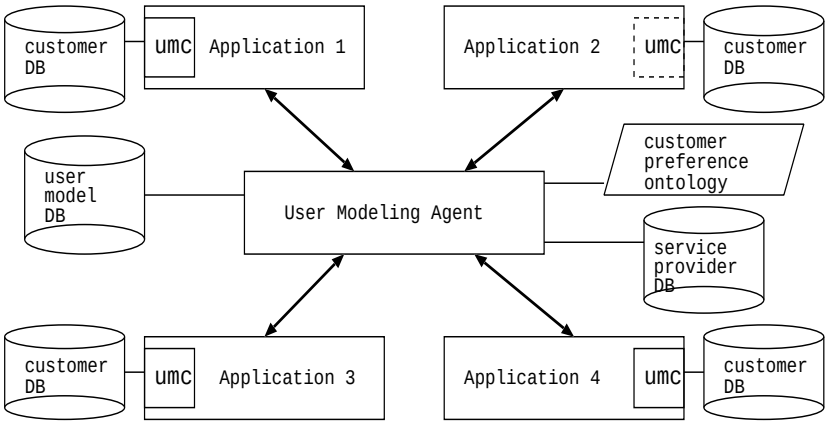


Fig. 1. Framework Architecture.

Modeling Agent. The rest of this presentation is organized as follows: Section II outlines some basic issues concerning customer information sharing between heterogeneous applications. Section III describes the architecture of our framework and the regulation of the propagation of information from service provider to service provider. Section IV describes the management of trust relationships between service providers. Section V compares our proposal to related work. Section VI discusses possible extensions to our work and closes the paper.

II. Background

In the development of a service supporting customer information sharing between applications the following issues are relevant:

1) In the propagation of the information, privacy preferences have to be taken into account [7], [8]. For instance, a customer might want to make her personal data available only to the service providers she is interacting with, she might allow the propagation to providers belonging to restricted categories, such as book sellers, or she might restrict the propagation of her personal data to service providers conforming to privacy policies [9].
2) Ontology mapping issues have to be addressed in order to enable the propagation of information in an open environment.
3) The information collected by each application has to be propagated to other applications according to specific trust relationships. For instance, a service provider may impose accessibility restrictions on the information it collects, or it may be interested in receiving information from selected sources. For instance, a book seller might want to share information only with other book sellers and to ignore data acquired by music sellers. Moreover, it might not want to share information with some particularly untrusted book sellers.

The issues described in the first two items above are addressed in initiatives that are proposing standard solutions to be adopted by the applications. For instance, the Platform for Privacy Protection initiative of the W3C Consortium (P3P, [9]) is defining a standard representation language for the specification of privacy preferences and privacy management policies. The ultimate goal is to enable the specification of privacy preferences at the customer side (e.g., in the user agent of a Web browser) and the automated verification of the acceptability of the policies adopted by the web sites the user is visiting.

As far as the binding task is concerned (item 2 above), this is very a complex issue and has usually been addressed by adopting ad hoc solutions. However, the current attempts to solve this issue tend to propose standard ontologies for the representation of user information and preferences, with the goal to make applications exploit a uniform representation language for the description of their users. Specifically, in the P3P proposal, a user information ontology has been defined to describe basic customer data such as contact addresses, socio-demographic information and clickstream data. Moreover, in order to enable service providers to declare the kind of user preference they want to collect, the ontology can be extended with additional concepts. This means that standard preference ontologies can be developed for the main sales domains, similar to the representation of products in the RosettaNet initiative [10]. Furthermore, in the research about Semantic Web, complex ontologies are being proposed to represent rich user preference information; e.g., see [11], [12].

In the rest of this paper, we will focus on the third issue, which has been relatively unexplored. For simplicity, we will describe the user preferences in a trivial <feature, value> representation language, as the focus of this presentation is in the controlled propagation of information, not on the kind of exchanged data. Moreover, given the trend towards the standardization of ontologies, we will assume that the User Modeling Agent adopts a general ontology and that the applications registered for information sharing adopt a subset of that ontology, without handling ontology mapping issues. Finally, we will assume that customers do not impose any restrictions on the propagation of their personal data although we believe that our framework can be extended to manage privacy preferences by conforming to the P3P

```
Customer preferences:
  Books:
    history: (Int:[0,1], Confidence:[0,1]);
    science: (Int:[0,1], Confidence:[0,1]);
    scienceFiction: (Int:[0,1],Confidence:[0,1]);
    literature: (Int:[0,1], Confidence:[0,1]);
    ...
  Music:
    rock: (Int:[0,1], Confidence:[0,1]);
    jazz: (Int:[0,1], Confidence:[0,1]);
    disco: (Int:[0,1], Confidence:[0,1]);
    ...
```

Fig. 2. Portion of the Customer Preference Ontology.

platform specifications without major problems.

III. Architecture of our Customer Information Sharing Framework

Before describing the Trust Management System, it is worth sketching the architecture of the customer information sharing service which controls the propagation of information between registered service providers.

We have designed a User Modeling Agent devoted to coordinating the propagation of information between service providers, by taking their mutual trust relationships into account. The architecture supports the cooperation between heterogeneous applications, that may (may not) exploit a local user modeling component for the management of the customer profiles. Our User Modeling Agent is also responsible for reconciling the information provided by the service providers, by merging the alternative user models in order to generate the preference information needed by each individual application.

Figure 1 shows the high-level architecture of our framework. The figure shows a scenario where four service providers have registered for information sharing. Each application has a local database (*customer DB*) storing its own customer information and may exploit a local user modeling component (*umc*) to manage the user models. The local user modeling component is shown as a small box within the application rectangle; in the example, three applications have their own component (plain boxes), while one application (*application 2*) only exploits the preference information provided by the User Modeling Agent (dashed box).

The *customer preference ontology* defines the representation of preferences adopted in the User Modeling Agent and in the registered applications. As the User Modeling Agent must be able to integrate the information retrieved from different service providers, the ontology is organized in subparts describing the customer preferences in different domains, e.g., the sales of books, music, and services such as insurance agencies. Figure 2 shows a portion of the ontology related to the books and music sales domains. For each concept:

- The preference is represented as an interest degree that takes real values in [0, 1]. The 0 value denotes total lack of interest, while 1 denotes maximum interest.
- The confidence degree describes the reliability of the estimated interest: it is a real number in [0, 1], where 0 denotes total lack of confidence (no evidence about the preference is available) and 1 denotes absolute confidence in the estimation.
  The confidence degree enables the User Modeling Agent to correctly integrate the information provided by the applications. In fact, each application is likely to provide evidence about few user preferences, leaving the other unknown and this is represented by setting the confidence to 0.

Similar to the approach adopted in other application domains (e.g., TV recommenders [13], [14]), the ontology is organized in a hierarchical way, as a tree of concepts, which supports a rather straighforward propagation of the interest and confidence information between concepts.

At the core of the architecture, the *User Modeling Agent* manages the registered applications. The agent exploits a *service provider DB* storing information about the applications registered for customer information sharing. As described in the following section, a registration service (the Trust Management System) enables service providers to join the set of registered applications and to specify trust relationships with the other service providers. The User Modeling Agent exploits these relationships to constrain the propagation of customer information within the pool of registered applications.

When an application SP invokes the User Modeling Agent to acquire the preferences of a customer C, the agent selects the trusted applications and requests C's user models. Then, the agent synthesizes the customer preferences, it generates the user model including the information needed by SP and sends the information to SP. The exchange of data between service providers and User Modeling Agent is carried out by means of SOAP messages storing the user models.

In order to merge the user preferences collected by different providers, the identifiers selected by the customer when registering for the various services have to be related to one another. As the identification of the customer across different applications is a very complex issue, global identifiers have been proposed, e.g., in the Microsoft Liberty Alliance project [15]. In our work, we adopt the global identifier approach: the User Modeling Agent maintains a central *user model DB* storing, for each customer, the identification data she entered when she registered at a service provider's site. In the absence of a global identifier (e.g., for customers who did not accept a global passport and who registered for a service before it registered for information sharing), multiple identities are treated as different customers.

IV. Trust Management

The trust relationships are specified by the service provider administrators when they register for customer information sharing and are stored in the *service provider DB* managed by the Trust Management System and exploited by the User Modeling Agent at information propagation time.

Similar to policy-based approaches [16], we adopted a concise and declarative representation of trust relationships based on the specification of service provider features and of

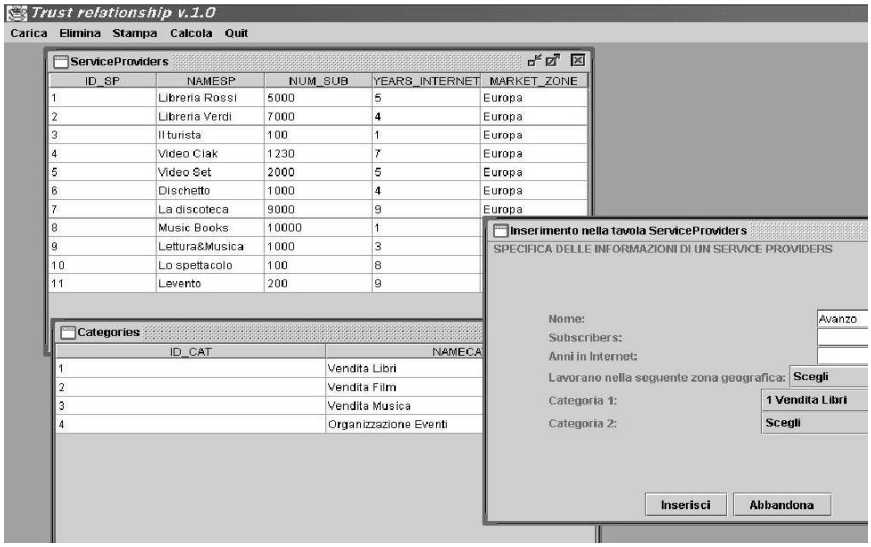


Fig. 4. Trust Management System: Introduction of Information about a Service Provider.

conditions on the propagation of data. However, we adopted an explicit trust management technique, based on the analysis of trusted party lists, instead of automatically providing access certificates. The reason is that, in an open e-commerce environment, the set of service providers having the right to receive the information collected by a service provider cannot be defined by means of necessary and sufficient conditions. More specifically, restriction conditions can be defined to select groups of entities eligible for information sharing. However, a one-by-one analysis is needed to revise the groups according to the requirements of the individual service provider, who may want to exclude candidates for business purposes. Notice that the evaluation of trust at the instance level is important not only because the customer information is very precious, but also because its dissemination is regulated by severe privacy rules that make both the service provider (as collector of personal data) and the middle agent(s) supporting information sharing responsible for any misuse of such data. Thus, each service provider administrator must be enabled to inspect and modify (by overriding general feature-based trust relationships) the list of parties to which the information sharing framework propagates the data.

```
Identification data:
   ID: SP1;
   Name: BookLand;
   URL: http://www.bookLand.com;
   ...
Categorization: bookSeller;
Features:
   NumberOfSubscribers: 3000;
   ...
Trust relationships:
   TAKE: {(bookSeller OR movieSeller) AND
           nrOfSubscribers>1000, 1), ...}
   NOT-TAKE: {musicSeller, ...}
   GIVE: {bookSeller OR movieSeller, ...}
   NOT-GIVE: {insuranceAgent, ... }
```

Fig. 3. Sample Service Provider Descriptor.

A. Description of Service Providers

Each service provider is described by the following data, stored in a table of the *service provider DB* (see Figure 3 for a sample descriptor):

1) Identification data: name, address, social security number, ...

2) Categorization: each service provider is classified in one or more categories. A taxonomy specifies the service provider types handled by the User Modeling Agent; e.g., *bookSeller*, *musicSeller* and *insuranceAgent*.

3) Features: number of subscribers, Quality of Service, ...

4) Trust relationships. These relationships are stored in separate fields, each one including one or more (alternative) relationships, separated by commas:

- *TAKE*: Conditions for the selection of the applications from which the service provider wants to receive customer information and degree of trust in the information.
  - The conditions are well-formed boolean expressions and may include categories and restrictions on the values of the service provider features.
  - The degree of trust is a real value in (0,1]. The value 1 denotes absolute trust, while values near to 0 denote lack of trust, i.e., the provider ignores the information coming from those providers.
- *NOT-TAKE*: Conditions for the selection of the applications from which the service provider does not want to receive information. These conditions have the same format as the previous ones but the trust degree is omitted because it is by default equal to 0.
- *GIVE*: Conditions imposed by the service provider on the dissemination of customer information to other service providers. These conditions are well-formed boolean

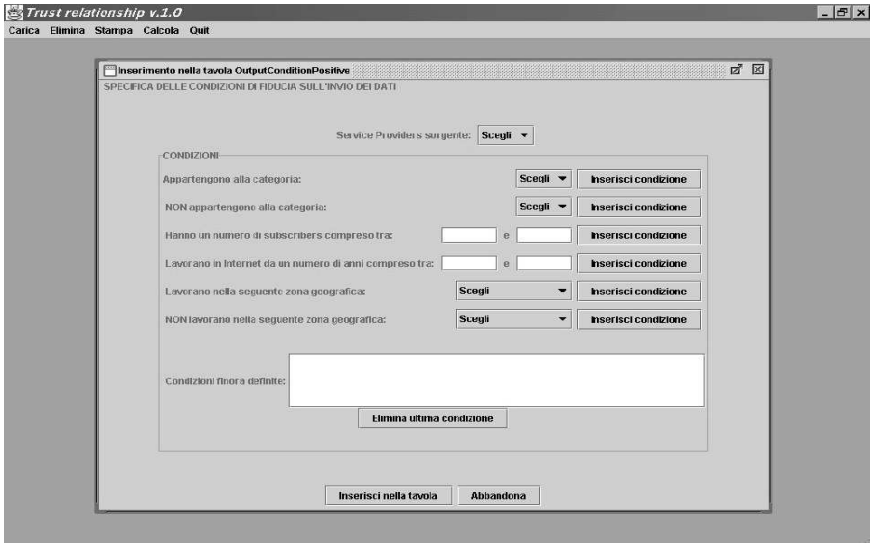


Fig. 5. Trust Management System: Definition of Trust Relationships.

expressions and may include categories and restrictions on the values of the service provider features.

- *NOT-GIVE*: Conditions for the selection of applications to which the service provider does not want to deliver its own customer information. The conditions have the same format as the *GIVE* ones.

Notice that by defining *TAKE* and *NOT-TAKE* relationships, the service provider assesses the usefulness and the quality of the preference estimates that might be provided by the other applications. For instance, the *TAKE* field of the descriptor of Figure 3 specifies that the *BookLand* service provider only trusts the information provided by book sellers and movie sellers having at least 1000 subscribers. Moreover, the *NOT-TAKE* field specifies that no feedback about customer preferences has to be taken from music sellers.

B. Management of the Service Provider Descriptors

The descriptor of a service provider is filled in by its administrator at registration time. In order to facilitate this activity, we have developed a Trust Management System that offers a graphical user interface for the introduction of the features of the service under specification and the conditions of the trust relationships. This system stores information about all the registered service providers and manages the network of trust relationships by summarizing them, in order to support an efficient propagation of information between applications.

Figure 4 shows a portion of the user interface of the Trust Management System, concerning the introduction of information about an individual service provider. At the right side, the screenshot shows a portion of the registration form (”Nome” - name; ”Subscribers”, ”Anni in internet” - years of activity in internet, etc.). At the left side, a window shows the list of the registered service providers. Figure 5 shows another page, supporting the definition of trust relationships. The service provider administrator is guided in the definition of trust conditions that specify which applications can use the customer information provided by the service under specification.[1] In particular, the system enables the administrator to include/exclude specific categories of applications, require a minimum/maximum number of customers, or number of years of activity in internet, and include/exclude specific marketing areas. Similar pages are generated to support the definition of conditions on the retrieval of customer information from other service providers. The system assists the administrator in the specification of trust relationships by performing consistency checks on the defined trust conditions. For instance, the same condition cannot be specified both in the *GIVE* and the *NOT-GIVE* fields.

Given the trust relationships specified by the administrator (*GIVE*, *NOT-GIVE*, *TAKE* and *NOT-TAKE* fields of the descriptor), the Trust Management System generates three trust relationship lists, *GIVE-IND*, *NOT-GIVE-IND* and *TAKE-IND*, by analyzing the descriptors of the other service providers (e.g., see Figure 6). Specifically:

- The *GIVE-IND* list is generated by selecting the service providers that satisfy at least one *GIVE* condition, that do not satisfy any *NOT-GIVE* condition and that do not trust any untrusted service provider (i.e., the transitive closure of the *GIVE-IND* relationship does not include any untrusted provider).
- The *NOT-GIVE-IND* is generated by subtracting the ap-

[1]We assume that the service provider administrator fills in the forms by providing correct data. The provision of false identities is a legal problem that cannot be handled at the technical level.

```
ID: SP1;
Trust relationships:
   TAKE-IND: {(SP2, 1), (SP10, 0.5), (SP45, 0), ...}
   GIVE-IND: {SP2, SP10, ...}
   NOT-GIVE-IND: {SP3, SP8, ...}
```

Fig. 6. Trust Relationships between Individual Service Providers.

TABLE I
SUMMARY OF TRUST RELATIONSHIPS FOR INFORMATION SHARING.

TRUST TABLE

Destination	Source	Filter
SP1	SP2	1.0
SP1	SP3	0.0
SP1	SP4	0.6
SP2	...	...

plications in the *GIVE-IND* list from the complete set of registered applications.

- The *TAKE-IND* list includes all the registered service providers and specifies, for each one, the level of trust in the customer information they provide. This is a real number in [0, 1] and has the same meaning adopted in the *TAKE* field of the descriptor. Untrusted service providers have a 0 trust level.
  The level of trust associated to service providers is computed as follows: each service provider that satisfies at least one *TAKE* condition, does not satisfy any *NOT-TAKE* condition and does not trust any service provider satisfying a *NOT-TAKE* condition has level equal to the minimum value associated to the provider by means of the *TAKE* conditions. All the other service providers receive a level of trust equal to 0. For instance, consider the *TAKE* and *NOT-TAKE* conditions reported in the descriptor of *SP1* in Figure 3. A service provider classified both as a *bookSeller* and *movieSeller* would receive a 0 level of trust because it satisfies a condition reported in the *NOT-TAKE* field. Notice that these conditions are evaluated in a pessimistic way (minimum value) because they are associated to the quality and usefulness of the customer information that is going to be received by a service provider. If some characteristics of an application have the potential to introduce noisy data, or irrelevant data, the quality of its contribution is reduced.

The generation of these lists is aimed at presenting details about the trusted and untrusted applications registered for customer information sharing. By exploiting the Trust Management System, the administrator of a service provider SP may inspect and modify (also by overriding the trust relationships that have been defined) the lists of service providers receiving information from SP or providing information to SP. Therefore, the administrator may periodically check the set of registered service providers and update the lists to include and/or exclude new applications. This is important for two reasons: first, the administrator needs to treat individual service providers in a special way (e.g., to trust a provider belonging to a generally untrusted category and vice versa). Second, as time passes, the set of registered applications may change: other service providers may modify their descriptors (a book seller might start to sell music, as well) and new service providers may register.

C. Summarizing Trust Relationships

Although the *GIVE-IND*, *NOT-GIVE-IND* and *TAKE-IND* lists provide complete information about the trust relationships between pairs of service providers, they fail to support the efficient propagation of the user models at run-time. In fact, each time the User Modeling Agent has to propagate the customer information from a service provider SP_j to another one SP_i, the agent should check:

- whether SP_i satisfies the *GIVE* restrictions specified by SP_j, and
- to which extent SP_i is trusting the information provided by SP_j (trust level in SP_i's *TAKE-IND* restrictions).

In order to support the efficient propagation of information between service providers, we have decided to pre-compile the trust relationships: in the *service provider DB* a *TRUST* table summarizes the trust relationships existing between all the registered service providers; see Table I. The table abstracts from the details of the *GIVE* and *TAKE* relationships, which represent unilateral viewpoints on the propagation of information, and describes the weight of the information provided by the various applications in the generation of the user model for each service provider. More specifically, in the table:

- The *Destination* column represents the service provider receiving the information.
- The *Source* column denotes the service provider that should provide the information.
- The *Filter* column includes real values in [0, 1] and specifies to which extent the information provided by the source application must be taken into account when integrating the customer's preferences to be sent to the destination application. As usual, if the filter takes a value close to 1, this means that the information provided by the source has to be propagated to the destination. Moreover, if the filter is 0, no information has to be propagated.[2]

The *TRUST* table is generated and revised off-line by our Trust Management System. The revision process is launched periodically, in order to update the table according to the changes in the pool of registered service providers; e.g., new registrations, removals, changes in the descriptors.

D. Run-time Customer Information Sharing

The idea behind customer information sharing is that, when an application SP_i invokes the User Modeling Agent to retrieve information about a customer C's preferences, the Agent exploits the *TRUST* table to select the service providers to be contacted. Only the applications whose filter is positive are considered in the generation of the user model and the value of the filter is exploited to merge the preference estimates provided by the applications. Specifically, the User Modeling Agent should retrieve C's preferences from the other registered applications according to the following principles:

[2]The filter takes the 0 value if the destination application is in the *NOT-GIVE-IND* list of the source or if the level of trust between destination and source in the *TAKE-IND* list is 0. Otherwise, the filter takes the trust level specified in the *TAKE-IND* list and thus corresponds to how strongly the destination application trusts the quality of information provided by the source.

1) The bidirectional trust relationships between SP_i and the other applications stored in the *TRUST* table guide the identification of the subset of applications to be considered by the User Modeling Agent and specify SP_i's trust in the provided information (*Filter* field of the table).
2) Within the set of selected applications, only those having C as a registered customer have to be considered.
3) The fact that C has registered in an application SP_j does not mean that SP_j has already acquired any preference information about C.

In order to take the first two factors into account, the User Modeling Agent consults the *TRUST* table to select a set of candidate applications and it queries the *user model DB* to identify the applications that have C as a registered customer. The agent exploits the *Filter* information stored in the *TRUST* table to tune the influence of their customer information in the generation of the user model. The trust level has to be taken into account when combining the contribution of the applications to the generation of the model. Ideally, the trusted applications should stronfly influence the generation of the user model, while the less trusted ones should marginally influence the process.

As far as the third factor is concerned, the contribution to the generation of the user model carried by each application A must be also tuned according to the confidence of A in the provided information (confidence degree assigned by the application, given the amount of evidence about the customer at disposal). As specified in the *customer preference ontology*, each customer preference has an associated confidence degree, describing the reliability of the information, i.e., whether there is evidence about the provided information or not.

We have selected a weighted addition formula to combine the information about the customer preferences provided by the applications invoked by the User Modeling Agent. For each requested preference P, the agent combines the interest estimates provided by the trusted applications SP_j as follows:

$$Int\_P = [\sum_{j=1}^{n} MIN(trust_{ij}, conf_j) * Int\_P_{SP_j}]/\delta \quad \textbf{(i)}$$

where $\delta = \sum_{j=1}^{n} MIN(trust_{ij}, conf_j)$

In the formula:

- Int_P is the interest value for P generated by the User Modeling Agent, given the contribution of the invoked service providers;
- n is the number of invoked applications;
- $trust_{ij}$ represents how strongly SP_i trusts SP_j (i.e., it is the *Filter* associated to SP_j in SP_i's *TRUST* table);
- $conf_j$ denotes the confidence associated to the interest by SP_j;
- δ is applied to normalize Int_P in [0, 1].

For each invoked application SP_j, the contribution to the computation of the interest value for P is thus weighted according to SP_i's trust level in SP_j and to SP_j's confidence in the estimated preference. The minimum of the two values is exploited to define the impact of the estimate according to a *Fuzzy AND*.

The formula (i) enables the User Modeling Agent to merge the information provided by the various applications according to the service providers' requirements, but also on the basis of a subjective evaluation of the reliability of the provided information. As confidence values are associated to individual preferences, they may change from invocation to invocation, depending on the observations of the customer behavior carried out by the applications.

V. Discussion and Related Work

Some policy-based approaches [16] have been proposed to manage the trust relationships between applications and to regulate the access to shared resources and data. For instance, the framework described by Kagal and colleagues [17] supports the automatic and distributed management of access rights to resources and information. The framework is implemented in a language supporting the specification of deontic concepts, such as rights and prohibitions to perform actions. Suitable *security agents* apply the defined policies to grant or cancel access and delegation rights to groups of agents in a controlled way, by delivering certificates.

Indeed, the purpose of our work differs from Kagal et al.'s work [17], [18], in relation to the type of rights we aim at regulating.

- Kagal et al. control different types of actions that the applications may perform on the resources, such as "reading", "writing", "executing" a file. Instead, we are only concerned with "reading" rights.
- At the same time, however, our framework enables the applications to define restrictions on the type of information they want to receive and controls the information flow accordingly.

VI. Conclusions and Future Work

We have presented a framework for customer information sharing that supports the controlled propagation of information among service providers. Our framework includes a registration service (the Trust Management System) exploited by service providers in order to join the pool of applications sharing information with one another. Moreover, the framework includes a User Modeling Agent that controls the information flow between applications and reconciles the information provided by the various service providers in order to generate the preference information needed by the requesting application.

We have developed a proof-of-concept implementation of the customer information sharing framework that supports the service provider administrator in the introduction of information about service providers and trust relationships. The framework is based on Java and uses JDBC technology to connect to the database where the trust information is stored in the corresponding tables.

Our framework handles bidirectional trust relationships to address the fact that service providers may want to:

- control the dissemination of information by imposing restrictions on the service providers that will receive data;

- impose restrictions on the service providers from which they want to receive information, in order to filter out irrelevant information sources available through the information sharing service.

As already specified, we have left the management of the customers' privacy preference aside, assuming that the customers do not impose restrictions on the dissemination of their personal data. In our future work, we will extend our proposal to the treatment of customer preferences, which can be done without major architectural changes. Specifically, taking the P3P specifications into account, the *user model DB* handled by the User Modeling Agent could be extended to store the individual customer's privacy preferences. Moreover, the overall service should require that, at registration time, the service providers publish their own P3P privacy policies. Having this information available, the User Modeling Agent could propagate the customer information between applications by taking into account not only the trust relationships, but also possible constraints imposed by the individual customer.

In our future work we will analyze the ontology issues concerning the binding between the service providers' local representations of the customer information and the one adopted in the customer information sharing service. Our goal is the development of an ontology binding tool supporting the administrator of a service provider to define the correspondences between the customer preferences defined in the application and those exploited by the main user model for information sharing.

In our future work we will also study the possibility of distributing the information sharing service for efficiency and reliability purposes. For instance, an interesting solution to study is a distributed User Modeling Agent in the line of peer-to-peer sharing networks, where the applications directly contact other trusted applications to gather customer profile information.

REFERENCES

[1] P. Resnick and H. Varian, Eds., *Special Issue on Recommender Systems*. Communications of the ACM, 1997, vol. 40, no. 3.

[2] J. Fink and A. Kobsa, "A review and analysis of commercial user modeling servers for personalization on the World Wide Web," *User Modeling and User-Adapted Interaction, Special Issue on Deployed User Modeling*, vol. 10, no. 2-3, pp. 209–249, 2000.

[3] M. Maybury and P. Brusilovsky, Eds., *The adaptive Web*. Communications of the ACM, 2002, vol. 45, no. 5.

[4] R. Burke, "Hybrid recommender systems: survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 289–322, 2002.

[5] M. O'Connor, D. Cosley, J. Konstan, and J. Riedl, "PolyLens: a recommender system for groups of users," in *Proc. European Conference on Computer Supported Cooperative Work (ECSCW 2001)*, Bonn, Germany, 2001.

[6] D. Billsus and M. Pazzani, "A personal news agent that talks, learns and explains," in *Proc. 3rd Int. Conf. on Autonomous Agents (Agents '99)*, Seattle, WA, 1999, pp. 268–275.

[7] A. Kobsa, "Personalized hypermedia and international privacy," *Communication of the ACM*, vol. 45, no. 5, pp. 64–67, 2002.

[8] A. Kobsa and J. Schreck, "Privacy through pseudonymity in user-adaptive systems," *ACM Transactions on Internet Technology*, vol. 3, no. 2, pp. 149–183, 2002.

[9] W3C, "Platform for Privacy Preferences (P3P) Project," http://www.w3.org/P3P/.

[10] "RosettaNet ebusiness standards for the Global Supply Chain," http://www.rosettanet.org/RosettaNet/Rooms/DisplayPages/LayoutInitial.

[11] UbisWorld, "Ubiquitous User, Modeling for Situated Interaction," http://www.u2m.org/.

[12] D. Heckmann, "A specialised representation for ubiquitous computing," in *Proc. Workshop on User Modelling for Ubiquitous Computing*, Johnstown, PA, 2003, pp. 26–28.

[13] L. Ardissono, C. Gena, P. Torasso, F. Bellifemine, A. Chiarotto, A. Difino, and B. Negro, "Personalized recommendation of TV programs," in *LNAI 2829. AI*IA 2003: Advances in Artificial Intelligence*. Berlin: Springer Verlag, 2003, pp. 474–486.

[14] L. Ardissono, C. Gena, P. Torasso, F. Bellifemine, A. Difino, and B. Negro, "User modeling and recommendation techniques for personalized Electronic Program Guides," in *Personalized Digital Television. Targeting Programs to Individual Users*. Kluwer Academic Publishers, 2004.

[15] Liberty Alliance Developer Forum, "Liberty alliance project specifications," http://www.projectliberty.org/specs/, 2004.

[16] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, no. 4, pp. 333–360, 1994.

[17] L. Kagal, S. Cost, T. Finin, and Y. Peng, "A policy language for pervasive systems," in *Proc. 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks*, Lake of Como, Italy, 2003.

[18] L. Kagal, T. Finin, and A. Joshi, "A policy based approach to security for the Semantic Web," in *Proc. 2nd Int Semantic Web Conference (ISWC2003)*, Sanibel Island, FL, 2003.

A Game-Theoretic Operational Semantics for the DALI Communication Architecture

Stefania Costantini Stefania Costantini Alessia Verticchio
Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito, I-67010 L'Aquila - Italy
{stefcost,tocchio}@di.univaq.it

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

I. Introduction

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

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

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

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

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

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

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

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

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

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

II. The DALI language

DALI [2] [4] is an Active Logic Programming language designed for executable specification of logical agents. A DALI agent is a logic program that contains a particular kind of rules, reactive rules, aimed at interacting with an external environment. The environment is perceived in the form of external events, that can be exogenous events, observations, or messages by other agents. In response, a DALI agent can perform actions, send messages, invoke goals. The reactive and proactive behavior of the DALI agent is triggered by several kinds of events: external events, internal, present and past events. It is important to notice that all the events and actions are timestamped, so as to record when they occurred. The new syntactic entities, i.e., predicates related to events and proactivity, are indicated with special postfixes (which are coped with by a pre-processor) so as to be immediately recognized while looking at a program.

The external events are syntactically indicated by the postfix *E*. When an event comes into the agent from its "external world", the agent can perceive it and decide to react. The reaction is defined by a reactive rule which has in its head that external event. The special token $:>$, used instead of $:-$, indicates that reactive rules performs forward reasoning. The agent remembers to have reacted by converting the external event into a *past event* (time-stamped). The set of past events in a way constitutes the set of the new beliefs that the agent has collected from her interaction with the environment.

Operationally, if an incoming external event is recognized, i.e., corresponds to the head of a reactive rule, it is added into a list called EV and consumed according to the arrival order, unless priorities are specified.

The internal events define a kind of "individuality" of a DALI agent, making her proactive independently of the environment, of the user and of the other agents, and allowing her to manipulate and revise her knowledge. An internal event is syntactically indicated by the postfix *I*, and its description is composed of two rules. The first rule contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction, specified in the second rule, may happen.

Internal events are automatically attempted with a default frequency customizable by means of directives in the initialization file. The user's directives can tune several parameters: at which frequency the agent must attempt the internal events; how many times an agent must react to the internal event (forever, once, twice,...) and when (forever, when triggering conditions occur, ...); how long the event must be attempted (until some time, until some terminating conditions, forever).

When an agent perceives an event from the "external world", it does not necessarily react to it immediately: she has the possibility of reasoning about the event, before (or instead of) triggering a reaction. Reasoning also allows a proactive behavior. In this situation, the event is called present event and is indicated by the suffix *N*.

Actions are the agent's way of affecting her environment, possibly in reaction to an external or internal event. In DALI, actions (indicated with postfix *A*) may have or not preconditions: in the former case, the actions are defined by actions rules, in the latter case they are just action atoms. An action rule is just a plain rule, but in order to emphasize that it is related to an action, we have introduced the new token $:<$, thus adopting the syntax $action :< preconditions$. Similarly to external and internal events, actions are recorded as past actions.

Past events represent the agent's "memory", that makes her capable to perform future activities while having experience of previous events, and of her own previous conclusions. Past events are kept for a certain default amount of time, that can be modified by the user through a suitable directive in the initialization file.

III. DALI Communication Architecture

A. The Architecture

The DALI communication architecture (Fig.1) consists of three levels. The first level implements the DALI/FIPA communication protocol and a filter on communication, i.e. a set of rules that decide whether or not receive or send a message. The second level includes a meta-reasoning layer, that tries to understand message contents, possibly based on ontologies and/or on forms of commonsense reasoning. The third level

consists of the DALI interpreter.

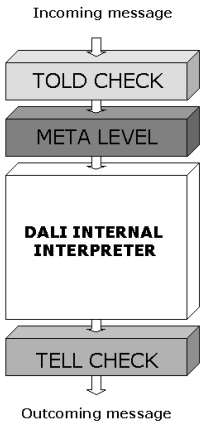


Fig. 1. The communication architecture of a DALI agent

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

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

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

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

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

Whenever a message is received, with content part *primitive(Content,Sender)* the DALI interpreter automatically looks for a corresponding *told* rule, which is of the form:

$$\begin{array}{l} told(Sender, primitive(Content)) : - \\ \qquad constraint_1, \ldots, constraint_n. \end{array}$$

where $constraint_i$ can be everything expressible either in Prolog or in DALI. If such a rule is found, the interpreter attempts to prove $told(Sender, primitive(Content))$. If this goal succeeds, then the message is accepted, and $primitive(Content))$ is added to the set of the external events incoming into the receiver agent. Otherwise, the message is discarded.

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

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

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

$$\begin{array}{l} tell(Receiver, Sender, primitive(Content)) : - \\ \qquad constraint_1, \ldots, constraint_n. \end{array}$$

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

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

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

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

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

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

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

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

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

B. Related Approaches

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

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

IV. Operational semantics

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

V. An example of application of the DALI communication filter

We will now demonstrate how the filter level works by means of an example, that demonstrates how this filter is powerful enough to express sophisticated concepts such as updating the level of trust. Trust is a kind of social knowledge and encodes evaluations about which agents can be taken as reliable sources of information or services. We focus on a practical issues: how the level of Trust influences communication and choices of the agents.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

VI. Concluding Remarks

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

References

[1] J. Barklund, S. Costantini, P. Dell'Acqua e G. A. Lanzarone, *Reflection Principles in Computational Logic*, Journal of Logic and Computation, Vol. 10, N. 6, December 2000, Oxford University Press, UK.

[2] S. Costantini. Towards active logic programming. In A. Brogi and P. Hill, editors, *Proc. of 2nd International Workshop on component-based Software Development in Computational Logic (COCL'99)*, PLI'99, (held in Paris, France, September 1999), Available on-line, URL
http://www.di.unipi.it/
brogi/ResearchActivity/COCL99/proceedings/index.html.

[3] S. Costantini. Many references about DALI and PowerPoint presentations can be found at the URLs: http://costantini.di.univaq.it/pubbls_stefi.htm and http://costantini.di.univaq.it/AI2.htm.

[4] S. Costantini and A. Tocchio, *A Logic Programming Language for Multi-agent Systems*, In S. Flesca, S. Greco, N. Leone, G. Ianni (eds.), *Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002*, (held in Cosenza, Italy, September 2002), LNAI 2424, Springer-Verlag, Berlin, 2002.

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

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

[7] R. A. Kowalski, *How to be Artificially Intelligent - the Logical Way*, Draft, revised February 2004, Available on line, URL http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html.

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

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

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

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

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

On the use of Erlang as a Promising Language to Develop Agent Systems

Antonella Di Stefano, Corrado Santoro
University of Catania - Engineering Faculty
Department of Computer and Telecommunication Engineering
Viale A. Doria, 6 - 95125 - Catania, Italy
EMail: {adistefa,csanto}@diit.unict.it

Abstract— **About 70% of agent programming platforms are written using Java^{TM}. However, the fact that many other platforms are based on *ad-hoc* programming languages, invented by the platform's authors themselves, suggests that something is missing, in Java^{TM}, to fulfill the requirements of agent applications. So the question is: What is the language that best fits the model of an autonomous software agent? We deal with such an issue in this paper, by deriving an abstract model for agents and proposing some parameters that allow to understand if a programming language and environment can be considered "best suited" for the development of agent systems. As a result, the paper evaluates Erlang, a functional language that presents some interesting characteristics for engineering and implementing agent-based applications. An Erlang-based platform, called eXAT and developed by the authors, is then presented. Finally, a comparison with a Java^{TM}-based approach explains why, in the authors' opinion, this language cannot be considered a good choice for the implementation of agent systems.**

I. Introduction

About 70% of agent programming platforms are written using Java^{TM} and many other are based on *ad-hoc* programming languages, invented by the platform's authors themselves. In many other cases, Java^{TM} platforms integrate additional tools, aiming at adding to agents some capabilities (e.g. "intelligence") that are missing in the original platform. These tools are, in general, based on programming languages, approaches and models different than those of Java^{TM}. For example, rule-production systems often used in conjunction with Java^{TM} platforms, such as JESS [1] or similar [2], [5], are based on a declarative/logic language. Another example is JADEX [29], the BDI extension for JADE [12], which forces the agent programmer to deal also with XML and OQL. In other cases, as in JACK [8], additional "agent-specific" keywords are added in the Java language, in order to make it able to better support agent-related concepts.

All of the statements above suggest that something is missing, in Java^{TM}, to fulfill the requirements of agent applications. The issue is that, even if Java^{TM} is widely known and easy to learn and use, it is not a "silver bullet" that magically solves any software application developing problem; like any problem domain must be faced and solved using an implementation approach—and language and tools—that **best fits** the specific domain, the natural questions for software agent implementation are: Which language should I have to use to realize my agent system? What is the language that best fits the model of an autonomous software agent? We believe that, notwithstanding the huge number of agent platforms today available, the questions above are not completely solved.

We agree that finding valid answers to these questions is not a simple task: an in-depth analysis is required, aiming at evaluating any approach currently proposed (both Java^{TM}-based and not), in order to (above all) experimentally verify the adherence of the platform or language to the agent model[1]. However, given that many platforms are Java^{TM}-based, we think that a good starting point is trying to evaluate this language with respect to other (and possibly new) alternative approaches.

According to such concepts, we provide, in this paper, the reasons that led us to consider Erlang [11], [10], [6] as a promising language for the development of agent systems, such that it has been employed for the implementation of our agent platform, called eXAT [4], [30], [13], [15], [14].

The Erlang language has gained interest, in the field of agent system. It is cited in the "Agent Software" list of the Agentlink web site [3] and some of its characteristics (in particular message reception and matching semantics) have inspired some concepts and constructs of the APRIL agent programming language [26]. Moreover, a recent work [31] proposes an Erlang-based BDI tool.

Given this, in pursuing our objective, we first provide an abstract agent model, based on formalizing an agent behavior by means of a finite state machine, which is the most common recurring model for software agents. Since this model is treated as the building block to design and implement agents, it is extended by including object-orientation; this allows for considering the typical concepts of software engineering, such as modularization, reuse of code, etc., which are fundamental also in the development of agent systems.

Then, in order to allow an evaluation of programming languages, we derived some *parameters/requirements* able to measure the *ability* of each given languages in supporting agent system design and implementation. Such requirements are derived by considering not only general design rules for software systems but, above all, the adherence of the

[1] Just for reference, the "Agent Software" web page of the Agentlink web site [3] reports 129 platforms/languages, but there are many other that are not cited there.

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

language—in terms of constructs syntax and semantics, and library functions—to the agent model derived before.

On this basis, we present the Erlang language and provide the reasons that, in our opinion, make this language very promising for agent implementation. Finally, we give a brief overview of the eXAT platform, showing how the combination Erlang + services offered by eXAT fits our purposes, in accordance with the requirements provided before.

The paper is structured as follows. Section II provides the agent model based on finite-state machines. Section III lists and describes the requirements to be taken into account in evaluating a language for agent implementation. Section IV presents the Erlang language and its ability in meeting the requirements derived before. Section V briefly describes the eXAT platform and the services provided, giving also some code samples. Section VI explains why, on the basis of the derived requirements, Java^{TM} is not well-suited for agent development. Finally, Section VII reports our conclusions.

II. The Agent Model

Let us start from a "classical" agent definition that can be formalized with the following statement:

> *An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future* [22].

Let us recall that an agent *senses* the environment and *acts* onto it on the basis of the inputs and its internal *state*. Combining this concept with the above definition leads to model an agent (behavior) by means of a function like

$$(Act, NewState) = f(Sense, CurrState) \qquad (1)$$

Since, in the agents' world, *Act*, *NewState*, *CurrState* and *Sense* are discrete variables, functions like (1) call for the use of the *finite-state machine* (FSM) abstraction for the development of agent behavior. Even if the literature reports many models for automata in general [23], [27], [28], we believe that the use of maps or functions with multiple clauses is best suited for the specification of the agent behavior according to the model of (1). For example, if we would specify the behavior function $b(\cdot,\cdot)$ of an agent that continuously waits for the arrival of a message, unless a timeout occurs causing stopping of its activity (see Figure 1), we can use a representation like the following:

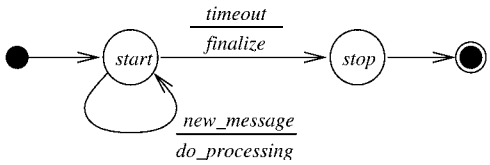


Fig. 1. A Simple Behaviour

Behavior Function b		
$b(new\_message, start)$	$=$	$(do\_processing, start)$
$b(timeout, start)$	$=$	$(finalize, stop)$

By choosing a proper syntax for specifying states, actions and environment senses (events), the use of functions like the one above represents a simple and flexible way to specify a FSM-based agent behavior.

A. Composing Behaviors

On the basis of the application to be realized, agent behaviors could be complex, thus requiring a FSM composed of a large number of states and transitions. Such a situation could be hard to handle during development stage. This means that the use of well-assessed software engineering techniques can help the programmer in tackling the problem of developing complex agent systems.

In fact, it can be noted that there are situations in which parts of an overall agent behavior could be reused in another different agent application. This happens, for example, when an agent implements one or more standard FIPA interaction protocols [21] (such as the contract-net [17], the request protocol [20], the English auction [19], etc.)[2].

In these cases, a natural way to improve agent engineering is to have ready-to-use components, each one implementing a simple and basic *sub-behavior*, which can be composed *in sequence*—to support serial activities—or *in parallel*—to support multiple concurrent activities (e.g. multiple interactions handling). Indeed, this approach is equivalent to using subroutines and co-routines in traditional imperative languages and it is also used in some agent platforms currently available, such as JADE [12].

Introducing the aspects above in the behavior model given by formula (1) implies the possibility of having the execution of a (sub-)behavior (or a set of sub-behaviours) as one of the possible agent actions. This new action does not provoke a direct effect on the environment, but only specifies how the agent has to behave in the immediate future.

As an example, if we want to specify the behavior $b_1(\cdot,\cdot)$ of an agent that:

- starts an English-auction [19] if it receives an "inform" message; or
- starts a Dutch-action [18] if a timeout occurs; and
- then, in both cases, stops,

we can write something like:

Behavior Function b_1		
$b_1(\text{“}inform\text{“}, start)$	$=$	$(behave(english\_auction), stop)$
$b_1(timeout, start)$	$=$	$(behave(dutch\_auction), stop)$

Behavior Function *english_auction*
/ definition of the behavior function for the English auction */*

[2]But reuse could be considered also for behavior patterns not strictly related to standard protocols.

Behavior Function *dutch_auction*
/ definition of the behavior function for the Dutch auction */*

B. Specializing and Extending Behaviors

The possibility of composing FSMs, according to the concepts illustrated in Section II-A, allows the "as-is" reuse of the same behavior in several multi-agent applications. However, in some cases, a behavior that has been previously designed could not be so general to allow its reuse for a specific purpose, but some changes need to be applied. In structured programming, where function/subroutine generalization is performed by means of parameters, specialization is done by assigning specific value to these parameters. With the object-oriented approach this process is made more flexible, since specialization is supported by means of (virtual) inheritance, which permits to add functionalities in the sub-class without requiring to change the original ancestor class' code. The same concepts, and in particular the virtual inheritance, can be applied in our context in order to support behavior extension, thus giving the possibility to reuse even a behavior not designed to be so general. This concept of behavior extension implies to specify a new behavior y', derived form y, that inherits from y all clauses, but changes some of them by:

- replacing an entire y's function clause with a new one;
- replacing one or more arguments of a y's function clause with new values;
- changing the action and/or the next state returned by a y's function clause;
- adding a new function clause.

Such an abstraction can be represented by labeling each function clause and then using labels, in the derived behavior, to indicate which function clause we are going to modify. This implies to write the behavior $b(\cdot,\cdot)$ introduced in the beginning of Section II as follows:

	Behavior Function b		
1	$b(new\_message, start)$	$=$	$(do\_processing, start)$
2	$b(timeout, start)$	$=$	$(finalize, stop)$

Thus, for example, if we want to write a new behavior b' that extends b and:

- performs a computation after the timeout has occurred; and then
- waits for another message before stopping,

we can write something like that:

	Behavior Function b'	:	extends b
$b.2$	$b(-,-)$	$=$	$(-, next)$
3	$b(new\_message, next)$	$=$	$(after\_finalize, stop)$

As the example shows, the specification of b' overrides clause 2 of b by changing the next state; then it adds a new clause (3) to wait for the message that triggers behavior finalization.

C. Discussion

The agent model described so far provides the possibility of *expressing*, *composing* and *extending* FSM-based behaviors. The notation we used for specifying behaviors is able to capture the basics of our model; it also shows the key features that ease the development of agents' behaviors, thus enabling *modularization* and *reuse* of components with a high degree.

Given these concepts, the next step is to find a programming language—or a platform or tool—that is able to concretely implement the provided abstractions without provoking loss of generality or power. Such a language has not only to comply with agent-related concepts but must also feature some general characteristics, derived from current requirements in engineering and implementing software systems. To this aim, the following Section will deal with such requirements, listing them and providing a short description highlighting why they are important.

III. Agent Development Requirements

A. General Requirements

a) Safety: The current trend in modern programming languages is to provide a *safe* environment for program execution. This means that the occurrence of situations, like dangling pointers, out of bound in array access, allocation errors, etc., which are the main causes of system crashes, are properly checked by the runtime environment and signaled to the program in execution e.g. by throwing an exception. Using safe languages undoubtedly improves the debugging of any software system. For this reason, safety is a characteristic preferable also in the development of agent systems.

b) Completeness: A good programming language (and environment) does not only have to provide an adequate syntax and semantics, but also a set of library functions to help the designer in facing the most recurring programming problems. Such libraries should include, for example, data handling (collections, lists, stacks, sets, etc.), user-interface (support for GUI), input/output (file handling and console I/O), etc.

c) Portability: Today, the most common computer platforms are either Windows- or Unix-based. Generally, the common trend is to use the Windows platform for graphical applications that require a sophisticated GUI, while Unix-based systems are employed to run network servers/services. Agent applications fall in both of the categories above: for example, we can have agents that interact with the user via a GUI and agents that offer their services on the net (e.g. agent-based web services). Since it is preferable to have a common environment for any platform, the language used to implement an agent system must be cross-platform. This also allows for agent applications to be fully portable.

B. Agent-Specific Requirements

d) Agent Model Compliance: As stated in Section II-C, any programming language chosen for the implementation of

agent systems has to be able to support agents that comply with the model derived in Section II. If to have the same model is not possible, the one provided should be as similar as possible to that in Section II.

e) Support for Rationality: Agents feature autonomy and pro-activeness, characteristics that are often supported by providing agents with a sort of intelligence (goal-oriented agents, BDI architectures, rule production systems, etc.). Such a support must be provided by the chosen language or by a library that, however, must be used with language constructs and syntax. This is required in order to have a common and integrated programming environment for the development of all the parts of an agents[3].

f) Support for Distribution: Multi-agent systems feature distribution; even if we can have MASs in which agents run on the same computer, in general the agents of a MAS reside in different interconnected hosts. To support this characteristic, the chosen language has to provide suitable abstractions and libraries to perform communication among programs running in different hosts. The possibility of hiding protocol and communication details to the programmer, who would use high-level tools (like e.g. RPC, RMI or CORBA), is obviously welcome.

IV. The Erlang Language

In order to find the language that best approximates the agent model introduced in Section II and meets the requirements listed in Section III, we started an investigation aiming at analyzing various existing programming languages. We voluntarily excluded all "agent-specific" languages and concentrated only on general purpose ones, because we noted that the former are rich of agent-specific constructs but lacks of many general-purpose statements and libraries, thus needing the integration of other environments to build a complete software system.

In this investigation, we found the *Erlang* language [11], [6] not only well-suited to develop agents based on the proposed model, but also able to provide constructs and abstractions for the implementation of a complete platform for rational agents, in accordance with the principles given in this paper[4]. The reasons for such a choice are listed below, while an evaluation of the language, using the requirements derived in Section III, will be reported in Section IV-A.

Erlang is a symbolic functional language; it supports functions with multiple clauses and/or guards. It is easy to see that this basic characteristic of Erlang perfectly fits the implementation of an agent behavior modeled as (1).

Erlang has a Prolog-like syntax, data handling and representation. Erlang is derived from Prolog[5] and thus inherits from this language many principles. Given that writing an agent often implies to add a sort of "intelligence", to be implemented by means of e.g. an expert system or a rule production engine, a benefit is indeed obtained from using a Prolog-like language.

Erlang is based on matching. An Erlang assignment expression is, in practice, a matching expression: left-hand side unassigned (unbound) variables are bound to right-hand side terms, and left-hand side terms are matched with right-hand side terms. For example, the assignment expression

```
[inform, X, Y] = [inform, sender, receiver]
```

matches the `inform` term, and binds `X` to `sender` and `Y` to `receiver`. This matching capability facilitates the specification of data patterns to be matched when a particular event occurs (e.g. the arrival of an ACL message formed in a specified way).

Erlang is a concurrent language; it is based on isolated processes that share nothing and interact by means of message passing. Multi-agent systems feature exactly the same characteristic, given that the word "processes" is changed in "agents".

Erlang is a distributed language; message passing semantics is independent of the physical location (i.e. network site) of the interacting processes. This characteristic perfectly fits the support for distributed multi-agent systems.

Erlang is a fault-tolerant language; processes are monitored and, when a process crashes, a programmed corrective action (e.g. restarting the process) is immediately performed. In order to implement a fault-tolerant agent system, a suitable architecture for supervision should be mandatory. Erlang provides it natively.

A. Evaluating Erlang

Having illustrated those basic characteristics of Erlang that make it suitable for the realization of agent systems, the next step, according to the principles dealt with in this paper, is to evaluate this language using the requirements introduced in Section III.

a) Safety: Erlang is safe. It is a symbolic language that handles primitive numeric types and "atoms". Composite types include lists (arrays) and tuples[6]. Erlang does not allow the use of pointers, while lists/tuples handling is protected against out-of-bounds accesses. Such (and other) runtime error conditions are signaled by means of exceptions, which can be also caught in order to perform user-defined error handling[7].

b) Completeness: The Erlang runtime environment is provided with a very large number of libraries, comparable to those of other more famous languages (like C/C++, Java, Python, etc.)[8].

[3]In this sense, and in the authors' opinion, solutions like JADE + JESS/-Drools, or JADE + JADEX, cannot be considered acceptable, since they force the programmer to deal with different programming languages that often (too much) differ in model, syntax and semantics.

[4]Erlang is a functional and concurrent language initially developed, in 1984, by Ericsson.

[5]The first implementation of Erlang was written in Prolog.

[6]Composite types include also strings and records, but strings are handled as "lists of integers", where each element is the ASCII code of the corresponding character, and records are treated as tuples.

[7]Other runtime conditions include also bad matching, calling a non-existent function, calling a non-defined function clause, etc.

[8]The list of Erlang libraries, together with an in-depth description of each of them, is reported in the documentation provided in the Erlang web site [6].

c) Portability: The Erlang environment is based on a virtual machine that is provided for many platforms (Windows, Linux, BSD, Solaris, etc.). Erlang programs are compiled in platform-independent bytecoded executables, which can thus directly run using the virtual machine of any platform[9]. Even if the performances of the Erlang virtual machine are quite good [10], [9], an ahead-of-time Erlang-to-native code compiler is also provided [24], [25].

d) Agent Model Compliance: As reported in the beginning of this Section, the fact that Erlang programming is based on functions with multiple clauses implies a one-to-one mapping of the agent model provided by (1) to native language constructs. However, in order to support the autonomous execution of a behavior modeled as in (1), a suitable engine should be needed, hence an *agent platform*. An agent platform is also needed to support behavior composition and extension, in accordance with the principles in Section II; indeed these abstractions do not have corresponding Erlang constructs and thus need to be supported by an ad-hoc runtime environment. However, even if the language lacks such constructs, its characteristics are able not only to allow an easy development a suitable runtime environment, but also to provide a flexible way to specify, in source programs, behavior composition and extension. This will be made more clear in Section V.

e) Support for Rationality: Even if Erlang is derived from Prolog and has many characteristics in common with this language, Erlang is not *logic* but *functional*. This means that it does not have a native support for the definition of e.g., Horn clauses, and thus for the introduction of inference in Erlang programs. However, the possibility of (*i*) handling types and symbols as in Prolog, (*ii*) expressing production rules as Erlang functions, (*iii*) supporting high-order computations and lambda functions, favor the implementation of rule-processing engines able to add Erlang program a sort of "intelligence".

f) Support for Distribution: Erlang allows the implementation of application protocols using sockets, as well as the support for distributed applications interacting using SOAP/XMLRPC, CORBA-IIOP[10] or simply HTTP. But the main feature of Erlang for distribution is the support of true location transparency in process interaction. In fact, the syntax and semantics of the language constructs for sending and receiving messages to and from processes do not change if the processes are local or remote. This is indeed a very interesting feature for the implementation of agent systems and platforms.

B. Remarks

The discussion reported above highlights that the requirements *a)*, *b)*, *c)* and *f)* are fully met by the Erlang language. Requirements *d)* and *e)* are instead partially met, but we stated that their support can be easily added by writing suitable Erlang libraries. We designed the eXAT platform for this purpose.

[9]Unless platform-specific features have been explicitly encoded by the programmer.

[10]To this aim, the Erlang runtime system provides an IDL complier that generates Erlang, Java^{TM} and C++ stubs.

V. The eXAT Platform

eXAT [4], [30], [13], [15], [14]—the name means ***erlang eXperimental Agent Tool***—is a platform for the development and execution of Erlang agents; the main services provided include:

- an execution engine for agent behaviors modeled as finite-state machines;
- a rule-processing engine for supporting the development of rule production systems;
- a communication module for the exchange of ACL messages[11] according to FIPA model and semantics [16].

Programming agents' behaviors, in eXAT, implies to model them by using a set of functions, with multiple clauses, that express what are the *events* that, bound to certain *states*, trigger the execution of certain *actions* and the change of *state*. In order to make behavior engineering more flexible, events are defined by specifying the *type* and the *data pattern* bound to that event. This allows, for example, to specify that an event is the arrival of an ACL message—the *type*—given that the message is an "inform" speech act—the *data pattern*. The event types handled by eXAT are:

- *acl*, the reception of an ACL message;
- *timeout*, the expiry of a given timeout;
- *eres*, an event occurring in a rule-processing engine (see Section V-B);
- *silent*, the silent (spontaneous) event.

As it will be explained later on, this decoupling between event types and bound patterns allows behavior extensions according to the inheritance concepts expressed in Section II-B.

Behavior specification in eXAT is easily performed by means of three Erlang functions with different clauses—**`action`**, **`event`** and **`pattern`**. Function `action` is used to assign, to each state name, a list of couples *event names* and *action function*, meaning that, at the occurrence of that event, the associated action function has to be executed. Function `event` indicates, for each event name, the *event type* and the *pattern name* relevant to the data associated to that event. Function `pattern` maps each pattern name with the relevant matching value, which depends on the type of the bound event; the possibility of using lambda functions adds flexibility in pattern specification.

As an example, the behavior depicted in Figure 1, supposing that the ACL message to wait for is an "inform" speech act encoded in LISP, can be implemented, in eXAT, by means of the following listing:

```
-module (b).

action (Self, start) ->
  [{new_message_event, do_processing},
   {timeout_event, finalize}].

event (Self, new_message_event) ->
```

[11]The exchange of ACL messages in eXAT relies on the Erlang native mechanism for exchanging data among Erlang processes. Thus, in the current version of eXAT, no FIPA standard message transport protocol is provided. This will be made available in the future releases of the platform.

```
  {acl, inform_pattern};
event (Self, timeout_event) ->
  {timeout, timeout_pattern}.

pattern (Self, inform_pattern) ->
  [#aclmessage {speechact = inform,
                language = 'LISP'}];
pattern (Self, timeout_pattern) -> 10000.

do_processing (Self, EventName, Data, ActionName) ->
  % Perform processing.
  % 'Data' is bound to the received message.
  object:stop (Self).

finalize (Self, EventName, Data, ActionName) ->
  % Finalize behavior.
  object:stop (Self).
```

The formal model of behavior in eXAT is thus expressed as follows[12]:

$$action : State \rightarrow \{(EventName, Action)\}$$

$$event : EventName \rightarrow (EventType, PatternName)$$

$$pattern : PatternName \rightarrow PatternSpecification$$

$$EventName \in \{silent, eres, acl, timeout\}$$

Even if the model above is expressed in a formulation different than that of formula (1), it is easy to check that its semantics is the same: we have only separated the various parts of a FSM in order to make specialization possible by means of inheritance, as it will be detailed in the next SubSection.

A. Composing and Extending eXAT behaviors

Behavior composition is performed, in eXAT, by calling a suitable **behave** function, in the body of an action implementation, which specifies the name of the next behavior to be executed. The function is synchronous, that is, it waits for the complete execution of the given behavior before returning to the caller. The sample behavior b_1, illustrated in Section II-A, that triggers an English or Dutch auction on the basis of a message or a timeout, is thus easily implemented using the following source code:

```
-module (b1).

action (Self, start) ->
  [{first_event, do_english_auction},
   {second_event, do_english_auction}].

event (Self, first_event) ->
  {acl, inform_pattern};
event (Self, second_event) ->
  {timeout, timeout_pattern}.

pattern (Self, inform_pattern) ->
  [#aclmessage {speechact = inform}];
pattern (Self, timeout_pattern) -> 10000.
  % Wait ten seconds.

do_english_auction (Self, EventName,
                    Data, ActionName) ->
  agent:behave (Self, english_auction),
  % behavior 'english_auction' is executed
  object:stop (Self).
  % stops current behavior when the 'english_auction' is over

do_dutch_auction (Self, EventName,
                  Data, ActionName) ->
  agent:behave (Self, dutch_auction),
  object:stop (Self).
```

[12]The state reached by the FSM after the occurrence of an event is encoded in the *Action* and, for this reason, it does not explicitly appear in these formulae.

Specialization is achieved, in eXAT behaviors, by means of *redefining* one or more parts of an existing behavior. In this case, the peculiar feature of eXAT relies on the *granularity of elements* on which redefinition is possible. eXAT behavior can be extended by redefining (*i*) *single function clauses* and (*ii*) *single elements* of data returned by `action`, `event` and `pattern` functions. In other words, from the point of view of the refined function, redefinition can be *total* or *partial*.

Total redefinition means to completely change the return value of the interested function or function clause, and this implies to modify (*a*) the couple {*event, action*} bound to a certain state—if the function is `action`—(*b*) the couple {*event type, pattern*} defining a certain event—when function `event` is considered—or (*c*) the specification of a given pattern—through redefinition of function `pattern`. For example, if we would design a behavior like *b1* above, but using a message (e.g. a "confirm") instead of a timeout to trigger the Dutch auction, we can write the following listing:

```
-module (b1_prime).

extends () -> b1.

event (Self, second_event) ->
  {acl, confirm_pattern}.

pattern (Self, confirm_pattern) ->
  [#aclmessage {speechact = inform}].
```

The reader may note the use of the `extends` function to indicate the ancestor behavior from which *b1_prime* inherits the basic FSM structure, and the functions `event` and `pattern`, which express the new condition that triggers Dutch auction execution.

Partial redefinition means instead to change only some elements of the data returned by a function, leaving the other elements as defined in the behavior superclass, e.g. one of the couples {*event, action*} bound to a certain state, the *event* of such a couple, the *event type* or the *pattern name* bound to a certain event, one element of a data pattern, etc.

B. Adding Intelligence

One of the parameters to evaluate the ability of a language for agent development is the possibility to support a sort of intelligence. Erlang does not feature this characteristic and, to overcome these issue, eXAT includes an Erlang-based rule production system, thus providing a development environment that uses the same language to program all the parts of an autonomous agent. The rule production system integrated in eXAT, called ERES, can be used to realize expert systems implementing the reasoning process for agents. ERES

allows the creation of multiple concurrent *rule production engines*, each one with its own *rules* and a *knowledge base*—the *mind*—that stores a set of *facts*—the *mental state*—represented by Erlang types (tuples or lists). Rules are written as function clauses with the form:

```
rule (pattern of the asserted fact) ->
        assert (fact to assert) -- and/or
        retract (fact to retract) -- and/or
        -- do other things
```

Rule processing is based on checking that one or more facts, with certain patterns, are present in the knowledge base and then doing something like asserting another fact, retracting an existing fact, etc. For example, supposing that we want to write the following inference rule:

If X is the child of Y and Y is female, then Y is X's mother; otherwise, if Y is male, then Y is X's father.

Representing the relations "child of", "mother of", "father of" and the "gender" respectively with the facts (Erlang tuples) {'child-of', X, Y}, {'mother-of', X, Y}, {'father-of', X, Y} and {Gender, X}, the inference rule above will be simply written as follows:

```
is_parent (true, Engine, Relation, Y, X) ->
  eres:assert (Engine, {Relation, Y, X});
is_parent (_, Engine, Relation, Y, X) -> nil.

rule (Engine, {'child-of', X, Y}) ->
  is_parent (eres:asserted (Engine, {female, Y}),
             Engine, 'mother-of', Y, X).
  is_parent (eres:asserted (Engine, {male, Y}),
             Engine, 'father-of', Y, X).
```

Fact patterns can be also given as lambda functions thus allowing the specification of complex matching expressions.

ERES engines are designed to be connected with behavior execution and message exchanging. The former connection can be performed by specifying, in a behavior, that an event is relevant to the assertion of a fact, with a given pattern, in a given ERES engine. In this case, the event type is "eres" and the bound data pattern expresses the way in which the fact to wait for is formed. For example, if we would trigger, in the behavior *b*1 of Section V-A, the Dutch auction when the fact {balance,X}, with $X > 3000$, is asserted in the ERES engine called "my_mind", we should write a behavior *b*1_*second* as follows:

```
-module (b1_second).

extends () -> b1.

event (Self, second_event) ->
  {eres, balance_pattern}.

pattern (Self, balance_pattern) ->
  {my_mind, read, {balance, fun(X) -> X > 3000 end}}.
```

Note the lambda function in the 'balance_pattern' to specify the fact pattern {balance,X}, with $X > 3000$.

The second connection of ERES engines is with the communication module of eXAT in order to support ACL semantics. We remind that FIPA-ACL specification defines message semantics by means of the so-called *feasibility precondition (FP)* and *rational effect (RE)*. However, even if well-specified, these conditions are not implemented in the majority of well-known agent platforms. eXAT fills this gap by allowing a direct connection between ACL message sending/receiving and the agent's mental state, which can be represented by the knowledge base of an ERES engine: *FP*s can be checked by looking at what is stored in the agent's mental state, while *RE*s can be achieved by suitably updating the agent's mental state. Such a feature is indeed very important in agent design, since it builds a semantic bridge between autonomy/pro-activeness and social behavior, thus constituting a fundamental way for realizing "true rational" agents.

VI. Evaluating Java^{TM}

A remark that emerges from the discussion reported till now is that the combination Erlang + eXAT is able to meet all the requirements listed in Section III. Now, in order to provide a more complete report of the reasons that led us to chose the Erlang approach, it is worthwhile to analyze the solutions based on Java^{TM}, checking if they are able (and to what extent) to meet the agent development requirements identified.

First of all, any reader can easily prove that the General Requirements, *a)*, *b)* and *c)*, are met by Java^{TM}; indeed they are among the basic characteristics of this language. Also requirement *f)*—support for distribution—is obviously met since Java^{TM} has been designed for distributed environments, even if the semantics of interaction among remote objects is not exactly the same as that of the local case. But the situation changes when we consider the other Agent-Specific Requirements. Agent compliance to model of Section II (requirement *d)*) is hard to obtain mainly for two reasons:

- Java^{TM} does not supports "methods with multiple clauses". Obviously the use of "if" statements inside a method's code does not provide the same semantics of multiple clauses.
- Java^{TM} is not a symbolic language and everything must be encapsulated inside an object. This characteristic, even if it provides a more "formally correct" programming environment, burdens the implementation process, since it requires many lines of code to be written to wrap concepts and symbols inside objects. Indeed a behavior model like that of Section II could be also built using Java^{TM}, but it requires a very complex object model to reach similar, but not the same, features. As an example, the behavior model of JADE [12], which is based on FSMs, is very flexible, but is not able to provide the same specialization feature we require (see [13], [15] for a brief comparison among eXAT and JADE).

Also requirement *e)*—support for rationality—is hard to obtain with Java^{TM}. This is demonstrated by the fact that all the projects aiming at including inference in Java^{TM} programs

or in Java^{TM} agent platforms employ non-Java approaches, abstraction and languages [1], [5], [7], [29].

VII. Conclusions

The concepts and statements reported in this paper do not aim at denigrating Java^{TM} nor the Java^{TM}-based agent platforms available today. Many of them are worth of note and are successfully employed in many agent-based projects. However, this does not imply that there could not exist approaches or language able to support agent development better than Java^{TM}. Our research goes in this direction. To this aim, we provided an alternative platform, called eXAT, that allows implementation of multi-agent system using the Erlang language. By means of requirement evaluation, we have shown, in this paper, that the combination Erlang + eXAT seems to be a valid and very interesting alternative for the implementation of agent systems. Surely, a better evaluation would imply the realization of some case-studies, aiming at understanding if the constructs, model and abstractions provided by eXAT are enough flexible and complete for the implementation of multi-agent applications. This is one of the topics that will be dealt with in our current and future research work.

References

[1] "http://herzberg.ca.sandia.gov/jess/. JESS Web Site," 2003.

[2] "http://www.ghg.net/clips/CLIPS.html. CLIPS Web Site," 2003.

[3] "http://www.agentlink.org/resources/agent-software.php," 2004.

[4] "http://www.diit.unict.it/users/csanto/exat/. eXAT Web Site," 2004.

[5] "http://www.drools.org. Drools Home Page," 2004.

[6] "http://www.erlang.org. Erlang Language Home Page," 2004.

[7] "JSR-000094 Java^{TM} Rule Engine API, http://www.jcp.org/aboutJava/communityprocess/review/jsr094/," 2004.

[8] "http://www.agent-software.com," 2004.

[9] J. Armstrong, B. Dacker, R. Virding, and M. Williams, "Implementing a Functional Language for Highly Parallel Real Time Applications," 1992.

[10] J. L. Armstrong, "The development of Erlang," in *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, A. Press, Ed., 1997, pp. 196–203.

[11] J. L. Armstrong, M. C. Williams, C. Wikstrom, and S. C. Virding, *Concurrent Programming in Erlang, 2nd Edition*. Prentice-Hall, 1995.

[12] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing multi-agent systems with a FIPA-compliant agent framework," *Software: Practice and Experience*, vol. 31, no. 2, pp. 103–128, 2001.

[13] A. Di Stefano and C. Santoro, "eXAT: an Experimental Tool for Programming Multi-Agent Systems in Erlang," in *AI*IA/TABOO Joint Workshop on Objects and Agents (WOA 2003)*, Villasimius, CA, Italy, 10–11 Sept. 2003.

[14] ——, "eXAT: A Platform to Develop Erlang Agents," in *Agent Exhibition Workshop at Net.ObjectDays 2004*, Erfurt, Germany, 27–30 Sept. 2004.

[15] ——, "Designing Collaborative Agents with eXAT," in *ACEC 2004 Workshop at WETICE 2004*, Modena, Italy, 14–16 June 2004.

[16] Foundation for Intelligent Physical Agents, "FIPA Communicative Act Library Specification—No. SC00037J," 2002.

[17] ——, "FIPA Contract Net Interaction Protocol Specification—-No. SC00029H," 2002.

[18] ——, "FIPA Dutch Auction Interaction Protocol Specification—-No. XC00032F," 2002.

[19] ——, "FIPA English Auction Interaction Protocol Specification—-No. SC00031F," 2002.

[20] ——, "FIPA Request Interaction Protocol Specification—-No. SC00026H," 2002.

[21] ——, "http://www.fipa.org," 2002.

[22] S. Franklin and A. Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents," in *Third International Workshop on Agent Theories, Architectures, and Languages (ATAL)*. Springer-Verlag, 1996.

[23] C. Hoare, *Communicating Sequential Processes*. Prentice Hall International, 1985.

[24] E. Johansson, M. Pettersson, and K. Sagonas, "A High Performance Erlang System," in 2^{nd} *International Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, Sept. 20–22 2000.

[25] ——, "The HiPE/x86 Erlang Compiler: System Description and Performance Evaluation," in *Sixth International Symposium on Functional and Logic Programming (FLOPS 2002)*, Sept. 15–17 2002.

[26] F. McCabe and K. Clark, "April: Agent Process Interaction Language," in *Intelligent Agents*, N. Jennings and M. Wooldridge, Ed. Springer, LNCS 890, 1995.

[27] R. Milner, *Communication and Concurrency*. Prentice Hall International, 1989.

[28] ——, *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge Univ Press, 1999.

[29] A. Pokahr, L. Braubach, and W. Lamersdorf, "Jadex: Implementing a BDI-Infrastructure for JADE Agents," *Telecom Italia Journal: EXP - In Search of Innovation (Special Issue on JADE)*, vol. 3, no. 3, Sept. 2003.

[30] C. Santoro, *eXAT: an Experimental Tool to Develop Multi-Agent Systems in Erlang - A Reference Manual*. Available at http://www.diit.unict.it/users/csanto/exat/, 2004.

[31] C. Varela, C. Abalde, L. Castro, and J. Gulias, "On Modelling Agent Systems with Erlang," in 3^{rd} *ACM SIGPLAN Erlang Workshop*, Snowbird, Utah, USA, 22 Sept. 2004.

A Multi-Agent System to Support Remote Software Development

Marco Mari, Lorenzo Lazzari, Alessandro Negri, Agostino Poggi and Paola Turci

***Abstract*—In this paper, we present a Web and multi-agent based system to support remote students and programmers during common projects or activities based on the use of the Java programming language. This system, called RAP (Remote Assistant for Programmers), associates a personal agent with each user. A personal agent helps its user to solve problems proposing information and answers, extracted from some information repositories, and forwarding answers received from other "on-line" users, that were contacted because their personal agents recommend them as experts in that topic. To be able to recommend their users, personal agents build and maintain a profile of them. This profile is centered on user's competences and experience and is built by extracting information through both the code she/he wrote and the positive answers the user gave to other users. A first prototype of the system is under implementation in Java by using the JADE multi-agent development framework. This prototype will be tested in practical courses on JADE shared among students of some American Latin and European Universities inside the European Commission funded project "Advanced Technology Demonstration Network for Education and Cultural Applications in Europe and Latin America".**

***Index Terms*—Cooperative systems, multi-agent systems, intelligent tutoring systems.**

I. Introduction

FINDING relevant information is a longstanding problem in computing. Conventional approaches such as databases, information retrieval systems, and Web search engines partially address this problem. Often, however, the most valuable information is not widely available and may not even be indexed or cataloged. Much of this information may only be accessed by asking the right people. The challenge of finding relevant information then reduces to finding the "expert" whom we may ask a specific question and who will answer that question for us. However, people may easily get tired of receiving banal questions or different times the same question, therefore, who needs help for solving a certain problem, should look for documents related to the problem and then eventually look for a possible expert on the topic.

This kinds of problems are very relevant in the software development because of the wide variety of software solutions, design patterns and libraries makes hard to take the best decision in every software development phase, and a developer can't always have the required expertise in all fields.

In this paper, we present a multi-agent based system, called RAP (Remote Assistant for Programmers), that integrated information and expert searching facilities for communities of student and researchers working on related projects or work and using the Java programming language. In the following section, we describe the RAP system, the current state of its implementation and some preliminary evaluation results, then we introduce related work and, finally, we give some concluding remarks and present some our future research directions to improve the RAP system.

Manuscript received September 27, 2004. This work is partially supported by the European Commission through the contract "@lis Technology Net (ALA/2002/049-055)" and by "Ministero dell'Istruzione, dell'Università e della Ricerca" through the COFIN project ANEMONE.

M. Mari is with DII, University of Parma, Parco Area delle Scienze 181A, 43100, Parma, Italy (phone: +39 0521 905712; e-mail: mari@ce.unipr.it).

L. Lazzari is with DII, University of Parma, Parco Area delle Scienze 181A, 43100, Parma, Italy (phone: +39 0521 905712; e-mail: lazzari@ce.unipr.it).

A. Negri is with DII, University of Parma, Parco Area delle Scienze 181A, 43100, Parma, Italy (phone: +39 0521 905712; e-mail: negri@ce.unipr.it).

A. Poggi is with DII, University of Parma, Parco Area delle Scienze 181A, 43100, Parma, Italy (phone: +39 0521 905728; e-mail: poggi@ce.unipr.it).

P. Turci is with DII, University of Parma, Parco Area delle Scienze 181A, 43100, Parma, Italy (phone: +39 0521 905708; e-mail: turci@ce.unipr.it).

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

II. The RAP System

RAP (Remote Assistant for Programmers) is a system to support communities of students and programmers during shared and personal projects based on the use of the Java programming language. RAP associates a personal agent with each user which helps her/him to solve problems: proposing information and answers extracted from some information repositories, and forwarding answers received by "experts" on the topic selected on the basis of their profile. A personal agent also maintains a user profile centered on her/his competences and experience built through the positive answers given to other users and by extracting information through the code she/he has written.

A. System Agents

The system is based on seven different kinds of agents: Personal Agents, Code Documentation Managers, Answer Managers, User Profile Managers, Email Manager, Starter Agent and Directory Facilitators.

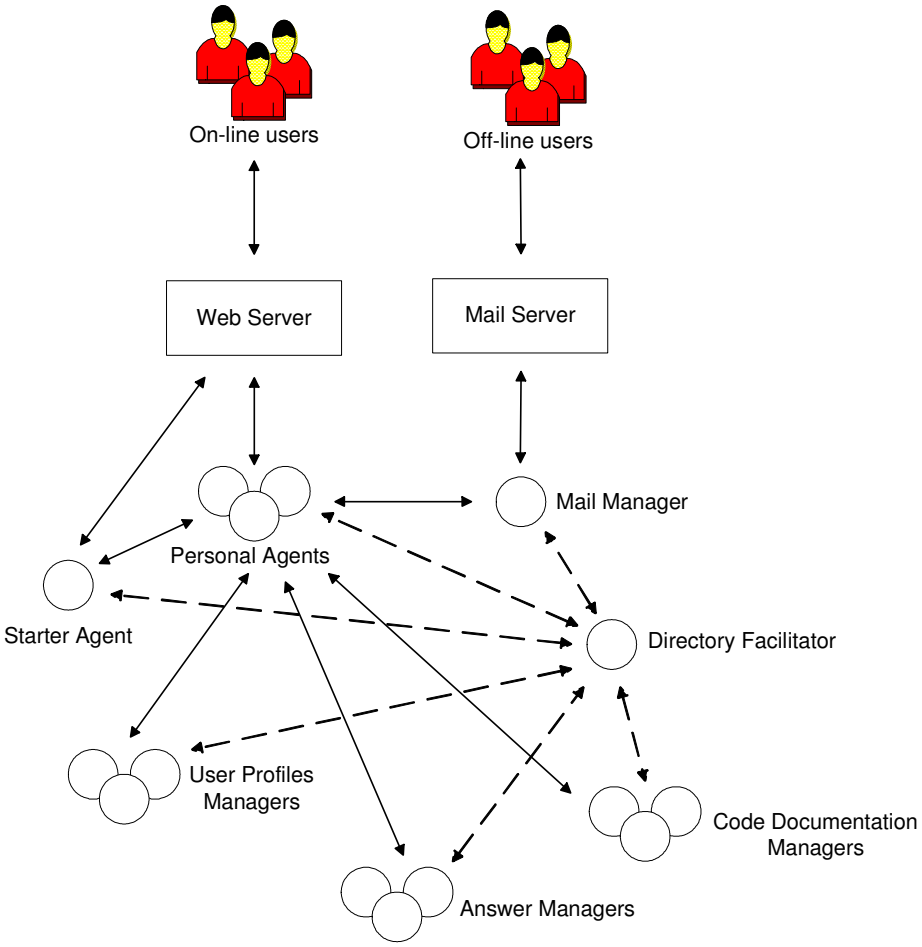


Fig. 1. RAP platform architecture.

Personal Agents are the agents that allow the interaction between the user and the different parts of the system and, in particular, between the users themselves. Moreover, this agent is responsible of building the user profile and maintaining it when its user is "on-line". User-agent interaction can be performed in two different ways: when the user is active in the system, through a Web based interface; when it is "off-line" through emails. Usually, there is a personal agent for each on-line user, but sometimes personal agents are created to interact with "off-line" users via emails.

User Profile Managers are responsible of maintaining the profile of "off-line" users and of activating personal agents when it is necessary that they interact with their "off-line" users via emails.

Code Documentation Managers are responsible of maintaining code documentation and of finding the appropriate "pieces of information" to answer the queries done by the users of the system.

Answer Managers are responsible of maintaining the answers done by users during the life of the system and of finding the appropriate answers to the new queries of the users. Besides providing an answer to a user, this agent is responsible of updating the score of the answer and forwarding the vote to either the personal agent or the user profile manager for updating the profile of the user that performed such an answer.

Email Managers are responsible for receiving emails from "off-line" users and forwarding them to the corresponding personal agents.

Starter Agents are responsible for activating a personal agent when either a user logs on or another agent request it.

Directory Facilitators are responsible to inform an agent about the address of the other agents active in the system (e.g., a personal agent can ask about the address of all the other personal agents, of the code documentation managers, etc.).

Figure 1 gives a graphical representation of a RAP platform and the interactions of personal agents and of the directory facilitator with the other agents of the platform. Note that a RAP platform can be distributed on different computation nodes and that a RAP system can be composed of different RAP platforms connected via Internet. Moreover, in figure 1 groups of three users or agents means that there can be one or more users and agents. Finally, in A RAP system there is a directory facilitator for each platform.

B. System Behavior

A quite complete description of the behavior of the system can be given showing the scenario where a user asks information to its personal agent to solve a problem in its code and the personal agent finds one (or more) "pieces of information" that may help her/him. The description of this

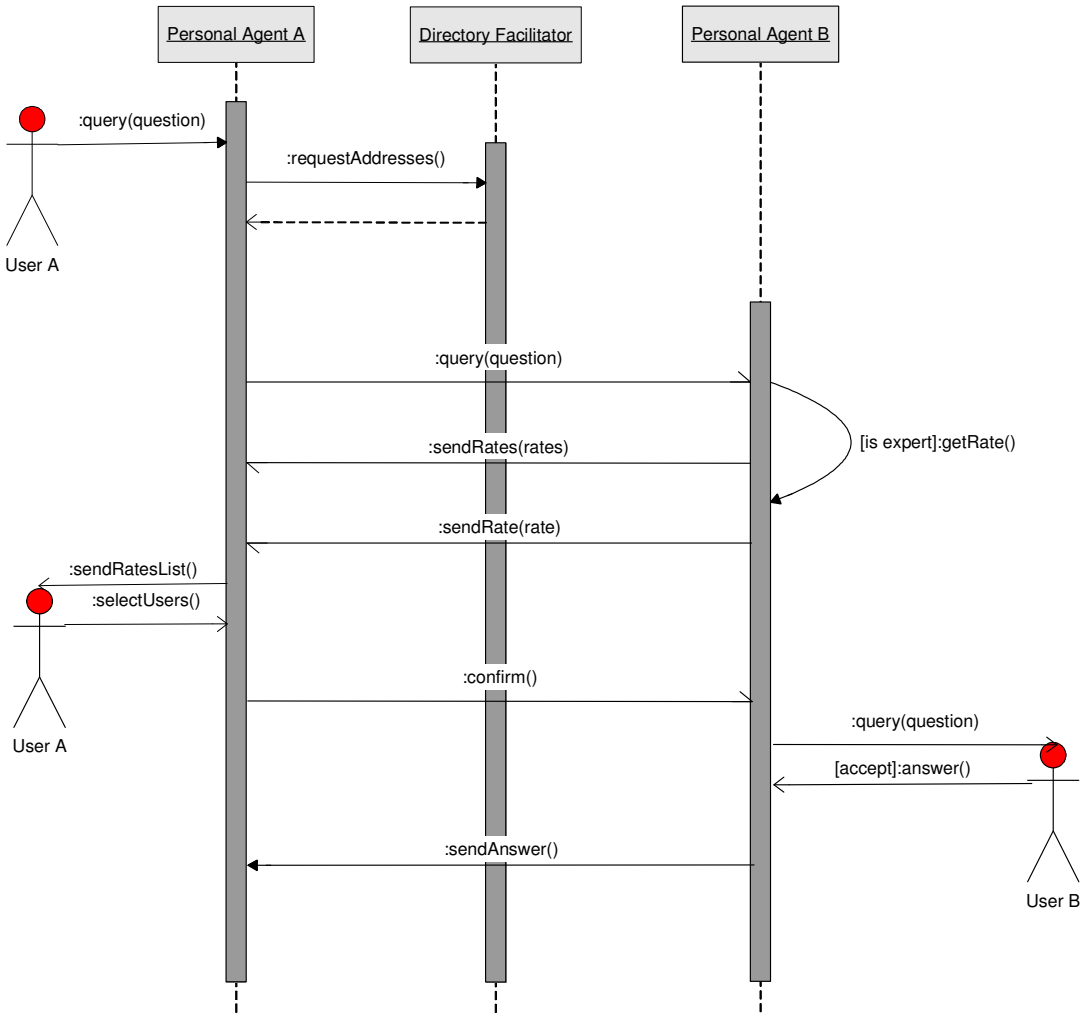


Fig. 2. UML interaction diagram describing how works to allow a user to ask a question and then receive the relative question from an "expert".

scenario can be divided in the following steps:

1) Select answer types
2) Submit a query
3) Find answers
4) Rate answer

Select answer types: the user can receive information extracted from code documentation, answers extracted from the answer repositories and new answers sent by the other users of the system. Therefore, before submitting the query, the user can select the types of answers (one or more) she/he likes to receive.

Submit a query: the user, through its user interface, provides the query to its personal agent. In particular, the user can query either about a class or an aggregation of classes for implementing a particular task or about a problem related to her/his current implementation. The query is composed of two parts. The first part (we call it "annotation") identifies the context of the query and can contains keywords provided by a system glossary and/or the identification of classes and/or methods in a univocal way (i.e., the user needs to specify the complete package name for a class and adds the class name for a method). The second part contains the textual contents of the query.

Find answers: the personal agents perform different actions and interact with different agents to collect the various types of answers.

For getting code documentation, the personal agent asks the directory facilitator about all the code documentation managers. After receiving this information, the personal agent forwards the query to all these agents. These agents search "pieces" of code documentation related to the query and send them to the personal agent associating a score with each "piece".

For getting answers from the answer system repositories, the personal agent asks the directory facilitator about all the answer managers. After receiving this information, the personal agent forwards the query to all these agents. These agents search answers related to the query and send them to the personal agent associating a score with each answer.

The reception of new answers from the system users is a more complex activity and its description can be divided in four further steps (Figure 2 shows the UML interaction diagram describing these phases):

3.1) Find experts
3.2) Receive experts rating
3.3) Select experts

3.4) Receive answers

Find experts: the personal agent asks the directory facilitator about the other active personal agents (i.e., the personal agents of the user that are "on-line") and all the user profile managers of the system (i.e., the agents managing the profile of the users that are not "on-line"). After receiving this information, the personal agent forwards the query to these personal agents together to the user profile managers.

Receive expert rating: all these agents (personal agents and user profile managers) compute the rating of their users to answer to this query on the basis of the query itself and of the user profile. The agents that compute a positive score (i.e., its user may give an appropriate answer to the query) reply to the querying personal agent with the rating of its user (in the case of a personal agent) or its users (in the case of user profile manager).

Select experts: the personal agent divides on-line and off-line users, orders them on the basis of their rating and, finally, presents these two lists to its user. The user can select more than one user and then the personal agent sends the query to the corresponding personal agents (for the "on-line" users) and to the corresponding user profile managers (for the "off-line" users).

Receive answers: the replying personal agents immediately present the query to their user and forward the answer as soon as the user provides it. User profile manager activates the personal agents of the involved users through the starter agent. These personal agents forward the query to their user via email and then terminate themselves. Users can answer either via email or when they log again on the system. In the case of email, the email manager starts the appropriate personal agent that extracts the answer from the email and forwards it. When the querying personal agent receives an answer, it immediately forwards it to its user.

Rate answers: after the reception of all the queries, or when the deadline for sending them expired, or, finally, when the user has already found an answer satisfying its request, the personal agent presents the list of read answers to its user asking her/him to rate them. After the rating, the agent forwards each rating to the corresponding personal agent, code documentation manager, answer manager or user profile manager that provides to update the user profile and/or the answer rating (when a user rates an answer retrieved from the answer repository, this rating is also used to updated the user profile of the user that previously proposed the answer). Note that in the case of rating of users answers, the rating cannot be known by the user that sent the answer and users that did not send answers automatically received a negative rating.

C. User and Document Profile Management

In our system, the management of user and document profiles is performed in two different phases: an initialization phase and an updating phase. Figure 3 gives a graphical description of this process.

In order to simplify, speed up and reduce the possibility of inaccuracy due to people's opinions of themselves and to incomplete information, we decided to build the initial profile of the users and documents in an automated way that, for the users, is very similar to the one used by Expert Finder system [21]. Profiles are represented by vectors of weighted terms whose value are related to the frequency of the term in the document or to the frequency of the use of the term by the user. The set of terms used in the profiles is not extracted from a training set of documents, but corresponds to those terms included in the system glossary, provided to the users for annotating their queries, and to the names of the classes and methods of the Java software libraries used by the community of the users of the system.

While document profiles are computed by using term frequency inverse document frequency (TF-IDF) [19] and profiles weighted terms correspond to the TF-IDF weight, each user profile is built by user's personal agent through the analysis of the Java code she/he has written. In this case, the weight of the terms in the profile corresponds to the frequency is not the TF-IDF weight, but the real frequency of the term in the code of the user (i.e., term frequency is not weighted on the basis of the frequency of the term in the code written by all the users). We used this approach for different reasons. First, we speed up and reduce the complexity of building user profiles. As a matter of fact, TF-IDF algorithm can be easily used in a centralized system where all the profiles and the data to build them are managed. Our context is more complex: the system is distributed, only the personal agent can access to the software of its user, for privacy and security reasons, and the profiles are maintained by the corresponding personal agents or by possibly different user profile managers when the personal agent is not alive. The second and most important reason is that the profile built by personal agents is only the initial user's profile. And it will be updated when the user writes new software and especially when the user helps other users answering their queries.

The updating of user and document profiles is done in three cases: i) a user asks about a problem and then rates some of the received answers, ii) a new software library is introduced in the ones used by the community or some new terms are introduced in the system glossary, and iii) a user writes new software.

In the first case, there are three possible behaviors according to the source of the answer (user, document repository or answer repository).

If the answer comes from a user, on the basis of the received rating her/his profile is updated, of course, only the part concerning the terms involved in the query annotation. Moreover, if the rating is positive, the answer is added to the answer repository and its profile is built from the query annotation and the rating of the answer.

If the answer comes from the document repository and the rating is positive, the answer is added to the answer repository, its profile is the original document profile updated by the rating of the answer.

Finally, if the answer comes from the answer repository and

the rating is positive, the part of the answer profile related to the terms involved in the query annotation is updated on the basis of the received rating. Moreover, in the case that this positive rated answer comes from a user and not from the document repository, also the part of the user profile related to the terms involved in the query annotation is updated on the basis of the received rating. Finally, the query corresponding to such positive rated answer is added in the repository (i.e., the answer was good for one or more previous queries, but also for the current one; queries are ordered by answer rating).

We decided to avoid the use of negative rates for updating the profile of the answers in the answer repository. In fact, if an answer is in the repository, it means that at least a user considered useful to solve her/his problem; therefore, if later on this answer received a negative rate it does only means that the answer is not appropriate for the last query, but it is still appropriate for the previous queries for which it received positive rates.

When a new software library is introduced in the list of libraries used by the users of the system or some new terms are introduced in the system glossary, all the document and user profiles must be updated. While document profiles are rebuilt on the basis of the new complete set of terms, user profiles are updated adding the weighted terms corresponding to the new term, of course with a weight equal to their frequency in the software written by the user.

Finally the user's profile is updated, adding only the new weighted terms, even when the user writes new software.

D. System Implementation and Experimentation

A first prototype of the RAP System is under development by using JADE [3]. JADE (Java Agent Development framework) is a software framework to aid the realization of agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems [6]. JADE is an Open Source project, and the complete system can be downloaded from JADE Home Page [9].

Given the distributed nature of JADE based agent systems, a RAP system can be distributed on a set of agent platforms connected usually via Internet and situated in different parts of the world. Each agent platform can be distributed on different computation nodes and is connected to a Web server, for allowing direct interactions with the users, and to a mail server, for allowing email interactions with the users. In each agent platform there is a unique starter agent and email agent, but there might be more than one user profile manager, code documentation manager, answer manager. This usually happens when the agent platform is distributed on different nodes in order to cope with performance issues due to the number of the users to be managed. Furthermore, distribution of a RAP platform on different computation nodes and agents replication can be introduced for reliability reasons (in this case, agents manage copies of data) too. Finally, there can be one or more directory facilitators. In the case of more than one directory facilitator, these agents build a hierarchical map of the agents of the system; therefore, when an agent is created, the only information it needs to know is simply the address of the main (root) directory facilitator.

A large part of the first prototype of the system has been completed. In particular, the subsystem supporting interactions among personal agents and the interaction between each pair of personal agent and "on-line" user has been completed. This subsystem has been used with success by a small set of students, connected by different labs or from their house, for the development JADE software within some course works. In these activities, students could formulate queries annotating it with terms extracted from a glossary derived from the Sun "Glossary of Java Related Terms" [20] and class and method names extracted from Java and JADE source code.

Moreover, we have evaluated the system with a simulation. We have asked 10 queries on Java programming to 10 students with experience in Java programming, but with advanced experience on different application fields and software libraries. Of course, the 10 queries were defined in order to put in the light the difference in the knowledge of the students involved. Then, we have evaluated the part of the RAP system involving only user interactions (no document and answer repository). A personal agent is responsible to perform the 10 queries and other 10 personal agents to provide the answers written by the different students. The evaluation has concerned mainly the comparison of the ordered list (built ordering experts on the basis of their profiles) provided by the querying agent to its user with an ordered list of the answers we did before performing the simulation. The simulation was reiterated a number of times equal to the possible orders of the query and the users profiles were reset each simulation. The initial user's profile was built on the basis of the content of the answers associated with this virtual user. Clearly, it cannot be considered a simulation of the real behavior of the system, but the obtained results have encouraged us in the completion of the system. In fact, the differences between the personal agent ordered list and our a priori ordered list decreases during the evaluation process and for the last query we have had an average error of the 5%.

As from the end of this year, the final RAP system will be tested in practical courses on JADE shared among students of some American Latin and European Universities inside the European Commission funded project "Advanced Technology Demonstration Network for Education and Cultural Applications in Europe and Latin America (@lis Technology Net)" [1]. Moreover, the system will be used by students and researchers, involved in the ANEMONE project [2], for cooperating in the realization of agent-based software.

III. RELATED WORK

In the last years a lot of work has been done in the fields of document and expert recommendation and in the development of tools and systems for supporting e-learning and, in particular, computer programming activities.

With the advent of the Web, document recommendation systems are become one of most important area of both

research and application of information technologies. All the most important proposed systems are applied to the recommendation of Web pages and are not specialized for computer programming documents, but usually allow the customization for different subjects. GroupLens is the first system that used collaborative filtering for document recommendation [18]. This system determinates similarities among users and then is able to recommend a document to a user on the basis of the rating of similar users on the recommend document. Syskill &Webert is a system with the goal of helping users distinguishing interesting Web pages on a particular topic from uninteresting ones [16]. In particular, this system recommends document to a user on the basis of her/his user profile that it builds and updates by using user's evaluations of the interestingness of the read documents. Adaptive Web Site Agent is an agent-based system for document recommendation [17]. This system works on the documents of a Web site and recommends documents to visitors integrating different criteria: user preferences for the subject area, similarity between documents, frequency of citation, frequency of access, and patterns of access by visitors to the web site.

Several prior systems support expertise recommendations. Vivacqua and Lieberman [21] developed a system, called Expert Finder, that recommends individuals who are likely to have expertise in Java programming. This system analyzes Java code and creates user profiles based on a model of significant features in the Java programming language and class libraries written by the user. User profiles are then used to assist novice users in finding experts by matching her/his queries with user profiles. A group of researchers at MITRE has also developed an expertise recommendation system called Expert Finder [11],[12]. This system finds experts by performing a query over a MITRE wide corporate database that includes information about 4500 MITRE employees. The entries in the database are manually maintained by each individual employee. After performing the query, the system filters the results and presents a list of employees who are likely to have some expertise in the queried topic. Expertise Recommender is another system that recommend people who are likely to have expertise in a specific area [13],[14]. A user garners recommendation from ER by picking a relevant identification heuristic, selecting a matching technique, and entering a description or terms related to a problem. Then, the system responds with a list of individuals who are likely to have expertise with the problem and who are a good social match for the person making the request. In this system, user profiles are built by processing user‘s day-to-day work products. MARS is a referral system based on the idea of social network [13]. This system is fully distributed and includes agents who preserve the privacy and autonomy of their users. These agents build a social network learning models of each other in terms of expertise (ability to produce correct domain answers), and sociability (ability to produce accurate referrals), and take advantage of the information derived from such a social network for helping their users to find other users on the basis of their interests.

A lot of work has been also done in the development of tools and systems for supporting e-learning and, in particular, computer programming activities. Hazeyama and Osada realized a system for collaborative software engineering education [7]. This system provides both generic collaboration services, useful in all the different phases of students course project, and services dedicated to a specific phase of such a project. In fact, the system offers a bulletin board subsystem and a notification service used by students and teachers along all the project, and, for example, provides a subsystem supporting students code inspection process: this subsystem provides a tool that allows to a teacher the annotation of students code with comments, and manages the interaction between the teacher and the students in the different phases of the inspection process (i.e., code submission, teacher feedback, updated code submission, etc.). WBT (Web Based Teaching) is an agent based collaborative learning support system providing community Web services [8]. The system is centered on a Web site containing teaching materials for computer programming practice and an electronic bulletin board system for question answering to assist students during their programming practice activities. In this system agents have the duty of distributing questions to the teacher or to "on-line" students that previously answered to similar questions. Mungunsukh and Cheng proposed an agent based learning support system for novice programmers in a distance-learning environment [15]. This system is dedicated to the learning of the VLB programming language and its activity can be divided in two phases: student observation and student support. In the first phase, the system attempts to understand students' behavior by observing their typing events, behaviors on different purpose of web browser of lessons, tasks and examples, error types made by students and debugging events on a programming editor. After the acquisition of information about the activities of the students, the system supports students with relevant information as, for instance, related examples and lessons for the problems they are working on, and problems which have similar solutions. I-MINDS is a multi-agent system that enables students to actively participate in a virtual classroom rather than passively listening to lectures in a traditional virtual classroom [10]. This system is based on three kinds of agents: teacher agents, student agents and remote proxy agents. Teacher agents interact with teachers and are responsible for: i) disseminating information to student agents and remote proxy agents, ii) maintaining student profiles and, on the basis of these profiles generating individual quizzes and exercises, iii) filtering students questions, and iv) managing classroom sessions progress. Student agents support the interaction with the teacher, maintain the profiles of the other students to identify potential "helpers" and, when it is necessary, solicits answers from such "helpers". Remote proxy agents support the interaction with the teacher and other students when a student is connected

with a low-speed internet connection (e.g., they filters messages to reduce the traffic). Guardia Agent is an agent-based system aimed at supporting students working on team projects [22]. This system is based on agents, one for each student, that autonomously monitor the progress of a group project, suggest new ways in which the students can act to improve the progress of the project (e.g., a new allocation of tasks), and enhance the communication between members of the group.

IV. Conclusions

In this paper, we present a system called RAP (Remote Assistant for Programmers) with the aim of supporting communities of students and programmers during shared and personal projects based on the use of the Java programming language. RAP associates a personal agent with each user and this agent maintains her/his profile and helps her/him to solve problems proposing information and answers extracted from some information repositories, proposing "experts" on these problems and then forwarding their responses.

RAP has similarities with WBT [8], I-MINDS [10] and, in particular, with the Expert Finder system [21]. In fact, both these three systems provide agents that recommend possible "helpers". However, none of them provides the integration of different sources of information (experts, answers archive and code documentation), and none of them integrates in the user profile information about user's day-to-day work products with information obtained from the answers the user provided to the other users of the system.

A first prototype of the RAP System is under development by using JADE [3],[9], a software framework to aid the realization of agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems [6]. A large part of the first system prototype has been completed and some tests have been already done. In particular the tests regarding the recommendation of experts have shown encouraging results.

The RAP system will be used in some practical courses on JADE by students of the partners of the "@lis Technology Net" project and by students and researchers, involved in the ANEMONE project [2], for cooperating in the realization of agent-based software. Moreover, the RAP system will be used as a service of the Collaborator system [5]. Collaborator is a system that provides a shared workspace supporting the activities of virtual teams through a set of services as, for example, chat and multimedia interaction, meeting scheduling and synchronous sharing of applications [4].

After the completion, experimentation of the first prototype, we plan to try to improve the quality of both document and expert recommendation by applying and then comparing the most considered recommendation techniques and, eventually, trying their integration.

References

[1] @LIS Technet Home Page (2003). Available from http://www.alis-technet.org.

[2] ANEMONE Home Page (2003). Available from http://aot.ce.unipr.it:8080/anemone.

[3] Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA-compliant agent framework.. Software Practice and Experience, 31, (2001) 103-128.

[4] Bergenti, B., Poggi, A., Somacher, M.: A Collaborative Platform for Fixed and Mobile Networks. Communications of the ACM, 45(11), (2002) 39-44.

[5] Bergenti, B., Poggi, A., Somacher, M., Costicoglou, S.: COLLABORATOR: A collaborative system for heterogeneous networks and devices. In. Proc. International Conference on Enterprise System (ICEIS03), Angers, France (2003) 447-480.

[6] FIPA Specifications (1996). Available from http://www.fipa.org.

[7] Hazeyama, A., Nakako, A., Nakajima, S., Osada, K.: Group Learning Support System for Software Engineering Education - Web-based Collaboration Support between the Teacher Side and the Student Groups. In Proc. Web Intelligence 2001, Maebashi City, Japan, (2001) 568-573.

[8] Ishikawa, T., Matsuda, H., Takase, H.: Agent Supported Collaborative Learning Using Community Web Software. In Proc. International Conference on Computers in Education, Auckland, New Zealand, (2002) 42-43.

[9] JADE Home Page (1998). Available from http://jade.tilab.com.

[10] Liu, X., Zhang, X. Soh, L., Al-Jaroodi, J., Jiang, H.: I-MINDS: An Application of Multiagent System Intelligence to On-line Education. In Proc. IEEE International Conference on Systems, Man & Cybernetics, Washington, D.C., (2003) 4864-4871.

[11] Mattox, D., Maybury, M. and Morey, D.: Enterprise Expert and Knowledge Discovery. The MITRE Corporation, McLean, VA, (2000). Available from http://www.mitre.org/support/papers/tech_papers99_00/maybury_enterprise/maybury_enterprise.pdf

[12] Maybury, M., D'Amore, R. and House, D.: Awareness of Organizational Expertise. The MITRE Corporation, MacLean, VA (2000). Available from http://www.mitre.org/support/papers/tech_papers99_00/maybury_awareness/maybury_awareness.pdf

[13] McDonald, D.W.: Evaluating expertise recommendations. In Proc. of the 2001 International ACM SIGGROUP Conference on Supporting Group Work, Boulder, CO, (2001) 214-223.

[14] McDonald, D.W.: Recommending collaboration with social networks: a comparative evaluation. In Proc of the Conference on Human Factors in Computing Systems, Ft. Lauderdale, FL, (2003) 593-600.

[15] Mungunsukh, H., Cheng, Z.: An Agent Based Programming Language Learning Support System. In Proc. International Conference on Enterprise System (ICEIS02), Auckland, New Zealand, (2002) 148-152.

[16] Pazzani, M., Billsus, D.: Learning and revising user profiles: The identification of interesting web sites. Machine Learning, vol. 27, (1997) 313–331.

[17] Pazzani, M., Billsus, D.: Adaptive Web Site Agents. Autonomous Agents and Multi-Agent Systems, 5, (2002) 205–218.

[18] Resnick, P., Neophytos, I., Mitesh, S., Bergstrom, P., Riedl, J.: GroupLens: An open architecture for collaborative filtering of netnews. In Proc. Conference on Computer Supported Cooperative Work, Chapel Hill, (1994) 175-186.

[19] Salton, G.: Automatic Text Processing. (1989), Addison-Wesley.

[20] Sun Java Glossary (2004). Available from http://java.sun.com/docs/glossary.html.

[21] Vivacqua, A. and Lieberman, H.: Agents to Assist in Finding Help. in Proc. ACM Conference on Human Factors in Computing Systems (CHI 2000), San Francisco, CA, (2000) 65-72.

[22] Whatley J.: Software Agents for Supporting Student Team Project Work. In Proc. International Conference on Enterprise System (ICEIS04), Porto, Portugal, (2004) 190-196.

GrEASe: Grid Environment based on Agent Services

Antonio Boccalatte, Alberto Grosso, Christian Vecchiola, Sara Fazzari, Silvia Gatto

***Abstract*— This paper presents an agent-based infrastructure for grid computing called GrEASe (Grid Environment based on Agent Services). Grids are typically complex, heterogeneous, and highly dynamic environments, and agent technology can satisfy the basic requirements of this kind of contexts. GrEASe is organized as a two layer structure: the lower one providing the resource independent functionalities and the upper one providing all the grid-specific services. All the features of the grid infrastructure have been modeled with the multi-behavioral agent model of the AgentService programming framework. This platform is also the runtime environment for the multi-agent system associated with each grid node.**

***Index Terms*— Grid Computing, Multi-Agent Systems**

I. Introduction

RESOURCE sharing is nowadays an important issue, not only because it offers many advantages in distributed computing, but also because data sharing is becoming more and more useful in many fields. Resources can be classified in three different groups: data, services, and computational power. By following this classification we can distinguish three types of grids [1]. Data Grids manage huge collections of geographically distributed data, which can be generated in many different ways: data streams are daily sent from satellites for weather forecasts and climatic changes analysis; large collections of data generated from scientific experiments allow geographically distributed researchers to collaborate to the same research project. Service Grids provide services that could not be obtained from a single platform: streaming multimedia services or collaborative applications. Computational Grids provide the aggregate power of a collection of processors spread over the network as a unique, big processor. Grids are an economic and efficient way to compute, since they bring to the end user an incredible set of resources with a relatively low cost.

A Grid infrastructure is a complex and high dynamical environment: multiple, heterogeneous, and distributed resources need to be managed and accessed by means of a uniform interface. Real applications need also a customized interaction according to the different privileges of the users. The depicted scenario can certainly benefit from Agent technology [2]. Agents are autonomous software entities with some level of intelligence; agents work better if they belong to a community such as a multi-agent system (MAS) [3]. Agents act in a distributed manner, cooperate, compete, and negotiate to solve a problem or to perform a task. These features make the agents an interesting technology to implement Grid infrastructures.

In this paper GrEASe, an agent-oriented architecture which provides services in a Grid is described. GrEASe is implemented by the use of the AgentService programming platform [4].

A brief overview on agent technology and multi-agent systems is provided in Section II and how this technology can be applied to grid computing is explained. Section III includes the description of AgentService programming platform. Section IV the describes the features of GrEASe, while Section V presents an interesting use case of such architecture followed by a possible application of GrEASe to a real scenario. Conclusions follow in Section VI.

II. Agents Technology And grid computing

A. Agents and Multi-Agent systems

A software agent is an autonomous software entity able to expose a flexible behavior. Flexibility is obtained by means of reactivity, pro-activity and social ability [3]. Reactivity is the ability to react to environmental changes in a timely fashion while pro-activity is the ability to show a goal directed behavior by taking the initiative. Social ability, that is the ability to interact with peers by means of cooperation, negotiation, and competition, is one of the most important features of agent oriented programming: agents do their best when they interoperate. Interaction is obtained by arranging agents in communities called multi-agent systems (MAS) [3]. MAS are generally decentralized open systems with distributed control and asynchronous computation: they

Manuscript received October 25 2004

A. Boccalatte is with Department of Communication, Computer and Systems Sciences, University of Genova, 16145 Genova Italy (phone: +39-010-353-2812; e-mail: nino@ dist.unige.it).

A. Grosso is with Department of Communication, Computer and Systems Sciences, University of Genova, 16145 Genova Italy (phone: +39-010-353-2284; e-mail: nino@ dist.unige.it).

C. Vecchiola is with Department of Communication, Computer and Systems Sciences, University of Genova, 16145 Genova Italy (phone: +39-010-353-2284; e-mail: nino@ dist.unige.it).

S. Fazzari was with Department of Communication, Computer and Systems Sciences, University of Genova, 16145 Genova Italy.

S. Gatto was with Department of Communication, Computer and Systems Sciences, University of Genova, 16145 Genova Italy.

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

provide a context for agents' activity with the definition of interaction and communication protocols. In addition they are scalable, fault-tolerant, reliable, and designed for reuse.

An abstract architecture specification of a generic multi-agent system has been proposed by the Foundation of Intelligent Physical Agents (FIPA), an international organization that promotes standards for agent technologies. The proposed architecture [6] is implemented by different multi-agent systems and has been taken as reference model in the comparison of different implementations of MAS.

B. Agents and Grid Computing

Agent technology has been a useful approach in different contexts: air traffic management [5], biologic systems modeling and simulation [7], workflow management [8], and on-line auction systems [9]. Moreover, different fields of computing have taken advantages from the agent oriented approach such as scheduling systems, collaborative smart agendas [10], information filtering [11], and soft-bots [12]. Agents are reliable components to build more flexible and fail safe systems, since autonomy and reactivity allow recovering from fault conditions. This is certainly necessary in high critical scenarios like air traffic management, but it is also a desirable in the case of grid computing. The social ability, such as cooperation, competition and negotiation, is equally fundamental in grids.

Grids are intrinsically distributed and complex systems, as they may require more than one step to provide a resource to a client. Interactions between nodes can change during time in order to make use of resources available at run time. Each node belonging to a grid needs to keep availability of the resources offering and benefits of a certain degree of autonomy and flexibility. Agent technology has been designed to model high dynamic and complex systems [13] and can fulfill many of the requirements related to the development of a grid infrastructure. By using agent technology, users and administrators of the resulting system can have a more friendly and understandable interface to interact with.

Some projects have already proved that the agent oriented approach could be an interesting solution in the field of Grid Computing.

A4, acronym for "Agile Architecture and Autonomous Agent" is a methodology for grid's resource managing. This approach, described in depth in [14] [15], is based on a flexible architecture, able to rapidly adapt to dynamic environmental changes. Agents are homogeneous and settled in a hierarchical structure, they have capabilities of service discovery and service advertisement.

MyGrid [16] is a Grid project which provides a collaborative environment for biologists working and living in different countries. The architectural design is based on agents and exploits their autonomy and their capability to implement complex interactions through negotiation messages in a generic Agents Communication Language (ACL). MyGrid relies on SoFAR (Southampton Framework for Agent Research) [17], that constitutes the agent oriented infrastructure used by MyGrid.

The Bond Agent System [18] is based on the JADE framework [19] and extends it by providing specific agent behaviours that abstract the concept of grid services.

The agent-oriented approach can take many advantages to field of Grid Computing. In particular it offers a flexible and high level approach that is, at the same time, powerful enough to handle all the different aspects of grid environments. Grids are dynamics by nature and agents have been modeled in order to get aware of the context in which they are situated and to dynamically interact with peers. Agents are also high level interfaces for humans, if compared to objects, and system designers can easily deal with them and organize the entire distributed system in a more clear way. All the presented projects rely on these features of agency and also GrEASe takes benefits from them by the means of the AgentService programming platform.

III. THE AGENTSERVICE PROGRAMMING PLATFORM

AgentService [4] is a multi-agent system development framework that provides a complete support to agent design, implementation, and management with a full run-time environment for agents scheduling, control and monitoring.

The framework has been developed with an extremely modular architecture in order to be customizable and portable over different architectures and operating systems. Modules cover:

- the storage subsystem (repository of all templates used to create agents in the platform);
- the persistence subsystem;
- the messaging subsystem;
- the logging subsystem.

Additional modules can be loaded into the platform in order to enrich and customize the platform services.

AgentService allows the definition of real, autonomous, and persistent agents. Agents have a multi-behavioral activity and organize their knowledge base in a set of persistent data structures shared among the different activities. Agents are scheduled and executed within the AgentService platform that provides them a set of services as defined by the FIPA2000 specification [6]:

- Agent Management System (AMS);
- Directory Facilitator (DF);
- Message Transport Service (MTS).

The agent model implemented in AgentService is based on the use of behaviors and knowledge. Behaviors include decisions and computational tasks; they dynamically determine the agent activity and influence its state.

Knowledge objects define the agent's knowledge base and consist of a set of shared data structures that can be persisted in order to preserve the agent's state and that are modified by the activity of Behavior objects.

AgentService provides to the developer a set of Agent Programming eXtensions (APX) [20] specifically designed to simplify the development and the implementation of agent oriented applications; they are a set of templates modeling the implementation of agents, behaviors and knowledge, represented as types in a C#-based programming language.

IV. GREASE ARCHITECTURE

A. Overall Overview

GrEASe (Grid Environment based on Agent Services) is an agent oriented infrastructure for grid computing. GrEASe architecture is structured in two layers: the lower one providing the basic services common to every grid, the upper one providing grid-specific functionalities. Both layers have been modeled by using an agent-oriented approach.

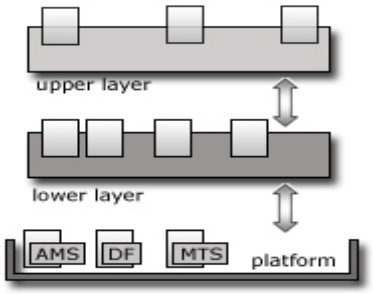


Figure 1 – Structure of a GrEASe node

B. The Lower Layer: Basic Infrastructure

The lower layer provides the basic grid functionalities classified as follows:

- node management: grids are dynamic environments where nodes can subscribe and unsubscribe at run-time. General information about the node status need to be accessed in order to provide the necessary interfaces to monitor and maintain the entire grid;
- resource querying and discovery: nodes can query and find resources by using a distributed dispatching system spread all over the nodes;
- authentication: users that access the resources need to be authenticated, since different policies are applied depending on the identity of the requestor;
- transport services: dispatch and receipt of the information and data.

The design and implementation of the lower layer led to the definition of different types of agents: NodeManager, Dispatcher, Authenticator, and Carrier.

NodeManager is the maintainer of the node and performs all the management operations. Three different behaviours have been designed in order to accomplish all the tasks of the NodeManager:

- node subscription and un-subscription from the Grid;
- monitoring services and information about the status of the node and of the resources;
- resource allocation and monitoring.

Dispatchers agents are spread all over the nodes and implement the resource querying and discovery process: each node has an instance of this type of agent. Dispatcher essentially forwards a request for a resource to the neighbor nodes, and waits for a response; at the same time it handles incoming requests from other dispatcher agents. Dispatcher is critical for performance of the resource search process and can support different search algorithms by simply changing the relevant behaviors.

Carrier agents implement the general file transfer service between nodes: agents inside the node instruct Carrier to send a file or are notified by the Carrier of an incoming file transfer for them. Different protocols (e.g. ftp protocol, or its secure version) can transparently be used to implement file transfer service, by defining the corresponding behaviors and selecting the most fitting ones for the specific context.

Authenticator agents are responsible of the user authentication process. The user profile is evaluated in order to grant:

- access to the grid system;
- access to the specified resource by applying the right policy.

Authenticator implements a two-level authentication strategy: first the credentials provided by the user are checked for the access to the grid system; then the availability of the resource is granted on the basis of the successful validation of the authorization criteria.

C. The Upper Layer: Grid-specific Components

Different types of grids are defined according to the different types of resources they share: processor-cycles, documents and data in general, or services. Therefore, specific requirements need to be fulfilled according to the different grid types. Resources of data-grids should be accessed at the same time by multiple clients. Conversely, in a computational-grid resources can be assigned only to a single client at time since the same processor cycles cannot be shared between multiple users.

The upper layer of the GrEASe architecture takes care of all the peculiar features related to the specific type of the chosen grid. The upper layer is defined by all those agents that strictly interact with the resources belonging to the grid and hosted in the node. For example let's define agents' behaviors according to the requirements of Computing Grids: the submission of a task to a node for computing means not only the transfer of the executable code of the task, but also the transfer of the requested input and output data. In addition, if tasks are not monolithic it may be convenient to monitor their progress. These functionalities can be encapsulated by the implementation of specific run-time behaviors, one for

handling the task execution, another for monitoring task progress. A similar approach can be adopted for Data and Service Grids.

V. GREASE IN ACTION

In order to see how GrEASe agents interact to provide a grid service the process of resource querying and discovery will be briefly described.

Figure 2 shows an instance of the AgentService platform running on each grid node (two expanded) that schedules resource agents and infrastructure agents. A client application asks NodeManager of the nearest node by providing user credentials to access the grid. NodeManager forwards the credentials to the co-located Authenticator and waits for feedback. Authenticator verifies the user credentials and in case of success sends an approval message to the user and notifies the NodeManager, which updates the user login status.

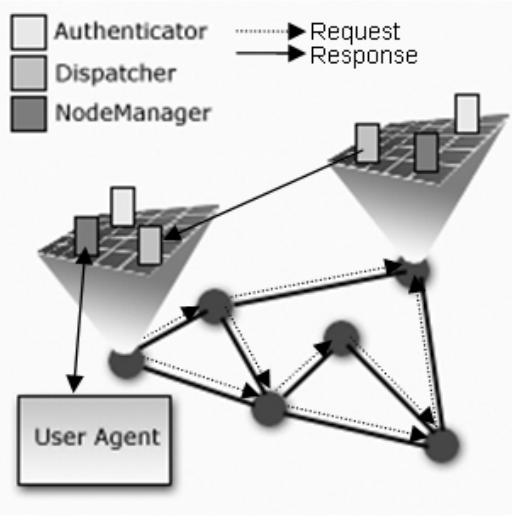


Figure 2 – Resource querying process

Once authenticated the users asks the NodeManager for a given resource. The NodeManager checks the resource availability on its node: if the resource is found it notifies the user, and following the user's confirmation, it instructs Carrier to dispatch the resource; if the resource is not available in the node, NodeManager forwards the request to the co-located Dispatcher who will distribute the request to others Dispatcher agents across the network, according to the selected resource search algorithm. Every Dispatcher reports the query to its NodeManager and the same process described before applies. If no resources are found in the grid, a time-out function associated to the query makes the query inactive. If more than one node answers that the resource is available, the user asks to send only the first answering node. All the messages across the nodes use the address provided by Directory Facilitator.

Knowledge objects used in this process are:

- Grid topology by Dispatcher;
- User credentials by Authenticator;
- Resource availability by NodeManager;
- Logged users by NodeManager.

The architecture of GrEASe is flexible enough to handle all the different scenarios of Grid Computing. An interesting application of GrEASe can be found in the modeling and the simulation of the peer-to-peer (p2p) nets for sharing data: such networks can be considered a sort of Data Grids:

- they provide to the end user a huge volume of data that is spread all over the network;
- the end user access all the data available in the network and the this access is independent from the physical location of data;
- nodes of the net can act either as servers for other nodes or as clients that feed data.

There are some aspects that make Data Grids different from peer-to-peer networks:

- peer-to-peer networks do not implement sophisticated access control techniques and do not have refined user profiles;
- peer-to-peer networks normally provide many different copies of the same data and do not worry about the synchronization of the different copies.

These aspects make peer-to-peer networks only less complex than Data Grids.

By the use of GrEASe it is possible to model each node of the network with an installation of the AgentService platform that runs the agents defined by the GrEASe architecture. The NodeManager will be responsible for the local resources of the node, while the Dispatcher and Carrier will be programmed in order to interact with peers also by using the most known p2p protocols: in this way the nodes of the GrEASe architecture can easily be integrated with the already existing p2p networks. Since peer-to-peer networks normally have simple user policies the Authenticator will provide only the basic functionalities of authentication when needed.

In peer-to-peer networks resources normally refer to simple files and for this reason there is no need to define particular agents that represents the resources inside the platform. The upper layer will be configured with particular agents:

- if the node is attached to an end user the upper layer will require a user-agent that handles the user requirements and control the behavior of the node for the user;
- in the case of the node is intended to measure and monitor the traffic that flows through it a special agent can be designed to track all this kind of information and report it to the user;

The introduction of the agents into the upper layer is rather simple because agents rely on the platform services to perform their activity: by querying the Directory Facilitator can dynamically discover the NodeManager; each agent uses the

message based system provided by the MTS to interact with the other agents and they can easily interact with the other "citizens" of the platform once they know the ontology that NodeManager, Dispatcher, Carrier, and Authenticator support. These ontologies are made available to each agent by the platform through the Directory Facilitator.

The architecture provided with GrEASe, the services offered by the AgentService platform, and the approach defined with the agent-oriented paradigm, allow a quick and not difficult implementation of the described example. In fact, designers can better concentrate on the peculiar aspects of the example rather than define the overall infrastructure and the programming model needed to implement the example.

VI. Conclusion and Future Work

Agent technology and in particular agent oriented decomposition has played a key role in the design and the implementation of GrEASe. The division of the tasks to the different types of agents has led to a flexible and customizable architecture. Interaction between agents is done with clean and fixed interfaces defined by the messages they exchange and this allows loose coupling among the different components.

The approach taken with GrEASe is different from the ones adopted by the other similar projects like A4 and the Bond Agent System: while A4 leverages on a hierarchical structure used to organized the resources in the grid, GrEASe adopts a two-level architecture that separates the features common to all the grid types from the features peculiar to the specific grid type. Moreover, GrEASe is not tied, as in the case of the Bond Agent System, to a strong BDI architecture but the use of AgentService allows a more open environment.

The architecture provided with GrEASe, the services offered by the AgentService platform, and the approach defined by the agent-oriented paradigm offer to developers a basic set of functionalities. In particular, the agent oriented approach and the fact that agents live inside a multi-agent system that relies on the services of a platform, are an important abstraction on which the GrEASe architecture is founded. GrEASe exploits the services of AgentService in order to deliver to the developer an high-level tool to model real applications in the field of Grid Computing. A simple example has been discussed in order to show to the user that the approach promoted by GrEASe can be an interesting solution.

Currently GrEASe implements the resource search and delivery in the three common grid types: Data, Computational and Service. The most natural expansion of future development is to allow the interaction between GrEASe and other existing legacy grids. In this case AgentService will be used to shape Interface Agents to access legacy grids in accordance to their individual established rules.

References

[1] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid. Enabling Scalable Virtual Organizations", *International Journal of Supercomputer Applications*, 2001.

[2] N.R. Jennings, and M. Wooldridge "Agent-Oriented Software Engineering", *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, 1999.

[3] G. Weiss, *Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence*, G. Weiss Ed., Cambridge, MA, 1999.

[4] A. Boccalatte, A. Gozzi, and A. Grosso, "Una Piattaforma per lo Sviluppo di Applicazioni Multi-Agente", *WOA 2003: dagli oggetti agli agenti – sistemi intelligenti e computazione pervasiva*, Villa Simius, Italy, September 2003.

[5] A. S. Rao, M. P. Georgeff, and D. Kinny, "A Methodology and Modelling Technique for Systems of BDI Agents Agents Breaking Away", *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (MAAMAW'96)*, published by Springer as Lecture Notes in Artificial Intelligence 1038, 1996.

[6] "FIPA Abstract Architecture Specification", FIPA standard SC00001L, http://www.fipa.org/specs/fipa00001/SC00001L.pdf.

[7] H. Van Dyke Parunak, "Go to the Ant: Engineering Principles from Natural Multi-Agent Systems", Forthcoming in Annals of Operations Research, special issue on Artificial Intelligence and Management Science.

[8] R. Sacile, E. Montaldo, M. Coccoli, M. Paolucci, and A. Boccalatte, "Agent-based architectures for workflow management in manufacturing", SSGRR 2000, L'Aquila I, Aug. 2000.

[9] C. Beam, and A. Segev, "Automated Negotiations: A Survey of the State of the Art", *Wirtschaftsinformatik*, Vol. 37-3, 1997, pp. 263-268.

[10] B. P. C. Yen, "Agent-based Distributed Planning and Scheduling in Global Manufacturing", in *Proc. of the thrid Annual International Conference on Industrial Engineering Thories, Applications and Practice*, Hong Kong, December, 2000

[11] P. Maes, and A. Moukas, "Amalthaea: An Evolving Multi-Agent Information Filtering and Discovery System for the WWW", *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 1, 1998, pp. 59-88.

[12] O. Etzioni, and D. Weld, "A Softbot-Based Interface to the Internet", *Communications of the ACM*, 37, 7, 1994.

[13] M. Wooldridge, "Intelligent Agents", in *Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence*, G. Weiss Ed., Cambridge, MA, 1999, pp. 27-78.

[14] J. Cao, D. P. Spooner, J. D. Turner, S. A. Jarvis, D. J. Kerbyson, S. Saini, and G. R. Nudd, "Agent-Based Resource Management for Grid Computing", in *Proc. of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, 2002.

[15] J. Cao, D. J. Kerbyson, G. R. Nudd, "Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing", in *Proc. of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '01)*, Brisbane, Australia, May 2001.

[16] L. Moreau, S. Miles, C. Goble, M. Greenwood, V. Dialani, M. Addis, N. Alpdemir, R. Cawley, D. De Roure, J. Ferris, R. Gaizauskas, K. Glover, C. Greenhalgh, M. Greenwood, P. Li, X. Liu, P. Lord, M. Luck, D. Marvin, T. Oinn, N. Paton, S. Pettifer, M. V Radenkovic, A. Roberts, A. Robinson, T. Rodden, M. Senger, N. Sharman, R. Stevens, B. Warboys, A. Wipat, and C. Wroe, "On the Use of Agents in a BioInformatics Grid", in *Proc. of the Third IEEE/ACM CCGRID'2003 Workshop on Agent Based Cluster and Grid Computing*, Sangsan Lee, Satoshi Sekguchi, Satoshi Matsuoka, and Mitsuhisa Sato ed., Tokyo, Japan, May 2003, pp 653-661.

[17] L. Moreau, N. Gibbins, D. DeRoure, S. El-Beltagy, W. Hall, G. Hughes, D. Joyce, S. Kim, D. Michaelides, D. Millard, S. Reich, R. Tansley, and M. Weal, "SoFAR with DIM Agents: An Agent Framework for Distributed Information Management", in *Proc. Of The Fifth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, Manchester, UK, Apr 2000, pp. 369–388

[18] M.A. Khan, S.K.Vaithianathan, K. Sivoncic, and L. Boloni, "Towards an Agent Framework For Grid Computing", *CIPC-03 Second International Advanced Research Workshop on Concurrent Information Processing and Computing*, Sinaia, Romania, 2003.

[19] F. Bellifemmine, G. Rimassa, and A. Poggi, “JADE – A FIPA compliant Agent Framework”, in *Proc. of the 4th international Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, London, 1999.

[20] A. Boccalatte, C. Vecchiola, and M. Coccoli, “Agent Programming Extensions relying on a component based platform”, in *Proc. of the 2003 IEEE International Conference on Information Reuse and Integration*, Las Vegas, NV, October 2003, pp. 24-31.

Design and development of a visual environment for writing DyLOG programs

Claudio Schifanella, Luca Lusso, Matteo Baldoni, Cristina Baroglio
Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
Tel. +39 011 6706711 — Fax. +39 011 751603
E-mail: {schi,baldoni,baroglio}@di.unito.it, lussoluca@tiscali.it

***Abstract*— In this article we present a visual development environment for writing DyLOG programs, explaining the motivations to this work and the main design choices. We will also analyze the main components of the system and the features offered to the user. The visual environment encompasses a fully new implementation of the DyLOG language, where Java is used instead of Sicstus Prolog, and an OWL ontology that will allow the use of the language in Semantic Web applications.**

I. Introduction

Engineering multi-agent systems (MASs) is a difficult task; one of the ways for achieving the successful industrial deployment of agent technology is to produce tools that support the developer in all the steps of design and implementation. Many researchers in the Agent Oriented Software Engineering (AOSE) community are developing complete environments for MAS design. Just to mention a few examples, AgentTool [1] is a Java-based graphical development environment to help users analyze, design, and implement MASs. It is designed to support the Multiagent Systems Engineering (MaSE) methodology [2], which can be used by the system designer to graphically define a high-level system behavior. Zeus [3] is an environment developed by British Telecommunications for specifying and implementing collaborative agents. DCaseLP (Distributed CaseLP, [4], [5], [6]) integrates a set of specification and implementation languages in order to model and prototype MASs. In this scenario, the quality of the tools that the designer can use strongly influences the *choice* of a given *specification language*. The availability of a visual environment that is intuitive to use, and simplifies the design of the agents in the system, can, actually, make the difference.

In this paper we present a visual environment (VisualDyLOG) for the development of DyLOG agents. DyLOG is a logic language for programming agents, based on reasoning about actions and change in a modal framework [7], that allows the inclusion, in an agent specification, also of a set of communication protocols. In ([8]) is proposed a methodological and physical integration of DyLOG into DCaseLP in order to reason about communication protocols.

By using VisualDyLOG, the user can specify in a simple and intuitive way all the components of a DyLOG program by means of a visual interface. The adoption of such an interaction device bears many advantages w.r.t. a text editor [9] and allows the programmer to work at a more abstract level, skipping the syntactical details of the language. Moreover, it is important to notice that the learning curve of logic languages is usually quite steep: the programming environment supplied by VisualDyLOG aims also at solving this problem.

An interesting application domain for agents develeoped by means of these tools is the Web, and in particular in the *Semantic Web*. Indeed, the web is more and more often considered as a means for accessing to interactive *web services*, i.e. devices that can be retrieved, invoked, composed in an automatic way. To this aim, there is a need for languages that allow web service specification in a well-defined way, capturing what the services do, how they do it, which information they need for functioning and so on, in order to facilitate the automatic integration of heterogeneous entities. Recently some attempt to standardize the description of web services has been carried on (DAML-S [10], OWL-S [11], WSDL [12]). While the WSDL initiative is mainly carried on by the commercial world, with the aim of standardizing registration, look-up mechanisms and interoperability, OWL-S (and previously, DAML-S) is more concerned with providing greater expressiveness to service descriptions in a way that can be *reasoned* about [13], by exploiting the *action metaphor*. In particular, we can view a service as an action (atomic or complex) with preconditions and effects, that modifies the state of the world and the state of agents that work in the world. Therefore, it is possible to design agents, which apply techniques for reasoning about actions and change to web service descriptions for producing new, composite, and customized services. These researches are basically inspired by the language Golog and its extensions [14], [15], [16]. In previous work, we have studied the use of DyLOG agents in the Semantic Web, and in particular, we have described the advantages that derive from an explicit representation of the *conversation policies* followed by web services in their description (currently not allowed by OWL-S). Actually, by reasoning on the conversation policies it is possible to achieve a better personalization of the service fruition [17], and it is also possible to compose services [18]. This research line has driven us to the implementation of an OWL [19] ontology, to be used as an interchange format of DyLOG programs, with the purpose of simplifying the use and the interoperation of DyLOG agents in a Semantic Web

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

context.

The paper is organized as follows. Section II is a very brief introduction to the main characteristics of the DyLOG language. Section III describes the developed editing environment while Section IV describes the developed OWL ontology and motivates the choice of the OWL language. Conclusions follow.

II. THE DyLOG LANGUAGE

Logic-based executable agent specification languages have been deeply studied in the last years [20], [21], [15]. In this section we will very briefly recall the main features of DyLOG; the interested reader can find a thorough description of this language in [22], [23].

DyLOG is a high-level logic programming language for modeling rational agents, based on a modal theory of actions and mental attitudes where *modalities* are used for representing *actions*, while *beliefs* model the agent's internal state. It accounts both for *simple* (atomic) and *complex actions*, or procedures. Atomic actions are either world actions, affecting the world, or mental actions, i.e. sensing and communicative actions producing new beliefs and then affecting the agent mental state. Atomic actions are described in terms of *precondition laws* and *action laws* that, respectively, define those conditions that must hold in the agent mental state for the action to be applicable, and the changes to the agent mental state that are caused by the action execution. Notice that besides the preconditions to a simple action execution, some of its effects might depend upon further conditions (*conditional effects*). Complex actions are defined through (possibly recursive) definitions, given by means of Prolog-like clauses and by action operators from dynamic logic, like sequence ";", test "?" and non-deterministic choice "$\cup$". The action theory allows coping with the problem of reasoning about complex actions with incomplete knowledge and in particular to address the temporal projection and planning problem in presence of sensing and communication.

Intuitively, DyLOG allows the specification of rational agents that reason about their own behavior, choose courses of actions conditioned by their mental state and can use sensors and communication for obtaining new information. The agent behavior is given by a *domain description*, which includes a specification of the agents initial beliefs, a description of the agent behavior plus a communication kit (denoted by CKit^{ag_i}), that encodes its communicative behavior. Communication is supported both at the level of *primitive speech acts* and at the level of *interaction protocols*. With regards to communication, a mentalistic approach, also adopted by the standard FIPA-ACL [24], is taken, where communicative actions affect the internal mental state of the agent. Some authors [25] have proposed a *social approach* to agent communication [25], where communicative actions affect the "social state" of the system, rather than the internal states of the agents. Different approaches are well-suited to different scenarios. DyLOG is a language for specifying an *individual, communicating agent*, situated in a multi-agent context. In this case it is natural to have access to the agent internal state.

We introduce an example that will be used in the rest of the paper, in order to better explain concepts. More details are included in the original work [26]. Let us consider the example of a robot which is inside a room. Two air conditioning units can blow fresh air in the room and the flow of the air from a unit can be controlled by a dial. In the following we report the code of the Simple Action *turn_dial(I)* that turns the dial of the unit *I* clockwise from a position to the next one. $\mathcal{B}_{robot}$ and $\mathcal{M}_{robot}$ are written as $\mathcal{B}$ (*belief*) and $\mathcal{M}$ (dual of $\mathcal{B}$) for simplicity.

A. A DyLOG implementation

At the basis of the development of the DyLOG programming environment there is a Java reimplementation of the language and of its interpreter. The choice of this implementation language is due to the great diffusion of it, to its well-known portability, and to the huge amount of available applications and frameworks.

The first step consisted in the development of the tuDyLOG package, which implements all the DyLOG constructs, offering to the programmer a set of classes and interfaces which allow the creation and editing of programs. tuDyLOG has been built upon *tuProlog* [27], a light-weight Prolog engine written in Java, which allows the instantiation and the execution of Prolog programs by means of a minimal interface. In this way, it is possible to exploit some of the mechanisms, made available by the *tuProlog* engine, which are used both in DyLOG and in Prolog, such as *unification*. Moreover, *tuProlog* supports interactions based on TCP/IP and RMI, a useful feature to the design of multi-agent systems. The implementation of the tuDyLOG package is currently being completed by extending the *tuProlog* engine so to obtain a tuDyLOG inference engine.

The structure of the classes which implement the language constructs follow the definition of a DyLOG program. A program is an instance of the class *DomainDescription*, which contains instances of the main kinds of program components: a set of initial observations, a set of actions that define the behavior of the agent, and a set of communicative actions. Each of such categories is represented by an adequate taxonomy, that reproduces the language specifications and offers a programming interface for operations like creation, modification, and deletion.

The connecting point between tuDyLOG and *tuProlog* is the class *DyLOGStruct*, an extension of the *tuProlog Struct* class: by means of *DyLOGStruct* every DyLOG construct can be turned into a corresponding *tuProlog* structure, with the possibility of exploiting the afore mentioned mechanisms. In this case we use a different notation (prefix notation) in order to meet the internal representation of the *tuProlog Struct* class. For example the first Precondition Law mentioned above is represented in this manner:

possible(turn_dial(I),if([belief(robot,in_front_of(I)),

$\Box(\mathcal{B}in\_front\_of(I) \wedge \mathcal{B}cover\_up(I) \supset \langle turn\_dial(I)\rangle\top)$
$\Box(\mathcal{B}flow(I, low) \supset [turn\_dial(I)]\mathcal{B}flow(I, high))$
$\Box(\mathcal{M}flow(I, low) \supset [turn\_dial(I)]\mathcal{M}flow(I, high))$
$\Box(\mathcal{B}flow(I, high) \supset [turn\_dial(I)]\mathcal{B}flow(I, off))$
$\Box(\mathcal{M}flow(I, high) \supset [turn\_dial(I)]\mathcal{M}flow(I, off))$
$\Box(\mathcal{B}flow(I, off) \supset [turn\_dial(I)]\mathcal{B}flow(I, low))$
$\Box(\mathcal{M}flow(I, off) \supset [turn\_dial(I)]\mathcal{M}flow(I, low))$
$\Box(\mathcal{B}flow(I, P) \supset [turn\_dial(I)]\mathcal{B}\neg flow(I, P))$
$\Box(\mathcal{M}flow(I, P) \supset [turn\_dial(I)]\mathcal{M}\neg flow(I, P))$

Fig. 1. The DyLOG code for the simple action *turn_dial(I)*.

belief(robot,cover_up(I))]))

III. VISUAL DYLOG

In this section we will show the main characteristics and features offered by VisualDyLOG. This environment is developed in Java using the *Eclipse* platform [28] and allows the development of a DyLOG program by means of a graphical user interface.

A. The Eclipse project

Eclipse is a platform designed for building integrated development environments (IDEs) and it represents a proven, reliable, scalable technology upon which applications can be quickly designed, developed, and deployed. More specifically, its purpose is to provide the services necessary for integrating software development tools, which are implemented as Eclipse plug-ins. The main characteristic of the design of Eclipse is, actually, that –except for a small runtime kernel– everything is a plug-in or a set of related plug-ins: the effect of this choice is an increase of the software reusability and extendability. Applications are deployed and distributed as stand-alone tools using the Rich Client Platform [29], which represents the smallest subset of Eclipse plug-ins that are necessary to build a generic platform application with a user interface.

Today Eclipse (originally released by IBM) is one of the most used platforms for developing applications, it has formed an independent and open eco-system, based on royalty-free technology, and it has established a universal platform for tools integration.

B. The environment

The VisualDyLOG environment is represented in Figure 2. We can distinguish different areas, each characterized by specific functionalities. With reference to the mentioned figure, area number (1) (the *Program View*) contains a whole view of the DyLOG program; it shows all the instances of the various constructs, kind by kind, and it also allows the creation and deletion of the instances. The area number (2) (*Editor View*) shows a visual representation of an instance contained in the Program View and selected by the user. The property values of such an instance are reported in the *Properties View* (area (4)). By working in the two latter views, the user can edit the selected instance. Since instances might in some cases be quite complex, there are situations in which the Editor View might show just a portion of the selected instance. Nevertheless a miniaturized overview of the whole instance will always be available in area number (3) (the *Outline View*). Last but not least, log messages are printed in the so called *Log View* (area number (5)).

VisualDyLOG internal architecture is based on the Graphical Editor Framework (GEF), an Eclipse plug-in. GEF, by exploiting the Model-View-Controller pattern, allows the creation of a graphical representation, given an existing data model. In our application, such a model is given by the instances of the package tuDyLOG, explained in Section II-A. In particular, by means of GEF:

- the graphical representation is modified after a change in the model has occurred;
- the model is changed by modifying the graphical representation of it, exploiting the "event-action" paradigm.

These notions are sketched by Figure 3. In Figure 4, instead, the *graphical notation* used to represent the main language constructs are shown.

It has been designed so to make the use of VisualDyLOG more intuitive: similar constructs are represented by shapes with the same morphology; such shapes recall flow chart symbols, contributing to a reduction of the learning process of new users.

C. An example of use

In this section we will show how to build a Simple Action by means of VisualDyLOG; in particular, we will use as an example the *turn_dial* action, whose DyLOG definition has been introduced in Section II (see Figure 1). The first step for creating a Simple Action consists in selecting the appropriate category from the Program View, and in assigning to it name and arity (the Program View also allows the creation of a new *action law* for a specific simple action). The new action will be added to the set of available Simple Actions in the Program View itself. By means of the Editor View, instead, it is possible to specify all the characteristics of the action: a working area is associated to each precondition law and to each action law that make the just created simple action; each such area can be selected and worked upon by clicking on a tab at the bottom of the Editor View. The palette at the right of the Editor View can be used to insert beliefs and terms in the working area. In order to edit a component it is necessary to click on the corresponding graphical representation of it and, then, modify

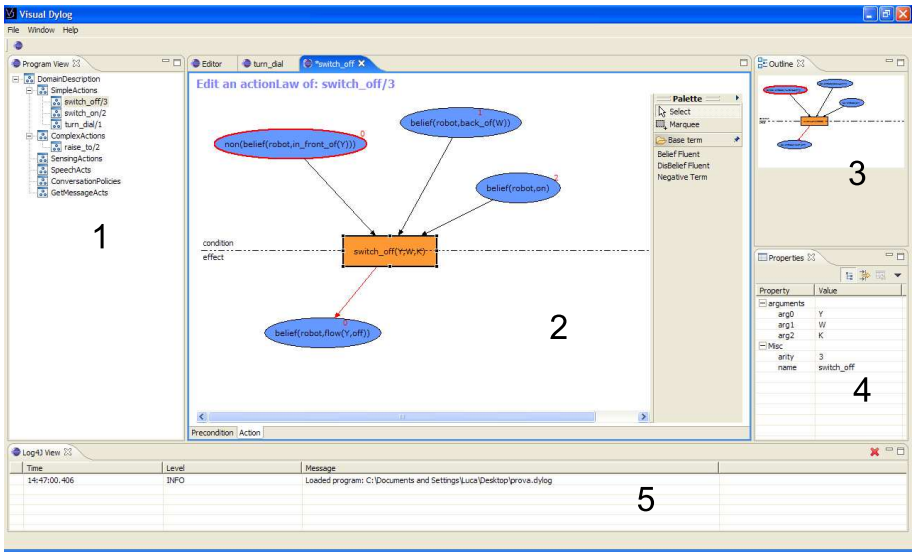


Fig. 2. A screenshot of the VisualDyLOG environment: (1) the Program View, (2) the Editor View, (3) the Outline View, (4) the Properties View, and (5) the Log View.

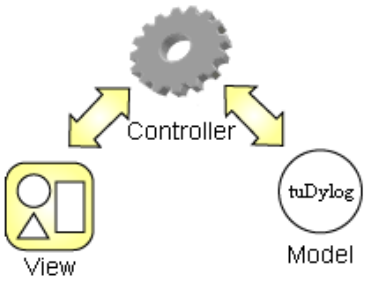


Fig. 3. GEF interaction model: the *Model*, in our case the tuDyLOG package, the *View* and the *Controller*, represented by GEF.

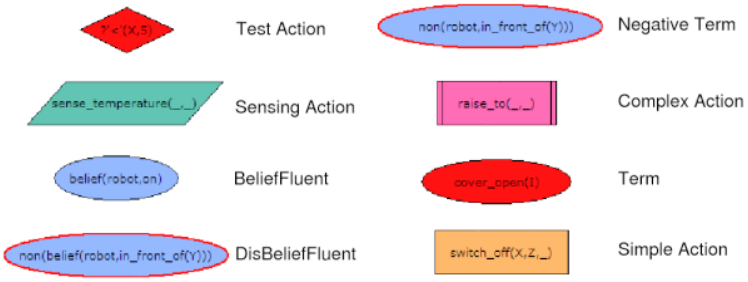


Fig. 4. The graphical notation used in VisualDyLOG

its properties by means of the Property View. In Figure 5 the working area used to create and modify the Precondition of the above mentioned action is shown. In the example *turn_dial* precondition consists of two fluents: the robot must be in front of the dial ($\mathcal{B}in\_fornt\_of(I)$) and the cover of the dial must be open ($\mathcal{B}cover\_up(I)$). In the figure, they are represented as light blue ovals. For the sake of simplicity, in the DyLOG representation of Figure 1 the agent name is omitted from the fluents. In the graphical representation, instead, it is the first argument of the fluents: since agents have a subjective view of the world, $robot$ is the agent to believe that $in\_front\_of(I)$ and $cover\_up(I)$ in order to execute the action. Notice also the prefix notation of fluents.

In Figure 6 the part of the interface devoted to the handling of one of the Action Laws is shown. The just described interaction schema is used also for the creation and editing of the other constructs of the language, such as complex actions (Figure 7), sensing actions, speech acts, and so forth.

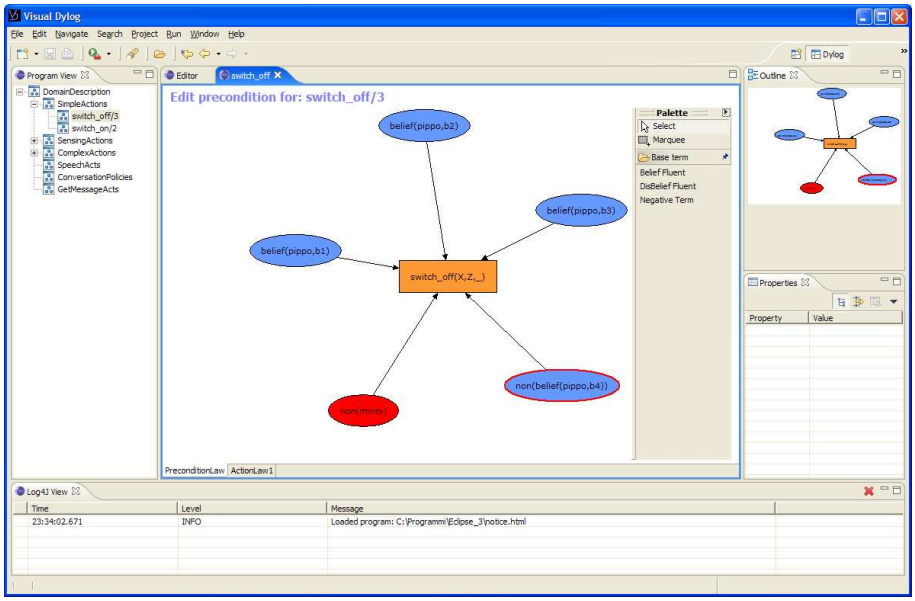


Fig. 5. Representation of a precondition law: beliefs are represented as light blue ovals, disbeliefs as blue ovals with a red border, terms as red ovals, while the action name is depicted as an orange rectangle.

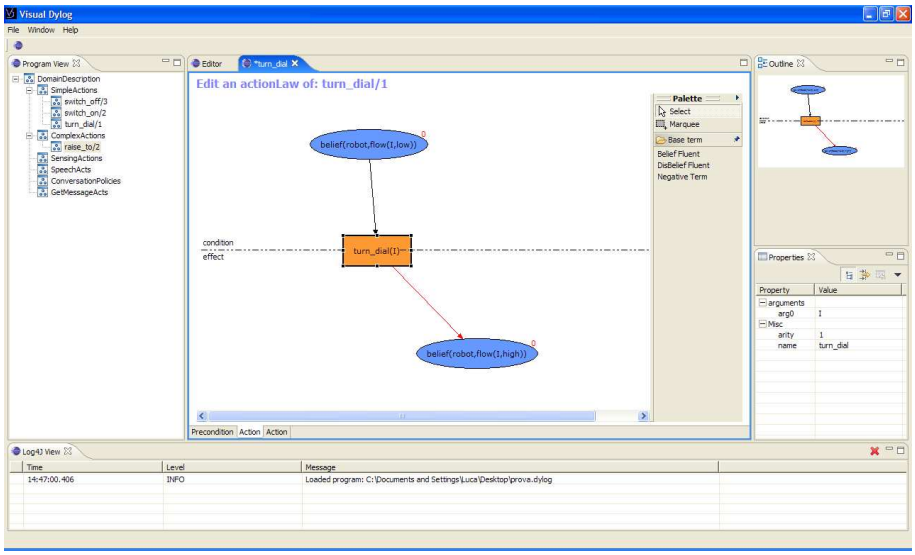


Fig. 6. Representation of an action law: the middle line divides preconditions to the effects from the action's effects themselves.

IV. An OWL ontology for DyLOG

In parallel with the work aimed at developing a programming environment for the language, we have also developed an ontology (called DyLOG Ontology) to be used for Semantic Web applications and, in particular, in the case of Semantic Web Services. We have already shown, in previous work, how the action metaphor and the mechanisms for reasoning about actions and change can fruitfully be exploited in many Semantic Web application frameworks [30], such as in educational applications and for the composition of web services. In order to allow the development of real applications over the web, there was a need of representing DyLOG programs in a way that is compatible with the infrastructure of the Semantic Web. Hence the choice of defining an OWL ontology.

OWL is a Web Ontology language [19], developed by the W3C community. The main characteristic of this language w.r.t. earlier languages, used to develop tools and ontologies for specific user communities is that it is compatible with the architecture of the World Wide Web, and with the Semantic Web in particular. In fact, OWL uses URIs (for naming) and the description framework provided by RDF, and adds the following capabilities to ontologies: the possibility of being distributed across many systems, full compatibility with Web standards for accessibility and internationalization, openess

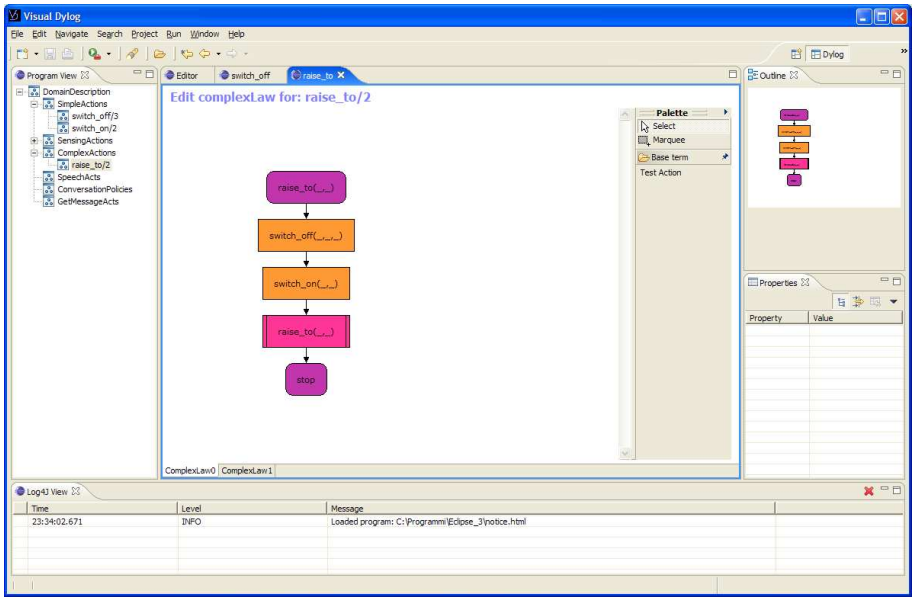


Fig. 7. Representation of a ComplexAction.

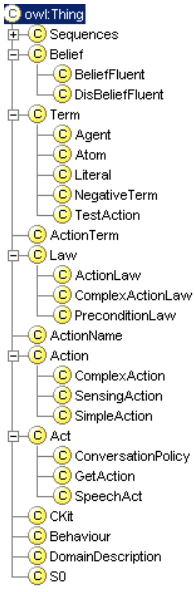


Fig. 8. The taxonomy of the DyLOG ontology

and extensiblility.

OWL builds on RDF and RDF Schema; it enriches the vocabulary so to allow the description of properties and classes. Some examples of add-ons are relations between classes (e.g. disjointness), cardinality (e.g. ”exactly one”), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

Recently, OWL has been used for defining a set of ontologies that can be considered as declarative languages and specifications for agents (more generally, web services) that are to be retrieved over the web in an intelligent way, so that they can interoperate and accomplish a common goal. A few examples are the OWL-S language [11], for web service functional description, FIPA OWL [31], an ontology for FIPA agents and ACL messages, and ConOnto [32], that allows the description of context aware systems. In the following we will describe the ontology that we have designed for describing DyLOG agents in a Semantic Web context.

A. *The* DyLOG *ontology*

As mentioned in the previous section, representing DyLOG programs by means of ontological terms allows the use of our language in the development of interoperating agents in a Semantic Web framework. Another advantage is the possibility to specify the syntactic constraints of the language directly within the ontology definition: for instance, a Simple Action must have one and only one Precondition Law; this constraint can be specified by imposing a proper restriction to the cardinality of the corresponding property. A reasoner can be used for verifying that the syntactic constraints are respected.

For representing a DyLOG program by means of the ontology it is necessary to start with an instance of class *DomainDescription*, which contains the properties for specifying the behavior, the communication policies and the initial observations (respectively *behaviour*, *ckit* and *s0*). Each such property is represented by an instance of a class that specifies all the characteristics of the corresponding DyLOG construct by means of properties and restrictions imposed to capture the syntactic constraints. In Figure 8 we report the taxonomy of the ontology, while in Figure 9 we present, as an example, the definition of the class *Simple Action* and its properties.

It is interesting to observe that, within a *Simple Action* instance, the order of the *Action Law* instances is meaningful

```
⟨owl:Class rdf:ID="SimpleAction"⟩
⟨rdfs:subClassOf rdf:resource="#Action"/⟩
⟨rdfs:subClassOf⟩ - ⟨owl:Restriction⟩
⟨owl:cardinality rdf:datatype="#int"⟩ 1 ⟨/owl:cardinality⟩
⟨owl:onProperty⟩
⟨owl:ObjectProperty rdf:about="#preconditionLaw"/⟩
⟨/owl:onProperty⟩ ⟨/owl:Restriction⟩ ⟨/rdfs:subClassOf⟩
⟨rdfs:subClassOf⟩ - ⟨owl:Restriction⟩ - ⟨owl:onProperty⟩
⟨owl:ObjectProperty rdf:about="#actionLawSeq"/⟩
⟨/owl:onProperty⟩
⟨owl:maxCardinality rdf:datatype="#int"⟩1⟨/owl:maxCardinality⟩
⟨/owl:Restriction⟩ ⟨/rdfs:subClassOf⟩ ⟨/owl:Class⟩

⟨owl:ObjectProperty rdf:ID="actionLawSeq"⟩
⟨rdfs:range rdf:resource="#ActionLawSeq"/⟩
⟨rdfs:domain⟩ - ⟨owl:Class⟩ - ⟨owl:unionOf rdf:parseType="Collection"⟩
⟨owl:Class rdf:about="#SpeechAct"/⟩
⟨owl:Class rdf:about="#SimpleAction"/⟩
⟨/owl:unionOf⟩ ⟨/owl:Class⟩ ⟨/rdfs:domain⟩ ⟨/owl:ObjectProperty⟩

⟨owl:ObjectProperty rdf:ID="preconditionLaw"⟩
⟨rdfs:range rdf:resource="#PreconditionLaw"/⟩
⟨rdfs:domain⟩ - ⟨owl:Class⟩ - ⟨owl:unionOf rdf:parseType="Collection"⟩
⟨owl:Class rdf:about="#SpeechAct"/⟩
⟨owl:Class rdf:about="#SimpleAction"/⟩
⟨/owl:unionOf⟩ ⟨/owl:Class⟩ ⟨/rdfs:domain⟩ ⟨/owl:ObjectProperty⟩

⟨owl:ObjectProperty rdf:ID="actionName"⟩
⟨rdfs:domain⟩ - ⟨owl:Class⟩ - ⟨owl:unionOf rdf:parseType="Collection"⟩
⟨owl:Class rdf:about="#Action"/⟩
⟨owl:Class rdf:about="#PreconditionLaw"/⟩
⟨/owl:unionOf⟩ ⟨/owl:Class⟩ ⟨/rdfs:domain⟩
⟨rdfs:range rdf:resource="#ActionName"/⟩ ⟨/owl:ObjectProperty⟩
```

Fig. 9. An excerpt from the OWL DyLOG ontology: definition of simple action.

because it might influence the program execution (like in prolog). Nevertheless, such an ordering cannot be represented directly in OWL. To this aim, we have defined an auxiliary structure (a linked list) that solves the problem. We have relied on this solution whenever an ordering had to be imposed over the instances of a given property.

In order to exploit the DyLOG ontology within the environment described in this article we added to tuDyLOG package functionalities to import and export a DyLOG program in Java representation to OWL and vice versa. This implementation uses libraries provided from Jena [33]: a framework, produced by HP labs, for develop Semantic Web applications.

V. Conclusions and future work

In this paper we have described VisualDyLOG, a graphical design and programming evironment for the modal logic language DyLOG. The project basically relies on two main choices. On a hand, a fully new, Java implementation of the DyLOG language has been developed, as an extension of the *tuProlog* package. The new package, named tuDyLOG actually exploits the basic mechanisms already offered by *tuProlog*, such as the methods for unification. The reason for changing implementation language (an implementation of DyLOG in Sicstus Prolog is already available) is that Java is more portable and allows us to exploit applications and frameworks that are already available, in particular, Eclipse: a well-known platform for building integrated development environments. By means of this platform it is easy to develop applications that can be deployed and distributed as stand-alone tools. The implementation of the graphical programming environment is almost complete; what still remains to do is the re-implementation of the DyLOG engine, which is on the way. Also the OWL ontology for DyLOG is ready to use and will be soon tested in a Semantic Web framework. In fact, we believe that this package will be very useful for the development of Semantic Web Services and we plan to use it in cooperation with the University of Hannover in an e-learning setting: integrating a DyLOG web service in the

Personal Reader architecture (see [34]).

REFERENCES

[1] AgentTool development system, "http://www.cis.ksu.edu/~sdeloach/ai/projects/agentTool/agentool.htm."

[2] S. A. DeLoach, *Methodologies and Software Engineering for Agent Systems*. Kluwer Academic Publisher, 2004, ch. The MaSE Methodology, to appear.

[3] ZEUS Home Page, "http://more.btexact.com/projects/agents.htm."

[4] E. Astesiano, M. Martelli, V. Mascardi, and G. Reggio, "From Requirement Specification to Prototype Execution: a Combination of a Multiview Use-Case Driven Method and Agent-Oriented Techniques," in *Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE'03)*, J. Debenham and K. Zhang, Eds. The Knowledge System Institute, 2003, pp. 578–585.

[5] I. Gungui and V. Mascardi, "Integrating tuProlog into DCaseLP to engineer heterogeneous agent systems," proceedings of CILC 2004. Available at http://www.disi.unige.it/person/MascardiV/Download/CILC04a.pdf.gz. To appear.

[6] M. Martelli and V. Mascardi, "From UML diagrams to Jess rules: Integrating OO and rule-based languages to specify, implement and execute agents," in *Proceedings of the 8th APPIA-GULP-PRODE Joint Conference on Declarative Programming (AGP'03)*, F. Buccafurri, Ed., 2003, pp. 275–286.

[7] M. Baldoni, L. Giordano, A. Martelli, and V. Patti, "Reasoning about Complex Actions with Incomplete Knowledge: A Modal Approach," in *Proc. of ICTCS'2001*, ser. LNCS, vol. 2202. Springer, 2001, pp. 405–425.

[8] M. Baldoni, C. Baroglio, I. Gungui, A. Martelli, M. Martelli, V. Mascardi, V. Patti, and C. Schifanella, "Reasoning about agents' interaction protocols inside dcaselp," in *Proc. of the International Workshop on Declarative Language and Technologies, DALT'04*, J. Leite, A. Omicini, P. Torroni, and P. Yolum, Eds., New York, July 2004, to appear.

[9] B. Shneiderman, *Designing the user interface*. Addison-Wesley, 1998.

[10] DAML-S, "http://www.daml.org/services/daml-s/0.9/," 2003, version 0.9.

[11] OWL-S, "http://www.daml.org/services/owl-s/."

[12] WSDL, "http://www.w3c.org/tr/2003/wd-wsdl12-20030303/," 2003, version 1.2.

[13] J. Bryson, D. Martin, S. McIlraith, and L. A. Stein, "Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web," 2002. [Online]. Available: citeseer.nj.nec.com/bryson02agentbased.html

[14] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl, "GOLOG: A Logic Programming Language for Dynamic Domains," *J. of Logic Programming*, vol. 31, pp. 59–83, 1997.

[15] G. D. Giacomo, Y. Lesperance, and H. Levesque, "Congolog, a concurrent programming language based on the situation calculus," *Artificial Intelligence*, vol. 121, pp. 109–169, 2000.

[16] S. McIlraith and T. Son, "Adapting Golog for Programmin the Semantic Web," in *5th Int. Symp. on Logical Formalization of Commonsense Reasoning*, 2001, pp. 195–202.

[17] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti, "Reasoning about interaction for personalizing web service fruition," in *Proc. of WOA 2003: Dagli oggetti agli agenti, sistemi intelligenti e computazione pervasiva*, G. Armano, F. De Paoli, A. Omicini, and E. Vargiu, Eds. Villasimius (CA), Italy: Pitagora Editrice Bologna, September 2003.

[18] ——, "Reasoning about interaction protocols for web service composition," in *Proc. of 1st Int. Workshop on Web Services and Formal Methods, WS-FM 2004*, M. Bravetti and G. Zavattaro, Eds. Elsevier Science Direct. To appear, 2004, electronic Notes in Theoretical Computer Science.

[19] OWL, "http://www.w3.org/2004/OWL/."

[20] K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross, and V. Subrahmanian, "IMPACT: a platform for collaborating agents," *IEEE Intelligent Systems*, vol. 14, no. 2, pp. 64–72, 1999.

[21] M. Fisher, "A survey of concurrent METATEM - the language and its applications," in *Proc. of the 1st Int. Conf. on Temporal Logic (ICTL'94)*, ser. LNCS, D. M. Gabbay and H. Ohlbach, Eds., vol. 827. Springer-Verlag, 1994, pp. 480–505.

[22] M. Baldoni, L. Giordano, A. Martelli, and V. Patti, "Programming Rational Agents in a Modal Action Logic," *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, 2004, to appear.

[23] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti, "Reasoning about self and others: communicating agents in a modal action logic," in *Theoretical Computer Science, 8th Italian Conference, ICTCS'2003*, ser. LNCS, C. Blundo and C. Laneve, Eds., vol. 2841. Bertinoro, Italy: Springer, October 2003, pp. 228–241.

[24] FIPA, "Fipa 97, specification part 2: Agent communication language," FIPA (Foundation for Intelligent Physical Agents), Tech. Rep., November 1997, available at: `http://www.fipa.org/`.

[25] M. P. Singh, "A social semantics for agent communication languages," in *Proc. of IJCAI-98 Workshop on Agent Communication Languages*. Berlin: Springer, 2000.

[26] M. Baldoni, L. Giordano, A. Martelli, and V. Patti, "Programming rational agents in a modal action logic," annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation. To appear.

[27] tuProlog Home Page, "http://lia.deis.unibo.it/research/tuprolog/."

[28] Eclipse platform, "http://www.eclipse.org."

[29] Eclipse Rich Client Platform, "http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/platform-ui-home/rcp/index.html."

[30] M. Baldoni, C. Baroglio, and V. Patti, "Web-based adaptive tutoring: An approach based on logic agents and reasoning about actions," *Journal of Artificial Intelligence Review*, 2004, to appear.

[31] FIPAOWL, "http://taga.umbc.edu/ontologies/fipaowl.owl."

[32] CONONTO, "http://www.site.uottawa.ca/~mkhedr/contexto.html."

[33] Jena Semantic Web Framework, "http://jena.sourceforge.net/."

[34] N. Henze and M. Herrlich, "The personal reader: a framework for enabling personalization services on the semantic web," in *Proc. of ABIS 2004*, 2004.

Using Method Engineering for the Construction of Agent-Oriented Methodologies

Giancarlo Fortino, Alfredo Garro, and Wilma Russo
D.E.I.S.
Università della Calabria
Via P. Bucci, 87030 Rende (CS), Italy
Email: {fortino, garro, russow}@deis.unical.it

***Abstract*— Great emphasis has been recently given to agent-oriented methodologies for the construction of complex software systems. In this paper two approaches for the construction of agent-oriented methodologies and based on methods integration are presented: *meta-model-driven* and *development process-driven*. The former is based on the MAS meta-model adopted by designers for the development of a MAS for a specific problem in a specific application domain. The latter is based on the instantiation of a software development process in which each phase is carried out using appropriate method fragments and by the mutual adaptation of the work products coming out from each phase.**

I. Introduction

In analysing and building complex software systems, a number of fundamental techniques for helping to manage complexity have been devised [3]:

- *Decomposition*: the basic technique for tackling large problems by dividing them into smaller, more manageable chunks, each of which can then be approached in relative isolation. It helps tackling complexity because it limits the designer's scope.
- *Abstraction*: the process of defining a simplified model of the system that emphasizes some details or properties, while suppressing others. It is useful because it limits the designer's scope of interest at a given time.
- *Organization*: the process of defining and managing the interrelationships between the various system's components. The ability to specify organizational relationships helps tackling complexity by enabling a number of basic components to be grouped together and treated as a higher-level unit of analysis, and by providing a means of describing the high-level relationships between the various units.

Recently the agent-oriented approach [13] has been widely recognized as very suitable for the development of complex software systems since it fully exploits the techniques listed above. In particular in the context of complex software systems:

- the agent-oriented decompositions are an effective way of partitioning the problem space;
- the key abstractions of the agent-oriented mindset (agents, interactions, and organizations) are a natural means of modelling;
- the agent-oriented philosophy for modelling and managing organizational relationships is appropriate for dealing with the existing dependencies and interactions.

The development of complex software systems by using the agent-oriented approach requires suitable agent-oriented modelling techniques and methodologies which provide explicit support for the key abstractions of the agent paradigm.
Several methodologies supporting analysis, design and implementation of Multi-Agent Systems (MAS) have been to date proposed in the context of Agent Oriented Software Engineering (AOSE) [14]. Some of the emerging methodologies are Gaia [16], MaSE [7], Prometheus [15], Tropos [4], Message [5], Passi [6], and Adelfe [2]. Although such methodologies have different advantages when applied to specific problems it seems to be widely accepted that an unique methodology cannot be general enough to be useful to everyone without some level of customization. In fact, agent designers, for solving specific problems in a specific application context, often prefer to define their own methodology specifically tailored for their needs instead of reusing an existing one. Thus, an approach that combines the designer's need of defining his own method-ology with the advantages and the experiences coming from the existing and documented methodologies is highly required.
A possible solution to this problem is to adopt the method engineering paradigm so enabling designers of MAS to use phases or models or elements coming from different methodologies in order to build up a customized approach for their own problems [12].
In particular, the development methodology is constructed by assembling pieces of methodologies (**method fragments**) from a repository of methods (**method base**). The method base is built up by taking method fragments coming from existing agent-oriented methodologies (such as Adelfe, Gaia, Message, Passi, Tropos, etc.) or ad hoc defined methods. Currently this approach is adopted by the FIPA Methodology Technical Committee (TC) [20].
It is therefore crucial to define guidelines for methods integration in order to both construct the methodology (retrieving the method fragments from the method base and integrating them) and apply it in the actual development life cycle.
In this direction, the paper proposes two approaches for

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

the construction of agent-oriented methodologies by using methods integration: (i) *meta-model-driven*, which is based on the MAS meta-model adopted by the designer for the development of a MAS for a specific problem in a specific application domain; (ii) *development process-driven*, which is based on the instantiation of a software development process in which each phase is carried out using appropriate method fragments.

The remainder of this paper is organized as follows. In section II and III the meta-model-driven and the development process-driven approaches are respectively described. In section IV, conclusions are drawn and on-going research activities delineated.

II. The MAS meta-model-driven approach

A method fragment [18] is a portion of methodology which is composed of the following parts:

1) A process specification, defined with a SPEM diagram [21], which defines the procedural aspect of the fragment;
2) One or more deliverables such as AUML/UML diagrams and text documents [1];
3) Some preconditions which represent a kind of constraint since it is not possible to start the process specified in the fragment without the required input data or without verifying the required guard conditions;
4) A list of elements (which is a part of the MAS meta-model subsumed by the methodology from which it was extracted) to be defined or refined through the specified process;
5) Application guidelines that illustrate how to apply the fragment and related best practices;
6) A glossary of terms used in the fragment in order to avoid misunderstandings if the fragment is reused in a context that is different from the original one;
7) Composition guidelines which describe the context/problem that is behind the methodology from which the specific fragment is extracted;
8) Aspects of fragment which are textual descriptions of specific issues such as platform to be used, application area, etc;
9) Dependency relationships useful to assemble fragments.

It should be noted that not all of these elements are mandatory; some of them (for instance notation or guidelines) could be not applicable or not necessary for some specific fragment.

To build his own methodology by exploiting the *meta-model-driven* approach, the designer must:

- choose or define a MAS meta-model suitable for the specific problem and/or the specific application domain;
- identify the elements that compose the meta-model of the MAS under development;
- choose the method fragments that are able to produce the identified meta-model elements;
- defining a development process characterized by a *method fragments execution order* on the basis of the relationship existing among the meta-model elements produced by each fragment.

Hence, the obtained methodology is able to completely *cover* the MAS meta-model for a given problem in a specific application domain.

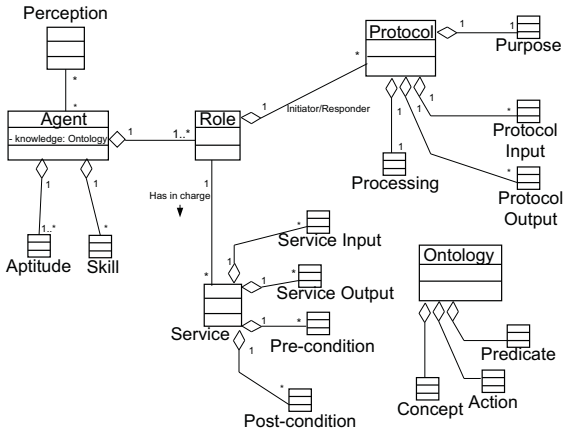


Fig. 1. An example MAS meta-model

An example MAS meta model is reported in Figure 1. Referring to the MAS meta-model of Adelfe, Gaia and Passi a set of methods fragments that are able to produce a piece of the MAS meta-model can be chosen. To completely cover the MAS meta-model selected fragments can be combined and, if necessary, new fragments can be defined (see Figure 2). Using this approach, the integration among the fragments is based on the relationships existing among the elements of the MAS meta-model. Thus, in order to obtain a completely and well-defined ad-hoc methodology, a proper *method fragments execution order* is to be defined.

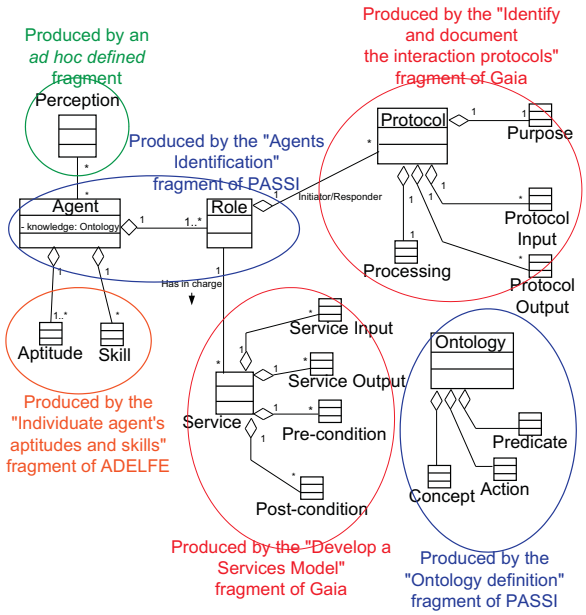


Fig. 2. An example of meta-model-driven methods integration

On the basis of the relationships shown in figure 2) the method fragments execution order is the following:

1) the Agents Identification fragment of Passi [19];
2) the concurrent execution of the ad-hoc defined fragment and the Individuate agent's aptitudes and skills fragment of Adelfe [17];
3) the concurrent execution of the Develop a Services Model fragment of Gaia and the Identify and document the interaction protocols fragment of Gaia [11];
4) the Ontology definition fragment of Passi [19].

III. The development process-driven approach

The development process-driven approach focuses on the instantiation of a software development process that completely covers the development of MAS (see Figure 3).

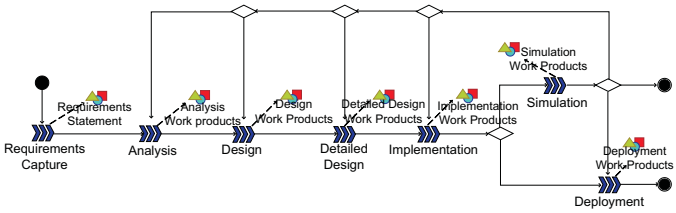


Fig. 3. An example of software development process

To build his own methodology by exploiting the *development process-driven* approach, the designer must:

- choose or define a software development process suitable for the specific problem and for the specific application domain;
- instantiate the development process by selecting, for each phase, suitable method fragments, chosen from agent-oriented methodologies proposed in the literature or ad-hoc defined.

An example software development process [8] is reported in Figure 3. Referring to the development phases specified by Tropos, Gaia, Passi and by a Statecharts-based methodology [10], a set of methods fragments that are able to carry out each phase of the development process are to be chosen. To completely cover the development process the selected fragments can be combined and, if necessary, new fragments can be defined (see Figure 4). Using this approach, the integration between the fragments is achieved by individuating and/or defining dependencies among work products produced by the fragments of the instantiated process. Notice that the work products produced in a given fragment might constitute the input for the next fragment provided that they contain all the in-formation required to its initialization (see Figure 5).

IV. Conclusions

This paper has proposed two approaches to the integration of methods fragments: *meta-model-driven* and *development process-driven*. These approaches are not mutually exclusive; rather, hybrid approaches containing features of both of them might be defined as well.

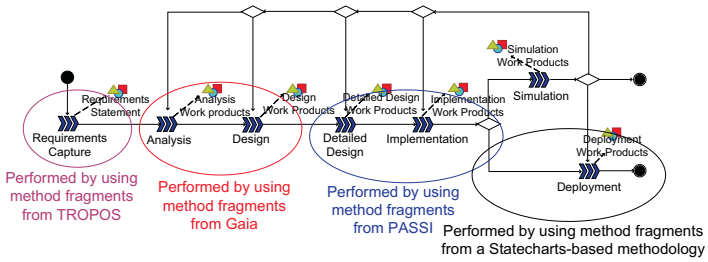


Fig. 4. Development process-driven methods integration

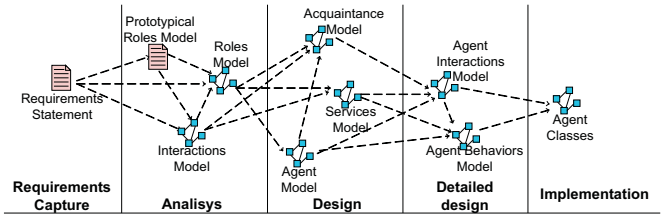


Fig. 5. Dependencies among work products of the instantiated process

The *meta-model-driven* approach provides the following advantage: flexibility for the definition of methodologies and meta-models of the MAS to be developed. Conversely, it has some drawbacks: (i) difficulty of integration of different fragments due to different semantics of the meta-model concepts; (ii) selection and/or definition of the meta-model to adopt for the specific problem and/or application domain.
The *development process-driven* approach is characterized by the following advantages: flexibility for the construction of methodologies by means of the instantiation of each stage of the development process. On the other hand, the disadvantages are the following: (i) process rigidity; (ii) low flexibility of the system meta-model since the meta-model of the adopted methodology must be used; (iii) adaptation among the work products which is sometimes difficult to achieve; (iv) choice and definition of the process to instantiate for the specific problem and/or application context. On going research activity is being focused on:

1) definition of adaptation techniques among work products produced by different methods and/or method fragments;
2) extraction from and definition of method fragments of already existing methodologies and the mutual adaptation among the defined method fragments. This activity is being carried out in the context of the FIPA Methodology TC;
3) the experimentation of the two presented approaches for the e-Commerce application domain [9].

References

[1] B. Bauer, J.P. Muller, and J. Odell. Agent UML: A Formalism for Specifying Multiagent Interaction. In Paolo Ciancarini and Michael Wooldridge, editors, *Agent-Oriented Software Engineering*, pages 91–103. Springer-Verlag, Berlin, 2001.

[2] C. Bernon., M.P. Gleizes, G. Picard, and P. Glize. The Adelfe Methodology For an Intranet System Design. In *Proc. of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS)*, Toronto, Canada, 2002.

[3] G. Booch. Object-Oriented Analysis and Design with Applications. Addison Wesley, 1994.

[4] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology, *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[5] G. Caire, F. Leal, P. Chainho, R. Evans, F. Garijo, J. Gomez, J. Pavon, P. Kearney, J. Stark, and P. Massonet. Agent Oriented Analysis using MESSAGE/UML. In *Proc. of the 2nd In-ternational Workshop on Agent-Oriented Software Engineering (AOSE)*, LNCS 2222. Springer-Verlag, Berlin, 2002.

[6] M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. Introducing Pattern Reuse in the Design of Multi-Agent Systems. In Ryszard Kowalczyk, Jorg P. Muller, Huaglory Tianfield, Rainer Unland, editors, *Agent Technologies, Infrastructures, Tools, and Applications for E-Services* , Lecture Notes in Artificial Intelligence (LNAI) volume 2592, pages 107–120, Springer-Verlag, Berlin Heidelberg, Germany, 2003.

[7] S. A. DeLoach, M. Wood, and C. Sparkman. Multiagent system engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, April 2001.

[8] G. Fortino, A. Garro, and W. Russo. From Modeling to Simulation of Multi-Agent Systems: an integrated approach and a case study. In Gabriela Lindemann, Jorg Denzinger, Ingo J. Timm, Rainer Unland, editors, *Multiagent System Technologies*, Lecture Notes in Artificial Intelligence (LNAI) volume 3187, pages 213–227, Springer-Verlag, Berlin Heidelberg, Germany, 2004.

[9] G. Fortino, A. Garro, and W. Russo. Modelling and Analysis of Agent-Based Electronic Marketplaces. *IPSI Transactions on Advanced Research*, 2004, to appear.

[10] G. Fortino, W. Russo, and E. Zimeo. A Statecharts-based Software Development Process for Mobile Agents. *Information and Software Technology*, 46(13):907–921, 2004.

[11] A. Garro, P. Turci, and M.P. Huget. Expressing Gaia Methodology using Spem. *FIPA Methodology TC, working draft v. 1.0/04-03-15, [http://fipa.org/activities/methodology.html]*.

[12] B. Henderson-Sellers. Method Engineering for OO Systems Development. *Communications of the ACM*, 46(10), 2003.

[13] N. R. Jennings. An Agent-Based Approach for Building Complex Software Systems. *Communications of the ACM*, 44(4), 2001.

[14] J. Lind. Issues in Agent-Oriented Software Engineering. In *Proc. of the First International Workshop on Agent-Oriented Software Engineering (AOSE)*, LNCS 1957, pages 45–58. Springer-Verlag, Berlin, 2001.

[15] L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *Proc. of the Third International Workshop on Agent-Oriented Software Engineering (AOSE)*, LNCS 2585, Springer-Verlag, Berlin, 2003.

[16] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[17] M. P. Gleizes et al. Adelfe fragments, rel.0, March 2004. *[http://www.pa.icar.cnr.it/ cossentino/FIPAmeth/docs/adelfe_fragments_v0.pdf]*

[18] Method Fragment Definition. *FIPA Methodology TC, working draft, Nov. 2003, [http://fipa.org/activities/methodology.html]*.

[19] M. Cossentino. PASSI fragments: All fragments, draft. rel 0.1, Feb. 2004. *[http://www.pa.icar.cnr.it/ cossentino/FIPAmeth/docs/passi_fragments_0_1.zip]*

[20] Foundation for Intelligent Physical Agents (FIPA) Specifications. *[http://www.fipa.org]*.

[21] Software Process Engineering Metamodel Specification, Version 1.0, formal/02-11-14. Object Management Group Inc. , November 2002.

A Personal Agent Supporting Ubiquitous Interaction

Giovanni Cozzolongo, Berardina De Carolis, and Sebastiano Pizzutilo

***Abstract*— D-Me (Digital_Me) is a multiagent system supporting ubiquitous and personal interaction with services available in active environments. It has been modeled as composed by two interacting entities: the *Environment*, in which various *services* are available, and a *Personal User Agent,* his/her digital "alter ego". A relation between these two entities is represented by the task the user intends to perform and the services the environment can provide for accomplishing user's tasks. Then, the personal user agent exploits several knowledge sources for proactively reminding or executing tasks according to the current context.**

***Index Terms*— Personal agents, ubiquitous computing, smart environments.**

I. Introduction

THERE are many different ways in which context information can be used to make applications more user friendly, flexible and adaptive especially in ubiquitous and pervasive computing where the context and usage needs change rapidly [1].

In ubiquitous computing (UbiComp) computers fade into the background, technology is present but invisible to users and the computation is possible everywhere and with any sort of device [2]. Then, interaction between users and services provided by a smart environment is very complex as it can happen at every time, in different situations and places. In this kind of situation, adaptation to user and context features seems to be important in order to decrease complexity and increase the conversational bandwidth of interaction (3 P.J. Brown, 1999). Context-awareness, then, refers to the ability of a system of extracting, interpreting and using context information intelligently in order to adapt its functionality to the current context of use [4,5].

Considering the interaction between a user and a context-aware system, there are at least two aspects that are worth mentioning: **information presentation** and **service fruition** [5]. As far as the first aspect is concerned, results of information services should be adapted not only to static user features, such as her background knowledge, preferences, sex, and so on, but also to more dynamic ones related to the context (i.e. activity, location, affective state and so on) [6]. The second aspect regards execution of users tasks triggered by context features. For instance user's tasks present in a to-do-list or agenda could be proactively reminded or executed when the user enters in an environment or is in a situation in which those task are enabled [7,8]. Moreover, their execution can be contextualized according to available resources, location and so on.

This paper presents an approach to address this second issues: taking advantage from user and context modeling for achieving effective ubiquitous interaction with services available in smart environments.

A way to approach this problem is to take inspiration from the personal interface agents research area [9,10]. In this paradigm, the user delegates a task to the agent that may operate directly on the application or may act in the background while the user is doing something else. An agent is, in this case, a representative of the user and acts on his/her behalf more or less autonomously. Moreover, it has to be able to communicate to the user in an appropriate way, without being too much intrusive, according to the context situation, user preferences, habits and needs. Then, importing this interaction metaphor in the Ubicomp vision, the ideal personal assistant, in addition, should exhibit a context-aware intelligence, doing exactly what the user expects him to do successfully in **the current context**.

Our work represents a first step in this direction. D-Me is a MultiAgent System (MAS) composed at least of two interacting entities: the **Environment**, a physical or logical place in which various *services* are available, and one or more *mobile users* interacting with ubiquitous services through a **Personal Agent**. A relation between these two entities is represented by the task the user wants to perform and the services that the environment can provide for accomplishing user's tasks. For this reason, in order to give to the user the possibility of delegating and controlling their D-Mes, when interacting with the environment, we developed, as a first prototype, a **Smart To-Do-List .**

A **To Do List** is a typical example of application that requires personalization and can take advantage from user and context modeling. Context-aware systems of this type remind the user of tasks based on the situational context. For example, if a user's to-do list contains the task 'buy food before going back home' and the user passes by a supermarket while going back home, then a useful context -aware reminder would notify the user to buy food. CyberMinder [7] and PDS[8] are examples of systems of this type. In particular, CyberMinder takes into account user's activities, location, time and user history as the context information. It can notice simple events (e.g., notifying a user of a meeting just based on time) or complex situations (e.g., reminding a user of an event using other people's context). The PDS system, in addition, utilizes machine learning in order to support a user's daily activities in an everyday computing setting. Another system that addresses the issue of context awareness of user interaction in real spaces is illustrated in [11]. In this system, two agents (one representing the user and the other representing the environment) cooperate for achieving context-aware

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

information presentation about the specific nature of the place the user is currently visiting.

Our approach takes advantage of the inherent properties of agents by adding to a simple context aware reminder proactivity and autonomy: if there is a task in the user to-do-list that can be completely or partially executed in the current context by requiring a service to the environment, its execution can be delegated to the personal agent given the appropriate autonomy level.

In this paper, we describe the D-Me MAS focusing on the description of the Personal Agent. In particular, Section II outlines some architectural requirements and describes the global organization of the D-Me system. Section III focuses on the main features of the user Personal Agent outlining how it exploits several knowledge sources for supporting "personal" interaction with the Environment. In Section IV, conclusions and future work are discussed.

II. Outline of the D-Me Architecture

Fig.1 illustrates the architectural schema of D-Me MAS developed according to FIPA specifications [12].

In this system, each user is represented by a Personal Agent (PA) that exploits some knowledge sources to remind and/or request, more or less autonomously, execution of environment services matching entries in the user To-Do-List that are enabled in a particular context.

On the other side, the environment is 'active': it is modeled as an organization of specialized agents:

- a **Keeper Agent** that coordinates the exchange of information between the other agents, acting as a directory facilitator (FIPA DF) and as a Agent Management System (FIPA AMS). Every other agent in the system has to register with it using the protocol appropriate for its role in the environment. For instance, a *Service Agent* should use a "*serviceRegister* " protocol, a D-Me agent uses its protocol.
- **Service Agents**, which provide services and are able to execute tasks;
- **D-Me Personal Agents**, representing users in the environment, that can look for contextually relevant services. A PA asks to the Keeper the address of other agents (personal or service) using a protocol that can look for them by name (a known agent) or some keywords (a type of service). In case of positive response, service execution can be asked directly to the corresponding agent, otherwise, the request is repeated until a timeout.
- a **Context Agent,** which can provide information about the environment context.

Users may interact with environment services in a remote way or by being physically in the environment and may move from one environment to another. Managing inter-environment communication is the task of the ***Environment Keeper*** with whom every Keeper must register.

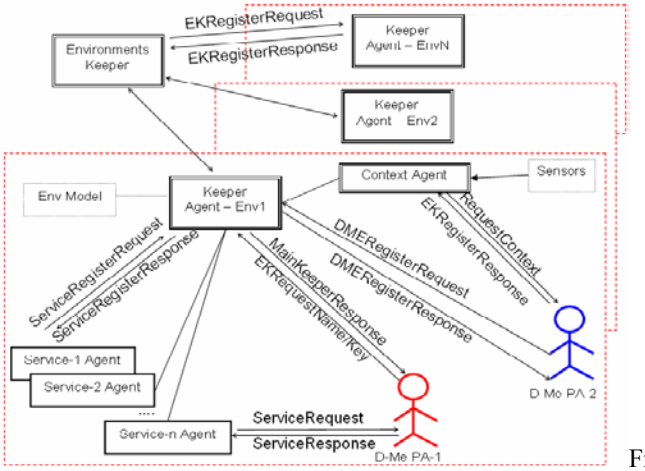


Fig. 1. D-Me MAS

From the implementation viewpoint, D-Me has been developed using the JADE toolkit [13]. In particular, the PA, that we will see in details later on, runs on a mobile device and has been implemented with JADE-Leap [14].

While the high-level communication protocols have been implemented using Agent Communication Language (ACL) messages, whose content refers to D-Me ontologies, the service discovery function has been developed using a framework for peer to peer communication, JXTA [15, 16]. Since FIPA has not yet delivered a definitive reference model for dynamic service discovery, we integrated the functionality of the D-Me Keeper agent with JXTA discovery middleware; this means that every time an agent registers in the environment, the Keeper will handle, besides standard information, also information about its public services. In this way, when an agent joins a platform, every other agent can be aware of its services. We are aware this is not a standard and definitive solution but, since our aim is not to create a new reference model for service discovery, we adopted a temporary solution until FIPA will provide this type of support [17].

In the rest of the paper, we will not going deep into this issue, since our aim is to show how the system works from the user point of view.

III. The D-Me Personal Agent

Fig.2 describes the D-Me PA. This agent is the core of the user's side of the D-Me system.

To support contextual service fruition, we developed a to-do-list application in which the user, through a friendly user interface, sets up a set of tasks to be performed in different context and environment and gives to his/her PA the autonomy to perform the task entirely or only in part [18]. When the user PA is in presence of a smart environment, that can provide services useful to the execution of scheduled tasks, it requests, on user behalf, their execution by passing to the environment user related information, that can be used to get personalized results.

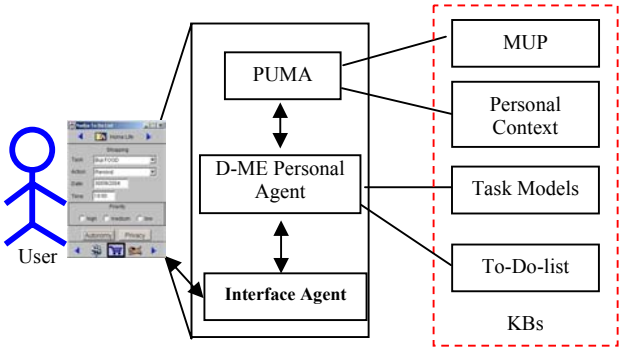


Fig. 2. D-Me Personal Agent

To achieve this aim, the PA is modeled as a BDI agent [19]; its reasoning mechanism is implemented as a cyclic behavior that continuously checks if, given the current set of agent beliefs (mental state) and given its desires (goals), some intentions and plan can be triggered and executed.

At this stage of the implementation, we modeled the knowledge for achieving the following macro-desires:

- *execute totally or in part tasks specified in the user to-do-list*: this desire is quite complex and it is achieved by accessing the specification of the task in the user to-do-list and executing the correspondent task model according to the associated constrains (autonomy, user and context features).
- *create new tasks if required*: sometimes the context triggers the execution of tasks that were not explicitly stated in the to-do-list. In this case this desire become active given the appropriate level of agent's autonomy on that family of actions.
- *get user-related information relevant for adapting task execution*: in order to adapt task execution and to communicate results to the user appropriately, the agent needs to know information about the user. These information can be stored in a user profile or can be inferred.
- *get context-related information relevant for contextualizing task execution*: as for user related data, assessing the current context situation is important especially for triggering and adapting task execution.
- *communicate personalized results*: results of tasks can be of various nature (information presentation, reminders, notifications, and so on). The way in which the agent communicates to the user is adapted to user interests, knowledge, preference and so on, but also to context features.

Then, In order to achieve these desires, the Personal Agent exploits the following knowledge sources:

i) the **to-do-list**, containing the description of the task and its constraints in terms of activation condition, priority, and autonomy level;

ii) the **formal description of the task**, that the agent can use in order to execute it;

iii) the **Mobile User Profile (MUP)**, containing situational information about the user managed by the Personal User Modeling Agent (PUMA);

iv) the **personal context** situation listing the value of sensors that can be detected from devices that the user wears (heart beat monitors, temperature, etc.), and

v) the environment **context situation** (light, noise, etc.) requested to the Context Agent.

These joint sets of information forms the agent's set of beliefs and can be used to trigger opportune intentions formalized as "plan recipes". Planning is a fundamental and yet computationally hard problem [20], since D-Me is potentially running on different types of personal devices with limited computational power, predefined plan recipes seem to be a good compromise between flexibility and resource constraints.

To demonstrate our solution approach, we use the following scenario as a running example throughout the remainder of this paper:

The user enters into the to-do-list a very urgent task: "buy food before going home (18.00)". She finished working and is now driving home. D-me reminds her, using the car display as an output device, the task in the list that should be performed outside the office before coming back home. In this case, D-Me reminds her to buy food. The user acknowledges the message and drives to the supermarket, where she usually shops. When the user goes into the supermarket the agent shows the list of missing food and the related special offers on her PDA or telephone. The list is obtained by matching the items provided by the home fridge agent, that checks the fridge content using tagged objects technology, and the supermarket special offers (obtained using the service discovery technology of the supermarket keeper).

Let's see in more details how these knowledge sources are used by the agent to support context-aware interaction with the environment.

A. D-Me Autonomy

D-Me Personal Agent exhibits an autonomous behavior when achieving its desires that has to match, somehow, the user delegation type. In particular, in the context of ubiquitous computing, we recognized the need to model autonomy at different levels:

- **Execution Autonomy**: related to execution of actions (tasks, subtasks, request of services, and so on).
- **Communication Autonomy**: related to the level of intrusiveness in communicating to the user. Can the agent take the interaction initiative in every moment or there are constraints related to the user and the context? Then, it is necessary to determine how much a message can be intrusive in a certain context.
- **Personal Data Diffusion Autonomy**: it is related to the autonomy of performing tasks requesting the diffusion of personal data like those contained in the user profile.
- **Resource Autonomy**: the agent may use critical resources of the user in order to executed delegated tasks (i.e. credit card number, time to schedule appointments).

Each dimension has an associated value that vary from "null" to "high" in a 5 values scale. The "null" value represents the absence of autonomy, the system has to execute what explicitly requested by the user. It cannot infer any information or decide to modify task execution without explicitly asking it to the user. The opposite value, "high", represents the maximum level of autonomy that gives to the

agent the freedom to decide what to do always according to constraints imposed by the user (i.e. budget limits). The other values determines an incremental growing of the autonomy in making decisions and inferring information [18].

Initially, as we will see later on, the user sets explicitly the autonomy level for a task in the to-do-list. During the interaction, autonomy levels are revised according to the type of feedback the user provides to the agent: **positive** feedback enforces the autonomy on that category of task , **negative** one reduces it. We are aware this is a simple mechanism, however it will give us the possibility to conduct a further study aiming at learning which is the most appropriate relation between the agent's level of autonomy and the type of user delegation on a category of tasks.

B. To-Do-List and Task Models

In order to give to the user personal agent the capability to reason on this information, it is necessary to specify the entry in the To-Do-List in terms of type or family of **task**, **environment** and **context information** relevant for task execution, **user** related **preferences**, agent's **autonomy**. To this purpose, we developed an interface in Java running on a PDA that enables user to input this information in a quite simple way (Fig. 3).

Fig. 3. A snapshot of the To Do List Interface.

A To-Do-List entry is then formalized in XML and stored in the set D-Me KBs. An example of entry corresponding to "buy food before coming back home" is the following:

```
<Knowledge
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-
instance"xsi:noNamespaceSchemaLocation="C:\DmeSystem
\dati\Knowledge.xsd" slotName="ToDoList">
  <Task slotName="taskDefinition" id="1" name="buy"
key="food   action="remind"   "   date=2509041800"
belongingScope="homelife"   environment="all"   p-
env="supermarket" priority="high" what="food list"
when="before"   whenevent="going   back   home"
remindBefore="1755" nextOk="3" nextError="4">
  <Autonomy   slotName="autonomy"   execution="high"
communication="high"          personalData="middle"
resourcesExploitation="low" />
</Task>
  ...
</Knowledge>
```

This specification states which is the task name, the type of associated D-Me action to be performed when the contextual situation triggers it (*remind* in this example), the scope of the task (*homelife*) that can be used to trigger user preferences in that scope, the *environment* in which the task should be reminded and the one in which the task should be performed (*p-env*). Additional information regards the priority, the deadline and the type of agent *autonomy* on that task.

In this example, the agent has an high execution and communication autonomy, a medium autonomy in communicating personal data to the environment and low autonomy on resource exploitation (in this example this is translated in the fact that the agent cannot buy and pay autonomously the food unless it is explicitly authorized by the user).

When user and context features triggers one of the tasks present in the user To-Do-List, the agent's desire of executing a task is achieved by selecting the appropriate plan in the D-Me KB.

In this case the *Remind(U, Do(Task , env, p-env, Cti)*) plan is selected. In this case, U denotes relevant user features, *Task* denotes *Buy(food)*, *env* the environment in which the remind can be notified (all), *p-env* the environment in which the user task can be performed (supermarket) and *Cti* represents the context at time *ti*.

C. P.U.M.A.: Personal User Modeling Agent

Mobile personalization can be defined as the process of modeling contextual user-information which is then used to deliver appropriate content and services tailored to the user's needs. As far as user modelling is concerned, a mobile approach, in which the user "brings" always with her/himself the user model on an personal device, seems to be very promising in this interaction scenario [21]. It presents several advantages: the information about the user are always available, updated, and can be accessed in a wireless and quite transparent way, avoiding problems related to consistency of the model, since there is always one single profile per user.

Based on this idea, in the context of our research on personalization of interaction in ubiquitous computing [22, 23], we have designed and implemented a Personal User Modeling Agent (PUMA).

In developing its architecture we considered the following issue: a personal device is used mainly in situations of user mobility. Normally, when the user is in more "stable" environments (i.e. home, office, etc.) he/she will use other devices belonging to that environment (i.e. PC, house appliances, etc.). In this view, the personal device can be seen as a "satellite" of other "nucleus" devices that the user uses habitually in his/her daily life. Then, the PUMA has to be able to handle this nucleus-satellite relation.

With this aim, instead of implementing a truly mobile agent, the PUMA is cloned and lives on all the user platforms/devices. However, although the chosen approach simplifies the implementation, it requires transferring knowledge needed for user modeling and opens consistency problems in maintaining a global image of user preferences, interests, habitual behavior, etc. In our approach, user models are organized in a hierarchy [24] whose nodes represent relevant interaction environments, task families, interest groups (Fig. 4).

In particular, the roots of the hierarchy represents user modeling scopes (interaction environments). Each node in the hierarchy represents a subset of user model data relevant to the specified domain, task, etc. Then the PUMA accesses and reasons on the Mobile User Profile portion that is in focus

according to the user task and environment.

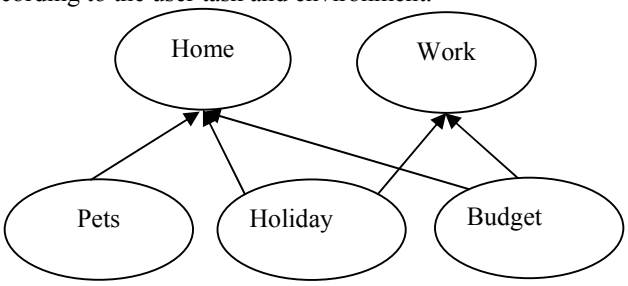


Fig. 4: An example of hierarchical User Model

This approach presents the main advantages of decreasing the complexity of representing an unified view of the user profile even if it requires particular attention in structure modelling and decomposition.

In another project we are testing how the same approach could be implemented using a hierarchy of Bayesian network instead of MUPs allowing in this way a better treatment of uncertainty that is typical of ubiquitous computing [25].

As far as **representation** is concerned, beside considering static long term user features (age, sex, job, general interests and so on) it is necessary to store information about more dynamic "user in context" features.

For instance, the fact that a user, when is shopping at the supermarket, *buy cookies only when there is a 3x2 special offer* is related to a contextual situation. If we want to give to the user PUMA the capability to reason on this type of facts, we need a representation language rich enough to formalize user properties related to contextual situation, understandable potentially by every environment, flexible and compact enough to be stored on the user personal device. In a first version of D-Me we developed our own ontology for describing mobile user profiles, however, since it was not the main aim of our research, in this second version of the prototype, we decided to adopt UbisWorld [26, 27] language as user model ontology of our Personal Agent. In this way we have a unified language able to integrate user features and data with situational statements and privacy settings that better suited our need of supporting situated interaction. This language is rich enough to deal with the representation and provide privacy-protection features. It allows representing all concepts related to the user by mean of the UserOL ontology, to annotate these concepts with situational statements that may be transferred to an environment only if the owner user allows this according to privacy settings. An example of a situational statement is the following:

```
<SituationalStatement version="Full_0.1">
<content>
   <subject><UbisWorld:Nadja /></subject>
   <predicate><UserOL:buying cookies /></predicate>
   <predicate-range><UserOL:normal,specialoffer,3x2/>
      </predicate-range>
   <object>special offer <object>
</content>
<restriction>< location>supermarket<location></restriction>
<meta>
   <owner><UbisWorld:Nadja /></owner>
   <privacy><UbisWorld:friends /></privacy>
   <purpose><UbisWorld:commercial /></purpose>
   <retention><UbisWorld:short /></retention>
   <viewer><UbisWorld:X-Supermarket /></viewer>
   <evidence>not-specified</evidence>
   <confidence>high</confidence>
</meta>
</SituationalStatement>
```

This approach can be used to represent some parts of the real world like an office, a shop, an house or an airport. It represents persons, objects, locations as well as times, events and their properties and features.

User preferences, interests, etc. are collected by the PUMA in two ways:

- using a graphical interface (Fig.5) in which the user can explicitly insert her preferences and related privacy settings regarding particular domains,
- other information (i.e. temporary interests) can be derived when the user insert a task in the To-Do-List.

User feedback and actions in the digital and real world may reproduce changes in the user model. The PUMA observes the user actions: when new information about the user can be inferred, it updates or adds a new slot in the MUP and sets the "confidence" attribute of that slot with an appropriate value that is calculated by the weighted average of all the user actions having an impact on that slot. The confidence attribute may be set to low, medium and high.

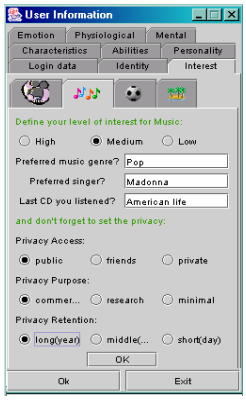


Fig. 5. An interface for initial setting of the PUMA.

Even if we have chosen the mobile approach, we cannot assume that the user will have with her/himself an handheld device and this type of device still presents hardware-related limits (capacity, computational speed, battery,…).

At this aim, in D-Me the PUMA could be stored on a Remote Server trusted by the user [28]. In the near future these technological constraints will be overcome by the spread of many personal an powerful device [29,30]

D. Context Information

Both entities, D-Me Agents and the Environment, need to sense and elaborate context information. In our approach, **Context** is grounded on the concept of "*user task executable in an environment*". Therefore, given a task in the user to-do-list, once the user has been classified according to the strategy of the UM component, its execution and results can be influenced by the context in which the interaction occurs and, in particular, by:

- **static environment features**: scope (daylife, social relations, budget, etc..), physical features, such as

desciption of objects relevant for interaction, type of environment (public, private).
- **dynamic environment features**: for instance noise light level and tagged object;
- **dynamic user features**, that identify the physical and social surroundings of the user that can be derived by specific data sensors (emotional state, location, activity the user is performing, time, ...);
- **device employed**.

At the present stage of the prototype, we do not work on hardware sensors. They will be realized in the next stage. At the moment we simulate their values through an interface that communicates relevant changes to the **Context Agent** that knows the global context situation at the considered time. The context situation relevant at time ti is represented in an XML structure compliant to the D-Me context ontology.

E. Interacting with the user

The Communication Behavior of the Personal Agent is used to interact with the user for communicating results of tasks or for asking information/confirmations required for task execution. We consider the following families of communication tasks:

- **request for input**. If, for instance, the to-do-list includes a task that requires additional information to be executed.
- **information provision**: Information may be presented when explicitly requested by the user or proactively prompted by D-Me because related to the current user task. In our scenario the supermarket special offers will be displayed as a consequence of the service discovery task.
- **request for confirmation**: if a task involves a step that requires a D-Me action and the agent does not have a full autonomy on that task, then the agent will ask the user for confirmation before performing it.
- **notification messages**. Proactive task execution is notified by D-Me, for instance, in the previous case, if the agent has the autonomy to perform an action then it will not ask for permission and will just notify it.
- **remind messages**. This is the typical message generated for the shopping task in our example.

User and context related factors are taken into account in generating the communication about a task in the following way [31]:

1. **user preferences and features**: results of information provision tasks are filtered, ordered and presented according to what has been inferred about the user starting from her profile data (interest, know-about, know-how). Possible user disabilities are taken into account for media selection.
2. **activity**: this influences information presentation as follows. If the user is doing something with a higher priority respect to the one of the communication task, then the message is postponed until the current activity ends. If the communication regards the current activity, media used in the message take into account the available body parts. Therefore, a voice input is preferable to a textual when, for instance, the user is running with her/his PDA asking for information about the next train to catch.
3. **location** of the user in the environment: texts, images and other media may be selected according to the type of environment (public vs. private, noisy vs. silent, dark vs. lightened, etc.) in which the users are and, more precisely, to their relative position to relevant objects in the environment.
4. **emotional state**: factors concerning the emotional state influence the level of detail in information presentation (short messages are preferred in stressing situation), the intrusiveness (bips and low priority messages are avoided when the user is already nervous), and the message content. For instance: if a user requests information in conditions of emergency, the agent will have to avoid engendering panic, by using reassuring expressions or voice timbre [32].
5. **device**: the display capacity affects the way information is selected, structured and rendered. For instance, natural language texts may be more or less verbose, complex figures may be avoided or substituted with ad hoc parts or with written/spoken comments.

To accomplish the communication task, the agent applies the following strategy: starting from XML-annotated results of a Service Agent, decides how to render them at the surface level taking into account the rules described above encoded in XSL.

IV. Discussion And Future Work

Effective ubiquitous interaction requires, besides techniques for recognising 'user in context' features, a continuous modeling of both the user and the context. Therefore, ubiquitous computing systems should be designed so as to work in different situations that depend on several factors: presence of a network connection, characteristics of interaction devices, user location, activity, emotional state and so on. However, in the near future, the network connectivity will be no more a problem, and we will not be worried about this constraint, as we are going towards an "interconnected world". Moreover the spread of technologies, such as for example RFID, will render the information about the context very rich and easy to use [33].

This work represents a step towards supporting personalized interaction between mobile users and a smart environment. Every user is represented by a D-Me Agent that, according to the content of her/his "To Do List", performs tasks on the user behalf by negotiating services with the smart environment.

Since the interaction happens through a personal agent, we started to consider the "delegation-autonomy" adjustment necessary for achieving cooperation between the user and his/her representative. However, more work in understanding how the user feedback influences the level of autonomy especially when this feedback is implicit (until now we considered only explicit feedback).

Moreover, as RFID are taking a key role in ubicomp we are investigating how to use them in such a system, so as to

"sense" the active tagged object. Those kind of object are part of the context and can influence the execution of several tasks as well as other information.

ACKNOWLEDGMENTS

We thanks students who cooperated in implementing the prototype described in this paper: in particular, Ignazio Palmisano, Luigi Iannone, and Roberto Tagliento. Finally, we thank Fiorella de Rosis to which we owe several fruitful ideas underlining this work.

REFERENCES

1. S. Greenberg. Context as a dynamic construct. Human-Computer Interaction, 2001, 16.
2. M. Weiser. Some computer science issues in ubiquitous computing. Commun. ACM, 1993, 36(7):75-84.
3. P. J. Brown, N. Davies, M. Smith, P. Steggles. Panel: towards a better understanding of context and context-awareness. In H-W Gellersen (Ed.) *Hand-held and ubiquitous computing: HUC'99 proceedings*, Springer, 1999.
4. G. Chen, D. Kotz, A Survey of Context-Aware Mobile Computing Research. Technical Report TR http://citeseer.nj.nec.com/chen00survey.html.
5. A. K. Dey, Understanding and Using Context. Personal and Ubiquitous Computing 5 (2001) 1, 4-7.
6. L. Ardissono, A. Goy, G. Petrone, M. Segnan and P. Torasso. Ubiquitous user assistance in a tourist information server. Lecture Notes in Computer Science n. 2347, 2nd Int. Conference on Adaptive Hypermedia and Adaptive Web Based Systems (AH2002), Malaga, pp. 14-23, Springer Verlag 2002.
7. H.E. Byun, K. Cheverst, User Models and Context-Awareness to Support Personal Daily Activities. Workshop on User Modelling for Context-Aware Applications. 2002.
8. A.K. Dey, G.D. Abowd, CyberMinder: A Context -Aware System for Supporting Reminders. Proc. Symposium on Handheld and Ubiquitous Computing, Bristol. (2000).
9. P. Maes, "Agents that Reduce Work and Information Overload," Communications of the ACM, Vol. 37#7, ACM Press, 1994.
10. H. Lieberman, T. Selker , Out of Context: Computer Systems That Adapt To, and Learn From, Context. IBM Systems Journal, Vol 39, Nos 3&4, pp. 617-631, 2000.
11. A.Celentano, D. Fogli, P. Mussio, F. Pittarello. Agents for Distributed Context-Aware Interaction. *AIMS '02, Workshop on Artifical Intelligence in Mobile Systems*, Lyon, France, July 22, 2002.
12. http://www.fipa.org.
13. http://sharon.cselt.it/projects/jade/
14. http://leap.crm-paris.com/
15. http://www.jxta.org
16. M. Pirker, M. Berger, M. Watzke,, An Approach for FIPA Agent Service Discovery in Mobile Ad Hoc Environments , Workshop on Agents for Ubiquitous Computing., Ubiagent04.
17. FIPA Agent Discovery Service Specification: http://www.fipa.org/specs/fipa00095/PC00095A.pdf
18. R. Falcone, C. Castelfranchi. Tuning the Collaboration Level with Autonomous Agents: a Principled Theory. AI*IA 2001: 212-224.
19. A. S. Rao and M. P. Georgeff, BDI-agents: from theory to practice, in "Proceedings of the First Intl. Conference on Multiagent Systems", San Francisco, 1995.
20. T. Bylander. The computational complexity of propositional STRIPS planning. Artificial Intelligence, 69:161–204, 1994.
21. A. Kobsa, Generic User Modeling Systems. UMUAI vol. II nos.1-2 pp.49-63. Kluwer Academic Publisher. 2001.
22. A. Cavalluzzi,B. De Carolis., S. Pizzutilo., G. Cozzolongo: Interacting with embodied agents in public environments. AVI 2004: 240-243.
23. G. Cozzolongo, B. De Carolis, S. Pizzutilo, Supporting Personalized Interaction in Public Spaces. In Proceedings of the Artificial Intelligence in Mobile Systems 2004. (Ubicomp Workshop). Baus J., Kray, C., Porzel, R. (Eds.). Nottingham, UK, 2004.
24. M. Samulowitz, Designing a Hierarchy of User Models for Context-Aware Applications. Position Paper at www.teco.edu/chi2000ws/papers/23_samulowitz.pdf
25. A. Jameson, Modeling Both the Context and the User. Personal and Ubiquitous Computing. Vol 5. Nr 1. Pp 29-33. 2001.
26. D. Heckmann: Ubiquitous User Modeling for Situated Interaction. User Modeling 2001: 280-282
27. D. Heckmann, Ubis World, www.u2m.org
28. B. De Carolis, S. Pizzutilo, I. Palmisano, A. Cavalluzzi, A Personal Agent Supporting Ubiquitous Computing. UM'03 9th International Conference on User Modeling. June 22-26, 2003.
29. www.vodafone.com
30. R.Want, http://www.intel.com/research/exploratory/personal_server.htm
31. B. De Carolis, F. de Rosis, S. Pizzutilo, Adapting Information Presentation to the "User in Context". IJCAI 2001 Workshop on AI in Mobile Systems, Seattle, 2001.
32. A Cavalluzzi, B De Carolis, V Carofiglio and G Grassano: Emotional dialogs with an embodied agent. In P Brusilovsky, A Corbett and F de Rosis (Eds): "User modeling '03".
33. T. Pederson, From Conceptual Links to Causal Relations -Physical-Virtual Artefacts in Mixed-Reality Space. PhD thesis, Dept. of Computing Science, Umeå university, report UMINF-03.14, ISSN 0348-0542, ISBN 91-7305-556-5., 2003, http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-137.

Un'Applicazione di E-government per la Gestione di Gare d'Appalto nella Pubblica Amministrazione

A. Grosso, M. Coccoli, and A. Boccalatte, *Dist – University of Genova*

***Abstract*—Lo sviluppo pervasivo delle nuove tecnologie dell'informazione, e in particolare di Internet, rappresenta un fattore di accelerazione e, al tempo stesso, lo strumento per "reinventare" le modalità di organizzazione e funzionamento delle amministrazioni pubbliche. In questo articolo viene presentato un sistema per la gestione automatica sul Web di gare d'appalto bandite dalla Pubblica Amministrazione in Italia basata sulla tecnologia ad agenti offerta dal framework AgentService. L'applicazione si avvantaggia quindi dell'elevata dinamicità e flessibilità delle comunità di agenti software e della interoperabilità offerta dai Web Service.**

***Index Terms*—e-government, multi-agent system, on-line auction.**

I. Introduzione

La diffusione di internet e la crescita del commercio elettronico stanno modificando alcune convenzioni del mondo economico, questo ha richiamato fortemente l'attenzione di governi e pubbliche amministrazioni, che sono intervenute a regolamentare il settore. Al momento attuale, sia da un punto di vista tecnico che legale, l'e-commerce può essere considerato una tecnologia matura e capace di attrarre sia imprese private che enti pubblici. Molte importanti istituzioni, ed in particolare la Comunità Europea e la Repubblica Italiana, stanno promuovendo lo sviluppo di servizi Internet per i cittadini, incoraggiando la nascita di sistemi informativi in grado di snellire la burocrazia e renderla più tempestiva [1].

Lo sviluppo pervasivo delle nuove tecnologie dell'informazione, ed in particolare di internet, rappresenta infatti un fattore di accelerazione e, al tempo stesso, lo strumento per:

- "reinventare" le modalità di organizzazione e funzionamento delle amministrazioni pubbliche;
- offrire ai cittadini, visti come "clienti", servizi più tempestivi, qualitativamente migliori e facilmente accessibili (quindi meglio distribuiti) attraverso l'uso della rete e della communication technology [2];
- contribuire, attraverso una maggiore interazione, a migliorare in modo significativo il rapporto tra apparati statali e cittadini;
- fornire servizi mirati, personalizzati, trasversali rispetto alle singole competenze e accessibili ovunque, in ogni momento.

La necessità di fornire servizi migliori, più efficienti, tempestivi e che non pesino eccessivamente sui bilanci è un'esigenza sentita oggi da ogni pubblica amministrazione a tutti i livelli e in qualsiasi parte del mondo [3].

Formalmente il governo elettronico (e-government) può essere definito come l'utilizzo delle nuove tecnologie telematiche nei rapporti tra la Pubblica Amministrazione (PA) e i cittadini, tra la PA e le imprese e tra gli organi della PA al loro interno (fra le diverse amministrazioni o i differenti livelli dello stato) [4]. Quindi, il "governo elettronico" interessa le applicazioni interne ed esterne delle tecnologie dell'informazione e della comunicazione (ICT) nel settore pubblico [5].

La Presidenza del Consiglio dei Ministri e il Ministro per l'innovazione e le tecnologie hanno emanato una Direttiva [6] che fissa le linee guida per l'anno 2004 in materia di digitalizzazione della pubblica amministrazione, indicando come punti cardine i servizi on-line per cittadini e imprese e la trasparenza dell'azione pubblica.

In questo articolo viene presentata un'applicazione di e-government per la gestione di gare d'appalto bandite dalla pubblica amministrazione, che propone un nuovo approccio per la gestione di aste on-line basato sulla tecnologia ad Agenti ed i Web Service. Web Auction Server System (WASS) è un sistema per la gestione delle contrattazioni nelle aste sul Web. WASS è pensato per essere utilizzato nella pubblica amministrazione italiana per rendere automatico ed economico il processo di acquisizione delle risorse, ma è in grado di operare anche in contesti differenti come ad esempio portali per il commercio elettronico. WASS è strettamente legato alla tecnologia offerta dal framework AgentService [7].

Nella sezione II viene introdotto l'utilizzo della tecnogia ad agenti nelle aste on-line, mentre nella sezione III è illustrato il workflow per l'acquisizione di beni e servizi nella pubblica amministrazione. La sezione IV presenta le principali

Manuscript received October 27, 2004.

A. Grosso, M. Coccoli, and A. Boccalatte are with the Department of Communication, Computer and System Sciences, University of Genova, via Opera Pia 13, 16145 Genova, Italia (phone: +39 103532284; fax: +39 103532154; e-mail: {agrosso, coccoli, nino}@dist.unige.it).

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

carratteristiche dei sistemi multi-agente e descrive le caratteristiche del framework AgentService. L'architettura e le funzionalità del WASS sono dettagliate nella sezione V, dopo di che vengono evidenziate le conclusioni.

II. Agenti e Aste On-Line

Il commercio elettronico basato sulle aste on-line sembra essere un'area in cui il web dimostra di essere più efficace rispetto ai sistemi tradizionali; questo è dovuto principalmente alla sua natura altamente interattiva, al coinvolgimento di molti fornitori rispetto alle vendite tradizionali di tipo singolo fornitore-compratore, ed infine ad una significativa riduzione dei costi. Se si considera anche il proliferare su internet di applicazioni per aste on-line come Auctionline, Onsale, InterAUCTION ed eBay, risulta evidente che la contrattazione basata sulle aste è divenuta una delle principali forme di commercio elettronico.

Le aste sul Web costituiscono un meccanismo conveniente per automatizzare le transazioni commerciali, ciò è principalmente dovuto alla semplicità con cui avvengono le interazioni nella negoziazione "multi-party", ma anche al fatto che le aste on-line sono in grado di minimizzare le scorte e ridurre significativamente i costi sia di gestione che di consegna. Inoltre è opportuno osservare che l'applicazione di aste on-line nel campo della PA può portare, in aggiunta ai già citati vantaggi, una maggior trasparenza nella contrattazione e assegnazione degli appalti, ciò è dovuto all'automatizzazione del servizio che limita al minimo l'intervento umano e quindi una possibile frode.

In generale, i sistemi per la gestione delle aste hanno un'elevata complessità; questa non è data solamente da oneri computazionali, ma principalmente è dovuta a problemi di progettazione, perché occorrerà focalizzarsi su come aumentare il rendimento e allo stesso tempo soddisfare le esigenze dei partecipanti/compratori.

Le aste sono un dominio applicativo altamente attrattivo per i ricercatori del settore dell'intelligenza artificiale (AI), che coinvolge lo sviluppo di "auction server" [8, 9], la definizione di agenti per la contrattazione e le euristiche [10]. D'altra parte, le aste non sono impiegate solamente per il commercio sul Web, ma costituiscono anche uno dei principali meccanismi di coordinazione per problemi di allocazione di risorse/task basati su agenti [11, 12, 13, 14].

L'interesse dei ricercatori nell'ambito della programmazione ad agenti è ormai una realtà. I concetti base di "agente autonomo" e "sistema multi-agente" (MAS), introdotti nel campo della Distributed Artificial Intelligence (DAI), possono essere applicati a contesti differenti per la distribuzione del controllo dei processi decisionali tra i componenti dei sistemi. Attualmente sono a disposizione un certo numero di strumenti software creati per rendere più semplice la programmazione orientata agli agenti: questi sono in genere composti da librerie e "tool" che guidano gli utenti durante la progettazione, l'implementazione ed il testing dei sistemi multi-agente. La tecnologia ad Agenti sembra quindi in grado di fornire il paradigma di programmazione adatto a modellare i sistemi di aste on-line. Questo è dovuto alle proprietà intrinseche degli agenti come l'autonomia e la proattività, che saranno trattate nei prossimi paragrafi.

III. Il Workflow nella Pubblica Amministrazione

Con il termine workflow, usato nelle sue diverse accezioni, ci si può riferire: ad un processo aziendale, alle specifiche di un processo generico, ad un software che implementi ed automatizzi un processo, o ad un'applicazione per il coordinamento delle persone e dei computer che creano il processo stesso.

Con l'introduzione di modelli e principi di contabilità economica e controllo di gestione, gli ordinamenti contabili delle amministrazioni e degli enti pubblici stanno cambiando radicalmente. La gestione della pubblica amministrazione diventa sempre più simile a quella delle aziende private: individuazione di programmi ed obiettivi, adozione di sistemi di programmazione, consuntivazione e controllo, pianificazione per obiettivi, monitoraggio dei risultati.

In questi ultimi anni si assiste all'introduzione nell'ente pubblico di una cultura aziendale, rivolta al conseguimento di risultati, obiettivo perseguito con decisione anche da molteplici interventi legislativi. È un processo di modernizzazione complesso, che riguarda i sistemi informatici, ma che ha anche un forte impatto organizzativo, che richiede nuove figure professionali e un intervento capillare per favorire l'evoluzione culturale parallelamente all'introduzione all'utilizzo di nuovi criteri gestionali.

Il concetto di workflow può quindi essere affiancato anche alla realtà delle istituzioni pubbliche comprendendo attività di razionalizzazione e, conseguentemente, di informatizzazione, dei processi di una generica amministrazione.

Tutti i sistemi di workflow assumono come elemento costitutivo primario il concetto di processo, inteso come entità fondamentale alla base della struttura logica e funzionale, su cui si fonda sia un'azienda privata che un ente pubblico.

Un processo è pertanto caratterizzato principalmente da:

- un prodotto che, trasferendo valore al cliente, rappresenta il vero obiettivo dell'organizzazione;
- un insieme di attività che rappresentano il flusso operativo del processo.

Per la produzione dei prodotti/servizi sono in genere coinvolte una o più strutture organizzative, attraverso una distribuzione di compiti e responsabilità, codificati in norme e procedure.

Per poter operare sui processi è necessario poterli rappresentare ed analizzare. E' importante quindi disporre di modelli per la loro rappresentazione in grado di evidenziare tutti i loro aspetti critici, quali ad es. le risorse consumate, il processo di trasformazione, il prodotto/servizio, le regole e i vincoli di trasformazione (controlli), i tempi e i costi, ecc.

IV. Agenti Software e Sistemi Multi-Agente

Un agente è definibile come un'entità computazionale in grado di agire in modo autonomo [15], acquisire informazioni dall'ambiente circostante ed agire secondo la propria base di conoscenza, scambiare informazioni con altri agenti o con esseri umani, e perseguire i propri obiettivi intraprendendo opportune iniziative. Gli agenti, perseguono il raggiungimento dei goal prefissati eseguendo delle funzioni o task che, frequentemente, appaiono vincolate da relazioni di interdipendenza o di conflittualità [16]. Un agente opera in un ambiente di esecuzione condiviso con altri agenti e applicazioni software; gli agenti sono in grado di interagire con l'ambiente in cui vivono, al fine di perseguire il proprio obiettivo [17]. Gli agenti sono adattativi, possono imparare dai cambiamenti dell'ambiente che li circonda: le capacità di apprendimento e adattabilità consentono all'agente di raggiungere con successo i propri obiettivi [16].

L'abilità sociale degli agenti costituisce una delle più importanti caratteristiche della programmazione orientata agli agenti. Un sistema multi-agente, MAS (multi-agent system), rappresenta una comunità sociale di membri interdipendenti che agiscono individualmente [18].

L'architettura che può essere considerata standard de facto per i sistemi multi-agente è quella descritta all'interno delle specifiche proposte dalla Foundation of Intelligent Physical Agents (FIPA) [19].

A. AgentService

AgentService è un ambiente completo per la progettazione, l'implementazione ed la distribuzione di applicazioni orientate agli agenti; fornisce quindi una specifica piattaforma di esecuzione degli agenti ed un linguaggio di programmazione agent-oriented. La Common Language Infrastructure (CLI) costituisce la base dell'ambiente di programmazione ed esecuzione di AgentService: gli agenti sviluppati con AgentService hanno pieno accesso al mondo dei componenti e all'ampia gamma di servizi offerti dalla CLI. Analizziamo ora gli elementi chiave proposti da Agent Service.

Il modello di agente: l'agente è composto da due elementi fondamentali: comportamenti e unità di conoscenza. I Behaviour rappresentano le attività concorrenti eseguite dall'agente. La Knowledge è composta da strutture dati persistenti condivise tra i behaviour e determina lo stato dell'agente.

Il framework ad agenti: una piattaforma di programmazione ad agenti che si basa sul modello sopra descritto e segue le specifiche architetturali indicate da FIPA. Gli agenti in esecuzione con i relativi comportamenti concorrenti sono ospitati all'interno di uno specifico dominio applicativo (AppDomain) della CLI. L'esecuzione e la sincronizzazione dei comportamenti concorrenti è gestita dalla piattaforma. AgentService che garantisce un elevato grado di scalabilità e sicurezza grazie alle caratteristiche offerte dagli AppDomain. In accordo al Document Object Model proposto con il framework, vi è una netta separazione tra la definizione di agente da parte del programmatore (design time agent) e la relativa istanza di agente in esecuzione (runtime agent). Tale separazione garantisce maggiore semplicità di programmazione ed assoluta autonomia all'agente. Le capacità sociali degli agenti si determinano tramite lo scambio di messaggi che si basa sul canale di comunicazione offerto dal sistema di "Remoting" della CLI.

Agent Programming Extensions (APX): un set di estensioni del linguaggio di programmazione C# mirate a semplificare lo sviluppo di applicazioni con AgentService. Il modello ad oggetti di AgentService è nascosto da APX, così che allo sviluppatore possa essere presentata una più semplice interfaccia orientata agli agenti che comporta limitati cambiamenti alla sintassi del C#.

B. Common Language Infrastructure

La Common Language Infrastructure è uno standard ECMA [20] e ISO-IEC [21] che definisce un ambiente virtuale di esecuzione. CLI è una piattaforma di programmazione orientata ai componenti in cui moduli di codice sono eseguiti in un contesto sicuro.

La Common Language Infrastructure è stata progettata per essere il target di differenti linguaggi di programmazione; offre una ricca libreria di classi ed un ampio set di servizi a runtime che garantiscono un'efficace esecuzione del codice. L'interoperabilità di linguaggio è una delle caratteristiche più innovative della CLI: moduli scritti in differenti linguaggi di programmazione possono interoperare con facilità senza bisogno di connettori software realizzati ad-hoc.

Sono disponibili implementazioni differenti della CLI per diversi sistemi operativi e diverse piattaforme hardware. Un'implementazione shared source della CLI è SSCLI comunemente nota con il nome di Rotor [22].

V. Web Auction Server System

In questo paragrafo viene presentato un sistema per la gestione di aste on-line. Web Auction Server System (WASS) garantisce ai fornitori un modo semplice ed automatico per competere in una contrattazione attraverso la tecnologia offerta dal framework AgentService e promossa grazie ai Web Service.

L'obiettivo del WASS è quello di fornire all'amministrazione pubblica italiana una via di accesso al mercato elettronico nel rispetto delle regole di workflow imposte dalla legislazione vigente. Le gare telematiche indette per l'acquisto di beni e servizi sono infatti regolamentate da una precisa normativa[1].

L'applicazione si prefigge quindi lo scopo di snellire le procedure amministrative per quel che riguarda l'approvvigionamento di beni o servizi da parte degli organi della Pubblica Amministrazione. E' stato realizzato un sistema di negoziazione che provvede a valutare in maniera automatica le offerte inviate dai fornitori partecipanti alla

[1] D.P.R. del 4 aprile 2002, n. 125, pubblicato sulla G.U. del 30 maggio 2002 e dalle successive linee guida

gara, predisponendo una graduatoria sulla base dei criteri scelti dall'amministrazione ordinante. Per l'abilitazione dei fornitori sono predisposti dall'amministrazione appositi bandi.

In particolare, l'applicazione si propone di:

- automatizzare il reperimento dei fornitori; attualmente avviene tramite contatto diretto oppure tramite gara pubblicata su un quotidiano di livello nazionale e può quindi dare luogo a esborsi di denaro;
- confrontare, tramite procedure automatiche, tutte le proposte raccolte e valutarne i risultati;
- migliorare i tempi di esecuzione dell'intero processo di acquisto, minimizzando soprattutto quelli imputabili alla burocrazia, riducendo i costi anche in termini di risorse umane e di documenti circolanti;
- aumentare la velocità di ricerca delle informazioni, predisponendo la memorizzazione su supporti di tipo digitale e quindi in database per un facile e rapido accesso ai dati.

Inoltre si introducono novità quali:

- la possibilità di attivare una gara d'appalto direttamente on-line;
- il controllo on-line l'andamento delle gare in tempo reale;
- la modifica delle informazioni presenti nelle basi di dati in maniera sicura e rapida senza dover compilare richieste o altri tipi di modulistica.

L'idea del WASS è nata dallo studio del flusso di documenti che avviene in relazione all'attività di approvvigionamento in una amministrazione comunale.

Il sistema proposto si basa, per ciò che riguarda la gara d'asta, sul paradigma ad agenti e, per la promozione e distribuzione del servizio, sui Web Service. La modellazione dei partecipanti all'asta attraverso l'uso di agenti software consente di sfruttare la caratteristica di alta flessibilità propria delle comunità di agenti (l'ingresso dinamico di nuovi agenti alla gara è una caratteristica nativa delle "agent society") e la loro proattività (ogni offerente è in grado di agire in maniera autonoma ed indipendente e può partecipare ad un asta o fare un rilancio senza dover necessariamente essere stimolato da un'altra entità).

Il sistema WASS si appoggia su di una base di dati per l'archiviazione delle informazioni relative a gare, fornitori e risorse da acquisire, e per la realizzazione di report. Oltre al data base, la struttura del WASS è formata da tre componenti fondametali:

- l'interfaccia web, per la parte grafica e di autenticazione. Rappresenta il mezzo di comunicazione fra gli utenti e il web service ed è materialmente il sito che rappresenta l'agenzia e dal quale partono tutti i servizi disponibili. Contiene tutti i controlli e form che servono per l'acquisizione dei dati necessari per l'esecuzione di query sul database e per l'immissione dei dati relativi all'appalto, alla verifica della situazione della gara, e della congruenza dei dati immessi.
- il Web Service, espone i servizi del WASS rendendoli accessibili alle pagine Web, consentendo ad esse l'accesso alla base di dati. Contiene materialmente le query che vengono richiamate dalle pagine web e restituisce i risultati delle interrogazioni alle stesse. Il Web Service è anche riferimento per la piattaforma ad agenti infatti contiene i metodi di avvio e gestione della contrattazione per la creazione di report sullo stato della gara.
- il sistema multi-agente, all'interno del quale gli agenti, creati con AgentService, rappresentano i fornitori e la Pubblica Amministrazione ed implementano l'intero meccanismo di contrattazione.

Possiamo ora riassumere il procedimento di attivazione ed esecuzione dell'asta. L'impiegato invia attraverso un pagina web la richiesta per una nuova gara d'appalto. La richiesta viene sottoposta al Web Service che accede al data base delle gare e costruisce un nuovo profilo di asta al quale verranno associati i possibili fornitori interessati in base alla categoria merceologica di appartenenza. La gara e la lista dei fornitori vengono quindi inviate alla piattaforma ad agenti, che attiva la contrattazione e dopo il tempo stabilito fornisce il risultato al Web Service che lo rende disponibile al sito Web.

A. Abilitazione dei fornitori alla gara

Le ditte fornitrici si iscrivono al sistema inviando una richiesta scritta alla Pubblica Amministrazione. In essa le ditte fornitrici devono inserire i dati identificativi della società:

- ragione o denominazione sociale;
- Partita IVA;
- Codice Fiscale;
- via e numero civico della sede legale della società;
- CAP della sede legale della società;
- città della sede legale della società;
- nazione della sede legale della società;
- rappresentante legale;
- categoria merceologica di appartenenza.
- caratteristiche dei beni forniti.

L'azienda deve inoltre dimostrare di essere in regola con i pagamenti INPS e INAIL e deve impegnarsi, qualora si aggiudichi una gara, a fornire i beni nella qualità e caratteristiche, che ha dichiarato di fornire, in sede di iscrizione.

Una volta accertate le credenziali l'ufficio della Pubblica Amministrazione comunicherà all'amministratore del sistema, che non fa parte della Pubblica Amministrazione, come gia specificato, i parametri della ditta. Quest'ultimo effettuerà la registrazione nel database, comunicando in maniera scritta alla ditta fornitrice l'avvenuta iscrizione con esito positivo.

La contrattazione è basata su agenti software è quindi necessario che ogni ditta fornitrice presenti tramite client Web le indicazioni per caratterizzare i comportamenti dei suoi agenti in modo da rendere completamente automatica la contrattazione. E' tuttavia possibile disabilitare e abilitare on line l'agente, per escluderlo o meno, da contrattazioni future, ed è prevista la costruzione di una serie di procedure per poter cambiare o aggiornare alcuni comportamenti degli agenti.

Il Sistema è impostato in modo da ricercare la ditta fornitrice da far partecipare alla gara in base alla categoria merceologica indicata in fase di registrazione.

B. Contrattazione

Attualmente, nella maggior parte dei casi, una gara di appalto o di fornitura utilizza il meccanismo dell'offerta a busta chiusa, in cui ogni partecipante effettua una offerta che non può più essere modificata e raggiunta la data di scadenza del bando, vengono aperte le buste e valutata l'offerta migliore. Nel sistema proposto viene invece istituita un'asta al ribasso, una delle tipologie d'asta indicate dalla regolamentazione degli appalti pubblici; tale modalità d'asta prevede che vengano formulate più offerte da parte di uno stesso fornitore. I valori delle offerte vincenti sono rese pubbliche in modo che tutti conoscano l'offerta migliore temporanea, mentre viene tenuto nascosto solo colui che l'ha formulata. Anche in questo caso, la contrattazione termina una volta scaduto il tempo.

All'interno del sistema WASS troviano due tipi di agenti, descritti in AgentService da due differenti template, che dovranno condurre la contrattazione:

- bidder agent, che rappresenta l'offerente, nel nostro caso interpreta il ruolo del fornitore;
- auctioneer agent, che rappresenta il banditore, nel nostro caso la pubblica amministrazione.

Analizziamo quindi come si articola lo svolgimento della gara. Dopo che la richiesta di appalto è stata sottoposta al WASS, il sistema si occuperà di trasmettere le informazioni necessarie al MAS; uno specifico agente di servizio in grado di interoperare con il *back-end* del Web Service si occuperà di comunicare all'agente banditore la descrizione della gara ed i possibili partecipanti selezionati dal WASS in base alla categoria merceologica.

L'auctioneer agent comunicherà ai bidder agent potenzialmente interessati l'apertura della nuova gara indicandone il tipo di contrattazione e la scadenza. A questo punto l'asta ha inizio. A tal proposito nell'applicazione sono stati implementati due tipi di aste al ribasso:

- semplice;
- pesata.

L'asta semplice è basata sul controllo dell'importo dell'offerta pervenuta. Ovviamente l'offerta con l'importo minore, si aggiudica la qualità di vincente temporaneo. Tutte le offerte successive vengono misurate in base al vincente temporaneo e vengono scartate tutte le offerte superiori. L'offerta minore alla scadenza si aggiudicherà l'asta.

L'asta pesata, si basa sul calcolo di un punteggio (P), pesato in base all'importo dell'offerta ed ai giorni di consegna.

$$P = a * W_i + b * W_t$$

I coefficienti W_i e W_t sono i pesi, mentre a è dato dal rapporto tra l'importo dell'offerta migliore e quello dell'offerta ricevuta, e b dal rapporto tra il miglior tempo di consegna (espresso in giorni) ed il tempo di consegna proposto. Si aggiudica l'asta colui che ottiene il punteggio maggiore al termine dei giorni previsti per la contrattazione.

Ogni volta che riceve una offerta, l'agente della pubblica amministrazione la confronta con l'offerta migliore temporanea, e periodicamente, calcola il vincitore momentaneo. Il controllo dell'offerta migliore viene effettuato da un comportamento dell'auctioneer agent; quindi grazie alla modularità dei behaviour di AgentService è possibile modificare con facilità il criterio della scelta del vincente in funzione del tipo di asta o di quanto indicato dalla gara di appalto. Basterà per questo che il programmatore fornisca all'agente banditore il nuovo comportamento, selezionandolo ad esempio dalle librerie di AgentService.

Alla fine di ogni giorno di contrattazione o, in ogni caso, dopo un determinato periodo di tempo, l'agente della pubblica amministrazione comunica il vincente a tutti gli altri agenti, in modo tale che possano eventualmente riformulare le loro offerte oppure decidere di abbandonare la contrattazione qualora avessero raggiunto i loro limiti di sconto applicabile imposti dai rispettivi fornitori.

Al termine dell'asta viene inviata una e-mail alla ditta vincitrice, nella quale vengono riepilogati i dettagli della gara, la descrizione completa degli articoli e delle loro caratteristiche tecniche, i tempi di consegna pattuiti, ecc. La ditta fornitrice dovrà rispondere alla e-mail, per confermare la fornitura, in caso contrario, trascorso un termine di tempo, si processerà la seconda migliore offerta.

Il progetto WASS, attraverso un servizio di report, fornisce in tempo reale una vista semplice e dettagliata dei messaggi che gli agenti si stanno scambiando nel corso di un processo di gara. Questo feed-back immediato su ciò che la piattaforma sta processando su un server remoto garantisce un elevato grado di trasparenza delle operazioni di contrattazione dal lato front-end sia del fornitore sia dell'impiegato statale. Al contempo il sistema mantiene l'anonimato dei fornitori partecipanti.

C. Bidder Agent

Analizziamo ora come è modellato il "template" dell'agente che rappresenta i partecipanti alla gara on-line. Secondo il modello proprio di AgentService, l'agente viene descritto attraverso i comportamenti, che ne caratterizzano l'attività, e le unità di conoscenza, che ne costituiscono il sapere.

Vediamo quindi quali sono le knowledge e i behaviour che definiscono il bidder agent:

Knowledge

- Active Auction: contiene le informazioni sulle aste a cui sta attualmente partecipando. Per ciascuna asta viene identificato il tipo, la quantità e il tipo di merci per cui si sta contrattando ed eventualmente il prezzo di partenza suggerito dall'acquirente;
- Auction Repository: contiene informazioni su ogni asta a cui l'agente ha partecipato, consiste in pratica in un archivio storico utile all'agente come base statistica per formulare offerte sempre più vincenti;
- Budget: è composta di tutti i dati necessari all'agente per formulare le offerte, quali ad esempio il prezzo limite e altre indicazioni stabilite dal fornitore che rappresenta;

Behaviour

- Communicator: concerne tutta l'attività di comunicazione

tra offerente e banditore: ricezione del bando dell'asta, invio delle offerte e ricezione delle informazioni sullo stato attuale dell'asta.

- Operator(s): implementa un particolare algoritmo di contrattazione strettamente legato al tipo di asta a cui l'agente partecipa.

- Manager: quando viene a conoscenza dell'inizio di un'asta decide in base al "Budget" se parteciparvi ed in caso di scelta favorevole con quale strategia contrattare con gli altri agenti.

Il *Template* del bidder agent può naturalmente essere modificato, definendo differenti *behaviour* e *knowledge*, o più facilmente personalizzando gli Operator. Ad ora sono implementati Operator con algoritmi per la contrattazione in aste al ribasso di tipo semplice, vickrey, busta chiusa, ed aste al rialzo di tipo inglese.

Vediamo ora come si articola l'attività del bidder agent durante la partecipazione ad un'asta. L'agente banditore dell'asta comunica l'inizio dell'asta attraverso il behaviour Communicator; quest'ultimo inserisce in Active Auction i parametri dell'asta e il prezzo di partenza e attiva il behaviour Manager.

Il Manager decide se partecipare o meno all'asta a seconda del prezzo limite e delle indicazioni contenute in Budget; se decide di parteciparvi sceglie l'algoritmo da utilizzare per calcolare l'offerta usando le strategie implementate in Operator. Il Manager poi si occuperà di aggiornare l'Auction Repository. Le offerte vengono poi inviate all'auctioneer agent tramite il Communicator, che, come visto nel paragrafo precedente, tra le tante offerte ricevute, stabilisce, in funzione del tipo di asta, quale è la vincente. Il banditore si occuperà quindi di notificare l'ammontare dell'offerta temporaneamente migliore a tutti gli agenti in gara per eventuali rilanci. Questo fino al sopraggiungere del tempo limite per l'asta.

VI. Conclusioni

Considerando il panorama estremamente eterogeneo per ciò che concerne l'Information Tecnology (IT) all'interno delle amministrazioni pubbliche, l'attenzione è stata rivolta ad aspetti quali l'interazione tra tecnologie diverse su varie piattaforme e su diversi dispositivi. L'adozione di standard aperti e l'assenza di vincoli di linguaggio o piattaforme da utilizzare risulta decisiva al fine di rendere raggiungibili i risultati prefissati. Da qui la scelta di uno strumento software quale il Web Service, basato su standard aperti e caratterizzato da portabilità e interoperabilità. Si è perciò voluto realizzare un'applicazione Web semplice, intuitiva e funzionale, che presentasse un alto grado di autonomia riducendo al minimo gli interventi, sia dell'amministrazione pubblica che dei fornitori. Per far fede a tale principio, è stato sviluppato un servizio che si avvalesse della tecnologia ad agenti proprio per la loro capacità di compiere azioni autonome in contesti complessi.

Gli agenti, grazie a caratteristiche quali autonomia e proattività, si candidano ad essere la strategia vincente per modellare il nascente quadro economico nel quale sempre più spesso sarà richiesto alla macchina di esibire comportamenti "intelligenti".

Il progetto WASS presenta ampi margini di miglioramento in quanto in questa sua prima realizzazione costituisce una solida infrastruttura software di base, sulla quale poter implementare ulteriori servizi e funzionalità. Il progetto è stato concepito proprio come punto di partenza estremamente "aperto" e flessibile in termini di:

- adattabilità ad eventuali nuovi vincoli normativi in materia di approvvigionamento per via telematica o ad esigenze peculiari di un ente pubblico. Queste possono risultare decisamente differenti per effetto della disomogeneità esistente tra le realtà comunali, provinciali e regionali sia a livello nazionale che in altri paesi;

- possibilità di implementare una più ampia casistica dei comportamenti (behaviour) per ogni singolo agente partecipante alla gara. Questo rende l'agente sempre più capace di adattarsi autonomamente alle differenti circostanze. Ad esempio, prevedendo cambi di comportamento di uno stesso agente fornitore in funzione dell'importo complessivo della commessa, dell'andamento della trattativa d'asta in corso, o della differente categoria merceologica oggetto della trattativa d'asta, ecc. In questo ambito la letteratura di riferimento dalla quale poter attingere nuove logiche comportamentali è rappresentata dalla Teoria dei giochi [23].

- possibilità di affiancare altre tipologie d'asta a quelle ad ora previste.

La suddivisione del progetto WASS in tre componenti distinti, permette di mantenere un ottimo livello di modularità e di chiarezza a servizio dello sviluppatore. In una visione di più ampio respiro e grazie a tale modularità, alla soluzione si potranno affiancare altri progetti in grado di integrarsi, interagire e di automatizzare i processi che precedono la gara d'appalto (studio di fattibilità, richiesta di finanziamento, autorizzazione, capitolato, pre-qualificazione), nonché i flussi documentali che gli stessi originano. Inoltre, per ottenere questa ulteriore semplificazione delle operazioni a carico degli impiegati statali, potrebbe essere necessaria una parallela riformulazione e standardizzazione di tali fasi.

Il sistema proposto, proprio per la sua architettura, ben si adatta ad operare anche in contesti differenti da quello del settore pubblico. A tal fine occorrerà, oltre alla personalizzazione dell'interfaccia, che i ruoli del banditore e dei partecipanti all'asta siano interpretati da soggetti differenti e che i relativi agenti che li rappresentano utilizzino gli appropriati algoritmi di contrattazione attraverso gli Operator della libreria del WASS (ad esempio una casa d'aste con un'asta all'inglese).

L'interesse verso l'informatizzazione dei processi delle pubbliche amministrazioni è dimostrato dal numero crescente di applicazioni che hanno per oggetto l'e-government; nel caso dell'acquisizione di risorse è interessante valutare e rapportare al WASS il progetto *eMarket* proposto dal Ministero delle Comunicazioni e Tecnologie Informatiche della Romania [24].

Come specificato dai realizzatori, il sistema *eMarket* è un progetto di commercio elettronico portato avanti dal governo della Romania nell'ambito dell' "European eGovernment Framework" nella forma di un mercato virtuale su internet. Il progetto pilota è iniziato nel marzo del 2002 con lo scopo di offrire una strada alternativa alle acquisizioni pubbliche. Le aste sono organizzate dalle istituzioni pubbliche e sono rese disponibili a qualsiasi società privata. Il meccanismo di offerta è molto semplice ed il sistema garantisce la vittoria al miglior offerente. L'*eMarket* sembra offrire molti dei vantaggi discussi per il WASS, primo fra tutti trasparenza e concorrenza nelle aste, ma il sistema di contrattazione è sostanzialmente differente dato l'impiego nel WASS della tecnologia ad agenti. Il WASS garantisce, per le caratteristiche prorpie dei MAS, un livello più elevato di automatizzazione del processo di contrattazione e maggiore flessibilità di utilizzo vista la facile riusabilità e personalizzazione dei componenti già realizzati e la possibilità di programmarne di nuovi progettati ad hoc per rispondere a esigenze differenti.

Considerando il mercato in cui si va a collocare un sistema fortemente autonomo come quello realizzato, è indubbia una certa perplessità da parte degli operatori nel delegare decisioni di importanza strategica "completamente nelle mani di un software". In tale contesto, potrebbe essere oggetto di studio una soluzione più equilibrata dal punto di vista del grado di interazione e decisione concesso ai suoi utenti, limitando gli agenti all'esecuzione automatica di task ripetitivi e quindi a strumenti computazionali per il supporto alle decisioni. Queste alternative potrebbero, in un secondo tempo, essere testate dal punto di vista delle prestazioni e confrontate con quelle del WASS.

BIBLIOGRAFIA

[1] eEurope 2005 Action Plan: http://europa.eu.int/information_society/eeurope/2005/all_about/action_plan/text_en.htm

[2] "E-government: nuovi paradigmi organizzativi e formativi nelle regioni e negli enti locali": http://www.di.unipi.it/parete/InItalia.html

[3] "E-government – maggior autonomia e iniziativa ai cittadini": http://www.eu.microsoft.com/italy/business/filePub/58382952wpEgovPA.pdf

[4] "E-government": http://cittadinionline.caltanet.it/egovern.shtml

[5] "La società dell'informazione": http://www.mininnovazione.it/ita/soc_info/politiche_governo/affariregionali.shtml

[6] DIRETTIVA 18 dicembre 2003 - Linee guida in materia di digitalizzazione dell'amministrazione per l'anno 2004. (GU n. 28 del 4-2-2004).

[7] A. Boccalatte, A. Gozzi, A. Grosso, C. Vecchiola, "AgentService", The Sixteenth International Conference on Software Engineering and Knowledge Engeneering (SEKE'04), Banff Centre, Banff, Alberta, Canada 20-24 June 2004.

[8] P. R. Wurman, , M. P.Wellman, and W. E.Walsh. The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents. In Second International Conference on Autonomous Agents (AGENTS'98), 1998.

[9] J. A. Rodrıguez-Aguilar, P. Noriega, C. Sierra, and J. Padget. Fm96.5 a java-based electronic auction house. In Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology(PAAM'97), 1997.

[10] P. Garcia, E. Gimenez, L. Godo, and J. A. Rodrıguez-Aguilar. Possibilistic-based design of bidding strategies in electronic auctions. In The 13th biennial European Conference on Artificial Intelligence (ECAI-98), 1998.

[11] F. Ygge and H. Akkermans. Making a case for multi-agent systems. In M. Boman and W. V. de Velde, editors, Advances in Case-Based Reasoning, number 1237 in Lecture Notes in Artificial Intelligence, pages 156–176. Springer-Verlag, 1997.

[12] F. Ygge and H. Akkermans. Power load management as a computational market. In Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), 1996.

[13] B. A. Huberman and S. Clearwater. A multi-agent system for controlling builging environments. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 171–176. AAAI Press, June 1995.

[14] M. P.Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. Journal of Artificial Intelligence Research, (1):1–23, 1993.

[15] Michael J. Wooldridge, Nicholas R. Jennings, Agent Theories, Architectures, and Languages: A Survey, Workshop on Agent Theories, Architectures \& Languages (ECAI'94), 1994.

[16] M. Wooldridge, "Intelligent Agents", in Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence, G. Weiss Ed., Cambridge, MA, pp. 27-78, 1999.

[17] G. Weiss, Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence, G. Weiss Ed., Cambridge, MA, 1999.

[18] Shen, W. and Norrie, D. (1999) "Agent-Based Systems for Intelligent Manufacturing: A state-of-the-Art Survey". Knowledge and Information Systems, 1(2):129-156.

[19] FIPA Abstract Architecture Specification, FIPA standard SC00001L: http://www.fipa.org/specs/fipa00001/SC00001L.pdf.

[20] Standard ECMA-335: Common Language Infrastructure (CLI) 2nd Edition, Dec. 2002, ECMA: http://www.ecma-international.org/publications/standards/Ecma-335.htm.

[21] Standard ISO/IEC 23271:2003: Common Language Infrastructure, March 28, 2003, ISO.

[22] Microsoft Shared Source CLI: http://msdn.microsoft.com/library/default.asp?url=/msdnmag/issues/02/07/sharedsourcecli/toc.asp

[23] S. Tijs. Introduction to Game Theory, New Delhi, Hindustan Book Agency, 2003.

[24] E-market, Electronic System for Public Acquisition: http://www.e-licitatie.ro/index_en.htm

Coordinated Change of State for Situated Agents

Giuseppe Vizzari and Stefania Bandini
Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano–Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
{giuseppe.vizzari, bandini}@disco.unimib.it

***Abstract*— Situated Multi Agent System models are characterized by the representation and exploitation of spatial information related to agents, the environment they inhabit and their positions. Specific coordination mechanisms exploiting the contextual spatial information can be defined. In particular this paper will focus on issues and proposed solutions related to the coordinated change of state for situated agents.**

I. Introduction

Agent coordination represents a very active and challenging area of the research in Multi-Agent Systems (MAS). The term coordination refers to the interaction mechanisms that allow autonomous agents to select and carry out their actions within a concerted framework. The separation of the agent computation model, specifying the behaviour of a single agent, from the coordination model is a proposal that goes back to the early nineties [7]. In particular, the concept of Linda tuple space [6] and the related coordination language is the most diffused metaphor adopted by current coordination languages and approaches. The basic model has been enhanced, on one hand at a technical level, in order to allow a distributed implementation of the conceptually centered tuple space [16]. On the other hand, tuple spaces have been also extended in order to allow the specification of tuple-based coordination media presenting reactive and programmable behaviours (see, e.g., [14], [15], [4]), and also allowing the specification and enforcement of organizational abstractions and constraints (e.g. roles, access control rules) to agent coordination [17].

Situated MASs (see, e.g., [1], [10], [20]) are particular agent based models which provide the representation and exploitation of spatial information related to agents and their position into the environment they inhabit. While the previously defined approaches to agent coordination provide general-purpose coordination languages and mechanisms, situated MASs present issues that could benefit from specific mechanisms for agent interaction. For instance, the concept of *field* (i.e. a signal that agents may spread in their environment, which can influence the behaviour of other entities) has been widely adopted for the generation of coordinated movements (see, e.g., [2], [9]). This kind of mechanism is devoted to the interaction of agents which may be positioned on distant points of their space, there can be situations in which agents which are in direct contact (considering a discrete representation of agents' environment) may wish to perform a coordinated change in the respective state (for instance in order to model the exchange of information) without causing modifications in the environment. In fact, field based interaction and other approaches focused on modelling agent environment [19], are intrinsically multicast interaction mechanisms that may be useful to represent actions and interactions that should be *observable* by other entities in the system. However this observability property should not automatically characterize all possible actions and interactions of a Multi Agent model. To this purpose, Multilayered Multi Agent Situated System (MMASS) [1] defines the *reaction* action which allows the coordinated change of the states of agents which are positioned in sites forming a clique (i.e. a complete subgraph) in the spatial structure of their environment. This operation, which also allows a direct exchange of information among the involved entities, is not observable by other agents. The aim of this paper is to describe issues related to coordinated changes in the state of situated agents, and propose approaches for the management of these issues, with specific reference to the reaction MMASS action.

The following section will better describe the problem, showing how existing situated MAS approaches tackle the issue of coordinated agent change of state. Section III will focus on the design and implementation of mechanisms supporting coordinated change of state of situated agents, discussing synchronous and asynchronous cases. Conclusions and future developments will end the paper.

II. Coordinated change of state in situated MASs

Despite most agent definitions emphasize the role of the environment, currently most model do not include it as a first class abstraction. The number of situated MAS models (that are models providing a representation of spatial features of agent environment) is thus relatively small, and the topic of coordinating the change of state of situated agents is still not widely analyzed.

One of the first approaches providing the possibility to define the spatial structure of agents' environment is represented by Swarm [12]. Swarm and platforms derived by it (e.g. Repast[1], Mason [8]) generally provide an explicit representation of the environment in which agents are placed, and often provide mechanisms for the diffusion of signals. Nonetheless they generally represent useful libraries and tools for the implementation of simulations, but do not provide a comprehensive, formally defined *interaction model*. In other words they do not provide support to the coordinated change of state among agents, but just define and implement a spatial

[1] http://repast.sourceforge.net

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

structure in which agents, and sometimes signals, may be placed. Moreover, they generally provide a sequential execution of agents' behaviours (that are triggered by the environment, which is related to the only thread of execution in the whole system). This approach prevents concurrency issues and allows to obtain compact and efficient simulations even with a very high number of entities. The price of these characteristics is essentially that agents are not provided with a thread of execution of their own (i.e. they have a very limited autonomy and proactiveness), and the execution of their behaviours is sequential (although not necessarily deterministic).

The Co-Fields [10] and the Tuples On The Air (TOTA) middleware [11] provide the definition and implementation of a field based interaction model, which specifically supports this kind of interaction that implies a local modification of agents' environment. However the defined interaction mechanism does not provide the possibility to have a coordinated change of agent state without such a modification.

A different approach to the modelling and implementation of situated MAS [20] instead focuses on the definition of a model for simultaneous agent actions, including centralized and (local) regional synchronization mechanisms for agent coordination. In particular, actions can be independent or interfering among each other; in the latter case, they can be mutually exclusive (*concurrent* actions), requiring a contemporary execution in order to have a successful outcome (*joint* actions), or having a more complex influence among each other (both positive or negative).

The previously introduced MMASS model provides two mechanisms for agent interaction. The first is based on the concept of *field*, that is a signal that may be emitted by agents, and will spread in the environment according to its topology and to specific rules specifying field diffusion functions. These signal may be perceived by agents which will react according to their specific behavioural specification. The model also defines the possibility for having a coordinated change of agent state through the *reaction* operation. The outcome of this joint action depends on three factors:

- agents' *positions*: reacting agents must placed in sites forming a complete subgraph in the spatial structure of the environment;
- agents' *behavioural specifications*: agents must include compatible reaction actions in their behavioural specification;
- agents' *willingness* to perform the joint action: one of the preconditions for the reaction is the agreement among the involved agents.

The following section will discuss issues related to the design and implementation of this operation, but several considerations are of general interest in the development of mechanisms supporting the coordinated change of state for situated agents.

III. Reaction

Reaction is an activity that involves two or more agents that are placed in sites forming a clique (i.e. a complete subgraph) and allows them to change their state in a coordinated way, after they have performed an agreement. The MMASS model does not formally specify what this agreement process consists of, and how the activities related to this process influence agent behaviour. This choice is due to the fact that such an agreement process could be very different in different application domains (e.g. user authentication, transactions). For instance, in some of these situations an agent should block its activities while waiting for the outcome of the agreement process, while in others this would be unnecessary. Especially in a distributed environment this agreement process could bring to possible deadlocks, and in order to better focus this subject, more details on internal mechanisms related to agent, to the environment and its composing parts must be given.

```
begin
turn:=0;
do
    begin
    localContext:=environment.sense(turn);
    nextAction:=actionSelect(localContext);
    outcome:=environment.act(nextAction,turn);
    if outcome<>fail then
        turn:=turn+1;
    end
while(true);
end
```

Fig. 1. Agent behaviour thread in a synchronous situation.

A. Synchronous environments

In synchronous situations a global time step regulates the execution of agents actions; in particular, every agent should be allowed to carry out one action per turn. In order to enforce synchronicity, the management of system time step and agent actions can be delegated to agents' *environment*, that they invoke not only for functional reasons (i.e. perform an action which modifies the environment) but also to maintain system synchronicity (i.e. agent threads are put into a wait condition until the environment signals them that the global system time step has advanced). This proposal assumes that agents are provided with one thread of execution, and also provides that the environment has at least one thread of execution of its own. In fact the environment is responsible for the management of field diffusion (more details on this subject can be found in [3]), other modifications of the environment (as consequences of agents' actions), and to enforce system synchronicity.

In the following, more details on agent and environment activities and threads of execution will be given; the situation that will be considered provides one thread for every agent, and a synchronous system. The described approach is valid both for centralized and for distributed situations; in the latter case one of the sites must be elected as a representative of the whole environment, and interactions with the environment can be implemented through a remote invocation protocol (e.g. RMI or others, according to the chosen implementation platform).

1) Agent behaviour management thread: The sequence of actions performed in the agent behaviour thread is the following:

```
begin turn:=0;
do
    begin
    until(forall i in 1..n, agent_i.actionperformed=true)
        begin
        collect(agent_i,action,agentTurn)
        if agentTurn=turn then
            begin
            manage(agent_i,action, turn);
            agent_i.actionperformed:=true;
            end
        else
            agent_i.wait();
        end
    turn:=turn+1;
    forall i in 1..n
        agent_i.actionperformed:=false;
    notifyAllAgents();
    end
while(true);
end
```

Fig. 2. Environment behaviour thread in a synchronous situation.

```
procedure reactionManagement(agent, action, turn)
    begin
    involvedAgents:=action.getReactionPartners();
    reactingAgents:=new list();
    reactingAgents.add(agent);
    agreed:=true;
    forall agent_i in involvedAgents
        begin
        if agent_i.agreeReaction(involvedAgents) = false then
            begin
            agreed:=false;
            break;
            end
        reactingAgents.add(agent_i);
        end
    if agreed=true then
        forall agent_i in reactingAgents
            agent_i.performReact(turn);
    else
        forall agent_i in reactingAgents
            agent_i.notifyFailure(turn);
    end
```

Fig. 3. Reaction management procedure in a synchronous situation.

- *sense its local context*: in order to understand what are the actions whose preconditions are verified, the agent has to collect information required for action selection, and more precisely:
  - active fields in the site it is positioned on and adjacent ones;
  - agents placed in adjacent sites, and their types;
- *select which action to perform*: according to the action selection strategy specified for the system (or for the specific agent type), the agent must select one action to be performed at that turn (if no action's preconditions are satisfied, the agent will simply skip the turn);
- *perform the selected action*: in order to perform the previously selected action, the agent must notify the environment, because the action provides a modification of agent's local context or even simply to maintain system synchronicity.

The last step in agent behavioural management cycle may cause a suspension of the related thread by the environment. In fact an agent may be trying to perform an action for turn t while other ones still did not perform their actions for turn $t-1$. A pseudo-code specification of agent behavioural thread sequence of activities is shown in Figure 1. Agents must thus keep track of current turn and of the previously performed action. In fact, as will be introduced in the following subsection, system dynamics might require an agent to reconsider its action when it is involved in a reaction process.

2) Environment management thread: The environment, more than just managing information on agents' spatial context, also acts as a monitor in order to handle concurrency issues (e.g. synchronization, agreements among agents). Agents must notify the environment of their actions, and the latter will manage these actions performing modifications to the involved structures (e.g. sites and active fields) related to the following turn. The state of the current one must be preserved, in order to allow its sensing and inspection by agents which still did not act in that turn.

The environment may also put an agent into a *wait* condition, whenever performing its action would break system synchronicity. This wait ends when all agents have performed their action for the current turn, and thus all entities are free to perform actions for the next one. The environment must thus keep track of the actions performed by agents in the current turn, and then notify waiting agents whenever system time advances. More schematically, a pseudo-code description of the environment thread of execution is shown in Figure 2. In particular the `manage` function inspects the specified action (which includes the required preconditions and parameters), checks if it is valid and then calls the appropriate subroutines which effectively perform actions.

The previously introduced sequences require a slight integration to specify how reaction actions are managed. In this case the beginning of an agreement process stops other agent actions until this process is over, either positively (when all other involved agents agreed) or negatively (when the agreement failed). In this way, also system time advancement is stopped until the reaction process is over, preserving system synchronicity.

The reaction is triggered by the agent which first requires the execution of this action to the environment. The latter becomes the leader of the group of involved agents, queries them asking if they agree to take part in the reaction, if an agreement is reached it signals them to change their state, then starts again the normal system behaviour, allowing the advancement of global system time step and thus agent execution. More schematically the environment procedure devoted to the management of reaction is shown in 3. An agent receiving a `notifyFailure` will have a `fail` outcome, and thus will not advance its time step and will start over again its behavioural cycle for the current turn. The `reactionManagement` procedure is one of the specific subroutines invoked by the the environment thread of execution previously shown in Figure 2 through the `manage` function.

3) Examples: A sample scenario illustrating the evolution of a centralized synchronous MMASS system is shown is Figure 4. Scenario (a) provides the presence of a set of agents (Agent-1, ..., Agent-n), which do not require the execution of reaction actions. The system dynamics is the following:

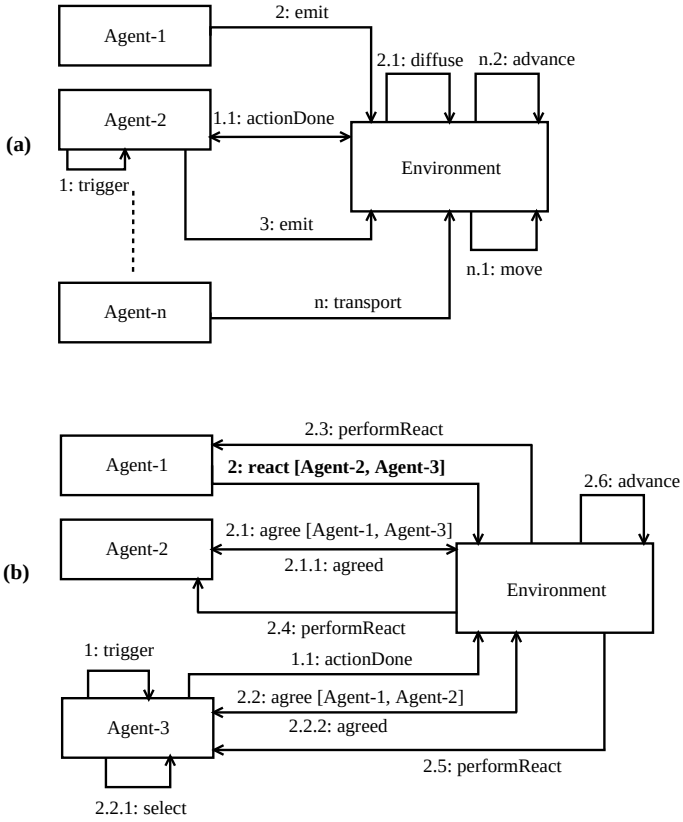


Fig. 4. A sample scenario illustrating the evolution of a centralized synchronous MMASS system. In (a) Agent-2 is put into a wait condition to preserve system synchronicity. In (b) an agreement process for a reaction among Agent-1, Agent-2 and Agent-n is shown.

- Agent-2 performs a trigger (action 1);
- Agent-1 emits a field (action 2) and as a consequence the environment performs its diffusion (action 2.1);
- Agent-2 also tries to perform an emission (action 3), but the environment puts it into a wait condition, as other agents did not perform their actions in that turn;
- agents that are not shown in the Figure perform their actions, which are managed by the environment;
- eventually Agent-n performs a transport action (action n), and as a consequence the environment performs its movement (action n.1), advances system time (action n.2) and eventually notifies agents. Agent-2 emit action (action 3) will now be managed.

A different case is shown in scenario (b), which exemplifies the sequence generated by a reaction request. Agent-1, Agent-2 and Agent-3 are positioned in sites forming a clique. In this case system dynamics is the following:

- Agent-3 performs a trigger (action 1);
- Agent-1 requires the environment to perform a reaction with Agent-2 and Agent-3 (action 2);
- as a consequence to this request, the environment asks Agent-2 if it intends to agree in preforming the reaction (action 2.1) and it receives a positive reply (action 2.1.1);
  the environment then asks Agent-3 if it wishes to reconsider its action for the current turn (action 2.2); the agent performs anew an action selection (action 2.2.1) and decides to agree(action 2.2.1);
- the environment indicates all involved agents that they must perform the reaction (actions 2.3 – 2.5) and then advances system time.

```
begin
do
    begin
    localContext:=mysite.sense();
    nextAction:=actionSelect(localContext);
    outcome:=site.act(nextAction);
while(true);
end
```

Fig. 5. Agent behaviour thread in an asynchronous situation.

4) Discussion: The previously described approach to the management of agents, their cycle of execution, their environment and reaction mechanisms provides a key role of the environment, which represents a sort of medium ensuring specific properties, and especially system synchronicity. This is a global feature of the system, and the simplest way to ensure it is to have a conceptually centralized unit to which all entities must refer in order to perform their actions. This medium and coordination models providing a centralized medium (e.g. a tuple space) seem thus similar, in fact, both provide an indirect interaction among agents and must tackle issues related to the concurrent access to shared resources. The main difference is the fact that, for instance, a Linda tuple space does not provide abstractions for the definition of spatial information (e.g. a topology, an adjacency relation), that should be modelled, represented and implemented. An interesting feature of advanced artifact based interaction models, and more precisely reactive and programmable tuple spaces, is the possibility to specify a behaviour for the artifact, which could be a way to implement interaction mechanisms defined by the MMASS model.

The described approach provides computational costs that could be avoided, in a centralized situation, by providing a single thread of execution, preventing synchronization issues by activating agents in a sequential (although non necessarily deterministic) way (i.e. adopting the approach exploited by Swarm–like simulation platforms). Whenever autonomy and proactiveness are not central elements in agent modelling, this could be a feasible and cost effective choice. It could be the case of simulations characterized by a large number of entities endowed with very simple behavioural specification. However, the described approach can useful when integrating into a single environment entities characterized by a higher degree of autonomy, proactiveness and heterogeneity (for instance, reactive and deliberative agents developed with deeply different approaches).

B. Asynchronous environments

In an asynchronous situation, the mechanisms for the management of agents and their interactions with the environment, are on one hand simpler than in a synchronous case (i.e. there is no need to ensure that every agent acts once per turn), but can also be more complex as there less constraints on action timings. In a centralized situation, it is still possible to delegate

```
begin
do
    begin
    reactionRequest:=mysite.getReactionRequest();
    newReactManager:=new ReactManagerThread(reactionRequest);
    newReactManager.start();
while(true);
end
```

Fig. 6. Agent reaction detection thread in an asynchronous situation.

the management of shared resources to an environment entity, whose task is actually simpler than in a synchronous situation as it does not have to maintain global system synchronicity, although it must guarantee the consistent access to shared resources. In a distributed and asynchronous situation, even if it would be possible to elect a single representative of agents' environment (like in the synchronous and distributed case, described in the previous Section), this possibility would represent a bottleneck and is not even necessary. In fact, the main reason for the presence of a single representative of agent environment was to assure system synchronicity. This Section will then focus on a distributed and asynchronous scenario, and will describe a distributed approach providing the collaboration of *sites*, instead of a single centralized environment, for the management of coordinated change of agents' states.

1) Agent related threads: As previously introduces, agents will now collaborate directly with the sites they are placed on, and their behavioural threads must thus be changed. A pseudo-code formalization of agent behaviour thread in an asynchronous situation is shown in Figure 5.

Another change that can be introduced in the agent is the presence of a distinct thread for the management of reaction requests. In fact the agreement process required by the reaction process can require a certain number of interaction among agents which are placed in computational units spread over a network. This means that a relevant delay may occur from the beginning of an agreement process and its outcome (either positive or negative). Being in an asynchronous situation there is no need to stop agent behavior in order to wait for this process to end. An agent may be provided with three kinds of threads:

- its behavioural thread, which is very similar to the one related to the synchronous situation, and whose structure is shown in Figure 5;
- a thread which is devoted to the detection of reaction requests; this thread is responsible to query the site for pending reaction requests (which may occur concurrently) and start the third kind of thread which will manage the agreement process; a pseudo-code formalization of this thread is shown in Figure 6;
- threads that are devoted to the effective management of the reaction process; a pseudo-code formalization of this thread is shown in Figure 7. This kind of thread must check if the agent effectively agrees to perform the reaction, through the `checkAgreement` invocation (only if it is not the one which actually started the reaction process). This means that first of all the agent must have a react action matching the one specified by the request

```
begin myReactAction:=this.getAction(reactionRequest); if
myReactAction<>null then
    begin
    if reactionRequest.author <> this then
        begin
        agreed:=checkAgreement(reactionRequest);
        site.replyReactReq(reactionRequest, agreed);
        end
    if agreed=true then
        begin
        agreemReached:=site.getReactAgreement(reactionRequest);
        if agreemReached=true then
            this.changeState(myReaction.nextState);
        end
    end
else
    site.replyReactReq(reactionRequest, false);
end
```

Fig. 7. Agent reaction management thread in an asynchronous situation.

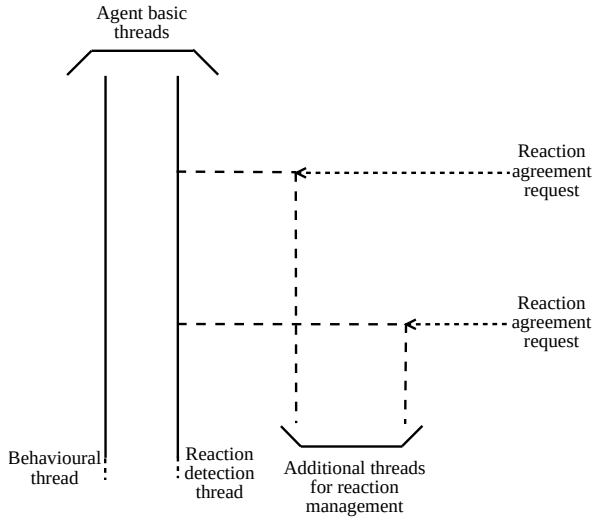


Fig. 8. Threads of execution related to a single MMASS agent in a distributed asynchronous environment.

(this is checked through the `getAction` invocation). Then it must wait the notification of the success or failure of the agreement (the `getReactAgreement` invocation may in fact suspend this thread) and, in the former case, change the agent state.

A diagram showing the three kinds of thread related to a single agent are shown in Figure 8.

2) Site related threads: Similar considerations on the internal structure of agents may be also done for sites. The latter act as a interfaces between agents and the rest of the environment, and must manage events generated both internally and externally. In particular, *internal events* are generated by an agent that is positioned on the site, and more precisely they are the following ones:

- *sense the local context*: the site must provide an agent with the information it needs to select which action it may perform (active fields in the site and adjacent ones, agents in adjacent positions and related types);
- *transport request*: when an agent attempts a transport action, the site it is positioned on must communicate with the destination one in order to verify if it is empty, and eventually allow the agent movement, which frees the

```
procedure reactionManagement(agent, action)
    begin
    involvedAgents:=action.getReactionPartners();
    reactingAgents:=new list();
    reactingAgents.add(agent);
    agreed:=true;
    forall agent_i in involvedAgents
        begin
        adjSite:=agent_i.getSite();
        adjSite.reqAgreement(action);
        end
    until(forall a in involvedAgents, a.gotResponse)
        begin
        if receiveAgreeResp(agent_i,action) = false then
            begin
            agreed:=false;
            break;
            end
        reactingAgents.add(agent_i)
        end
    if agreed=true then
        forall agent_i in reactingAgents
            begin
            adjSite:=agent_i.getSite();
            adjSite.performReact(action);
            end
    else
        forall agent_i in reactingAgents
            begin
            adjSite:=agent_i.getSite();
            adjSite.performReact(adjSite);
            end
    end
```

Fig. 9. Reaction management procedure for the leader site in an asynchronous situation.

current site;

- *reaction request*: upon reception of a reaction request by the overlaying agent, the site must propagate it to involved agents' sites, which in turn will notify them. The site must wait for their replies and then notify all involved entities of the agreement operation outcome; in other words, the site where the reaction is generated is the *leader* of the group of involved sites; a pseudo-code formalization of the reaction management procedure for the leader site is shown in Figure 9;
- *field emission*: when a field is generated in a site it must be added to the set of active fields present in the site, and it must be propagated to other adjacent sites according to the chosen diffusion algorithm.

With reference to reaction, and especially on the selection of a leader site, there are some additional elements that must be integrated with the previous description of site behaviour. In an asynchronous environment, there is the possibility that two agents concurrently start two related reactions. For instance, given three agents A, B and C, placed in sites forming a clique, agent A and Agent B require their respective sites to react among themselves and with agent C. There is not a single site which started the reaction, so a leader must be chosen. Whenever this kind of situation occurs an election protocol must be invoked. The first and probably simplest solution, is to associate a unique identifier related to every site (a very simple way of obtaining it could be the adoption of a combination of the IP address and TCP port related to the site) and assume that the one with the lowest identifier becomes the leader of the reaction group, and others will behave as the reaction request was generated by the leader.

```
procedure reactionManagement(site, action)
    begin
    if this.agent <> null then
        begin
        this.agent.notifyReaction(action);
        agreed:=getReactReply(agent,action);
        site.replyReact(agreed);
        if agreed=true then
            if site.reqAgreement()=true then
                this.agent.setReactAgreement(action,true);
        end
    else
        site.replyReact(false);
    end
```

Fig. 10. Reaction management procedure for non-leader sites in an asynchronous situation.

Externally generated events are consequences of internal events generated by agents in other sites; more precisely they are the following ones:

- *inspect the site*: upon request, the site must provide to adjacent sites information related to active fields and to the presence (or absence) of an agent in it;
- *diffusion propagation*: when a field generated in a different site is propagated to the current one the latter must evaluate its value through the related diffusion function and, if the value is not null, it must propagate the field to other adjacent sites according to the adopted diffusion algorithm;
- *reaction request*: upon reception of a reaction request by the leader of a reaction group, the site must forward it to the overlaying agent, wait for its response and transmit it back to the leader; then it must wait for the outcome of the reaction and notify the overlaying agent; a more schematic description of non-leader sites behavior for management of reaction is shown in Figure 10;
- *transport*: when a remote agent attempts a transport action, the destination site must verify if its state has changed from the previous inspection performed by the agent, and if it is still empty will allow the transport action, blocking subsequent incoming transports.

Site is thus responsible for many concurrent activities; the proposed structure of threads for a site is shown in Figure 11: there are two threads respectively detecting internal and external events, and these two threads generate additional ones in order to effectively manage them.

3) Inter-thread communication: Both agents and sites are provided with a set of threads which must be able to communicate among themselves in a safe and consistent way. For instance, agent reaction management thread in an asynchronous situation communicates to the underlying site by means of a `replyReactRequest` invocation (see Figure 7). The latter performs a write operation on a thread-safe queue, that is a structure with synchronized accessors (observers and modifiers) that may be accessed by site threads but also by the ones related to the agent that is placed on it. The `replyReactRequest` invocation inserts an event in this queue, and notifies threads that were waiting for the generation of events. In this case the thread interested in the agent reply to the reaction request is the one related to the underlying site which effectively manages the agreement process with other

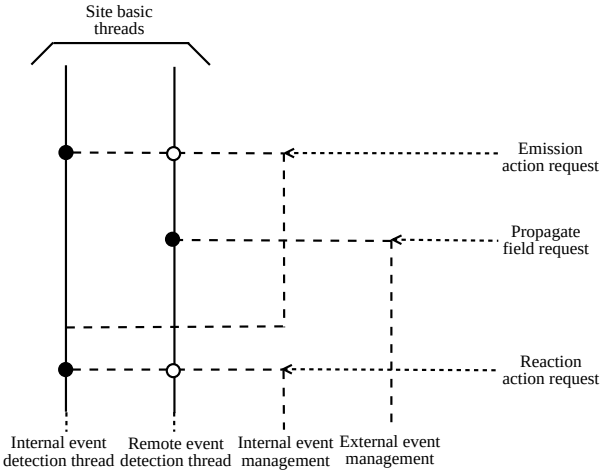


Fig. 11. Threads of execution related to a single MMASS site in a distributed asynchronous environment.

involved entities. It could be either the leader, which is put into a wait condition by the and the `receiveAgreeResp` invocation (see Figure 9), or any other involved site, which is put into a wait condition by a `getReactReply` invocation (see Figure 10).

4) Precautions on network communication: What was still not considered is the possibility to have failures in network transmission, even if to design a robust distributed protocol for reaction management is not the focus of this work. Moreover the chosen technologies supporting network communication could implement mechanisms assuring a reliable form of communication. However, considering the simple loss of messages related to the orchestration of reaction, a simple protocol providing the transmission acknowledgements and the definition of timeouts in order to avoid deadlock situations could be easily implemented. Whenever this kind of issue is detected, the agents' threads related to the management of reaction could simply try to repeat the whole process from the beginning. Moreover, the fact that every agent is related to multiple threads of control, greatly reduces the dangers and issues related to possible deadlocks. In fact, the agent behaviour thread is separated from the management of reactions, and the same can be said for what concerns site specific functions (e.g. threads related to field diffusion are separated from those managing reactions). In this way a failure in a reaction process does not hinder the possibility of the agent to continue its common behaviour, leaving aside the specific reaction that caused the problem. This price of these advantages is that agents and sites are more complex from a computational perspective, and require more resources both in terms of memory and processor time.

There are also some functional requirements that must be considered: the execution of an action during an agreement process might change the preconditions that brought an agent to accept the proposed agreement. In specific cases this could represent a serious issue, and in this case the possibility of the reaction management thread to temporarily block the agent behavioural one should be introduced, suitably exploiting the inter thread interaction mechanism.

5) Discussion: Some of the concurrency issues that were described in this Section are common also in direct agent interaction models. In fact, they are generally designed to work in an asynchronous situation in which messages may be sent and received at any time. In order not to miss any message, the communication partners require some kind of indirection mechanism and structure. For instance, the abstraction of *mailbox* is adopted by Zeus [13], and Jade [18] uses *queues* for managing agent messages. In both cases, specific threads of execution, in addition to those that are related to agents, are adopted to manage communication channels and message exchange.

Unlike the synchronous approach, in this case no single entity managing the coordinated change of state among agents is provided. While managing this kind of operation in a distributed way provides a more complex implementation of sites, to which this activity is delegated, this approach seems more suitable in distributed situations, unless synchronization is absolutely necessary. In fact, a single entity managing this operation may represent a bottleneck and a single point of failure, hindering system robustness.

IV. Conclusions and future developments

The paper has discussed issues related to the coordinated change of state for situated MASs, proposing specific solutions for synchronous and asynchronous situations. In particular, the MMASS reaction action was considered as a specific case of coordinated change of state in situated agents, but several considerations are of general interest in the design and implementation of mechanisms supporting this form of coordinated action in situated MASs. In particular the approach described in [20] provides a similar approach to situated agents coordination: in fact it provides a centralized synchronization, similar to the one provided by the environment described in Section III-A. A distributed mechanism for agent coordination is also described, but it provides a personal synchronizer for every agent while in the approach described in Section III-B every site is responsible for providing this kind of service to the hosted agent.

This work is part of a wider research aimed at the design and development of a platform supporting the development of MMASS based systems. In this framework, another work focused on supporting field diffusion [3], while agent movement will be object of a through analysis: in fact this mechanism requires an attention to functional aspects (e.g. concurrent agents' attempts to move towards the same empty site) and also non-functional ones related to agent mobility in distributed environments. In particular the latter represents a whole area in agent research and software engineering in general (see, e.g., [5]).

References

[1] Stefania Bandini, Sara Manzoni, and Carla Simone, "Heterogeneous agents situated in heterogeneous spaces.," *Applied Artificial Intelligence*, vol. 16, no. 9-10, pp. 831–852, 2002.

[2] Stefania Bandini, Sara Manzoni, and Giuseppe Vizzari, "Situated cellular agents: a model to simulate crowding dynamics," *IEICE Transactions on Information and Systems: Special Issues on Cellular Automata*, vol. E87-D, no. 3, pp. 669–676, 2004.

[3] Stefania Bandini, Sara Manzoni, and Giuseppe Vizzari, "Towards a specification and execution environment for simulations based on mmass: Managing at–a–distance interaction," in *Proceedings of the 17th European Meeting on Cybernetics and Systems Research*, Robert Trappl, Ed. 2004, pp. 636–641, Austrian Society for Cybernetic Studies.

[4] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli, "Mars: a programmable coordination architecture for mobile agents," *IEEE Internet Computing*, vol. 4, no. 4, pp. 26–35, 2000.

[5] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, 1998.

[6] David Gelernter, "Generative communication in linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, 1985.

[7] David Gelernter and Nicholas Carriero, "Coordination languages and their significance," *Communications of the ACM*, vol. 35, no. 2, pp. 97–107, 1992.

[8] Sean Luke, G. C. Balan, Liviu A. Panait, C. Cioffi-Revilla, and S. Paus, "Mason: a java multi-agent simulation library," in *Proceedings of Agent 2003 Conference on Challenges in Social Simulation*, 2003.

[9] Marco Mamei, Letizia Leonardi, and Franco Zambonelli, "A physically grounded approach to coordinate movements in a team," in *Proceedings of the 1st International Workshop Mobile Teamwork*. 2002, pp. 373–378, IEEE Computer Society.

[10] Marco Mamei, Franco Zambonelli, and Letizia Leonardi, "Co-fields: Towards a unifying approach to the engineering of swarm intelligent systems," in *Engineering Societies in the Agents World III: Third International Workshop (ESAW2002)*. 2002, vol. 2577 of *Lecture Notes in Artificial Intelligence*, pp. 68–81, Springer–Verlag.

[11] Marco Mamei and Franco Zambonelli, "Programming pervasive and mobile computing applications with the tota middleware," in *2nd IEEE International Conference on Pervasive Computing and Communication (Percom2004)*. 2004, pp. 263–273, IEEE Computer Society.

[12] Nelson Minar, Roger Burkhart, Chris Langton, and Manor Askenazi, "The swarm simulation system: A toolkit for building multi-agent simulations," Working Paper 96-06-042, Santa Fe Institute, 1996.

[13] Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee, and Jaron C. Collis, "Zeus: A toolkit for building distributed multiagent systems," *Applied Artificial Intelligence*, vol. 13, no. 1-2, pp. 129–185, 1999.

[14] Andrea Omicini and Enrico Denti, "From tuple spaces to tuple centres," *Science of Computer Programming*, vol. 41, no. 3, pp. 277–294, 2001.

[15] Andrea Omicini and Franco Zambonelli, "Coordination for Internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sept. 1999, Special Issue: Coordination Mechanisms for Web Agents.

[16] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman, "Lime: Linda meets mobility," in *Proceedings of the 21st International Conference on Software Engineering (ICSE99)*. 1999, pp. 368–377, ACM press.

[17] Alessandro Ricci, Mirko Viroli, and Andrea Omicini, "Agent coordination context: From theory to practice," in *Cybernetics and Systems 2004*, Robert Trappl, Ed., Vienna, Austria, 2004, vol. 2, pp. 618–623, Austrian Society for Cybernetic Studies, 17th European Meeting on Cybernetics and Systems Research (EMCSR 2004), Vienna, Austria, 13–16 Apr. 2004. Proceedings.

[18] Giovanni Rimassa, *Runtime Support for Distributed Multi-Agent Systems*, Ph.D. thesis, University of Parma, January 2003.

[19] Luca Tummolini, Cristiano Castelfranchi, Alessandro Ricci, Mirko Viroli, and Andrea Omicini, ""Exhibitionists" and "voyeurs" do it better: A shared environment approach for flexible coordination with tacit messages," in *1st International Workshop on "Environments for MultiAgent Systems" (E4MAS 2004)*, Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, Eds., 2004, pp. 97–111.

[20] Danny Weyns and Tom Holvoet, "Model for simultaneous actions in situated multi-agent systems," in *First International German Conference on Multi-Agent System Technologies, MATES*. 2003, vol. 2831 of *LNCS*, pp. 105–119, Springer–Verlag.

Timed Coordination Artifacts with ReSpecT

Alessandro Ricci
DEIS
Università di Bologna – Sede di Cesena
via Venezia 52, 47023 Cesena (FC), Italy
Email: aricci@deis.unibo.it

Mirko Viroli
DEIS
Università di Bologna – Sede di Cesena
via Venezia 52, 47023 Cesena (FC), Italy
Email: mviroli@deis.unibo.it

***Abstract*— Environment-based approaches to Multi-Agent Systems (MAS) advocate the use of abstractions mediating the interaction between agents, providing an alternative viewpoint to the standard speech-act-based approach. A remarkable example is rooted in the notion of *coordination artifact*: embodied entities provided by the MAS infrastructure to automate a specific coordination task, and featuring peculiar engineering properties such as encapsulation, predictability, inspectability and malleability. An example technology supporting this scenario is TuCSoN, where coordination artifacts are built as tuple centres programmed with the ReSpecT logic language.**

In most application scenarios characterised by a high degree of openness and dynamism, coordination tasks need to be time-dependent so as to be able to specify and guarantee necessary levels of liveness and of quality of service. Moreover, temporal properties are also fundamental for intercepting violations in the agent-artifact contract, which is at the root of the engineering approach underlining coordination artifacts. Accordingly, in this paper we introduce an extension to the ReSpecT language allowing to define timed coordination artifacts in the TuCSoN infrastructure. This is achieved by adding the management of trap events, fired and intercepting using the same mechanism currently used by ReSpecT to handle communication events, thus in a uniform and coherent way. Examples are provided to show the expressiveness of the language to model temporal-based coordination tasks.

I. Introduction

In the context of Environment-based approaches to interaction on Multi-Agent Systems (MAS), the notion of *coordination artifact* has been introduced as the root of an engineering methodology for agent coordination [10], [15]. The key idea of this approach is to equip the MAS with a coordination infrastructure, providing abstractions — called coordination artifacts — perceived by the agents as run-time entities living in the environment. Coordination artifacts are designed with the goal of automating a specific coordination task, provided to the agents as a service, and featuring peculiar engineering properties such as encapsulation, predictability, inspectability and malleability [10], [15].

An example technology supporting this scenario is TuCSoN [11], [14]. In TuCSoN, the nodes of the network can be populated by *tuple centres* playing the role of coordination artifacts. Tuple centres are LINDA-like blackboards, whose reactive behaviour can be programmed using the logic-based language ReSpecT, so as to make the tuple centres encapsulating any coordination task, from simple synchronization policies up to complex workflows. In particular, ReSpecT is shown to be Turing-complete, thus allowing any coordination algorithm to be specified.

However, in most application scenarios characterised by a high degree of opennes and dynamism, coordination tasks need to be time-dependent. On the one hand, it is very useful to specify (and then enforce) given levels of liveness and of quality of service — e.g. requiring agents to interact with the coordination artifact at a minimum/maximum frequency. On the other hand, temporal properties are also fundamental aspects concerning interception of violations in the agent-artifact contract: an agent might be required to provide a service within a given deadline, or might require the artifact to do the same. As shown in [15], it is sensible e.g. to let coordination artifacts provide agents with operating instructions featuring timed properties, which can be correctly enforced only through timed coordination tasks.

The need for specifying timed coordination policies emerged in a parallel way in the field of distributed systems as well. For instance, in JavaSpaces [4] primitives `read` and `take` — looking for a tuple analogously to `rd` and `in` in LINDA — comes with a timeout value: when the timeout expires the request immediately returns a failure. Similarly, tuples can provide a *lease* time when inserted in the space: when the lease expires the tuple is automatically removed. All these primitives, and others based on time, can actually be the basis for structuring more complex coordination scenarios, such as e.g. auctions and negotiations protocols including time-based guarantees and constraints.

In this work we discuss how the basic ReSpecT tuple centre model has been extended to support the definition and enaction of time-aware coordination policies. The basic idea is to exploit the programmability of the coordination medium extended with a temporal framework to get the capability of modelling *any* time-based coordination patterns, realised directly by specifying a suitable behaviour of the artifact.

The rest of the paper is organised as follows: Section II discusses in details the ReSpecT extended model, Section III provides some concrete examples exploiting the extended model, Section IV provides some reflections on the features of the approach and finally Section V provides related works, conclusion and future works.

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

σ	::=	{reaction(*p*(t),(*body*)).}	Specification
p	::=	*cp* | *rp*	ReSpecT primitives
cp	::=	out | in | rd	Comm. primitives
rp	::=	in_r | rd_r | out_r | no_r	Reaction primitives
body	::=	[*goal*{,*goal*}]	Specification body
ph	::=	pre | post	Direction predicates
goal	::=	*ph* | *rp*(t)	Goals

Fig. 1. The syntax of a ReSpecT specification

II. Extending ReSpecT with Time

We describe here informal semantics of a significant fragment of the ReSpecT language: the reader interested in a formal presentation should refer to [9], [8]. Then, we describe how this model can be extended so as to deal with timing aspects, that is, with the ability to trigger trap events at a specified time (in the future).

A. The Basic Model

ReSpecT [8] is a logic-based language to program the reactive behaviour of tuple centres [9].

Tuple centres are *coordination media* extending the basic model of LINDA tuple spaces [5]. Similarly to LINDA, they accept and serve requests for inserting a tuple t (by primitive out(t)), removing a tuple matching template tt (by primitive in(tt)), and reading a tuple matching template tt (by primitive rd(tt))[1]. With respect to LINDA, ReSpecT tuple centres specialise the tuple space model with logic tuples (Prolog-like terms with variables) and unification as the matching criterion; differently from LINDA tuple spaces, tuple centres can be programmed so that whenever an external communication event occurs a computation reactively starts which may affect the state of the inner tuple space. External communication events can either be *(i)* a *listening*, reception of a request from a coordinated process (either a in, rd, out), or *(ii)* a *speaking*, the production of a reply towards a coordinated process (either the reply to a in or rd)[2].

The ReSpecT language can be used to declare a set σ of *reaction specification tuples* (RSTs), using the syntax of Figure 1.

Each RST has a head and a body. When a communication event *p*(t) occurs, all the RSTs with a matching head are activated, that is, their bodies — each specifying an atomic computation over the tuple centre — are used to spawn a pending reaction waiting to be executed. Being specified by a body, reactions are composed by a sequence of reaction primitives *rp* resembling LINDA primitives, which are used to remove a tuple (in_r), read a tuple (rd_r), insert a tuple (out_r), and check for the absence of a tuple (no_r). This sequence can contain a direction predicate *ph*, pre or post, which is used to filter between reactions to a listening or a speaking. In particular, we here consider therefore five kinds of external communication events: listening of a out, rd, or in, and speaking of a in and rd.

Reactions are non-deterministically picked and executed, by atomically executing all its reaction primitives. Their effect is to change the state of the tuple centre, and to fire new reactions, as long as they match some other RST — whose head can specify a reaction primitive (internal communication events) other than a communication primitive (external communication events). This recursive creation of reactions is the mechanism by which ReSpecT achieves expressiveness up to reaching Turing-completeness [3].

Primitives in_r, rd_r, and no_r might fail (the former two when the tuple is absent, the latter when it is present), in which case the reaction execution fails, and its effect on the tuple centre is rolled back. The computation fired by the external communication event stops when (if) no more pending reactions occur: when this happens the tuple centre waits until the next communication event occurs.

B. The Extended Model

First of all, the model is extended with a notion of current time of the tuple centre *Tc*: each tuple centre has its own clock, which defines the passing of time [3]. Actually, tuple centre time is a physical time, but it is value considered to be constant during the execution of an individual reaction: in other words, we assume that *Tc* refers to the time when the reaction started executing. This choice is coherent with ReSpecT philosophy concerning reactions, which are meant to be executed atomically (in the case of successful reactions).

In order to get *Tc* in ReSpecT programs a new primitive is introduced:

current_time(*?Tc*)[4]

This primitive (predicate) is successful if *Tc* (typically a variable) unifies with the current tuple centre time *Tc*. As an example, the reaction specification tuple

```
reaction(in(p(X)),(
    current_time(Tc),
    out_r(request_log(Tc,p(X)))
)).
```

inserts a new tuple with timing information each time a request to retrieve a tuple p(X) is executed, realising a temporal log of the requests.

The model is then extended with the notion of *trap event* or simply *trap*, which is an event generated when the tuple centre reaches a specific time point. A trap occurs because

[1] Tuple centres can also deal with usual predicative primitives inp(tt) and rdp(tt) of LINDA, but these are not considered here for the sake of simplicity and without loss of generality.

[2] we use here the term *listening* related to events following the basic terminology adopted in [9]

[3] In current implementation the temporal unity is the millisecond

[4] A Prolog notation is adopted for describing the modality of arguments: + is used for specifying input argument, - output argument, ? input/output argument, @ input argument which must be fully instantiated

of a *(trap) source*, characterised by a unique identifier *ID*, a time *Te* and a description tuple *Td*. The language is extended with the possibility to generate and manipulate trap events and sources. In particular we introduce the two following features:

- internally in the tuple centre, a coordination law (i.e. one or more reaction specification tuples) might install a trap source, which causes a trap to occur at a specific time. For instance, we may want to generate a trap described by the tuple `expired(T)` a certain interval *LeaseTime* after the insertion of a tuple `leased(T)`;
- the tuple centre reacts to a trap event analogously to communication events, by means of proper reaction specification tuples. In the case above, we may want the tuple *T* to be removed when the trap described by `expired(T)` occurs.

In order to support trap generator installation, the language is extended with two new primitives:

```
new_trap(-ID,@Te,+Td)
kill_trap(@ID)
```

The first is successful if `Te` is an integer equal or greater than zero. Its effect is to install a new trap source — with `ID` as identifier — which enters a queue of installed sources. When tuple centre time *Tc* time will be equal or greater than current time plus `Te`, a trap event described by the tuple `Td` will be then generated and inserted into the queue of triggered trap events, whereas its source is deinstalled — i.e. removed from its queue. Notice that because of the success/failure semantics of ReSpecT semantics, if the reaction including an invocation to primitive `new_trap` fails, no trap source is installed, actually. An example involving the `new_trap` primitive is as follows:

```
reaction(out(leased(T,LeaseTime)),(
    new_trap(_,LeaseTime,expired(T))
)).
```

The reaction is triggered when a tuple matching `leased(T,LeaseTime)` is inserted, and it installs a new trap source which will generate a trap described by the tuple `expired(T)` after `LeaseTime` units from then. Primitive `kill_trap` is instead used to deinstall a source given its identifier: such a primitive fails if not installed sources has is characterised by the identifier provided.

Then, the language has been extended with the possibility to write reactions triggered by the occurrence of trap events. The syntactical and semantic models of trap reactions are analogous to the reactions to communication events:

```
reaction( trap(Tuple), Body)
```

`Body` specifies the set of actions to be executed when a trap with a description tuple matching the template `Tuple` occurs. In the following simple example

```
reaction(trap(expired(T)),( in_r(T) )).
```

when a trap described by a tuple matching the template `expired(T)` occurs, the tuple specified in `T` is removed from the tuple set. Notice that if the tuple is not present the `in_r` fails causing the whole reaction to fail — as the trap event is occurred, however, the trap source is erased.

Trap events are listened one by one as soon as the tuple centre is not executing a reaction; that is — according to the tuple centre semantics [9], [8] — when it is in the idle state, or between a listening and a speaking stage, or during a reacting stage (between the execution of two reactions). When a trap event is listened, it is first removed from the trap event queue, the set of the reactions it triggers is determined — by matching the reaction head with the trap description tuple — and then executing sequentially all such reactions. As for the ReSpecT reacting stage, the order of execution of the reactions is not deterministic.

An important semantic aspect of this extension concerns the priority of reactions fired by external communication events (standard execution) with respect to those of trap events (trap execution). The model and implementation described here feature higher priority of reactions fired by trap events. This means that if during the standard executions of a reaction chain a trap event occurs, the chain is broken, and the reactions fired by the trap are executed. It's worth noting that the individual reactions are still atomic, not interruptible as in the basic ReSpecT model: traps event in the trap queue are listened (and related reactions executed) after the completion of any reaction eventually in execution. Then, chains of reactions can be broken, not individual reactions. This is fundamental in order to preserve the semantic properties of ReSpecT model [8]. Also reactions triggered by a trap event are atomic, and they cannot be interrupted or suspended: in other words, trap handlers are not interruptible and cannot be nested.

As will be discussed in Section IV, the possibily of breaking reaction chains is important to build robust coordinating behaviour, in particular with respect to possible bugs generating terminating reaction chains.

Nevertheless, it is worth mentioning here that other semantics are possible and interesting. By giving higher priority to the standard execution, one ensures that traps never interfere with it. In exchange of the better isolation of code achieved, in this case one can no longer guarantee the same timing constraints: trap executions must wait for the standard execution to complete. Notice that such aspects are mostly orthogonal to the actual applicability of temporal coordination laws as shown e.g. in next section. Moreover, a straightforward generalisation of our model can be realised by specifying the priority level of a trap (higher, lower, or equal to the that of external communication events) at the time its source is installed[5].

III. Examples

In this section we describe some simple examples of how temporal coordination primitives and coordination laws can be modelled on top of extended ReSpecT. It's worth noting that these examples – even if simple – appear in several research work in literature as a core of timing features extending the

[5]This interesting feature which is subject of current research is not described in this paper for brevity.

```
1  reaction( in(timed(Time,Tuple,Res)), (
     pre, in_r(Tuple),
     out_r(timed(Time,Tuple,yes)))).

2  reaction( in(timed(Time,Tuple,Res)), (
     pre,no_r(Tuple),
     new_trap(ID,Time,expired_in(Time,Tuple)),
     out_r(trap_info(ID,Time,Tuple)) )).

3  reaction( trap(expired_in(Time,Tuple)),(
     in_r(trap_info(ID,Time,Tuple)),
     out_r(timed(Time,Tuple,no)) )).

4  reaction( out(Tuple),(
     in_r(trap_info(ID,Time,Tuple)),
     kill_trap(ID),
     out_r(timed(Time,Tuple,yes)) )).
```

TABLE I

ReSpecT SPECIFICATION FOR MODELLING A TIMED IN PRIMITIVE

```
1  reaction( out(leased(Time,Tuple)), (
     new_trap(ID,Time,lease_expired(Time,Tuple)),
     in_r(leased(Time,Tuple)),
     out_r(outl(ID,Time,Tuple)) )).

2  reaction( rd(Tuple),( pre,
     rd_r(outl(ID,_,Tuple)),
     out_r(Tuple) )).

3  reaction( rd(Tuple),(post,
     rd_r(outl(ID,_,Tuple)),
     in_r(Tuple) )).

4  reaction( in(Tuple),( pre,
     in_r(outl(ID,_,Tuple)),
     out_r(Tuple),
     kill_trap(ID) )).

5  reaction( trap(lease_expired(Time,Tuple)), (
     in_r(outl(ID,Time,Tuple)))).
```

TABLE II

ReSpecT SPECIFICATION FOR MODELLING TUPLES WITH A LEASE TIME

basic model; typically, in the literature there is a specific extension for each timing feature described here: on the contrary, we remark the generality of our approach, which is meant to support these and several other time-based coordination patterns on top of the same model.

A. Timed Requests

In this first example we model a timed `in` primitive, i.e. an `in` request that keeps blocked only for a maximum amount of time. An agent issues a timed `in` by executing primitive `in(timed(@Time,?Template,-Res)`. If a tuple matching *`Template`* is inserted within *`Time`* units of time, the requested tuple is removed and taken by the agent as usual with *`Res`* being bound to the `yes` atom. Conversely, if no matching tuples are inserted within the specified time, *`Res`* is bound to `no` atom. Table I reports the ReSpecT specification which makes it possible to realise the behaviour of this new primitive. When the `in` request is issued, if a tuple matching the template is present a proper tuple satisfying the request is created (reaction 1). Instead, if no tuple is found, a trap source is installed for generating a trap at the due time (reaction 2). Also, a tuple `trap_info` is inserted in the tuple set, reifying information about the installed trap source, required for its possible removal. If a tuple matching a template of a pending timed `in` is inserted on time, the related trap source is removed and a proper tuple matching the timed `in` request is inserted (reaction 4). Finally, if the trap occurs — meaning that no tuples have been inserted on time matching a pending timed `in` — then a tuple matching the timed `in` request carrying negative result is inserted in the tuple set (reaction 3).

B. Tuples in Leasing

In this example we model the notion of *lease*, analogously to the lease notion in models such as JavaSpaces [4] and TSpaces [16]. Tuples can be inserted in the tuple set specifying a lease time, i.e. the maximum amount of time for which they can reside in the tuple centre before automatic removal.

An agent insert a tuple with a lease time by issuing an `out(leased(@Time,@Tuple))`. Table II shows the ReSpecT specification programming the tuple centre with the desired leasing behaviour . When a tuple with a lease time is inserted in the tuple centre, a trap source is installed for generating a trap when the tuple centre time reaches the lease due time (reaction 1). Also a tuple `outl` is inserted in the tuple set with the information on the trap source and the leased tuple (note that the flat tuple with the lease time is not directly present in the set). Then, for each `rd` issued with a template matching a leased tuple, a flat tuple satisfying the request is first inserted in the tuple set (reaction 2), and then removed after the `rd` has been satisfied (reaction 3). An `in` request instead causes directly the removal of the lease tuple and of the trap source (reaction 4). Finally, if a trap event occurs (meaning that the lease time of a tuple expired), the `outl` tuple carrying information about the presence of the leased tuple is removed (reaction 5).

C. Dining Philosophers with Maximum Eating Time

The *dining philosopher* is a classical problem used for evaluating the expressiveness of coordination languages in the context of concurrent systems. In spite of its formulation, it is generally used as an archetype for non-trivial resource access policies. The solution of the problem in ReSpecT consists in using a tuple centre for encapsulating the coordination policy required to decouple agent requests from single requests of resources — specifically, to encapsulate the management of *chopsticks* (for details refer to [9]).

Each philosopher agent *(i)* gets the two needed chopsticks by retrieving a tuple `chops(C1,C2)`, *(ii)* eats for a certain amount of time, *(iii)* then provides back the chopsticks by inserting the tuple `chops(C1,C2)` in the tuple centre, and *(iv)* finally starts thinking until next dining cycle.

A pseudo-code reflecting this interactive behaviour is the following:

```
1  reaction(in(all_timed(Time,Tuple,OutList)),(
     new_trap(ID,Time,inat(Time,Tuple,OutList)),
     out_r(current_in_all(ID,Time,Tuple,[])),
     out_r(remove_in_all(ID)))).
2  reaction( out_r(remove_in_all(ID)),(
     in_r(remove_in_all(ID)),
     rd_r(current_in_all(ID,Time,Tuple,L)),
     in_r(Tuple),
     in_r(current_in_all(ID,Time,Tuple2,L)),
     out_r(current_in_all(ID,Time,Tuple2,[Tuple|L])),
     out_r(remove_in_all(ID)))).
3  reaction( out_r(remove_in_all(ID)),(
     in_r(remove_in_all(ID)),
     rd_r(current_in_all(ID,_,Tuple,_)),
     no_r(Tuple))).
4  reaction( out(Tuple),(
     in_r(current_in_all(ID,_,Tuple,L)),
     in_r(Tuple),
     out_r(current_in_all(ID,_,Tuple,[Tuple|L])))).
5  reaction( trap(inat(Time,Tuple,OutList)), (
     in_r(current_in_all(ID,Time,Tuple,L)),
     out_r(all_timed(Time,Tuple,L)))).
```

TABLE III

ReSpecT SPECIFICATION MIMICKING AN INALL WITH A DURATION TIME

```
while (true){
    think();
    in(chops(C1,C2));
    eat();
    out(chops(C1,C2));
}
```

The coordination specification in ReSpecT (first 6 reactions of Table IV, bottom) mediates the representation of the resources (`chops` vs. `chop` tuples), and most importantly avoid deadlocks among the agents.

Here we extend the basic problem by adding a further constraint: the maximum time which philosophers can take to eat (i.e. to use the resources) is given, stored in a tuple `max_eating_time(`*`MaxEatingTime`*`)` in the tuple centre. To keep the example simple, if this time is exceeded, the chopsticks are regenerated in the tuple centre, avoiding the starvation of the philosophers waiting for them, and the chopsticks eventually inserted out of time are removed.

The solution to this problem using the extended ReSpecT model accounts for adding only the ReSpecT specification (the agent code and related protocols are untouched) with the reactions 7–10 described in Table IV (bottom), and extending reaction 4 with the part in italics. Essentially, the new reactions install a new trap source as soon as a philosopher retrieves his chopsticks (reaction 7). If the philosopher provides the chopsticks back in time (before the occurrence of the trap), then the trap source is removed (reaction 8). Otherwise, if the trap event occurs, the triggered trap reaction recreates the missing chopsticks tuples in the tuple centre and inserts a tuple `invalid_chops` which prevent chopsticks insertion out of fime (reaction 9). This prevention is realised by checking the existence of the tuple `invalid_chops` when the tuple `chops` are released by a philosopher (reaction 10).

It is worth noting that keeping track of the maximum eating

```
% a request of the chopsticks is reified with a
% required tuple
1  reaction(in(chops(C1,C2)),(pre,out_r(required(C1,C2)))).

% if both the chopsticks are available, a chops
% tuple is generated
2  reaction(out_r(required(C1,C2)),(
     in_r(chop(C1)),in_r(chop(C2)),out_r(chops(C1,C2)))).

% with the retrieval of the chops tuple,
% the chopsticks request is removed
3  reaction(in(chops(C1,C2)), (post,in_r(required(C1,C2)))).

% the release of a chops tuple still valid (on time)
% causes the insertion of individual chopsticks,
% represented by the two chop tuples
4  reaction(out(chops(C1,C2)), (
     current_agent(AgentId),
     no_r(invalid_chops(AgentId,C1,C2)),
     in_r(chops(C1,C2)),out_r(chop(C1)),out_r(chop(C2)))).

% a chops tuple is generated if there is
% a pending request, and both chop tuples
% are actually available
5  reaction(out_r(chop(C1)), (rd_r(required(C1,C)),
     in_r(chop(C1)),in_r(chop(C)),out_r(chops(C1,C)))).
6  reaction(out_r(chop(C2)), (rd_r(required(C,C2)),
     in_r(chop(C)),in_r(chop(C2)),out_r(chops(C,C2)))).


% a chopsticks request causes also creating a
% new trap generator, keeping track of its information
% in the chops_pending_trap tuple
7  reaction(in(chops(C1,C2)),( pre,
     rd_r(max_eating_time(Tmax)),
     new_trap(ID,Tmax, expired(C1,C2)),
     current_agent(AgentId),
     out_r(chops_pending_trap(ID,AgentId,C1,C2)))).

% when chops are released on time, the trap
% generator is removed
8  reaction(out(chops(C1,C2)),(
     in_r(chops_pending_trap(ID,C1,C2)),
     kill_trap(ID))).

% trap generation causes the insertion back
% of the missing tuples and the insertion of tuple
% keeping track of the invalid chops
9  reaction(trap(expired(C1,C2)),(
     no_r(chop(C1)), no_r(chop(C2)),
     current_agent(AgentId),
     in_r(chops_pending_trap(ID,AgentId,C1,C2)),
     out_r(invalid_chops(AgentId,C1,C2)),
     out_r(chop(C1)), out_r(chop(C2)))).

% chopsticks released that are invalid (due to
% time expiration) are immediately removed
10 reaction(out(chops(C1,C2)), (
     current_agent(AgentId),
     in_r(invalid_chops(AgentId,C1,C2)),
     in_r(chops(C1,C2)))).
```

TABLE IV

ReSpecT SPECIFICATION FOR COORDINATING DINING PHILOSOPHERS WITH A MAXIMUM EATING TIME

time as a tuple (`max_eating_time` in the example) makes it possible to easily change it dynamically, while the activity is running; this can be very useful for instance in scenarios where this time need to be adapted (at runtime) according to the workload and, more generally, environmental factors affecting the system.

Finally, it's worth remarking that the approach is not meant to alter the autonomy of the agent, for instance by means of some form of preemption in the case of timing violations; on the contrary – as a coordination model – all the constraints and (timed based) rule enforcing concerns the interaction space.

D. An Artifact for Timed Contract Net Protocols

As a final example, we describe a coordination artifact modelling and embodying the coordinating behaviour of a time-aware Contract Net Protocol (CNP). CNP is a well-known protocol in MAS, used as basic building block for bulding more articulated protocols and coordination strategies [13]. Following [6], we consider the CNP in a task allocation scenario: a master announces a task (service) to be executed, potential workers interested provide their bids, the announcer collects the bid and selects one; after confirming his bid, the awarded bidder becomes the contractor, taking in charge of the execution of the task and finally providing task results.

We extend the basic version with some timing constraints. In particular we suppose that: *(i)* the bidding stage has a duration, established at a "contract" level; *(ii)* there is a maximum time for the announcer for communicating the awarded bidder; *(iii)* there is a maximum time for the awarded bidder for confirming the bid and becoming the contractor; *(iv)* there is a maximum time for the contractor for executing the task.

According to our approach, a coordination artifact can be used to embody the coordinating behaviour of the time-aware CNP, fully encapsulating the social/contractual rules defining protocols steps and governing participant interaction, including temporal constraints. The coordination artifact is realised as a tuple centre – called `tasks` –, programmed with the ReSpecT specification reported in Table VI. Table V shows the pseudo-code representing the interactive behaviour of the master (top) and workers (bottom).

The usage protocol of the artifact for the master consists in: making the announcement (by inserting a tuple `announcement`), collecting the bids (by retrieving the tuple `bids`), selecting and informing the awarded bidder (by inserting the tuple `awarded_bid`) and, finally, collecting the result (by retrieving the tuple `task_done`); for the workers, the usage protocol accounts for reading the announcement (by reading the tuple `announcement`), evaluating the proposal and providing a bid (by inserting a tuple `bid`), reading the master decision (by retrieving the tuple `bid_result`), and – in the case of awarding – confirming the bid (by inserting the tuple `confirming_bid`), performing the task and, finally, providing the results (by insering the tuple `task_result`).

The artifact behaviour in ReSpecT described in Table VI reflects the various stages of the CNP protocol, and traps are used for modelling the timing constraints related to the various stages: from bidding, to awarding, confirming, and task execution A brief description of the artifact behaviour follows: when a new announcement is done (reaction 1), the information about the new CNP are created (tuple `task_todo` and `cnp_state`) and a new trap source is installed, generating a trap when the bidding time is expired. At the trap generation (reaction 2) – meaning that the bidding stage is closed – all the bids inserted are collected (reaction 3), the information concerning the protocol state updated, and a new trap source is installed, generating a trap when the awarding time is expired. If the master provides information about the awarded bidder before this trap generation, the trap source is killed, the tuples concerning awarded and non-awarded bidders are generated (reactions 5, 8, 9), and a new trap source for managing confirmation expire is installed (reaction 5). If no awarded bidder is provided on time or a wrong (unknown) awared is communicated, the tuple reporting the CNP state is updated accordingly, reporting the error (reactions 4, 6, 7). If the awarded bidder confirms on time his bid (reaction 10), the execution stage is entered, by updating the CNP state properly and installing a new trap generator for keeping track of task execution time expiration. Otherwise, if the confirm is not provided on time, the related trap event is generated and listened (reaction 11), aborting the activity and updating accordingly the CNP state tuple. Finally, if the contractor provides the task result on time (reaction 13), the trap generator for task execution is killed, the tuples concening the terminating CNP are removed and the result information are prepared for being retrieved by the master. Otherwise, if the contractor does not provide information on time, the trap is generated and the artifact state is updated accordingly, reporting the error (reaction 12).

```
tasks ? out(announcement(task(TaskId,TaskInfo,MaxExecTime)))
tasks ? in(bids(TaskId,BidList))
Bid ← selectWinner(BidList)
tasks ? out(awarded_bid(TaskId,AgentId))
tasks ? in(task_done(TaskId,Result,Duration))
```

```
tasks ? rd(announcement(task(TaskId,TaskInfo,MaxExecTime)))
MyBid ← evaluate(TaskInfo)
tasks ? out(bid(TaskId,MyId,MyBid))
tasks ? in(bid_result(TaskId,MyId,Answer))
if (Answer=='awarded') {
    tasks ? out(confirm_bid(MyId))
    Result ← perform(TaskInfo)
    tasks ? out(task_result(TaskId,MyId,Result))
}
```

TABLE V
SKETCH OF THE BEHAVIOUR OF THE AGENTS PARTICIPATING TO THE TIMED CONTRACT NET PROTOCOL: MASTERS *(Top)* AND WORKERS *(Bottom)*

IV. DISCUSSION

The approach aims to be general and expressive enough to allow the description of a large range of coordination patterns based on the notion of time. An alternative way to solve

```
% When an announcement is made, a trap generator is
% installed for generating a timeout for bidding time
1  reaction(out(announcement(task(Id,Info,MaxTime))),(
     out_r(task_todo(Id,Info,MaxTime)),
     out_r(cnp_state(collecting_bids(Id))),
     rd_r(bidding_time(Time)),
     new_trap(_,Time,bidding_expired(Id)))).

% When the bidding time has expired, the master can
% collect the bids for choosing the winner. A trap
% generator is installed for defining the maximum
% awarding time
2  reaction(trap(bidding_expired(TaskId)),(
      in_r(announcement(_)),
      in_r(cnp_state(collecting_bids(TaskId))),
      out_r(collected_bids(TaskId,[])),
      out_r(cnp_state(awarding(TaskId))),
      rd_r(awarding_time(Time)),
      new_trap(_,Time,awarding_expired(TaskId)))).
3  reaction(out_r(collected_bids(TaskId,L)),(
      in_r(bid(TaskId,AgentId,Bid)),
      out_r(bid_evaluated(TaskId,AgentId,Bid)),
      in_r(collected_bids(TaskId,L)),
      out_r(collected_bids(TaskId,
                [bid(AgentId,Bid)|L])) )).

% When the awarding time has expired, the bidders are
% informed of the results. If no winner has been
% selected the protocol enters in an error state,
% otherwise the protocol enters in the confirming
% stage, setting up a maximum time for it
4  reaction(trap(awarding_expired(TaskId)),(
      in_r(cnp_state(awarding(TaskId))),
      out_r(check_awarded(TaskId)))).
5  reaction(out_r(check_awarded(TaskId)),(
      in_r(check_awarded(TaskId)),
      rd_r(awarded_bid(TaskId,AgentId)),
      in_r(bid_evaluated(TaskId,AgentId,Bid)),
      out_r(result(TaskId,AgentId,awarded)),
      out_r(cnp_state(confirming_bid(TaskId,AgentId))),
      rd_r(confirming_time(Time)),
      new_trap(ID,Time,confirm_expired(TaskId)),
      out_r(confirm_timer(TaskId,ID)),
      out_r(refuse_others(TaskId)))).
6  reaction(out_r(check_awarded(TaskId)),(
      in_r(check_awarded(TaskId)),
      rd_r(awarded_bid(TaskId,AgentId)),
      no_r(bid_evaluated(AgentId,Bid)),
      out_r(cnp_state(aborted(TaskId,wrong_awarded))))).
7  reaction(out_r(check_awarded(TaskId)),(
      in_r(check_awarded(TaskId)),
      no_r(awarded_bid(TaskId,AgentId)),
      out_r(cnp_state(aborted(TaskId,award_expired))))).
8  reaction(out_r(refuse_others(TaskId)),(
      in_r(bid_evaluated(TaskId,AgentId,Bid)),
      out_r(result(TaskId,AgentId,'not-awarded')),
      out_r(refuse_others(TaskId)))).
9  reaction(out_r(refuse_others(TaskId)),(
      in_r(refuse_others(TaskId)) )).

% At the arrival of the confirm from the awarded
% bidder, a timeout trap is setup for checking the
% execution time of the task
10  reaction(out(confirm_bid(TaskId,AgentId)),(
      in_r(confirm_bid(TaskId,AgentId)),
      in_r(cnp_state(confirming_bid(TaskId,AgentId))),
      current_time(StartTime),
      out_r(cnp_state(executing_task(TaskId,StartTime))),
      in_r(confirm_timer(TaskId,IdT)),
      kill_trap(IdT),
      rd_r(task_todo(TaskId,_,MaxTime)),
      new_trap(IdT2, MaxTime, execution_expired),
      out_r(execution_timer(TaskId,IdT2)))).

% The occurrence of the confirm expired trap means
% that the confirm from the awarded bidder has not
% arrived on time, causing the protocol to be aborted
11  reaction(trap(confirm_expired(TaskId)),(
      in_r(cnp_state(confirming_bid(TaskId,AgentId))),
      in_r(confirm_timer(TaskId,_)),
      rd_r(awarded_bid(TaskId,AgentId)),
      out_r(cnp_state(aborted(TaskId,
              confirm_expired(AgentId)))))).

% The occurrence of the execution expired trap means
% that the awarded bidder has not completed the
% task on time, causing the protocol to be aborted
12  reaction(trap(execution_expired(TaskId)),(
      in_r(cnp_state(executing_task(TaskId,StartTime))),
      in_r(execution_timer(TaskId,_)),
      rd_r(awarded_bid(TaskId,AgentId)),
      current_time(Now),
      Duration is Now - StartTime,
      out_r(cnp_state(aborted(TaskId,
              execution_expired(AgentId,Duration)))))).

% The awarded bidder provided task result on time
% terminating correctly the protocol
13  reaction(out(task_result(TaskId,AgentId,Result)),(
      in_r(task_result(TaskId,AgentId,Result)),
      in_r(awarded_bid(TaskId,AgentId)),
      in_r(execution_timer(TaskId,Id)),
      kill_trap(Id),
      in_r(cnp_state(executing_task(TaskId,StartTime))),
      in_r(task_todo(TaskId,Info,MaxTime)),
      current_time(Now),
      Duration is Now - StartTime,
      out_r(task_done(TaskId,Result,Duration)) )).
```

TABLE VI

BEHAVIOUR OF THE ARTIFACT REALISING A TIMED CNP, ENCODED IN THE ReSpecT LANGUAGE

the problem consists in adopting helper agents (sort of *Timer* agents) with the specific goal of generating traps by inserting specific tuples in the tuple centre a certain time points. With respect to this approach and also to other approaches, the solution described in this work has several advantages:

- *Incapsulation of coordination* — Managing traps directly inside the coordination medium makes it possible to fully keep coordination encapsulated, embedding its full specification and enactment in a ReSpecT program and tuple centre behaviour. Conversely, using helper agents to realise part of the coordination policies which cannot be expressed directly in the medium causes a violation of encapsulation. Among the problems that arise, we have: less degree of control, more problematic reusability and extensibility, more complex formalisation.
- *Timed-coordination* — The approach is not meant to provide strict guarantees as required for real time systems: actually, this would be difficult to achieve given also the complexity of ReSpecT behaviours, based on first order logic. However, the model is expressive and effective enough to be useful for several kind of timed systems in general. Also, the management of time events directly inside the medium makes it possible to have some guarantees on the timings related to trap generation and

trap reaction execution. These guarantees would not be possible in general adopting external agents simulating traps by inserting tuples at (their) specific time. The reacting stage of a tuple centre has always priority with respect to listening of communication events generated by external agents; this means that in the case of complex and articulated reaction chains, the listening of a trap event (i.e. reacting to tuples inserted by timer agents) could be substantially delayed, and possibly could not happen. On the contrary, this cannot happen in the extended model, where a trap event is ensured to be listened and the related reactions to be executed — with higher priority.

- *Well-founded semantics* — The extension realised to the basic model allows for a well-defined operational semantics extending the basic semantics of tuple centres and ReSpecT with few constructs and behaviours. In particular, the basic properties of ReSpecT – in particular atomic reaction execution – are all preserved. This semantics has been fundamental for driving the implementation of the model and will be important also for the development of verification tools.
- *Compatibility, reuse and minimality* — The extension does not alter the basic set of (Linda) coordination primitives, and then it does not require learning and adopting new interfaces for agents aiming to exploit it: all the new features are at the level of the coordination medium programming. This in particular implies that the new model can be introduced in existing systems, exploiting the new temporal features without the need to change existing agents.
- *"The hills are alive"* — Coordination artifacts with temporal capabilities can be suitably exploited to model and engineer *living environments*, i.e. environments which spontaneously change with some kind of transformations, due to the passage of time. A well known example is given by environments in the context of stigmergy coordination approaches with multi-agent systems [12]; in this context, the pheromones (part of the agents – ants – environment) evaporate with the passing of time according to some laws which heavily condition the emerging coordination patterns. Tuple centres can be exploited then to model and enact the living environment: tuples can represent pheromones (placed to and perceived from the environment by mean of the basic coordination primitives), and tuple centre behaviour can embed the rules describing how to transform pheromones with the passage of time.

Concerning the implementation of the model, the tuple centre centralisation vs. distribution issue arises. The basic tuple centre model is not necessarily centralised: however, the extension provided in this work — devising out a notion of time for each medium — leads quite inevitably to realise tuple centres with a specific spatial location. This is what already happens in TuCSoN coordination infrastructure, where there can be multiple tuple centres distributed over the network, collected and localised in infrastructure nodes. It is worth mentioning that this problem is not caused by our framework, but is inherent on any approach aiming at adding temporal aspects to a coordination model.

However, according to our experience in agent based distributed system design and development, the need to have a distributed implementation of individual coordination media is a real issue only for very specific application domains. For the most part of applications, the bottleneck and single point of failure arguments against the use of centralised coordination media can be answered by a suitable design of the multi-agent system and an effective use of the coordination infrastructure. At this level, it is fundamental that a software engineer would know the scale of the coordination artifacts he is going to use, and the quality of service (robustness in particular) provided by the infrastructure.

V. Related Works and Conclusion

The contribution provided by this work can be generalised from tuple centre to — more generally — the design and development of general purpose time-aware coordination artifacts in multi-agent systems [10].

Outside the specific context of coordination models and languages, the issue of defining suitable languages for specifying the communication and coordination in (soft) real time systems have been studied for long time. Examples of such languages are Esterel [1] and Lustre [2], both modelling synchronous systems, the former with an imperative style, and the latter based on dataflow. In coordination literature several approaches have been proposed for extending basic coordination languages with timing capabilities. [7] introduces two notions of time for Linda-style coordination models, relative time and absolute time, providing different kind of features. Time-outs have been introduced in JavaSpaces [4] and in TSpaces [16].

The approach described in this work is quite different from these approaches, since it extends the basic model without altering the basic Linda model from the point of view of the primitives, but acting directly on the expressiveness of the coordination media. Also, it does not provide specific time capabilities, but — following the programmable coordination media philosophy — aims at instrumenting the model with the expressiveness useful for specifying any time-based coordination pattern.

The model has been implemented in the version 1.4.0 of TuCSoN coordination infrastructure, which is available for downloading at TuCSoN web site [14]. Ongoing work is concerned with defining a formal operational semantics of the extended model, consistent and compatible with the basic one defined for ReSpecT [9], [8]. The formal semantics is important in particular to frame the expressiveness of the model compared to existing models in literature concerned with timed systems, and to explore the possibility of building theories and tools for the verification of formal properties.

Future work will stress the approach with the engineering of real world application domain involving time in the coordination activities.

REFERENCES

[1] G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[2] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: a declarative language for real-time programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 178–188. ACM Press, 1987.

[3] E. Denti, A. Natali, and A. Omicini. On the expressive power of a language for programming coordination media. In *Proc. of the 1998 ACM Symposium on Applied Computing (SAC'98)*, pages 169–177. ACM, February 27 - March 1 1998. Track on Coordination Models, Languages and Applications.

[4] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces: Principles, Patterns, and Practice*. The Jini Technology Series. Addison-Wesley, 1999.

[5] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[6] M. Huhns and L. M. Stephens. Multiagent systems and societies of agents. In G. Weiss, editor, *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*, pages 79–118. MIT Press, 1999.

[7] J.-M. Jacquet, K. De Bosschere, and A. Brogi. On timed coordination languages. In D. Garlan and D. Le Métayer, editors, *Proceedings of the 4th International Conference on Coordination Languages and Models*, volume 1906 of *LNCS*, pages 81–98, Berlin (D), 2000. Springer-Verlag.

[8] A. Omicini and E. Denti. Formal ReSpecT. In A. Dovier, M. C. Meo, and A. Omicini, editors, *Declarative Programming – Selected Papers from AGP'00*, volume 48 of *Electronic Notes in Theoretical Computer Science*, pages 179–196. Elsevier Science B. V., 2001.

[9] A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, Nov. 2001.

[10] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 286–293, New York, USA, 19–23 July 2004. ACM.

[11] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999. Special Issue: Coordination Mechanisms for Web Agents.

[12] V. D. Parunak. 'Go To The Ant': Engineering principles from natural agent systems. *Annals of Operations Research*, 75:69–101, 1997.

[13] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *Proceedings of the 1st International Conference on Distributed Computing Systems*, pages 186–192, Washington D.C., 1979. IEEE Computer Society.

[14] TuCSoN home page. `http://lia.deis.unibo.it/research/TuCSoN/`.

[15] M. Viroli and A. Ricci. Instructions-based semantics of agent mediated interaction. In N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 102–110, New York, USA, 19–23 July 2004. ACM.

[16] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T Spaces. *IBM Journal of Research and Development*, 37(3 - Java Techonology):454–474, 1998.

Commutation as an Emergent Phenomenon of Residential and Industrial Location Decisions: from a Microeconomic to a MMASS-based Model

Alexander Kaufmann(°),Sara Manzoni(*), Andreas Resetarits(°)

Abstract*— In this paper we describe one of the results of the research activities that have been conducted by an interdisciplinary research group composed by computer scientists and economists during the Exystence Thematic Institute on "Regional Innovation Systems and Complexity" (Wien, September 2004). The main aim of work is to apply the Multilayered Multi Agent Situated Systems (MMASS) to model socio-economic processes in residential and industrial development. Some of the group members have previously experienced in modeling this type processes according to a microeconomic agent-based approach (and they have already developed a simulation system). The specific model we considered in this work assumes that commuter traffic in urban regions can be studied as an emergent phenomenon of the decisions of individual heterogeneous agents (i.e. households decide on residence, firms on location). We will show that the adoption of the MMASS approach provides modelers with the necessary expressive power that the problem requires and, at the same time, it allows to obtain a model that is simpler both to be developed and to be used. The typical use of this type of model is, as in the case we describe, to develop a simulation system that implements it. Thus, a software tool (like the one provided by MMASS) that allows to design and develop simulations can be fruitfully exploited by domain experts that are interested in model domain validation and domain analysis. In this paper we report the first phase of one of the researches that will be conducted during a research framework that involved the Austrian Research Center Seibersdorf (ARCS) and the Department of Computer Science, Systems and Communication (DISCo) of the University of Milano-Bicocca.

***Index Terms*—MAS-based modeling of complex system, complex system in economics and land use, MMASS modeling**

I. Introduction

THIS paper reports a research activity that has been conducted during the Thematic Institute "Regional Innovation Systems and Complexity" within the Exystence framework (http://www.complexityscience.org). The working group was composed by computer scientists and economists and they collaborated in order to define a common framework where to conduct a joint research on complex systems that could be fruitful and interesting for both the involved research disciplines.

The main objectives of this collaboration can be summed up as follow:

- Investigation on the notion of agents in microeconomics and its classification according to concepts and notions that are traditionally considered by agent research in computer science (i.e. agent architectures and behavioral models, interaction models within Multi-Agent Systems, relationship between agents and their surrounding environment, …).
- Investigation on the use of agent-based simulations in economics. This part of the work concerns an overview of the main motivations and goals that bring economists in developing software simulation systems in their researches and work (e.g. model validation, prevision, analysis, and so on). Moreover, particular attention is paid to the identification of the set of requirements and tools from the viewpoint of system developers (e.g. computational models, software platforms) and of simulation users (e.g. analysis approaches and tools).
- Definition of a common framework starting from analogies and differences emerged from the analysis of the different use of the notion of agents and interactions within computer science and economics. The common framework aims at concerning the conceptual, modeling, as well as computational point of views.

This working group aims to the definition of a set of methodological and software tools to support researchers and experts in landuse management in their research activities. In order to reach this long-term research goal, the working group has identified a set of activities to be conducted together or by group members individually. On one hand, a set of activities

Manuscript received November 2, 2004.

S. Manzoni is with the Department of Computer Science, Systsems and Communication *(DISCo) at the University of Milano-Bicocca* (e-mail: manzoni@disco.unimib.it).

A. Kauffman and A. Reseitarits are with the *Systems Research (technology-economy-environment department) at the Austrian Research Center Seibersdorf (ARCS).*

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

will be conducted by the members of the working group individually (even if a coordinated way). An example of these individual activities consists in overviewing available computational models (e.g. agent-based, based on Multi Agent Systems and Cellular Automata or their composition, and so on), focusing in previous experiences in adopting agent approach in Economics. The main aim of this activity is to formalize a set of fundamental simulation requirements that are coming from Economics (in particular from new emerging approaches to study complex systems from an economic viewpoint).

On the other hand, this paper will describe one of the activities that will be conducted by interdisciplinary working groups (here we report the one that has been conducted during the Wien Thematic Institute hosted by ARCS). The aim of this activity was to experiment the application of the Multilayered Multi-Agent Situated Systems [Bandini et al., 2002] to model socio-economic processes in residential and industrial development. Some of the group's members have previously experienced in modeling this type of processes according to an agent-based approach and a microeconomic simulation tool has already been developed. The specific model we considered in this work assumed and demonstrated that commuter traffic in urban regions can be studied as an emergent phenomenon of the decisions of individual heterogeneous agents (i.e. households decide on residence, firms on location).

In this paper we will show that the adoption of the MMASS approach provides modelers with the necessary expressive power that the problem requires and, thus, allows representing the commutation problem as an emergent phenomenon of the residential-industrial complex systems composed by situated interacting agents. At the same time, the MMASS allows to obtain a model that is simpler to be developed, updated and, thus, used. The typical use of this type of model is, as in the case we describe, to develop a simulation system that implements it. Thus, after having introduced field data about the area that is object of the study (i.e. the Wien area in this case), domain experts analyze the simulation runs in order to reach the simulation aims (maybe, for instance, the validation of the model itself, or the prevision or explanation of known phenomena). During this process, very often, the originally developed model may require to be updated. In fact, different versions of a model are usually developed, enriching first versions with previously not considered elements, additional parameters that had previously been disregarded, ignored or unknown, and so on.

The paper is organized as follow: first, we draw an overview about the adoption of distributed approaches (based on agents, MAS and CA) in economic theory (focusing in land use and traffic simulation contexts). Then, we will briefly overview the agent-based microeconomic model that has been previously proposed to study commuting as an emergent phenomenon, and we propose a model of the same problem according to the MMASS approach, pointing out the motivations of the adoption of MMASS among other MAS-based modeling tools. Finally the paper will end with some considerations on this proposal.

II. WHY AN AGENT-BASED MODEL?

Over much of its history economic theory has been preoccupied with explaining the optimal allocation of scarce resources. As a consequence of the notion of an optimal solution equilibrium between supply and demand of goods has become the central concept in economics. In order to be able to analyze partial and total equilibrium models, they have to be extremely simplified. It is especially the, usually necessary, assumption of homogeneity (i.e. a single agent called 'representative') that misses important aspects of economic reality. Traditional economics focuses primarily on the market as a selection mechanism, but neglects the market as a cause of variation and innovation. Of course, there have been many theories (e.g. [Schumpeter, 1999]) dealing with the evolution of economic systems, but they always lacked the rigor of equilibrium economics. For evolutionary models new methods were required, and agent-based modeling approach suggests interesting research directions. This approach is certainly adequate for analyzing economic models characterized by heterogeneity of agents, bounded and contradicting rationalities of agents, strategic behavior, imperfect information, imperfect competition, and other factors leading to out-of-equilibrium dynamics [Arthur et al., 1997]. Agent-based modeling helps to understand the economy as a co-evolutionary system, linking the economic macrostructure to the microeconomic behavior of individual agents (Batten, 2000). However, for a really evolutionary model of the economy, it is not sufficient to build agent-based models only to explain the emergence and change of relations between agents (e.g. as suggested by network models). Agent-based modeling has also to contribute to the understanding of the emergence and change of behavioral norms, organizations and institutions, which, at present, seems to be a much more difficult task [Tesfatsion, 2003].

Self-organization models, used to explain urban development or traffic flows, are not new. Until now, most models have focused on one of these issues only. So far there have been only few attempts to deal with urban development and traffic flows in a combined model in order to understand their mutual interdependence. As far as urban development is concerned, the limits of equilibrium-based approaches have led to an increased interest in simulation which is better able to capture the complex dynamics of interactions between heterogeneous agents. Cellular Automata (CA) have been the most frequently applied method [Portugali, 1999]. The fact that already simple rules can lead to complex dynamics and the direct applicability on spatial processes have made CA to a widely used tool for analyzing patterns of urban development that are characterized by self-organization. One of the first CA-models in economic research analyzed the emergence of social segregation caused by the preference of people to live in the neighborhood of other people belonging to the same

social class [Schelling, 1969]. Other CA-models concerned land use patterns and their change over time (e.g. [Colonna et al., 1998]). As far as traffic is concerned, simulation has been used as a tool to improve traffic planning and management of traffic flows. For this purpose CA as well as MAS-based models have been proposed (e.g. [Raney et al., 2002]; [TRANSIMS]). Agent-based traffic simulation models are especially useful, because they enable the identification of each individual car, truck, bike or pedestrian. As a consequence, it is possible to analyze individual objectives, route plans, search and decision strategies [Batten, 2000] as well as effects of learning and changes of strategies on the traffic flows [Raney et al., 2002].

Within our interdisciplinary research, we claim that economy researches requires dedicated and more specific tools (both at the methodological and software levels) to be applied to this growing and interesting direction. Moreover, we claim that researches and studies on agents in computer science are ready to provide these modeling and computational tools in order to fruitfully support economy theory.

III. The microeconomic model

The microeconomic model by which this work has been inspired is based on the decisions of individual heterogeneous agents: households decide on residence, firms on location. Commuting is both a result of decisions of individual agents (i.e. an emergent feature in the urban system), and a feedback factor influencing the decisions of households and firms. The here described model focuses on self-organization of households and firms, while other agents (e.g., regulation of land use by municipalities) are taken as given.

A. Residential and industrial choice of location and commuter flows

The model consists of two classes of agents: *households* of employed persons and *firms*. Both classes are heterogeneous with respect to preferences on location. Specific types of households prefer given residential locations as well as specific firms prefer given sites. They regard different location factors and they attribute different weights to certain factors. In particular, on one hand *households* looking for residence take the following factors into account:

- the residential density at their location and in the surroundings;
- the availability of private services at their location and in the surroundings;
- the green space at their location and in the surroundings;
- the distance to the city centre.

On the other hand, *firms* looking for their optimal location focus on the following factors:

- the industrial density (as an indicator for the price of a certain location) at their location and in the surroundings;
- the ratio between demand and competitors at their location and in the surroundings;
- the proximity of related firms (suppliers, services, customers) within a cluster at their location and in the surroundings;
- the distance to the next transport node (highway exit, railroad station).

In the model the notion of distance between two locations does not indicate the topological distance, but it is given by an estimate of the time needed to reach a location from the other one taking into account the type of available connections (e.g. roads, underground line, train line). In the model experimentations these values have been computed according to collected field data and considering the availability of the different transportations in the experimentation territory (i.e. Wien urban territory).

The behavior of households in trying to find out their residential location is based on a location utility function and a cost function which considers commuting and relocation in case of changing residence. Commuting is a result of the choice of residence and the randomly determined new job opportunities or losses. Employed persons and jobs, accordingly, are differentiated by levels of qualification, so that not any job is accessible for every employed person. The behavior of firms is based on their location utility and a cost function of relocation in case of changing the site.

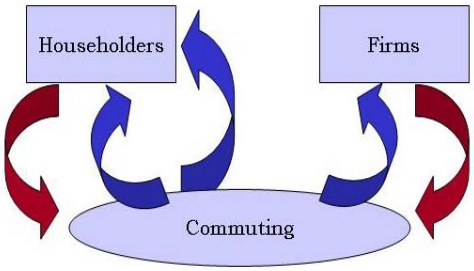


Figure 1: Influences and feedbacks between householdders, firms and commuting

Combining the decisions on residential and industrial locations, as well as the random job matching, leads to commuter flows between the locations which, in turn, enter the residential choice of households. Further feedback (represented in Figure 1) concerns the change in residential and industrial density, both being factors on which households and firms base their respective decision-making processes. Moreover, other factors that influence residential and industrial development and commutation are determined exogenously (for instance: location preferences of firms, changes in residential preferences life cycle of households, changes in job opportunities and employment, zoning and restrictions of land use, provision of traffic infrastructure).

The chart in Figure 2 gives a short overview of the whole microeconomic model. In the following sections we describe its modules in more detail.

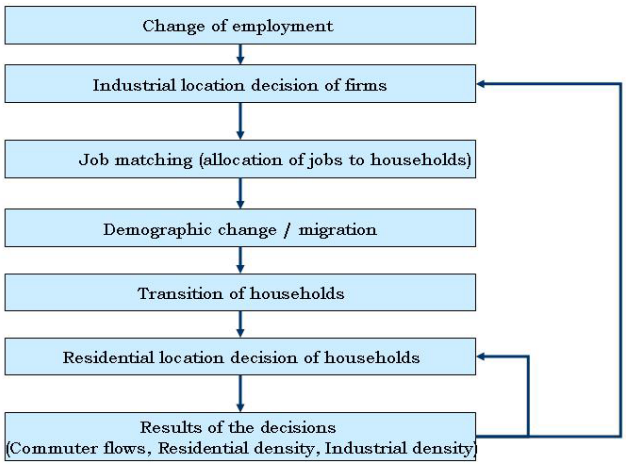


Figure 2. Structure of the model

B. *Endogenous processes*

As previously introduced, there are two different types of location decisions performed by the two types of agents that the model considers: *households* decide on their residence, while *firms* on the production site. Information needed for both decisions can be either perfect or distance-dependent. In a first model version, we supposed agents to have perfect information all over the urban region; in a second one, a distance discount parameter (i.e. σ) has been introduced in order to let agent sensitivity to information decreases with the distance.

Using information regarding the relevant location factors any household maximizes its residential utility and takes into account commuting and relocation costs (Table 1 lists and schematically describes all the involved parameters):

$$H: \alpha R + \beta S + \delta G + \gamma D_{ic} - (C3 + C1) \rightarrow \max$$

TABLE 1: RESIDENTIAL LOCATION FACTORS

Households H	H	
Residential density	R*	$R_i = H_i/B_i$, $R^*_I = R_i + \Sigma_j R_j e^{-\sigma D_{ij}}$, normalized: % of R'
Private services (relative supply)	S*/H	$S^*_I = S_i + \Sigma_j S_j e^{-\sigma D_{ij}}$, normalized: % of H
Green space	G*	$G^*_I = G_i + \Sigma_j G_j e^{-\sigma D_{ij}}$, normalized: % of A
Distance between I and j	D_{ij}	
Downtown distance	D_{ic}	normalized: % of D_{max}
Residential relocation cost	C1	
Residential area	B_i	

On the contrary, according to the information regarding the relevant location factors (either in perfect or distance-dependent information versions) any firm maximizes its location utility considering relocation cost (see Table 2):

$$F: \varphi I + \lambda P + \mu X + \pi D_{in} - C2 \rightarrow \max$$

TABLE 2: INDUSTRIAL LOCATION FACTORS

Firms F	F	
Industrial density	I*	$I_i = F_i/M_i$, $I^*_i = I_i + \Sigma_j I_j e^{-\sigma D_{ij}}$, normalized: % of I'
Demand / competition ratio	P*	$P^*_i = (H_i + \Sigma_j H_j e^{-\sigma D_{ij}}) / (S_i + \Sigma_j S_j e^{-\sigma D_{ij}})$
Cluster (relative supply)	X*/F	$X^*_I = X_i + \Sigma_j X_j e^{-\sigma D_{ij}}$, normalized: % of F
Distance between I and j	D_{ij}	
Transport node distance	D_{in}	Normalized: % of D_{max}
Industrial relocation cost	C2	
Industrial area	M_i	

Both types of agents, households as well as firms, are heterogeneous regarding their location preferences (households also with regard to their qualification). According to their type (i.e. 'household' or 'firm'), agents apply the above utility function, but they differ with respect to the weights associated to each location factor (see, respectively, Table 3 and Table 4). Symbols in Tables 3 and Table 4 indicate the relevance of each parameter and the type of its effects (i.e. either positive or negative).

TABLE 3: CLASSES OF HOUSEHOLDS AND PREFERENCES

		Residential preferences			
		α R	β S	δ G	γD_{ic}
Highly qualified suburbanites	(Q=1)	--	0	++	-
Highly qualified urbanites	(Q=1)	+	++	0	++
Less qualified suburbanites	(Q=2)	-	0	++	-
Less qualified urbanites	(Q=2)	0	+	0	+

TABLE 4: CLASSES OF FIRMS AND PREFERENCES

		Location preferences			
		φ I	λ P	μ X	πD_{in}
Private services	(S)	0	++	+	0
Cluster firms	(X)	-	0	++	-
Large scale manufacturing	(V)	--	0	0	-
Utilities	(U)	0	0	0	0

In order to solve conflicts in case of density constraints, in the here presented model, "First come – first locate" strategy with random order (i.e., reordering of agents after each step) was applied. Alternative strategies such as comparison of the added value (e.g. those with the highest value are allowed to locate, the others either stay where they are), or have to choose second-/third-best locations, are possible but have not still been applied.

C. *Exogenous processes*

All the actual parameters and several parameters for the model experimentation have been determined exogenously (several sets of parameters are tested in the simulation runs). This concerns residential preferences, industrial location preferences, generation and loss of jobs, zoning and maximum density and transport infrastructure.

Industrial location preferences are constant; they do not change during the simulation period. On the other hand, residential preference for suburban or urban locations changes

probabilistically according to an assumed household life cycle (see Table 6). When a household reaches age 60 we assume that it retires, stops commuting and does not change location. Transition probability is estimated according to the frequency of households with and without children per age class (mean value of all municipalities is more than one municipality is considered). Moreover, the qualification level does not change according to age.

TABLE 6: TRANSITION PROBABILITIES OF HOUSEHOLDS

			Probability per age class		
			-30	*31-45*	*45-60*
Highly qualified suburbanites	→	Highly qualified urbanites	low	very low	Negligible
Highly qualified urbanites	→	Highly qualified suburbanites	low	very high	High
Less qualified suburbanites	→	Less qualified urbanites	very low	Negligible	Negligible
Less qualified urbanites	→	Less qualified suburbanites	Negligible	low	very low

The generation and loss of jobs is defined by the respective national industrial activity. The latter changes randomly within a specific industry the job opportunities offered by a certain firm and, after matching with people looking for jobs, it leads to the actual employment of any firm.

As far as spatial information is concerned, the regulation of land use (zoning), the upper limits of density and the provision of infrastructure (traffic capacity) are determined exogenously and may change discretely over time

IV. THE MMASS-BASED MODEL

Among models based on Multi Agent Systems (MAS [Ferber, 1999]), within our research framework we decided to adopt the Multilayerd Multi Agent Situated Systems (MMASS [Bandini et al., 2002]. The main motivations of this decision are strictly related to problem features and peculiarities (i.e. relevance of spatial features of agent environment, strong role of agent situatedness in their behaviors and interactions …) that we will overview in the following section. Then our proposal of applying the MMASS approach to model the above described problem[1] will be described.

A. Why MMASS?

Some features that we identified as interesting in relation to the considered problem are:

- It explicitly describes the spatial structure of agent environment (i.e. space): a multilayered network of sites where each node can host an agent, and represents a part of the distributed medium for the diffusion of signal emitted by agents to interact (e.g. to provide information to other agents).
- MMASS agents can be characterized by heterogeneous behaviors that are space-dependant: an action is performed by an agent since it belongs to some given type, it is currently characterized by some given state and it is situated in a given spatial location.
- Interactions between MMASS agents are heterogeneous and space-dependant (i.e. the distance between agents is an element that determines its nature – e.g. synchronous vs asynchronous, direct vs indirect, local vs at-a-distance):
  - MMASS agents interact according to direct interaction mechanism (i.e. MMASS *reaction*) when they are situated in adjacent positions and have previously agreed to synchronously change their states;
  - not adjacent agents can interact according to an indirect interaction mechanism based on emission-diffusion-perception of signals (i.e. *MMASS fields*) emitted by agents themselves
- Multilayered spatial structure (i.e. multiple situated MAS can coexist and interact): MMASS allows the modeler to exploit multiple layers in order to represent a high-complexity system (like the one of the reference problem) as composed by multiple interacting systems of lower complexity. Heterogeneous aspects that contribute to the behavior and dynamics of the whole system can be described by distinct MAS situated in distinct (but interconnected) layers of the spatial structure

B. The proposal

In order to apply the MMASS approach to represent the above described problem model, we first distinguished *territorial elements (i.e. territory)* from those entities that populate the territory and behave according to their type, their state and the state of the local environment they are currently situated.

We describe a territory as a discrete set of locations where either residential or industrial buildings are allowed (other location types have not been considered). A suitable representation of the territory set of locations in a graph-like structure (see the top of Figure 3), where each node of the graph represents a territory area (and its type), and graph edges represent connections between territory areas. In this representation, an edge exists between two locations only when some transportation infrastructure (e.g. road, train line) exists between them. Useful available information can be associated to each graph node and edge. For instance, edges can be labeled with information about the type of available transportation, the average number of cars per hour when it represent a road, mean delay time if it represents public transportations, and so on.

In adopting this type of representation of the territory, we have adopted a first feature of the MMASS model that is, the

[1] A detailed description of the MMASS approach is out the scopes of this paper. Details on the model can be found in [Bandini et al., 2002]; for some examples of its applications within the research context of modeling and simulation of complex systems see [Bandini et al., 2004a] and [Bandini et al., 2004b].

possibility to describe the structure of the environment that is populated by a set of active entities (i.e. agents). MMASS agents can represent thus those system entities that perform some kind of decision-making process (according to their features and state and the ones of the environment they are situated in).

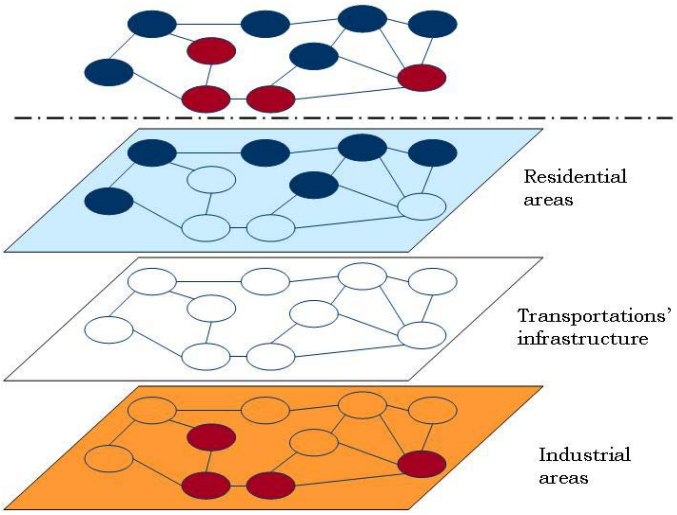


Figure 3. Multi-layered representation of the territory

The second feature of the MMASS approach that we exploited concerns the possibility to represent the environment where agents are situated according to a multilayered structure. Thus, given a territory, we represent it according to a structure composed by three layers. Two layers (the ones on the top and bottom of Figure 3) are devoted to represent those territorial areas in which, respectively, residential and industrial buildings are allowed. Each layer can be seen as a sort of "view on the territory graph representation" where only subsets of the graph nodes are considered.

The main motivation of this choice is related to the fact that, in this way, at residential and industrial layers we can represent two distinct complex sub-systems (i.e. "Householders' System" and "Firms' System" respectively). In fact the effect of householders' decisions, first of all, occurs within the system they are part of, but at the same time, householders and firms belong to two different complex systems. The third sub-system we considered is represented by "Commuting". We will not describe here into details the behavior, architecture and interaction abilities of agents that constitute each system since they are mainly based on the firms' and householders' models that have been described in Section 2.

According to system description (see Section 2 and Figure 1), we have identified three main influences that can occur between these three sub-systems (Figure 4):

1. *Householders' and Firms' systems* → *Commuting*: commuting is the result of decisions of householders and firms;
2. *Decisions in Firms' System* → *Householders' System*: decisions in the Firms' System influences the Householders' System since a firm may move to a location that may cause a change in decisions of some householders. This influence is not bidirectional since the availability of 'manpower' in the surroundings has not been considered by domain experts as a fundamental factor in firms' decisions-making process.
3. *Commuting* → *Householders' System*: the level of commuting is one of the main elements in householders' decisions (while it is not a factor influencing firms' decisions on their location).

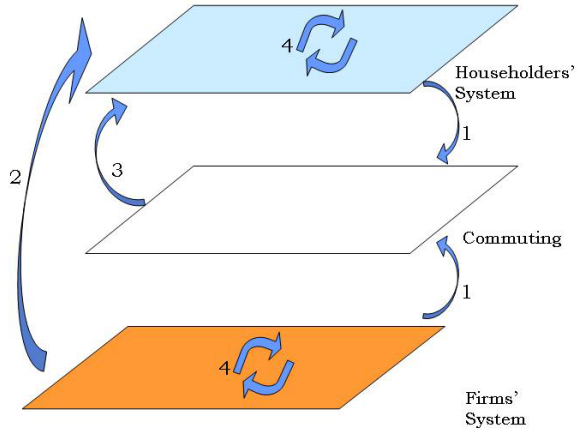


Figure 4. Influences between Systems

C. *Some observations on the proposal*

From the MMASS-based model description, we can draw some first observations and conclusions about the suitability of the adoption of the MMASS approach for the considered problem. In fact MMASS allows modelers to

- represent all the elements of the microeconomic reference model (that already demonstrated to fruitfully allow to represent the considered problem);
- better separate different elements involved in the complex system dynamics (e.g. territorial and decisional ones);
- explicitly represent influences, feedbacks and interactions between sub-systems;
- simpler update, and incrementally improve, the model.

Moreover, MMASS, despite other MAS-based modeling approaches, allows domain experts to simpler develop simulation software in order to experiment, validate, and update the model according to the problem requirements. In fact, a simulation platform for models based on MMASS is already available [Bandini et al., 2004c].

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we have described a microeconomic agent-based model of a complex system where commuting is strongly involved in system dynamics (it is the result of householders and firms decisions and, at the same time, it is involved in their decisions). We have not included in this paper a discussion on the quality of this model. For this work, this model is the reference model and it is out of the scopes of this paper to validate it and verify its suitability.

Thus, we have proposed a MMASS-based model of the same problem (Section 3). The aim of this work was not to propose a modeling approach that improves the suitability or validity of the microeconomic model. On the contrary, we have proposed this modeling approach since it provides interesting features related to the reference scenario and to the goals of the microeconomic model. The main features of MMASS approach that can be useful in this work have been listed and some of them have been exploited in its application.

The here presented work is still ongoing and next activities will concern:

- specification of agent behavioral models: this work will be performed according to the behaviors of agents described by the microeconomic model (see households' and firms' utility functions);
- detailed specification of interactions and influences between sub-systems;
- development of a simulation system based the MMASS-based model: in performing this activity, we will exploit the tools provided by the MMASS platform [Bandini et al., 2004c] that will allow us to produce a simulation tool in the short time.

REFERENCES

[Arthur et al., 1997] Arthur, W.B., Durlauf, S.N., Lane, D.A. (eds.), *The economy as an evolving complex system II*, Perseus Books, 1997

[Bandini et al., 2002] Bandini, S., S. Manzoni, C. Simone, *Heterogeneous Agents Situated in Heterogeneous Spaces*, Applied Artificial Intelligence, Vol. 16, n. 9-10, 2002, pp. 831-852.

[Bandini et al., 2004a] Bandini, S., S. Manzoni, G. Vizzari, *Multi-Agent Approach to localization Problems: the Case of Multilayered Multi-Agent Situated System*, Web Intelligence and Agent Systems International Journal, IOS PRESS, 2004. (in press).

[Bandini et al., 2004b] Bandini, S., S. Manzoni, G. Vizzari, *Situated Cellular Agents: A Model to Simulate Crowding Dynamics*, IEICE TRANSACTIONS on Information and Systems, Vol.E87-D(3), march 2004, pp.669-676.

[Bandini et al., 2004c] Bandini, S., S. Manzoni, G. Vizzari, *Towards a platform for MMASS based simulations: focusing on field diffusion,* To appear in Applied Artificial Intelligence, Taylor & Francis, 2004.

[Batten, 2000] Batten, D.F., Discovering artificial economics. How agents learn and economies evolve, Westview Press, 2000.

[Colonna et al., 1998] Colonna, A., di Stefano, V., Lombardo, S.T., Papini, L., Rabino, G.A., *L.A.U.D.E.: Learning automata for urban development exploration. The case study of Rome urban system*, ERSA-conference 1998, Vienna, 1998.

[Ferber, 1999] Ferber, J., Multi-Agent Systems: An Introduction to distributed artificial intelligence, Addison-Wesley, Harlow (UK), 1999.

[Portugali, 1999] Portugali, J., *Self-organization and the city*, Springer, 1999.

[Raney et al., 2002] Raney, B., Cetin, N., Völlmy, A., Nagel, K., *Large scale multi-agent transportation simulations*, ERSA-conference 2002, Dortmund, 2002.

[Schelling, 1969] Schelling, T.S., *Models of segregation*, American Economic Review, 59(2), 488-493, 1969.

[Schumpeter, 1939] Schumpeter, J.A., Business cycles. A theoretical, historical, and statistical analysis of the capitalist process, McGraw-Hill, 1939.

[Tesfatsion, 2003] Tesfatsion, L., *Agent-based computational economics*. ISU Economics Working Paper no. 1, 2003.

[TRAweb] *Transportation analysis simulation system (TRANSIMS)*, http://transims.tsasa.lanl.gov/, http://www.transims.net/.

Structuring Organizations by Means of Roles Using the Agent Metaphor

Guido Boella
Dipartimento di Informatica - Università di Torino - Italy

Leendert van der Torre
CWI - Amsterdam - The Netherlands

***Abstract*— In this paper we propose to define the organizational structure of multiagent systems using the agent metaphor. The agent metaphor is not only used to model software agents, but also social entities like organizations, groups and normative systems. We argue that mental attitudes can be attributed to them - beliefs, desires and goals - and also an autonomous and proactive behavior in order to explain their behavior. We show how the metaphor can be applied also to structure organizations in functional areas and roles, which are described as agents too. Thus, the agent metaphor can play a role similar to the object oriented metaphor which allows structuring objects in component objects. Finally, we discuss how the agent metaphor addresses the problems of control and communication in such structured organizations.**

I. Introduction

Software engineering is used to provide models and techniques to develop complex software system. It is necessary to make it easier to handle the complexity arising from the large number of interactions in a software system [1]. Models and techniques allow expressing knowledge and to support the analysis and reasoning about a system to be developed. As the context and needs of software change, advances are needed to respond to changes. For example, today's systems and their environments are more varied and dynamic, and accommodate more local freedom and initiative [2].

For these reasons, agent orientation emerged as a new paradigm for designing and constructing software systems [1], [2]. The agent oriented approach advocates decomposing problems in terms of autonomous agents that can engage in flexible, high-level interactions. In particular, this is a natural representation for complex systems that are - as many real systems are - invariably distributed [1]. Compared to the still dominant software paradigm, namely object orientation, agent orientation offers a higher level of abstraction for thinking about the characteristics and behaviors of software systems. It can be seen as part of an ongoing trend towards greater interactivity in conceptions of programming and software system design and construction. Much like the concepts of activity and object that have played pivotal roles in earlier modelling paradigms - Yu [2] argues - the agent concept can be instrumental in bringing about a shift to a much richer, socially-oriented ontology that is needed to characterize and analyze today's systems and environments.

The shift from the object oriented perspective to the agent oriented one is not, however, without losses. Booch [3] identifies three tools which allow coping with complexity: "1) Decomposition: the most basic technique for tackling any large problem is to divide it into smaller, more manageable chunks each of which can then be dealt with in relative isolation. 2) Abstraction: the process of defining a simplified model of the system that emphasises some of the details or properties. 3) Organisation: the process of identifying and managing interrelationships between various problem solving components."

In the agent oriented approach, however, decomposition, abstraction and organization are not yet addressed with the same efficacy as in the object oriented approach, where an object can be composed of other objects, which can be ignored in the analysis at a certain level of abstraction. The agent metaphor is sometimes proposed as a specialization of the object metaphor [4]: agents do not only have - like objects - a behavior which can be invoked by the other agents, but they also autonomously act and react to changes in the environment following their own goals and beliefs. In contrast, the component view of objects in the object metaphor could to be lost. The property of agents, i.e., sociality, closest to the property allowing the aggregation of objects to form more complex objects is not enough to overcome the gap. In particular, multiagent systems offer as aggregation methods the notion of group or of organization. According to Zambonelli *et al.* [5] "a multiagent system can be conceived in terms of an organized society of individuals in which each agent plays specific roles and interacts with other agents". At the same time, they claim that "an organization is more than simply a collection of roles (as most methodologies assume) [...] further organization-oriented abstractions need to be devised and placed in the context of a methodology [...] As soon as the complexity increases, modularity and encapsulation principles suggest dividing the system into different suborganizations". According to Jennings [1], however, most current approaches "possess insufficient mechanisms for dealing with organisational structure". Moreover, what is the semantic principle which allows decomposing organizations into suborganizations must be still made precise.

The research question of this paper, thus, is: how can the agent oriented paradigm be extended with a decomposition structure isomorphic to the one proposed by the object oriented paradigm? How can a multiagent system be designed and constructed as an organization using this structure?

The methodology we use in this paper is a normative multiagent framework we proposed in [6], [7], [8], [9]. The

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

basic idea of this framework is: agents attribute mental attitudes, like beliefs, desires and goals, to the other agents they interact with and also to social entities like groups, normative systems, and organizations. Thus these social entities can be described as agents too, and at the same time, the components of organizations, namely, functional areas and roles, can be described as agents, as in the ontology we present in [7]. We call them *socially constructed agents*.

This paper is organized as follows. In Section II we discuss the progress from object orientation to agents and socially constructed agents. In Section III we present the formal model and in Section IV we discuss the issue of control and communication in an multiagent system structured as an organization. A summary closes the paper.

II. From objects to socially constructed agents

The trend in software and requirements engineering and in programming languages paradigms has been from elements that represent abstract computations towards elements that represent the real world: from procedural to structured programming, from objects to agents. Agent systems have no central control authority, instead each agent is an independent locus of control, and the agent's task drives the control. Delegating control to autonomous components can be considered as an additional dimension of modularity and encapsulation. Intentional concepts such as goals, beliefs, abilities, commitments, *etc*., provide a higher-level characterization of behavior. One can characterize an agent in terms of its intentional properties without having to know its specific actions in terms of processes and steps. Explicit representation of goals allows motivations and rationales to be expressed. The agent concept provides a local scope, for reconciling and making tradeoffs among competing intentionality, such as conflicting goals and inconsistent beliefs. By adopting intentional modelling, the networks of dependencies among the agents can be modelled and reasoned about at a high level of abstraction. Moreover, cooperation among agents cannot be taken for granted. Because agents are autonomous, the likelihood of successful cooperation is contingent upon many factors. However, an agent that exists within a social network of expectations and obligations has behaviors that are confined by them. The agent can still violate them, but will suffer the consequences. The behavior of a socially situated agent is therefore largely predictable, although not in a precise way.

Given that agents are nowadays conceived as useful abstractions for modelling and engineering large complex systems, the need for a disciplined organizational principle for agent systems emerges clearly in the same way as the formalizatoin of the object decomposition principle does in the case of object oriented systems.

One of the main features of the object perspective is that objects are composed by other objects and that objects can be replaced by other objects with the same properties (e.g., the same interface). This is not entirely true for agents. According to Jennings [1], "the agent oriented approach advocates decomposing problems in terms of autonomous agents", but no further decomposition seems possible. To overcome this flatness limitation, the organization metaphor has been proposed, e.g., by [10], [5]. Organizations are modelled as collections of agents, gathered in groups [10], playing roles [1], [11] or regulated by organizational rules [5]. What is lacking is a notion of organization as a first class abstraction which allows decomposing into subproblems the problem which a system wants to solve, using a recursive mechanism (as the object decomposition is) until autonomous agents composing a multiagent system are reached.

The desired solution is required to model at least simple examples taken from organizational theory in Economics as the following one. Consider a simple enterprise which is composed by a direction area and a production area. The direction area is composed by the CEO and the board. The board is composed by a set of administrators. The production area is composed by two production units; each production unit by a set of workers. The direction area, the board, the production area and the production units are *functional areas*. In particular, the direction area and the production areas belong to the organization, the board to the direction area, *etc*. The CEO, the administrators and the members of the production units are *roles*, each one belonging to a functional area, e.g., the CEO is part of the direction area.

This recursive decomposition terminates with roles: roles, unlike organizations and functional areas, are not composed by further social entities. Rather, roles are played by other agents, real agents (human or software) who have to act as expected by their role.

The object metaphor is not adequate to deal with such a structure, because each entity can be better described in terms of belief, desires and goals, and of its autonomous behavior. We talk, e.g., about the decisions of the CEO, or about the organization's goal to propose a deal, about the belief of the production area that the inventory is finished, *etc*. Hence, at first sight, these entities can be described as autonomous agents. But this is not sufficient, since the agent metaphor does not account for the decomposition structure of an organization relating it with its functional areas and roles. Moreover, organizations, functional areas and roles do not exist in the same sense as (human or software) agents do. Thus, if we want to follow this intuition, the agent metaphor must be extended. Inspired by Searle [12]'s analysis of social reality we define organizations, functional areas and roles as *socially constructed agents*. These agents do not exist in the usual sense of the term, but they are abstractions which other agents describe as if they were agents, with their own beliefs, desires and goals, and with their own autonomous behavior. The argument goes as follows:

1) agents can attribute to other (human or software) agents mental attitudes and an autonomous behavior to explain how they work, regardless of the fact that they really have any mental attitudes (the *intentional stance* of Dennett [13]);
2) according to Searle [12], agents create new social entities like institutions - e.g., money and private property -

by means of collectively attributing to existing entities - e.g., paper bills - a new functional status - e.g., money - and new qualities.
3) if the new functional status is composed by mental attitudes and autonomous behavior, the new entities are described as agents: *socially constructed agents*.
4) hence, socially constructed agents, *qua* agents, can create new socially constructed agents by attributing mental attitudes to them, in turn.

Agents create organizations by collectively attributing them mental attitudes; organizations, as socially constructed agents, can create new social entities like functional areas and roles which are the components of the organization. Functional areas, as agents, can in turn apply the agent metaphor to create subareas and further roles, and so on. Roles are descriptions of the behavior which is expected by agents who, with their own mental attitudes, play these roles: the role's expected behavior is described in terms of mental attitudes, since roles are considered socially constructed agents. Modelling roles by attributing them mental attitudes allows a more expressive way to describe the expected behavior with respect, e.g., the scripts proposed by Activity Theory [14]. In this manner, we have a way to structure an organization in components with an homogeneous character - since they are all agents - in the same way as the object orientation allows structuring objects by means of objects. An advantage of this way of structuring an organization is that its components can be described as agents with beliefs, desires and goals. Hence, the same decomposition approach advocated by [1] is used for structuring an organization: it is decomposed in a set of autonomous agents: not only real ones, but socially constructed agents like functional areas and roles; socially constructed agents do not exist, but they are only used as abstractions in the design analysis to structure an organization. At the end of the process there are only human or software agents which, to coordinate their behavior, behave as if they all attribute the same beliefs, desires and goals to the organization. This is a subjective approach to coordination [14].

Another reason why organizations, functional areas and roles should be all considered as agents - and not simply groups - is that they have private properties and agents who are employed in them; so a department can possess a building and machines, employ people, *etc*. Moreover they are the addressees of obligations (e.g., to pay the employees), permissions (e.g., a role can use a certain machine) and powers (e.g., the role of CEO can take decisions). This is what is also meant by the law when such social entities are defined as "legal persons": they are considered persons with obligations and rights [15]. Finally, organizations and functional areas, as legal institutions, are normative agents themselves: they are agents who can pose obligations on the roles and on the employees, e.g., by giving orders to them, or endow them with permissions and powers.

There is a difference with the decompositional view of the object oriented perspective which must be noticed. The parts of an object exist by themselves and the object itself exists only as long as its (essential) parts exist. In contrast, in an organization the perspective is reversed: the "components" of the organization exist only as long as the organization exists, while the organization itself can exist even without its components. The role of CEO does not have sense if the organization which the role belongs to does not exist anymore. The reason is that an organization as a social entity has no physical realization. The organization exists because of the attribution of mental attitudes by the agents of a society. In turn, functional areas and roles exist only as long as the organization attributes mental attitudes to them. An important consequence of this view is that an organization can restructure itself while continuing to exist.

As [16], [10] claim, a multiagent system should not make any assumption about the implementation of the agents. As Yu [2] notices, the agent perspective does not mean necessary that entities should be implemented with mental attitudes:

> Agent intentionality is externally attributed by the modeller. From a modelling point of view, intentionality may be attributed to some entity if the modeller feels that the intentional characterization offers a useful way for describing and analyzing that entity. For example, some entity that is treated as an agent during modelling may end up being implemented in software that has no explicit representation and manipulation of goals, *etc*.

Socially constructed agents defined in terms of beliefs, desires and goals are only an abstraction for designing the system. Moreover, the behavior of roles is described by mental attitudes, but this does not require that the agents playing roles in the organizations are endowed with beliefs and motivations: it is sufficient that their behavior conforms to that of the role they are playing.

In Figure 1, we summarize the approach: the multiagent system in the oval is composed of three real agents (boxes) who collectively attribute beliefs (B), desires (D) and goals (G) to the organization (parallelogram). The organization, in turn, attributes mental attitudes to two functional areas and functional areas to three roles. The organization and the functional areas are attributed also norms (V), facts (f), institutional facts (i) and decisions (the triangle d).

III. The conceptual model

We introduce the conceptual model necessary to cope with socially constructed agents: first the multiagent system with the attribution of mental attitudes to agents, then the normative system.

First of all, the structural concepts and their relations. We describe the different aspects of the world and the relationships among them by introducing a set of propositional variables X and extending it to consider also negative states of affairs: $L(X) = X \cup \{\neg x \mid x \in X\}$. The relations between the propositional variables are given by means of conditional rules written as $R(X) = 2^{L(X)} \times L(X)$: the set of pairs of a set of literals built from X and a literal built from X, written as $l_1 \wedge \ldots \wedge l_n \rightarrow l$ or, when $n = 0$, $\top \rightarrow l$. The rules are used to

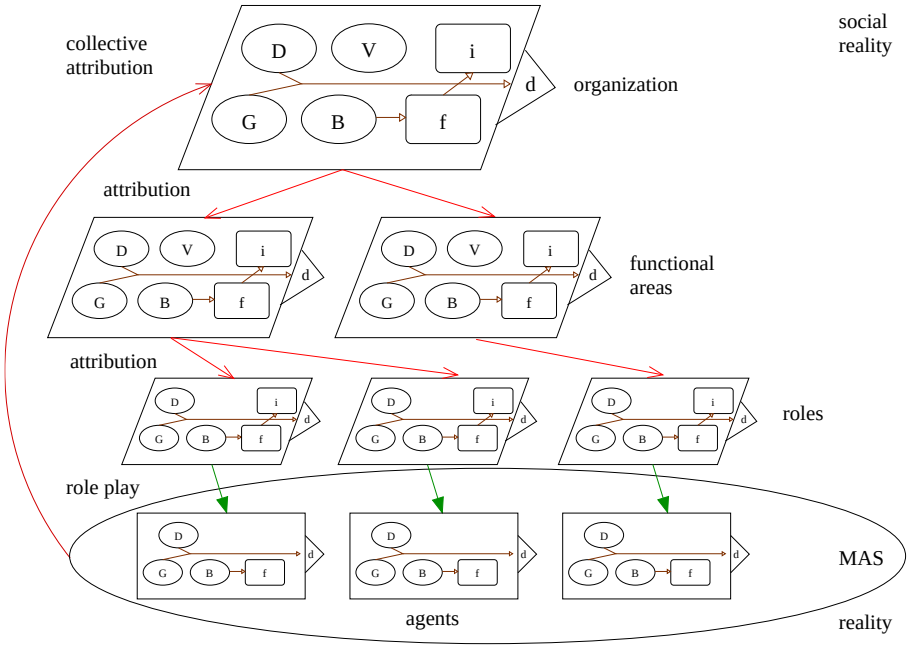


Fig. 1. The attribution of mental attitudes.

represent the relations among propositional variables existing in beliefs, desires and goal of the agents.

Then there are the different sorts of agents A we consider. Besides real agents RA (either human or software) we consider as agents in the model also socially constructed agents like organizations OA, functional areas FA, and roles RO. The different sorts of agents are disjoint and are all subsets of the set of agents A: $RA \cup OA \cup FA \cup RO \subseteq A$. All these agents have mental attitudes; by mental attitudes we mean beliefs B, desires D and goals G.

Mental attitudes are represented by rules, even if they do not coincide with them: $MD : B \cup D \cup G \rightarrow R(X)$. When there is no risk of confusion we abuse the notation by identifying rules and mental states. To resolve conflicts among motivations we introduce a priority relation by means of $\geq$: $A \rightarrow 2^M \times 2^M$ a function from agents to a transitive and reflexive relation on the powerset of the motivations $M = D \cup G$ containing at least the subset relation. We write $\geq_a$ for $\geq (a)$. Moreover, different mental attitudes are attributed to all the different sorts of agents by the agent description relation $AD : A \rightarrow 2^{B \cup D \cup G \cup A}$. We write $B_a = AD(a) \cap B$, $A_a = AD(a) \cap A$ for $a \in A$, *etc.*

Also agents are in the target of the agent description AD relation for the following reason: organizations, functional areas and roles exist only as profiles attributed by other agents. So they exist only as they are described as agents by other agents, according to the agent description relation. The AD relation specifies that an agent $b \in OA \cup FA \cup RO$ exists only as far as some other agents $\{a \in A \mid b \in A_a\}$ attribute to it mental attitudes. The set $(FA \cup RO) \cap A_o$ represents the immediate "components" of the organization or functional area $o \in OA \cup FA$. The decomposition structure of an organization ends with roles. Roles are described as agents, but they do not create further socially constructed agents; rather, roles are associated with agents playing them, $PL : RO \rightarrow RA$.

We introduce now concepts concerning informational aspects. First of all, the set of variables whose truth value is determined by an agent (decision variables) [17] are distinguished from those P which are not (the parameters). Besides, we need to represent also the so called "institutional facts" I. They are states of affairs which exist only inside normative systems and organizations: as Searle [12] suggests, money, private property, marriages, *etc.* exist only as part of social reality; since we model social reality by means of the attribution of mental attitudes to social entities, institutional facts can be modelled as the beliefs attributed to these agents, as done by [8]. Similarly, we need to represent the fact that social entities like normative systems and organizations are able to change their mental attitudes. The actions determining the changes are called creation actions C. Finally, inspired by Lee [18] we introduce the notion of documents DC: "we use the term 'document' since most information parcels in business practice are mapped on paper documents".

As concerns the relations among these concepts, we have that parameters P are a subset of the propositional variables X. The complement of X and P represents the decision variables controlled by the different agents. Hence we associate with each agent a subset of $X \setminus P$ by extending again the agent description relation $AD : A \rightarrow 2^{B \cup D \cup G \cup A \cup (X \setminus P)}$. We write $X_a = AD(a) \cap X$.

Moreover, the institutional facts I are a subset of the parameters P: $I \subseteq P$. When a belief rule $Y \wedge c \rightarrow p \in B_a$ has an institutional fact $p \in I$ as consequent, we say that $c \in X$ *counts as* p in context Y - using Searle [12]'s terminology - for agent $a \in OA \cup FA \cup RO$.

The creation actions C are a subset of the institutional facts $C \subset I$. Since agents are attributed mental attitudes, we represent their modification by adding new mental attitudes expressed as rules. So the creation action relation $CR : \{b, d, g\} \times A \times R(X) \rightarrow C$ is a mapping from rules (for beliefs, desires and goals) to propositional variables, where $CR(b, a, r)$ stands for the creation of $m \in B_a$, $CR(d, a, r)$ stands for the creation of $m \in D_a$, and $CR(g, a, r)$ stands for the creation of $m \in G_a$, such that the mental attitude m is described by the rule $r \in R(X)$: $r = MD(m)$.

Finally, the document creation relation $CD : DC \rightarrow X$ is a mapping from documents to decision variables representing their creation. We write $CD(d) \in X_a$ for the creation of document $d \in DC$.

We define a multiagent system as $MAS = \langle RA, OA, FA, RO, X, P, B, D, G, AD, MD, \geq, I, C, DC \rangle$.

We introduce obligations posed by organizations and functional areas by means of a normative multiagent system. Let the norms $\{n_1, \ldots, n_m\} = N$ be a set. Let the norm description $V : OA \cup FA \rightarrow (N \times A \rightarrow X)$ be a function from agents to complete functions from the norms and agents to the decision variables: we write V_o for the function $V(o)$ and $V_o(n, a)$ for the decision variable of agent $o \in RA \cup OA \cup FA$ representing that it considers a violation of norm n by agent $a \in A$.

$NMAS = \langle RA, OA, FA, RO, X, P, D, G, AD, MD, PL, \geq, I, C, DC, N, V \rangle$ is a normative multiagent system .

Following [6], obligations are defined in terms of goals of the addressee of the norm $\mathbf{a}$ and of the agent $\mathbf{o}$. The definition of obligation contains several clauses. The first one defines obligations of agents as goals of the normative agent, following the 'Your wish is my command' strategy, the remaining ones are instrumental to the respect of the obligation.

Agent $\mathbf{a} \in A$ is *obliged* by normative agent $\mathbf{o} \in OA \cup FA$ to decide to do $x \in L(X_{\mathbf{a}} \cup P)$ with sanction $s \in L(X_{\mathbf{o}} \cup P)$ if $Y \subseteq L(X_{\mathbf{a}} \cup P)$ in $NMAS$, written as $NMAS \models O_{\mathbf{ao}}(x, s|Y)$, if and only if there is a $n \in N$ such that:

1) $Y \rightarrow x \in D_{\mathbf{o}} \cap G_{\mathbf{o}}$: if agent $\mathbf{o}$ believes Y then it desires and has as a goal that x.
2) $Y \cup \{\sim x\} \rightarrow V_{\mathbf{o}}(n, \mathbf{a}) \in D_{\mathbf{o}} \cap G_{\mathbf{o}}$: if agent $\mathbf{o}$ believes Y and $\sim x$, then it has the goal and the desire $V_{\mathbf{o}}(n, \mathbf{a})$: to recognize it as a violation by agent $\mathbf{a}$.
3) $Y \cup \{V_{\mathbf{o}}(n, \mathbf{a})\} \rightarrow s \in D_{\mathbf{o}} \cap G_{\mathbf{o}}$: if agent $\mathbf{o}$ believes Y and decides $V_{\mathbf{o}}(n, \mathbf{a})$, then it desires and has as a goal that it sanctions agent $\mathbf{a}$.
4) $\top \rightarrow \sim s \in D_{\mathbf{a}}$: agent $\mathbf{a}$ desires $\sim s$, which expresses that it does not like to be sanctioned.

Since obligations are defined in terms of mental states, they can be created by means of the creation actions C introducing new desires and goals, as shown by [8]. In this paper, we will use the shorthand $CR(\mathbf{o}, O_{\mathbf{ao}}(x, s|Y))$ to represent the set of creation actions necessary to create an obligation $O_{\mathbf{ao}}(x, s|Y)$.

IV. Control and communication in organizations

Instead of having a single global collection of beliefs and motivations, modelling organizations as socially constructed agents allows allocating different beliefs B_a, desires D_a and goals G_a to separate agents $a \in A_{\mathbf{o}}$ composing the organization $\mathbf{o} \in OA$. Agents can be thought of as a locality for intentionality. In this way it is possible to distribute subgoals of $G_{\mathbf{o}}$ among the different functional areas and roles $a \in A_{\mathbf{o}}$ to decompose problems in a hierarchical way and to avoid to overburden them with too much goals. In particular, the goals G_r attributed to role $r \in RO$ represent the responsibilities which agent $b \in A$ playing that roles ($PL(r) = b$) has to fulfill.

The beliefs attributed to the organization ($B_{\mathbf{o}}$) and attributed by the organization to its components (B_m and $m \in A_{\mathbf{o}}$) represent their know how and the procedures used to achieve the goals of the organization; these beliefs are represented for example by statutes and manuals of organizations. As in case of goals, different beliefs B_a can be distributed to functional areas and roles $a \in A_{\mathbf{o}}$. In this way the organization can respect the incapsulation principle and preserve security and privacy of information, as requested by [10].

The beliefs, desires and goals of the components of an organization play also another role. They express the institutional relations among the different components: in particular, the control and communication relations among the functional areas and roles. Both issues will be addressed using the notion of *document*. Documents are the way information parcels are represented in organizations and represent also the records of decisions and information flow.

The institutional relations of control and communication among the components of an organization are defined in terms of the "counts as" relation. For Jones and Sergot [19], the "counts as" relation expresses the fact that a state of affairs or an action of an agent "is a sufficient condition to guarantee that the institution creates some (usually normative) state of affairs". As [19] suggest this relation can be considered as "constraints of (operative in) [an] institution". In Section III we propose to model "counts as" relations by means of belief rules of the socially constructed agents. They express how an organization, a functional area or a role provide an institutional classification of reality.

In an organization it is fundamental to specify how agents can control other agents by giving orders to them [10], [5]; the control is achieved by the command structure of an organization. In fact, organizations can be seen as burocracies according to [20]. Control has two dimensions: how the organization and its functional areas can pose obligations (commands) to roles, and who has the power to create these obligations (since, as organizations and their units are socially constructed agents, they do not act). For example, a production unit can decide to give a production order to its members and the decision of the production unit can be taken by a director of that unit. The basic block of control is the creation of obligations. As described in the conceptual model, an

agent can change its own mental attitudes. In particular, an organization $\mathbf{o}$ can change its desires and goals so to create a new obligation $O_{a\mathbf{o}}(x, s \mid Y)$ by means of the creation action $CR(\mathbf{o}, O_{a\mathbf{o}}(x, s \mid Y))$. It is possible to create sanction-based obligations addressed to agent $a \in A$ since the agents involved in organizations are depended on them, for example, for the fact that organizations pay them salaries and decide benefits.

The creation actions C of an organization $\mathbf{o}$ are parameters, hence they are not directly controlled by it: the organization does not act directly, but only by means of the actions of the agents composing it. Creation actions achieve their effect to introduce new obligations if some other action "counts as" a creation action for the organization: this relation is expressed by a belief rule of the organization $\mathbf{o}$, e.g., $c \rightarrow CR(\mathbf{o}, O_{a\mathbf{o}}(x, s \mid Y)) \in B_{\mathbf{o}}$. Since there is no other way for making true the creation action, only the organization itself can specify who create new obligations. In particular, $c \in X_r$ can be an action $CD(d)$ of a role $r \in RO$ of producing a document $d \in DC$: in this way the organization $\mathbf{o}$ specifies that the role r has control over some other role $a \in RO$ such that $a \in A_{\mathbf{o}}$. The document d represents the record of the exercise of the power of agent r. Also functional areas are modelled as agents in an organization: hence, the same mechanism can be used to specify that an agent r has control over role $a \in RO$, where r and a can belong to the same functional area $m \in FA$ ($\{r, a\} \subseteq A_m \cap RO$).

Since the "counts as" relation can be iterated, it is possible to specify how a role $r \in RO$ belonging to a functional area $m \in FA$ ($r \in A_m$) of an organization $\mathbf{o} \in OA$ can create an obligation $O_{a\mathbf{o}}(x, s \mid Y)$ directed to a functional area or role $a \in FA \cup RO$ directly belonging to the organization: $a \in A_{\mathbf{o}}$. This is possible since an action $c \in X_r$ of role r can count as an institutional fact $p \in I$ for the functional area m: $c \rightarrow p \in B_m$. In turn, the institutional fact p can count as the creation of an obligation $O_{a\mathbf{o}}(x, s \mid Y)$ by the organization $\mathbf{o}$: $p \rightarrow CR(b, \mathbf{o}, O_{a\mathbf{o}}(x, s \mid Y)) \in B_{\mathbf{o}}$; this obligation is directed towards agent a which belongs to the organization $\mathbf{o}$. These relations are only possible since the beliefs B_m of the functional area m are attributed to agent m by the organization $\mathbf{o}$ itself, since $m \in A_{\mathbf{o}}$. For example, a decision of the CEO counts as an obligation of the entire organization since the direction functional area to which the CEO belongs considers the CEO's decision as made by itself and the organization, in turn, considers the decision of the direction as having the obligation as a consequence. In this way, the organization, when it creates its components by attributing mental attitudes to them, at the same time, constructs its control structure.

The second issue is communication among roles. It is often claimed [10] that the organizational structure specifies the communication possibilities of agents. Agents can communicate almost by definition and standard communication languages have been defined for this aim [21]. What the organization can specify is their possibility to communicate to each other in an institutional way by means of documents; as Wooldridge *et al.* [22] claim, organizations specify "systematic institutionalized patterns of interactions".

Communication among socially constructed agents is based on the same principle as control. It relies on the fact that the beliefs of a functional area or of a role are attributed to them by the higher level socially constructed agent which they are attributed mental attitudes by. In this way we can express the fact that a document created by a role $r \in RO$ communicates some belief p to an organization or functional area $m \in OA \cup FA$ it belongs to $r \in A_m$: $CD(d) \rightarrow p \in B_m$, where $CD(d) \in X_r$ is an action creating a document $d \in DC$. This is read as the fact the action of role r "counts as" the official belief p of agent m. The document d represents the record of the communication between r and m.

Analogously, we can specify official communication among roles. A role $r \in RO$ communicates to a role $a \in RO$ that $p \in P$ if there is some action $CD(d) \in X_r$ creating a document $d \in DC$ such that $CD(d) \rightarrow p \in B_a$. Note that B_a are not the beliefs of the agent $b \in RA$ playing role a ($b = PL(a)$). Rather they are the beliefs attributed to the role by the functional area $m \in FA$: since the role a is created by the functional area m, those beliefs are attributed to a by the functional area m. When an agent $b \in RA$ which plays the role $a \in RO$ knows that document d has been created, it has to act as if it had the belief p, while it is not requested to be psychologically convinced that p is true. Otherwise agent b does not stick to its role anymore and it becomes liable to having violated its duties.

V. Summary

In this paper we propose a way to model the organizational structure of multiagent systems. Organizations are composed by functional areas and roles; functional areas, in turn, are composed by functional areas and roles. Roles are played by agents. Using the methodology of attributing mental attitudes to social entities, we show that organizations and their components can be described as agents: socially constructed agents. Since socially constructed agents are agents, they can construct, in turn, other agents which constitute their components. This strategy allows creating a decomposition structure as rich as the one in object orientation. Moreover, it allows progressively decomposing an organization in simpler agents described by beliefs and motivations to manage the complexity of a multiagent system. Finally, since agents can be subject to obligations and endowed with permissions and powers, all the social entities composing an organization can be the addressees of norms and powers; at the same time, socially constructed agents can be normative systems imposing obligations on their components, i.e., organizations can be modelled as burocracies [20].

This paper is part of a wider project modelling normative multiagent systems. In [8] we model normative systems by means of the agent metaphor: we attribute them beliefs, desires and goals: beliefs represent the constitutive rules of the organization while regulative rules, like obligations, are modelled in terms of goals of the system. In [6] we extend the model to virtual communities and we use the agent metaphor to describe local and global policies. In [9], constitutive rules

are used to define contracts and games among agents are extended to allow an agent to change the obligations enforced by the normative system. Roles have been introduced in [23]. This paper constitutes a step forward in this project in that the agent metaphor is used to explain how organizations can create other social entities like functional areas and roles and, at the same time, specify their behavior. In this way we account for their definitional dependency characteristic of social entities [24]. Our ontology of social reality is presented in [7].

Future work concerns defining the relation between roles described as agents and the agents playing those roles. Moreover, contracts, described in [9] can be introduced to regulate the possibility to create new obligations, new roles and new social entities inside an organization [10].

REFERENCES

[1] N. R. Jennings, "On agent-based software engineering," *Artificial Intelligence*, vol. 117(2), pp. 277–296, 2000.

[2] E. Yu, "Agent orientation as a modelling paradigm," *Wirtschaftsinformatik*, vol. 43(2), pp. 123–132, 2001.

[3] G. Booch, *Object-Oriented Analysis and Design with Applications*. Reading (MA): Addison-Wesley, 1988.

[4] B. Bauer, J. Muller, and J. Odell, "Agent UML: A formalism for specifying multiagent software systems," *Int. Journal of Software Engineering and Knowledge Engineering*, vol. 11(3), pp. 207–230, 2001.

[5] F. Zambonelli, N. Jennings, and M. Wooldridge, "Developing multiagent systems: The Gaia methodology," *IEEE Transactions of Software Engineering and Methodology*, vol. 12(3), pp. 317–370, 2003.

[6] G. Boella and L. van der Torre, "Local policies for the control of virtual communities," in *Procs. of IEEE/WIC WI'03*. IEEE Press, 2003, pp. 161–167.

[7] ——, "An agent oriented ontology of social reality," in *Procs. of FOIS'04*, Torino, 2004.

[8] ——, "Regulative and constitutive norms in normative multiagent systems," in *Procs. of KR'04*, 2004, pp. 255–265.

[9] ——, "Contracts as legal institutions in organizations of autonomous agents," in *Procs. of AAMAS'04*, 2004, pp. 948–955.

[10] J. Ferber, O. Gutknecht, and F. Michel, "From agents to organizations: an organizational view of multiagent systems," in *LNCS n. 2935: Procs. of AOSE'03*. Springer Verlag, 2003, pp. 214–230.

[11] M. McCallum, T. Norman, and W. Vasconcelos, "A formal model of organisations for engineering multi-agent systems," in *Procs. of CEAS Workshop at ECAI'04*, 2004.

[12] J. Searle, *The Construction of Social Reality*. New York: The Free Press, 1995.

[13] D. Dennett, *The intentional stance*. Cambridge (MA): Bradford Books/MIT Press, 1987.

[14] A. Ricci, A. Omicini, and E. Denti, "Activity theory as a framework for mas coordination," in *Procs. of ESAW'02*, 2002, pp. 96–110.

[15] O. Pacheco and J. Carmo, "A role based model of normative specification of organized collective agency and agents interaction," *Autonomous Agents and Multiagent Systems*, vol. 6, pp. 145–184, 2003.

[16] V. Dignum, J.-J. Meyer, and H. Weigand, "Towards an organizational-oriented model for agent societies using contracts," in *Procs. of AAMAS'02*. ACM Press, 2002, pp. 694–695.

[17] J. Lang, L. van der Torre, and E. Weydert, "Utilitarian desires," *Autonomous Agents and Multiagent Systems*, pp. 329–363, 2002.

[18] R. Lee, "Documentary Petri nets: A modeling representation for electronic trade procedures," in *Business Process Management, LNCS 1806*. Berlin: Springer Verlag, 2000, pp. 359–375.

[19] A. Jones and M. Sergot, "A formal characterisation of institutionalised power," *Journal of IGPL*, vol. 3, pp. 427–443, 1996.

[20] W. Ouchi, "A conceptual framework for the design of organizational control mechanisms," *Management Science*, vol. 25(9), pp. 833–848, 1979.

[21] T. W. Finin, Y. Labrou, and J. Mayfield, "KQML as an agent communication language," in *Software Agents*, J. Bradshaw, Ed. Cambridge: MIT Press, 1995.

[22] M. Wooldridge, N. Jennings, and D. Kinny, "The Gaia methodology for agent-oriented analysis and design," *Autonomous Agents and Multi-Agent Systems*, vol. 3(3), pp. 285–312, 2000.

[23] G. Boella and L. van der Torre, "Attributing mental attitudes to roles: The agent metaphor applied to organizational design," in *Procs. of ICEC'04*. IEEE Press, 2004.

[24] C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi, and N. Guarino, "Social roles and their descriptions," in *Procs. of KR'04*, 2004.

A Conceptual Framework for Self-Organising MAS

Andrea Omicini*, Alessandro Ricci*, Mirko Viroli*, Cristiano Castelfranchi†, Luca Tummolini†
*DEIS, Alma Mater Studiorum, Università di Bologna
via Venezia 52, 47023 Cesena, Italy
Email: andrea.omicini@unibo.it, mirko.viroli@unibo.it, aricci@deis.unibo.it
†Institute of Cognitive Sciences and Technologies, CNR
viale Marx 15, 00137 Roma, Italy
Email: c.castelfranchi@istc.cnr.it, tummoli@ip.rm.cnr.it

***Abstract*— In this seminal paper, we sketch a general conceptual framework for self-organising systems (SOSs) that encompasses both stigmergy and MAS coordination, and potentially promotes models of self-organisation for MASs where interaction between cognitive agents is mediated by the environment, by means of artifacts provided by the agent infrastructure. Along this line, we first introduce the notions of *Behavioural Implicit Communication* (BIC) as a generalisation of stigmergy, and of *shared environment* (s-env) as a MAS environment promoting forms of observation-based coordination (such as BIC-based ones) that exploit cognitive capabilities of intelligent agents to achieve MAS self-organisation.**

I. Introduction

Self-organisation is typically associated to natural systems, where global coherent behaviour emerges from a multiplicity of local interactions between non-intelligent system components, in absence of global centralised control. For instance, physical systems like molecules of magnetic materials, biological systems like cytoskeletal filaments in cytoplasm of eukaryotic cells [1], social systems like insect societies [2], all exhibit forms of local interaction between very simple system components that result in higher-level forms of organisation, which can be reduced neither to the individual component's behaviour, nor to explicit external control or constraints over system's evolution. Self-organisation is also found in (human) social systems, where it emerges from non-directed local interactions between humans [3]. Robustness, fault-tolerance and adaptability to changes are typical features of those sorts of self-organising systems (SOSs henceforth) that computer scientists and engineers are nowadays trying to capture and bring to computational systems.

By definition, SOSs are those systems that exhibit some forms of global order (organisation, structure, architecture, ...), or direction, that emerge as the result of apparently non-ordered, non-directed local behaviour. Correspondingly, fundamental definitory features of SOSs are the lack of centralised control, and locality of interaction between components.

The very fact that natural SOSs often exhibit global "intelligent" (in a very broad sense) behaviours in spite of their non-intelligent individual components (magnetic particles, cytoskeletal filaments, ants) has led a good deal of the SOS research in computer science to focus on SOSs based of very simple software components. This is the case, for instance, of most of the literature on ant-based systems, trying to capture the principle of self-organisation by mostly focusing on the patterns of interaction between ant-like components, rather than on their inner structure and functioning, as in the case of stigmergy coordination [4].

This has changed in the last few years, with Multi-Agent Systems (MASs henceforth) taking momentum in the SOS field [5]. There, the most typical model for local interaction between components (agents) is based on direct communication: according to [6], self-organising MASs are typically driven by social interaction (communication, negotiation, coordination) among autonomous entities. This is the case, for instance, of the AMAS theory [7], where self-organisation depends on the ability of the agents to be locally "cooperative" – based on their ability to subjectively interpret interactions with other agents and the environment. Also, this corresponds to well-known patterns of self-organisation in human organisations [3].

On the other hand, when interaction among agents is mediated (so indirect, as opposed to direct interaction) by the environment, it typically happens that cognitive abilities of agents are not adequately exploited to the aim of self-organisation. According to [8, page 316], there is

> "a fundamental flaw in many studies of self-organisation: the assumption that the subunits of a self-organised system are dumb"

This is the case, for instance, of stigmergy [9] and swarm intelligence [10] applied to MAS coordination, where no use of agent cognitive capabilities is assumed to achieve self-organisation.

Given such premises, in this seminal paper we assume as our conceptual target those forms of self-organisation which are based on mediated interaction through the environment (á la stigmergy), but where intelligence of components plays a relevant role. So, we first demystify the apparent dichotomy between stigmergy coordination and social communication, showing a larger range of options: interaction between cognitive agents is not always reducible to communication, communication is not always explicit, and stigmergy (once properly defined [11]) does not exhaust the whole range of interaction through the environment. This is achieved by adopting the theory of *Behavioural Implicit Communication* (BIC), which models a wide range of social behaviours, and works as a critical decentralised coordination mechanism which is mainly

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

responsible for social order in human societies [11]. Such a mechanism is shared with animal societies, where it takes the form of stigmergy (which can then be thought as a BIC sub-category), and in the context of MAS provides a more comprehensive theory for self-organisation based on local interactions mediated by the environment that also covers cognitive agents.

Then, we focus on the environmental properties that enable BIC, and devise out the notion of *shared environment* (s-env) as a MAS environment promoting forms of observation-based coordination (such as BIC-based ones) that exploit cognitive capabilities of intelligent agents to achieve MAS self-organisation. In particular, the environment should support observability of agent's behaviour, and enable awareness of observation, through suitably-designed MAS infrastructures. Along this line, a formal model for MAS encompassing both BIC and s-env is introduced, that works as a model for MAS infrastructures enabling and promoting advanced forms of self-organisation for MAS based on cognitive agents, where agents interact through suitable abstractions provided by the infrastructure.

Some meaningful examples are finally discussed, that show how forms of self-organisation can emerge in MASs based on cognitive agents by exploiting the observability features provided by shared environments, focusing in particular on the BIC approach.

II. Self-Organisation through Behavioural Implicit Communication

A. Interaction, Communication, Observation

In this section we briefly introduce various kind of interaction which can be found in complex systems, remarking in particular the relevance of indirect interaction and implicit communication – based on observation and awareness – as far as coordination and self-organisation activities are concerned.

Forms of indirect interaction are pervasive in complex systems, in particular in systemic contexts where systems take the form of structured societies with an explicit organisation, with some cooperative activities enacted for achieving systemic goals. In such contexts, in order to scale with activity complexity, sorts of *mediating artifacts* are shared and exploited to enable and ease interaction among the components. Mediating artifacts of different kind can be identified easily in human society, designed and exploited to support coordination in social activities, and in particular in the context of cooperative work: examples are blackboards, form sheets, but also protocols and norms. Mediation is well focused by some theories such as Activity Theory [12] and Distributed Cognition, [13] adopted in the context of CSCW and HCI, exploring how to shape the environment in terms of mediating artifacts in order to better support cooperative work among individuals. Stigmergy is another well-known form of indirect interaction, exploiting directly the environment as mediating artifact: individuals interact by exploiting shared environmental structures and mechanisms to store and sense kind of signs (such as pheromones in the case of ant-based systems), and processes transforming them (such as evaporation/aggregation of pheromones) [2].

With respect to interaction, communication adds *intentionality*. A famous claim of the Palo Alto psychotherapy school says that "any behaviour is communication" [14]: more generally, we consider communication as any process involving an intentional transfer of information from an agent X (sender) to an agent Y (receiver), where X is *aimed at* informing Y. Agent X's behaviour has the goal or the function of informing agent Y. Agent X is executing a certain action "in order" to have other agents receiving a message and updating their beliefs or epistemic state. Communication is an intentional or functional notion in the sense that it is always goal oriented such that a behaviour is selected also for its communicative effect[1]. In the context of cognitive MAS – composed by intelligent agents – explicit types of (high level) communication are typically adopted for supporting coordination and self-organisation, mainly exploiting common semantics and ontologies.

However, in complex societies explicit communication is only part of the story: not all kinds of communication exploit codified (and hence rigid) actions. Humans and animals are for instance able to communicate also without a *predefined* conventional language, by observing their normal behaviour and practical actions. More generally, also forms of *implicit* communication play a key role as kind of interaction. Looking to societies of individuals provided with cognitive capabilities (humans, agents, ...), *observation* and *awareness* can be counted among the main basic mechanisms that enable forms of implicit communication, which allows for coordination and autonomous organisation activities. An agent's behaviour could be observed by another agent, and interpreted / used as information by the observing agent; but also, being aware to be observed, an agent could use its behaviour as a means to communicate.

So, our claim here is that implicit communication – based on observation and awareness – can be very effective as basic brick to build flexible coordination and self-organisation in the context of artificial societies, composed by cognitive agents. While we agree with [15] that coordination is a causal process of correlation between agents' actions typically involving an information flow between an agent and its environment, we do not consider always this flow as a process of communication. Consider a case where an hostile agent, whose actions are "observable", is entering a MAS. If another agent becomes aware of his presence, can observe him, should we say that the hostile agent is communicating his position? Or, differently, is the escaping prey communicating to the predator her movements? Also, even if an agent's perception of the action of another

[1] An agent's behaviour can be goal oriented for different reasons. An intentional agent (i.e. a BDI agent) is a goal governed agent (the goal is internally represented) which instantiates a communicative plan to reach the goal that another agent is informed about something. However, also simple reactive agents (i.e. insect-like) can act purposively (hence can communicate) if their behaviour has been shaped by natural or artificial selection, by reinforcement learning or by design (in the interest of the agent itself). In these latter cases the behaviour has the *function* of communicating in the sense that it has been selected *because of* a certain communicative effect.

agent is necessary implemented as information transition from a sender to a receiver, this implementation of interaction should not be necessarily considered as "communication" and the passed information should not be always labelled as a "message". From the external viewpoint of the designer a message passing of this sort is designed in order to inform the agent who is observing. However from the viewpoint of the agent a simple perception is not necessarily communication.

With respect to existing approaches on self-organisation using intelligent agents (such the AMAS approach [7]), we do not adopt direct communication as the main form of interaction, instead we aim at exploring implicit communication as a form of indirect interaction, based on observation and awareness as its basic bricks. With respect to existing approaches based on indirect interaction – such as stigmergy or computational fields [16] – we aim at considering societies composed by individuals with high level cognitive capabilities able to observe and reason about observations and actions.

B. Behavioural Implicit Communication

In cognitive MAS, communication is normally conceived as implemented through specialised actions such as speech acts defined in the FIPA ACL protocol [17]. Such protocols are inspired by natural language or expressive signals where meaning is associated to a specific action by convention.

Here we are interested in the case where the agent is aware of being observed (other agents believe that he is performing a given practical action) and he "intends that" [18] the other are interpreting his action. This sort of communication without a codified action but with a communicative intention is what we intend for behavioural Implicit Communication [11]. What is relevant here is that the agent's execution plan is aimed to achieve a pragmatic goal as usual: i.e. an agent A is collecting trash to put it in a bin (as in [19]).

A general definition for BIC is: the agent (source) is performing a usual practical action α, but he also knows and lets or makes the other agent (addressee) to observe and understand such a behaviour, i.e. to capture some meaning μ from that "message", because this is part of his (motivating or non motivating) goals in performing α. To implicitly communicate, the agent should be able to contextually "use" (or learn to use or evolve to use) the *observed* executed plan also as a sign, the plan is used as a message but it is not shaped, selected, designed to be a message.

An agent B has the same goal but observing the other's action he decides to clean another side of the road. Since the agent A knows that an agent B is observing him, the practical action he is executing can be used *also* as a message to B such as "I am cleaning here". Such a possibility can lead agents to avoid a specific negotiation process for task allocation and can finally evolve in an implicit agreement in what to do.

Three different conditions are necessary to support such a form of communication.

- The first is relative to environmental properties. The "observability" of the practical actions and of their traces is a property of the environment where agents live, one environment can "enable" the visibility of the others while another can "constrain" it, like sunny or foggy days affect our perception. An environment could also enable an agent to make himself observable or on the contrary to hide his presence on purpose.
- The second is related to the capacity of agents to understand and interpret (or to learn an appropriate reaction to) a practical action. A usual practical action can be a message when an agent knows the way others will understand his behaviour. The most basic message will be that the agent is doing the action α. More sophisticated form would imply the ability to derive pragmatic inference from it (what is the goal of doing? What can be implied?).
- The third condition is that the agent should be able to understand (and observe) the effect that his actions has on the others so that he can begin acting in the usual way *also* because the other understand it and react appropriately.

behavioural Implicit Communication is in this sense a parasitical form of communication that exploits a given level of visibility and the capacity of the others to categorise or react to his behaviour.

So, BIC can be considered a generalisation of stigmergy. The need for an environment for a MAS is often associated with the goal of implementing stigmergy as decentralised coordination mechanism. Besides, being *the production of a certain behaviour as a consequence of the effects produced in the local environment by previous behaviour* or *indirect communication through the environment* [4], stigmergy seems very similar to the form of communication we are arguing for.

However these general accepted definitions make the phenomenon too broad. It is too broad because it is unable to distinguish between the communication and the signification processes. As we have seen in 2.1 we do not want to consider the hostile agent's actions or the escaping prey as communicative actions notwithstanding that the effects of their actions elicit and influence the actions of other agents. Besides, every form of communication is mediated by the environment exploiting some environmental channel (i.e. air).

As in BIC, real stigmergic communication does not exploit any *specialised communicative* action but just usual practical actions (i.e. the nest building actions). In fact we consider stigmergy as a subcategory of BIC, being communication via long term *traces*, physical *practical* outcomes, *useful* environment modifications which preserve their practical end but acquire a communicative function. We restrict stigmergy to a special form of BIC where the addressee does not perceive the *behaviour* (during its performance) but perceives other *post-hoc traces* and outcomes of it.

Usually stigmergy is advocated as a coordination mechanisms that can achieve very sophisticated forms of organisation without the need for intelligent behaviour. However there also exist interesting forms of stigmergic communication at the intentional level. Consider a sergeant that – while crossing a mined ground – says to his soldiers: "walk on my prints!".

From that very moment any print is a mere consequence of a step, plus a stigmergic (descriptive "here I put my foot" and prescriptive "put your foot here!") message to the followers.

C. Forms of Observation-based Coordination

Coordination is that additional part or aspect of the activity of an agent specifically devoted to deal and cope with the dynamic environmental interferences, either positive or negative, i.e. with opportunities and dangers/obstacles [20]. Coordination can either be non social as when an agent coordinate with a moving object. The conceptual framework introduced so far makes it possible to frame some basic forms of coordination in terms of observation and awareness, which will be the key for enabling self-organisation of systems:

- *Unilateral* — *X* intends to coordinate with *Y* by observing *Y*'s actions.
- *Bilateral* — In this case we have the unilateral form of coordination for both agents, so: *X* intends to coordinate with *Y* by observing*Y*'s actions, and viceversa: *Y* intends to coordinate with *X* by observing *X*'s actions.
- *Unilateral-AW* — In this case we have a unilateral form of coordination, but with a first form of awareness: *X* intends to coordinate with *Y* by observing *Y*'s actions, and *Y* is aware of it (i.e. knows to be observed).
- *Reciprocal* — In this case the we have both a bilateral form of observation based coordination and awareness by both the agents: *X* intends to coordinate with *Y* by observing *Y*'s actions, *Y* is aware of it, *Y* intends to coordinate with *X* by observing *X*'s actions and *X* is aware of it.
- *Mutual* — This case extends the reciprocal form by introducing the explicit awareness of each other intention to coordinate: *X* intends to coordinate with *Y* by observing *Y*'s actions, *Y* is aware of it, *Y* intends to coordinate with *X* by observing *X*'s actions, *X* is aware of it, and *X* is aware of *Y* intention to coordinate and *Y* is aware of *X* intention to coordinate.

behavioural implicit communication is necessary for mutual coordination while it is possible and useful in the other kinds of observation-based self-organisation.

D. The Role of behavioural Implicit Communication in Dynamic Social Order

Global social order cannot be mainly created and maintained by explicit and formal norms, supported only by a centralised control, formal monitoring, reporting and surveillance protocols. Social order needs to be self-organising, spontaneous and informal, with spontaneous and decentralised forms of control and of sanction [21]. In this respect, BIC plays a crucial role. Sanctions like the act of excluding or avoiding cheaters are messages; the same for the act of exiting (quitting commitments). The act of monitoring the others' behaviour is a message for social order; the act of fulfilling commitments, obeying to norms, are all implicitly communication acts. Behavioural Implicit Communication has a privileged role also for establishing commitments, locally negotiating

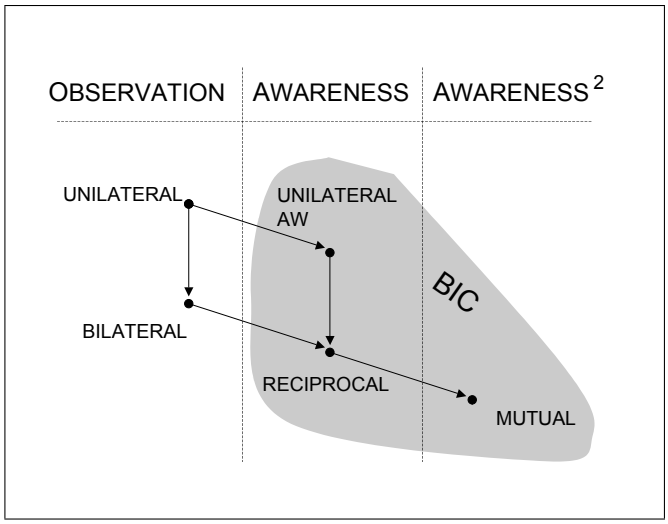


Fig. 1. Forms of coordination in relation to observation capability and awareness. Squared awareness means awareness of awareness. BIC appears with awareness, but is fully exploited when considering mutual coordination.

rules, monitoring correct behaviours, enforcing laws, letting spontaneously emerge conventions and rules of behaviours.

Accordingly, a self-organising society of artificial agents should be able to let emerge a sort of 'social contract' analogous to the one we find in human societies. Such a social contract will first be established mainly by implicit communication, then tacitly signed and renewed.

In what follows, we give some examples of this crucial role.

- *Imitation for rule propagation* — One of the main functions of imitation (i.e., repeating the observed behaviour of Y – the model) is for achieving a basic form of implicit communication. The condition is that Y (the model) can observe (be informed about) the imitative behaviour of X. By simply imitating the peer, the agent can propagate a tacit message like "I use the same behaviour as you, I accept (and spread) it as convention; I conform to it". This BIC use of imitation is probably the first form of mimetic propagation through communication and plays a key role in convention establishment. X interprets the fact that Y repeats its innovation as a confirmation of its validity (good solution) and as an agreement about doing so. Then, X will expect that Y will understand again its behaviour next time, and that Y will use again and again it, at least in the same context and interaction.
- *The fulfilment of social commitments* — Differently from the acts of conforming to already existing norms, agents (when observable) can implicitly communicate the fulfilment of their social commitments. A conforming behaviour is a form of demonstrative act primarily intended to show that one have done the expected action. Thus, the performance of the act is also aimed at informing that it has been performed.

  This is especially important when the expectation of X's act is based on obligations impinging on X, and Y is

monitoring X's non-violation of his duty. Either X is respecting a prohibition, or executing an order, or keeping a promise. A social-commitment of X to Y of doing the act, in order to be really (socially) fulfilled, requires not only that agent X performs the promised action, but also that the agent Y knows this. Thus, when X is performing the act in order to keep his promise and fulfil his commitment to Y, he also intends that Y knows this. Even in absence of explicit and specific messages, any act of social commitment fulfilment can be an implicit communication act about that fulfilment.

A second order meaning of the conforming act can also be: "I'm a respectful guy; I'm obedient; I'm trustworthy", but this inferential meaning is reached trough the first meaning "I'm respecting, obeying, keeping promises". This second order meanings can circulate and boost the reputation process that is a key informal sanction system for dynamic social order [22].

- *Local reissuing of norms* — Moreover, one of the functions of norm obedience is the confirmation of the norm itself, of the normative authority of the group, and of conformity in general. Consequently, one of the functions of norm obeying behaviours is that of informing the others about norm obedience. At least at the functional level, X's behaviour is implicit behavioural communication.

  Frequently, X either is aware of this function and collaborates on this, thus he intends to inform the others about his respect of norms, or he is worrying about social monitoring and sanctions or seeking for social approval, and he wants the others see and realise that he is obeying the norms. In both cases, his conforming behaviour is also an intentional behavioural/implicit communication to the others.

  At the collective level, when an agent respects a norm, he pays some costs for the commons and immediately moves from the mental attitude of norm addressee (which recognised and acknowledge the norm and its authority, and decided to conform to it) to the mental set of the norm issuer and controller [23]: he wants the others to respect the norm, pay their own costs and contribution to the commons.

III. A BIC-oriented Shared Environment for Self-Organisation

So, to promote advanced forms of self-organisation in MAS featuring cognitive agents, MAS environment should be shaped so as to allow for observability and awareness of agents behaviour.

Generally speaking, agents that live in a *common environment* (*c-env*) are agents whose actions and goals interfere (positively or negatively). In a pure *c-env*, agent actions and their traces are state transitions which can ease or hamper the individual agents' goals. An example is a ground that is common for different insect species but where no interspecies communication is possible. Agents can observe just the state of the environment, and then act on that basis, achieving a given self-organisation, still with no access to the actions of their peers. Even a trace is seen as part of the environment and not as a product of other agents. So, a generic property of a *c-env* is that it provides agents with the means to keep track of its state and possibly affect it.

As far as observation-based self-organisation is concerned, we here propose a stronger notion of environment, called *shared environment* (*s-env*). This is a particular case of a *c-env* that enables (i) different forms of observability of each other action executions, as well as (ii) awareness of such observability, thus supporting unilateral, bilateral, reciprocal, and mutual coordination.

A. Observability in Shared Environments

Each *s-env* is defined by the level of observability that it can afford. The level of observability is the possibility for each agent to observe another agent behaviour, namely, to be informed when another agent executes a given action. For instance, the most general kind of *s-env* can be defined by the fact that each agent *can* observe the execution of all the actions of all others agents. A prototypical model of this sort of environment is the central 'square' of a town. Other levels of observability may limit the ability of agents to observe given actions of other agents – e.g. considering sort of invisible actions – or to observe only given agents and not others – e.g. considering obstacles preventing observation.

The level of observability of an *s-env* is easily understood by a *power* relation $Pow : A \times A \times Act$, where A is the set of agents – ranged over by meta-variables x, y, and z – and Act is the set of usual practical actions which may be subject of observation through the *s-env* – ranged over by meta-variables α and β. When $\langle x, y, \alpha \rangle \in Pow$, also written $Pow(x, y, \alpha)$, it means that action $\alpha \in Act$ executed by agent y is observable by agent x through the *s-env*.[2] This means that in that *s-env*, it is structurally possible for x to observe the executions of action α by y. We naturally say that x has the role of observer agent, y that of observed agent, α that of observed action. We extend the notation for power relation using sets of agents or actions, e.g. writing $Pow(x, B, \alpha)$ with $B \subseteq A$ for $Pow(x, y, \alpha)$ holding for all $y \in B$, or $Pow(x, y, Act)$ in place of $Pow(x, y, \alpha)$ for all $\alpha \in Act$.

Pow relation can be then conceived as specifying the rules defining the set of 'opportunities and constraints' that afford and shape agents' observability within the environment. A specific rule is an opportunity or a constraint *for a specific agent* and in particular it is only relative to the agent's active goals while interacting with that environment.

Whereas relation Pow is introduced to statically describe the set of opportunities and constraints related to agents' observability, an *observation* relation Obs (a subset of Pow) has to be introduced to characterise the state of the s-env at a given time. When $Obs(x, y, \alpha)$ holds, it means that agent x

[2]Observability of an action should be intended here in its most general acceptation, that is, accounting for all the properties that need to be observed – so, not only the executing agent, but also time of execution, information possibly carried along, and so on.

is actually observing executions of action α by agent y. That is, $Obs(x, y, \alpha)$ means that an execution of action α by agent y will be perceived by x. Hence, notice that we differentiate between the potential ability to observe, which is a typical property of the environment where the agents live in, and the actual observability, which might be driven by the explicit motivation of agents. Indeed, since $Obs \subseteq Pow$, observation is constrained by the level of observability featured by the *s-env*.

The meaning of the observation relation can be understood by taking into account the agent's viewpoint over observation. We first introduce the concept of agent *epistemic state (ES)*, representing the beliefs the agent has because of his observation role. The ES of an agent x includes its *environmental knowledge* about observation, which is then given by information *(i)* on the agents he is observing, *(ii)* on the agents that are observing him, and *(iii)* on the action executions that he is observing.

The first two kinds of knowledge can be addressed by supposing the agent may, at a given time, have some knowledge about the current state of relation Obs. In particular, write $B_z obs(x, y, \alpha)$ for agent z believing that x is observing, from that time on, executions of action α performed by z. On the other hand, to represent the third kind of knowledge, we write $B_z(done(y, \alpha))$, meaning that agent z believes that y has executed action α.[3]

B. Epistemic Actions

The epistemic state of an agent evolves through *epistemic actions*, which are actions aimed at acquiring knowledge from the environment [25]. Such an aim is expressed as an agent intention: accordingly, we also define the concept of *motivational state* (MS) of an agent, which includes all the intentions an agent has at a given time. Then, an epistemic action is fired by an agent intention, by which the *s-env* reacts updating its state as well as the epistemic state of the agent. So, we have different kinds of epistemic actions, each fired by a different motivation: they are used e.g. to know who is observing who, to have an agent observing another, to avoid an agent observing another, and so on.

A first case of epistemic action is used by the agent which is willing to know whether he is observing another agent, whether another agent is observing him, or generally, whether an agent x is observing actions α of an agent y. So, suppose the MS of z includes intention $I_z check(x, y, \alpha)$, which means that agent z intends to know whether x observes executions of α by y. Then, eventually an epistemic action is executed by which the ES of agent z will include the belief about whether $Obs(x, y, \alpha)$ holds or not.

Similarly, an agent may have the intention $I_x obs(x, y, \alpha)$ in exploiting the observability power of the environment to observe y's actions α. When such an intention appears in the MS of agent x, the s-env conceptually intercepts it and enacts the corresponding observations. This means that (i) the s-env adds $B_x obs(x, y, \alpha)$ to the agent's epistemic state (agent x knows that he is observing actions by agent y), and (ii) relation Obs is added the rule $Obs(x, y, \alpha)$ (the s-env makes agent x observing actions α by agent y). In other words, we can think that the appearance of an intention in the motivation state of the agent causes the execution of an epistemic action toward the environment, enabling agent observations.

Similarly, an agent may want to stop observing actions. When the intention $I_x drop(x, y, \alpha)$ appears in the agent motivational state, the effects of $obs(x, y, \alpha)$ are reversed.

Now we are ready to link the MS state of the agent, Obs rules and the ES state of the agent. According to the semantics of the actions, the execution of an action α by agent y (written $done(y, \alpha)$) causes the creation of a new belief $B_x done(y, \alpha)$ in the epistemic state of all the agents x of the environment such that $Obs(x, y, \alpha)$ holds.

C. Formal Model

To make our arguments more precise we introduce a formal framework to describe the notions of ES, MS, epistemic actions, and observation in a precise way, which is meant to serve as an actual design for an infrastructure providing a *s-env*. In particular, we provide a syntax and an operational semantics for modelling MAS according to the conceptual framework defined in previous sections.

Throughout this model, composition operator $||$ is assumed to be commutative, associative, to absorb the empty configuration 0, and to consume multiple copies of the same element – that is, $x \,||\, x \equiv x$. Accordingly, any grammar definition of the kind

$$X ::= 0 \mid x_1 \mid \ \ldots \ \mid x_n \mid X \,||\, X$$

defines elements of the syntactic category X as compositions (without repetitions) of terms $x_1, \ldots, x_n$. Given one such composition X, we write $x_j \in X$ and $x_j \notin X$ with the obvious meaning. The syntax of MAS configurations is reported in Figure 2.

Metavariable S ranges over configurations of the MAS, which at our abstraction level are simple compositions of agent configurations (ES and MS) and environment configurations (Pow and Obs). Environment configurations are composition of terms, each denoting either the power of agent x to observe action α executed by agent y ($Pow(x, y, \alpha)$), or the fact that the environment is making x observe actions α executed by agent y ($Obs(x, y, \alpha)$). Agent configurations are compositions of mental properties, namely beliefs (B) and intentions (I) qualified by the agent x, and about a formula ϕ. As described above, these properties are used to represent the ES and MS of agent x, namely its knowledge and motivations. Notice that we model a MAS configuration as a composition of both agent and environment properties without a neat separation: in fact, at our level of abstraction such a distinction is not necessary, for epistemic actions involving both kinds of properties in a uniform way.

[3]The syntax we introduced clearly reminds standard modal logics for beliefs as in [24], however, it is not our goal here to introduce any logics for agent reasoning. This is why we still refer to the weaker notion of epistemic state instead of beliefs state – and motivational state instead of intentional state as described below.

S	$::=$	$0 \mid A \mid E \mid S \,\|\|\, S$	MAS configuration
E	$::=$	0	environment configuration
	$\mid$	$Pow(x, y, \alpha)$	x has the power to observe y's α
	$\mid$	$Obs(x, y, \alpha)$	x is observing y's α
	$\mid$	$E \,\|\|\, E$	composition
A	$::=$	0	agent configuration
	$\mid$	$B_x\phi$	belief of x
	$\mid$	$I_x\phi$	intention of x
	$\mid$	$A \,\|\|\, A$	composition
ϕ	$::=$		formulas
		$obs(x, y, \alpha)$	x is observing y's α
	$\mid$	$coord(x, y, \alpha)$	x coordinates with y through α
	$\mid$	$check(x, y, \alpha)$	check whether x is observing y's α
	$\mid$	$drop(x, y, \alpha)$	prevent x from observing y's α
	$\mid$	$done(x, \alpha)$	x executes actions α
	$\mid$	$\neg\phi \mid I_x\phi \mid B_x\phi$	structured formulas

Fig. 2. Syntax of MAS configurations.

Elements ϕ are formulas which can be believed and/or intended by an agent. Atomic formulas are: *(i)* $obs(x, y, \alpha)$, used to express that x is observing executions of α by y, *(ii)* $coord(x, y, \alpha)$, used to express that x coordinates its behaviour with y by observing executions of α, *(iii)* $check(x, y, \alpha)$, used to check if x is observing executions of α by y, *(iv)* $drop(x, y, \alpha)$, used to prevent x from observing executions of α by y, and *(v)* $done(x, \alpha)$, used to express that x executes/has executed α. Moreover, formulas can be structured ones: $\neg\phi$ expresses negation of ϕ, $I_x\phi$ and $B_x\phi$ that agent x intends/believe ϕ. A number of assumptions on such formulas are clearly to be made as usual, e.g. that $\neg\neg\phi \equiv \phi$ or $B_x\phi \equiv B_xB_x\phi$. This amounts to define a logics for beliefs and intentions: however, this aspect can be treated in a fairly standard way, therefore its details are not reported for they play no significant role in this paper – they are more about agent internal architecture rather than agent interaction through the environment.

On top of this syntax for MAS configurations, we introduce an operational semantics, describing what are the allowed evolutions of such configurations. This describes the dynamic aspects of our model, providing details on preconditions and effects to epistemic actions and observation in general. As usual [26], operational semantics is defined by a set of rewrite rules, reported in Figure 3. Each rule defines a MAS configuration to be rewritten as interaction of the agent with the *s-env* occurs: the left-hand side reports preconditions, the right-hand effects, and the above part (when present) further preconditions for the applicability of the rule.

Rule [CHECK] says that if agent z intends to check/know if x is observing y's action α, and this is the case, then such an intention will be turned into a positive belief. Dually, rule [N-CHECK] deals with the case where this is not the case ($Obs(x, y, \alpha)$ does not occur in the system configuration), so that z will believe that $obs(x, y, \alpha)$ does not hold.

Rule [DROP-Y] says that if agent z knows that x is observing y's action α (which is the case) and wants to stop him, term $Obs(x, y, \alpha)$ is dropped from the environment and z's belief is updated correspondingly. By rule [DROP-N] we deal with the similar case, but supposing the agent had a wrong belief (x was not actually observing y's actions α), which is dealt with trivially.

Rule [ASK] is about agent z willing that x observes y's actions α: if this is allowed ($Pow(x, y, \alpha)$), x's beliefs will be updated along with the environment state.

Rule [OBS-R] and [OBS-F] recursively define how the environment broadcasts information about an action to all the observers. When agent x wants to execute α, each observer y (rule [OBS-R]) will be recursively added the belief $B_y done(x, \alpha)$: when none needs to be managed, x intention can simply become a fact, that is, he will believe the action to be executed ([OBS-F]).

The final, trivial rule [AGENT] is used to represent the fact that at any given time some agent configuration can change autonomously, thus modelling any belief revision or intention scheduling.

Notice that formulas $B_z coord(x, y, \alpha)$ or $I_z coord(x, y, \alpha)$ never appear in this semantics. This is because the fact that an agent coordinates its behaviour with another is not an aspect influencing/influenced by the environment: it is rather a mental property characterising the forms of observation-based coordination an agent participates to thanks to the *s-env* support.

D. Formalising Observation-based Coordination

We put to test our formal framework showing how the forms of coordination devised in Subsection II-C can be represented through our syntax.

$$\frac{Obs(x,y,\alpha) \in S}{I_z check(x,y,\alpha) \,||\, S \rightarrow B_z obs(x,y,\alpha) \,||\, S} \quad \text{[CHECK]}$$

$$\frac{Obs(x,y,\alpha) \notin S}{I_z check(x,y,\alpha) \,||\, S \rightarrow B_z \neg obs(x,y,\alpha) \,||\, S} \quad \text{[N-CHECK]}$$

$$\frac{-}{I_z drop(x,y,\alpha) \,||\, B_z obs(x,y,\alpha) \,||\, Obs(x,y,\alpha) \,||\, S \rightarrow B_z \neg obs(x,y,\alpha) \,||\, S} \quad \text{[DROP-Y]}$$

$$\frac{Obs(x,y,\alpha) \notin S}{I_z drop(x,y,\alpha) \,||\, B_z obs(x,y,\alpha) \,||\, S \rightarrow B_z \neg obs(x,y,\alpha) \,||\, S} \quad \text{[DROP-N]}$$

$$\frac{-}{I_z obs(x,y,\alpha) \,||\, Pow(x,y,\alpha) \,||\, S \rightarrow B_z obs(x,y,\alpha) \,||\, Pow(x,y,\alpha) \,||\, Obs(x,y,\alpha) \,||\, S} \quad \text{[ASK]}$$

$$\frac{I_x done(x,\alpha) \,||\, S \rightarrow I_x done(x,\alpha) \,||\, S'}{I_x done(x,\alpha) \,||\, Obs(y,x,\alpha) \,||\, S \rightarrow I_x done(x,\alpha) \,||\, Obs(y,x,\alpha) \,||\, B_y done(x,\alpha) \,||\, S'} \quad \text{[OBS-R]}$$

$$\frac{Obs(y,x,\alpha) \notin S}{I_x done(x,\alpha) \,||\, S \rightarrow B_x done(x,\alpha) \,||\, S} \quad \text{[OBS-F]}$$

$$\frac{-}{A \,||\, S \rightarrow A' \,||\, S} \quad \text{[AGENT]}$$

Fig. 3. Operational Semantics of Agent Configurations.

Given two agents x and y, an action α, and the system configuration S we introduce the following predicates:

- Unilateral

$$Uni(x,y,\alpha,S) \triangleq Obs(x,y,\alpha) \in S \ \wedge \ I_x coord(x,y,\alpha)$$

Agent x is in unilateral coordination with y (in system S, through action α), if he is observing y's actions α and he intends to coordinate with y through such actions.

- Unilateral with Awareness

$$UniAW(x,y,\alpha,S) \triangleq Uni(x,y,\alpha,S) \ \wedge \ B_y obs(x,y,\alpha) \in S$$

The form of coordination is unilateral with awareness if x is in unilateral coordination with y and if y knows to be observed by x.

- Bilateral

$$Bi(x,y,\alpha,S) \triangleq Uni(x,y,\alpha,S) \ \wedge \ Uni(y,x,\alpha,S)$$

x and y are in bilateral coordination if they are both in unilateral coordination with each other.

- Reciprocal

$$Rec(x,y,\alpha,S) \triangleq UniAW(x,y,\alpha,S) \ \wedge \ UniAW(y,x,\alpha,S)$$

x and y are in reciprocal coordination if they are both in unilateral coordination with awareness.

- Mutual

$$Mut(x,y,\alpha,S) \triangleq Rec(x,y,\alpha,S) \ \wedge \ B_x I_y coord(y,x,\alpha) \ \wedge \ B_y I_x coord(x,y,\alpha)$$

Finally, x and y are in mutual coordination if they are in reciprocal coordination and, moreover, they both know that the other agent intends to coordinate through the observed action α.

IV. Conclusions

In this paper we focused on some properties of MAS infrastructures for cognitive agents supporting forms of self-organisation, based on the BIC theory. Even though not dealing with internal aspects of agents, we consider agents provided with some cognitive capabilities, differently from current environment-based approach in self-organisation, typically based on reactive agents (e.g. ants).

MASs built on top of a BIC-oriented infrastructure exhibit the basic enabling principles which typically characterise self-organisation:

- *Local interaction* — In the framework there is an explicit notion of locality of interaction: agent observability and awareness are related to a notion of locality that is dynamic, depending on the adopted topology, which is defined by the infrastructure and can be changed over

time. The enacting of $Pow(x, y, \alpha)$ rules by MAS infrastructure implicitly defines such a topology in terms of what actions can be observed by whom at any time.

- *Decentralised control* — Control is decentralised and encapsulated in cognitive agents, which exhibits an autonomous behaviour with respect to the environment.
- *Emergent patterns* — Patterns of MAS self-organisation emerge from agent interacting through a suitably shaped environment, by exploiting observation capabilities provided by the infrastructure.

Besides these basic principles, other interesting aspects that are often considered when dealing with self-organising systems can be re-casted in our framework:

- *Individual-based models* — Individual-based models are currently considered the right approach for the quantitative and qualitative study of SOS [26], tracking each individual state and behaviour. The model presented in the paper is indeed individual-based, since a MAS is composed by individual agents with their own cognitive state and behaviour, eventually playing different kinds of roles inside the system.
- *Openness* (in the thermodynamic acceptation) — In order to keep thermodynamic systems self-organised there must be a continuous flow of energy from the environment: our MASs are characterised by an analogous form of openness, since agents are meant to exchange information within the environment – which is outside the system – by means of perceptions and actions.
- *Non-linearity and feedbacks* — Non-linearity and (positive) feedback that typically characterise SOS can be obtained with forms of mutual coordination, realising kind of non-linear chains of observation and awareness.
- *Dissipative structures* — In our framework, infrastructure structure / services exploited by agents for enhancing their observation / awareness capability can play the role of dissipative structures, typically considered in SOS [27] as a key to export entropy out of the system.

Most of complex system scenarios calls for systems with self-organising capabilities but immersed in an environment that can have (social) norms and constraints, typically specified at design time and that enforced at runtime. We think that in order to cope with such (apparently conflicting) aspects, MAS infrastructure can play a key role [28]. On the one side, it can provide mechanisms and abstractions enabling forms of interaction enabling MAS self-organisation – thus promoting system's unpredictability. On the other side, such mechanisms and abstractions can play a regulatory role, by enforcing laws and norms constraining and ruling agent interaction space – thus promoting system's predictability. We believe that our approach will support MAS engineers in finding the most suitable balance between such a dilemma of "global vs. local control" in MASs.

REFERENCES

[1] F. Nedelec, T. Surrey, and É. Karsenti, "Self-organisation and forces in the microtubule cytoskeleton," *Current Opinion in Cell Biology*, vol. 15, no. 2, pp. 118–124, Feb. 2003.

[2] P.-P. Grassé, "La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie: essai d'interpretation des termites constructeurs," *Insectes Sociaux*, vol. 6, pp. 41–83, 1959.

[3] H. Haken, *Synergetics: An Introduction. Nonequilibrium Phase Transition and Self-Organization in Physics, Chemistry, and Biology*. Springer-Verlag, 1977.

[4] O. Holland and C. Melhuis, "Stigmergy, self-organization, and sorting in collective robotics," *Artificial Life*, vol. 5, no. 2, pp. 173–202, 1999.

[5] G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Eds., *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, ser. LNAI, vol. 2977. Springer-Verlag, May 2004. [Online]. Available: http://www.springer.de/cgi/svcat/search_book.pl?isbn=3-540-21201-9

[6] G. Di Marzo Serugendo, N. Foukia, S. Hassas, A. Karageorgos, S. Kouadri Mostéfaoui, O. F. Rana, M. Ulieru, P. Valckenaers, and C. Van Aart, "Self-organisation: Paradigms and applications," in *Engineering Self-Organising Systems*, ser. LNAI, G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Eds. Springer-Verlag, May 2004, vol. 2977, pp. 1–19. [Online]. Available: http://www.springer.de/cgi/svcat/search_book.pl?isbn=3-540-21201-9

[7] D. Capera, M.-P. Gleizes, and P. Glize, "Self-organizing agents for mechanical design," in *Engineering Self-Organising Systems*, ser. LNAI, G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Eds. Springer-Verlag, May 2004, vol. 2977, pp. 169–185. [Online]. Available: http://www.springer.de/cgi/svcat/search_book.pl?isbn=3-540-21201-9

[8] T. D. Seeley, "When is self-organization used in biological systems?" *Biological Bulletin*, vol. 202, pp. 314–318, 2002.

[9] Hadeli, P. Valckenaers, C. Zamfirescu, H. Van Brussel, B. Saint Germain, T. Hoelvoet, and E. Steegmans, "Self-organising in multi-agent coordination and control using stigmergy," in *Engineering Self-Organising Systems*, ser. LNAI, G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Eds. Springer-Verlag, May 2004, vol. 2977, pp. 105–123. [Online]. Available: http://www.springer.de/cgi/svcat/search_book.pl?isbn=3-540-21201-9

[10] R. Tolksdorf and R. Menezes, "Using swarm intelligence in Linda," in *Engineering Societies in the Agents World IV*, ser. LNAI, A. Omicini, P. Petta, and J. Pitt, Eds. Springer-Verlag, June 2004, vol. 3071, pp. 49–65, 4th International Workshop (ESAW 2003), London, UK, 29–31 Oct. 2003. Revised Selected and Invited Papers. [Online]. Available: http://www.springer.de/cgi/svcat/search_book.pl?isbn=3-540-22231-6

[11] C. Castelfranchi, "When doing is saying – the theory of behavioral implicit communication," 2003, draft. [Online]. Available: http://www.istc.cnr.it/doc/62a_716p_WhenDoingIsSaying.rtf

[12] B. Nardi, Ed., *Context and Consciousness: Activity Theory and Human Computer Interaction*. Cambridge, MA: MIT Press, 1996.

[13] E. Hutchins, *Cognition in the Wild*. Cambridge, MA: MIT Press, 1995.

[14] P. Watzlavich, J. Beavin Bavelas, and D. D. Jackson, *Pragmatics of Human Communication: A Study of Interactional Patterns, Pathologies, and Paradoxes*. New York: W.W. Norton & Co., 1967.

[15] H. V. D. Parunak, S. Brueckner, M. Fleischer, and J. Odell, "A design taxonomy of multi-agent interactions," in *Agent-Oriented Software Engineering IV*, ser. LNCS, P. Giorgini, J. Müller, and J. Odell, Eds. Springer-Verlag, 2004, pp. 123–137, 4th International Workshop (AOSE 2003), Melbourne, Australia, 15 July 2003, Revised Papers.

[16] M. Mamei and F. Zambonelli, "Self-organization in multi-agents systems: A middelware approach," in *Engineering Self-Organising Systems*, ser. LNAI, G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Eds. Springer-Verlag, May 2004, vol. 2977, pp. 233–248. [Online]. Available: http://www.springer.de/cgi/svcat/search_book.pl?isbn=3-540-21201-9

[17] FIPA, *FIPA Communicative Act Library Specification*, 2000, `http://www.fipa.org`.

[18] B. J. Grosz and S. Kraus, "Collaborative plans for complex group action," *Artificial Intelligence*, vol. 86, pp. 269–357, 1996.

[19] A. S. Rao, "A unified view of plans as recipes," in *Contemporary Action Theory*, G. Hölmstrom-Hintikka and R. Tuomela, Eds. Kluwer Academic Publishers, 1997, vol. 2: Social Action.

[20] C. Castelfranchi, "Modelling social action for AI agents," *Artificial Intelligence*, vol. 103, pp. 157–182, 1998.

[21] ——, "Engineering social order," in *Engineering Societies in the Agents World*, ser. LNAI, vol. 1972. Springer-Verlag, Dec. 2000, pp. 1–18, 1st

International Workshop (ESAW'00), Berlin (Germany), 21 Aug. 2000, Revised Papers.

[22] R. Conte and M. Paolucci, *Reputation in Artificial Societies. Social Beliefs for Social Order*. Boston: Kluwer Academic Publisher, 2002.

[23] R. Conte and C. Castelfranchi, *Cognitive and Social Action*. London: University College of London Press, 1995.

[24] M. D. Sadek, "A study in the logic of intention," in *3rd Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA, USA, 1992, pp. 462–473.

[25] C. Castelfranchi and E. Lorini, "Cognitive anatomy and functions of expectations," in *Cognitive Modeling of Agents and Multi-Agent Interactions*, 2003, workshop at IJCAI 2003. Proceedings.

[26] G. Plotkin, "A structural approach to operational semantics," Department of Computer Science, AArhus University, Denmark, Tech. Rep. DAIMI FN-19, 1991.

[27] G. Nicolis and I. Prigogine, *Exploring Complexity: An Introduction*. W.H. Freeman & Co., 1989.

[28] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini, "Coordination artifacts: Environment-based coordination for intelligent agents," in *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., vol. 1. New York, USA: ACM, 19–23 July 2004, pp. 286–293. [Online]. Available: http://portal.acm.org/citation.cfm?id=1018409.1018752

Engineering Trust in Complex System through Mediating Infrastructures

Alessandro Ricci
DEIS
Università di Bologna – Sede di Cesena
via Venezia 52, 47023 Cesena (FC), Italy
Email: aricci@deis.unibo.it

Andrea Omicini
DEIS
Università di Bologna – Sede di Cesena
via Venezia 52, 47023 Cesena (FC), Italy
Email: aomicini@deis.unibo.it

***Abstract*— Starting from the many research results on trust in state-of-the-art literature, we first point out some open problems related to trust in multiagent systems (MAS), focussing in particular on the issue of the engineering of agent societies, and on the role of agent infrastructures. Then, we discuss two infrastructural abstractions – coordination artifacts, and agent coordination contexts –, and show how they can be exploited for modelling and engineering trust within MAS.**

I. Trust in Complex System Engineering

One of the most relevant problems of our contemporary society is its dependency on information technologies systems which are getting more and more complex and difficult to control. Accordingly, the problem of *trust* between humans and information technology comes out from the inability to provide simple and accessible models to make systems behaviour somehow understandable and predictable for the users themselves. This does not affect only end-users, but also (and, in some sense, mostly) the engineers and developers that are responsible of system design and construction. In particular, the difficulty of conceiving trustworthy models for the engineering of complex and complex systems emphasises the fact that the impetuous technological progress chararcterising our society is a necessary but not sufficient condition for the widespread generation and adoption of innovative processes.

As a simple example, the possibility of checking system behaviour and functioning by inspecting its source code once it is made available (the myth of Open Source wave) is simply not feasible, according to current state-of-the-art models and tools. Turning from the notion of ”program” to the notion of ”system” involves a paradigm shift: the behaviour of a program (as a sequence of instructions of certain (virtual) machines) is, in principle, inspectable, understandable and predictable. Instead, it is typically not possible to formalise nor to have a complete understanding of the behaviour of a software system (as a collection of heterogeneous and independent components interacting in a distributed environment) [23].

According to the current major research lines, the complexity of modern and forthcoming systems can be managed only by introducing models that account for *societies* of heterogeneous actors (objects, components, agents, processes, humans..) which interact and communicate in dynamic and unpredictable environments: at least, this is a suitable model for current web-based systems. So, trust is one of the most important social issues for human as well as for artificial systems: this is evident if we consider scenarios such as e-commerce or e-government, where the edge between human and artificial societies tends to blur: these contexts make it clear that all the social issues involved in human societies, trust in primis, must be faced also in the construction of complex artificial systems.

Accordingly, the applicability (reuse) of models for human societies in the context of artificial systems is a primary issue, exploiting, for instance, the explaination and prediction capabilities of theories both as a scientific and engineering tool to validate engineering constraints of systems [22]. This is especially important if we aim at considering trust beyond conceptually simple applications such as digital signature or e-commerce transactions, facing contexts where trust matters not only for a human actor (users or engineers) w.r.t. the system, but for every human and artificial actor that constitute system society.

Trust is then recognised as a fundamental aspect of engineering systems with MAS: however, trust characterisation and models as found in state-of-the-art literature do not cover some issues which we consider fundamental for the engineering of agent societies. First, a well-defined notion of social trust is missing: few approaches deal with an infrastructure (and then social, objective) support to trust, being mostly focussed on the subjective perception and modelling of trust by individuals. Even the approaches considering forms of social trust (referred as system-level trust in literature) fail to provide a comprehensive model of the trust phenomenon at the social level (including the notion of observation, traceability of actions, etc), limiting their approach to provide some specific mechanisms. Then, trust frameworks (models and mechanisms) are focussed essentially on the behaviour of a individual component (agent), and no account is given for characterising trust at a systemic level, i.e. trust in a group or society of agents in being able to achieve their social tasks. Linked to this point, current models and mechanisms are developed mostly in competitive contexts, where agents are totally self-interested; instead we are interested in modelling trust in systems where agent cooperatively work for a global (system) outcome. In this case we have several points of view concerning trust: trust

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

of the users relying on a systems of cooperating agents, trust of the engineers in the system he designed, trust of the collectivity of the components (agents) with respect to a specific one, trust of an individual components (agent) of the system with respect to the collectivity.

In this paper, then, first we extend the notion of trust to consider also these issues, more related to an engineering point of view on systems and based on infrastructural support to trust. The extension will relate trust to coordination and organisation issues, as fundamental engineering dimensions of systems. Then, we show how some infrastructural abstractions recently introduced for engineering of MAS coordination and organisation – namely coordination artifacts and agent coordination contexts – can play an effective role in defining trust according to our wider vision.

The remainder of the paper is organised as follows: first, in Section II a brief account of state-of-the-art models for trust in MAS is provided; then, Section III remarks some points missing from such models, discussing a wider characterisation of trust including engineering issues. Accordingly, Section IV and Section V discuss how coordination and organisation infrastructural abstraction can play a fundamental role for characterising this enhanced notion of trust. Finally, conclusions are reported in Section VI.

II. Modelling Trust in Agent Societies

Trust has been defined in several ways in distinct domains (see [15] for a comprehensive survey, and [4] for a general description). A definition that is frequently adopted in trust models is:

> "Trust is a belief an agent has that the other party will do what it says it will (being honest and reliable) or reciprocate (being reciprocative for the common good of both), given an opportunity to defect to get higher payoff"
> (adapted from [1])

The various approaches to trust in MAS have been recently classified in two main classes, for some extend in counter-position and complimentary: *individual-level trust* and *system-level trust* [15].

Roughly speaking, in individual-level trust all the burden about trust is in charge of individual agents, and depends on their ability to model and reason about the reciprocative nature, reliability or honesty of their counter-parts. In system-level trust instead the actors in the systems are forced to be trustworthy by the *rules of the encounter* [18] (i.e. protocols, mechanisms) that regulate the systems. So the burden about trust is shifted from agents to some system support, which is realised by designing specific protocols and mechanisms of interaction (i.e. the rules of the encounter). A typical example are auctions.

So the point of view of individual-level trust accounts for an agent situated in an open environment trying to choose the most reliable interaction partner from a pool of potential agents and deliberating which strategy to adopt on it. Following the classification described in [15], trust models for individual-level can be classified in this case either *learning (evolution) based*, *reputation based* or *socio-cognitive based*. In the first, trust is viewed as an emergent property of direct interaction between self-interested agents, who are endowed with strategies that can cope with lying and non-reciprocative agents. Reputation models [19] instead enable agents to gather information in richer forms from their environment and make rational inferences from the information obtained and their counterparts. The models then specify strategies to *gather ratings* that define the trustworthiness of an agent, using relationships existing between member of the community; reasoning methods to gather information from *aggregation of ratings* retrieved from the community (borrowing the concept of social network from sociology); and mechanisms to promote ratings that *truly* describe the trustworthiness of an agent. Finally, the socio-cognitive models adopt a higher level view, modelling the subjective perception of trust in terms of cognitive constructs [3], in contrast to the quantitative view of trust which characterises previous approaches. While the first two models are all based on an assessment of the outcome of interactions between agents, the basic context for socio-cognitive approaches is that of *task delegation* where an agent x wishes to delegate a task to agent y. In doing so agent x needs to evaluate the trust it can place in y by considering the different *beliefs* it has about motivations of agent y.

In the overall, trust at the individual level concerns strategies learnt over multiple interactions, the reputation of potential interaction partners, and believed motivations and abilities regarding the interaction. Some problems affecting these approaches have been remarked in the literature: it can be computationally expensive for an agent to reason about all the different factors affecting trust in its opponents; then, agents are limited in gathering information from various sources that populate its (open) environment. Given these limitations, system-level trust approaches shift the focus on the rules of encounter so as to ensure that interaction partners are *forced* to be trustworthy. The mechanisms that dictate these rules of encounter (auctions, voting, contract-nets, market mechanisms, etc) enable agent to trust each other by virtue of the different constraints *imposed by the system*. Always following [15], these system-level mechanisms can be classified in *trustworthy interaction mechanisms*, *reputation mechanisms* and *distributed security mechanisms*. Mechanisms of the first class are adopted to prevent agents from lying or speculating while interacting (auctions are an example, see [20] for an overview); reputation mechanisms [24] make it possible to model the reputation of agents at system level, i.e. it is the system that manage the aggregation and retrieval of ratings (as opposed to reputation models which leave the task to the agents themselves). Finally, the latter class includes security mechanisms and infrastructures which are considered essential for agents to trust each other and each other communication (examples are public key encryption and certificate infrastructures)[14], [5].

According to [15], complex systems require both types of trust approach, individual- and system-level. While the

individual-level trust *models* enable agent to reason about its level of trust and of its opponents, the system-level *mechanisms* aim to ensure that opponents' actions can be trusted. It's worth noting that this dichotomy have been remarked also for another dimension focussing on interaction, i.e. coordination, where approaches have been classified as subjective (all the coordination burden on agent and their capabilities) and objective (the coordination burden in charge of abstractions provided by suitable infrastructures)[10].

III. Extending Trust for Engineering Societies

The characterisation of trust in state-of-the-art literature as described in previous section do not give emphasis enough to some issues that we consider as fundamental in the engineering of agent societies. These aspects can be summed up in the following points:

- *Social Trust* – We need to consider in a more general and systematic way the support that infrastructures can provide as a service for societies engineered on their top, beyond specific mechanisms or protocols. This accounts for generalising system-level trust approaches, devising basic abstractions and services on top of which to build trust strategies. Among such basic services, a support for *observation* and *traceability* of both agent action, and interaction among agents and agent-environment. These basic services can be suitably exploited and composed to keep track – for instance – of action history of a specific agent, making it available to some other agents, with the permissions to inspect such information. This infrastructural support is extremely effective when dealing with *open* systems, with heterogeneous agents dynamically participating to the activities of different societies and organisations: infrastructures can provide services to agent organisations to keep track and make information available about agent performance in its interaction life, acting in different contexts, as a kind of "criminal record" publicly available; thus, respecting privacy of the agent, i.e., making available only what is needed to be observed according the type of activities the agent is going to participate.
- *Trust in Societies* – individual-level and system-level approaches share a focus on (trust on) the behaviour of an individual agent. However, in the engineering of complex systems it emerges the need of modelling the notion of trust also related to *groups* or *societies* of agents, delegated of the execution of some social task. More generally we are looking to a systemic acceptation of trust: how much a system (as a structured society of agents) can be trusted in its behaviour, in its ability to achieve the global objectives as outcomes of the cooperative work of its agents? So we are interested in characterising trust also in cooperative scenarios, not only in competitive ones as it typically happens in the literature.
- *Constructive Trust* – As in the system-level (objective) case, we are interested in infrastructural abstractions (services) for creating and managing trust. However, differently from system-level approaches, we characterise these abstractions not only as *barriers*, basically creating trust by enforcing *norms* constraining agent actions and interactions. We are interested also in framing trust from a *constructive* point of view: I can have trust in an system because of the availability of services which provide some (objective) guarantees that not only certain interaction cannot happen, but also that some social tasks can be effectively executed, specifying for instance the workflow or plan useful for achieving the global objective. Considering system-level approaches, it is like modelling trust on the rules of encounters which make it possible to achieve some social goal.
- *Trust and Organisation* – As mentioned in the context of system-level trust, security support has a certain impact on trust in a system [14]. However, when engineering complex systems, some important aspects concerning security – such as access control – cannot be dealt without considering the organisation and coordination model adopted [11]. As an important example, Role-Based Access Control (RBAC) models and architectures – well-known in the research literature concening security in complex information systems, and recently introduced also in MAS [21] – make it possible to model security (access control) policies in the context of role-based organisational models. Accordingly, the presence of such an organisational model can have a significant impact on trust models and services, which can be characterised also considering the notions of roles and related organisational policies.

In the overall, the social and engineering acceptation of trust that emerges from the above points aims to be wider than the one usually found in the literature, and can be framed in the idea of agent societies used as metaphors to model trust in information technology in the most general way. This includes both trust between humans and systems – i.e. trust between users and systems and trust between designers/engineers and systems – and trust between systems and systems – i.e. trust among system components and trust among components of different systems. The interpretation of systems in terms of societies, promoted by MAS approaches, makes it possible to face these issues within the same conceptual framework, adopting a uniform approach to explore general models and solutions, relevant in computer science as well as in the other related fields.

A possible way to bring to practice such generalised acceptation of trust is to relate them to the coordination and organisation dimensions (and the related models) which characterise the engineering of agent societies. In next sections we follow this line, by presenting two infrastructural abstractions which we have recently introduced in MAS engineering, namely *coordination artifacts* and *agent coordination contexts*, and discussing their role in modelling and engineering such a notion of trust in MAS.

IV. Artifacts for Trust

From the research studies carried on in human (cooperative) activity – mainly with Activity Theory [2], [6] – it clearly emerges the fundamental role of tools or *artifacts* in the development of (social) activities in complex systems framed as societies [7], [16]. According to these studies, every non trivial human activities is *mediated* by some kind of artifacts. An artifact acts as the glue among two or multiple actors, as the tool that enables and mediates their interaction, ruling / governing the resulting global and "social" behaviour; consequently, an artifact can be considered *the* conceptual place encapsulating all the complexity of the social behaviour that it enables, allowing its factorisation, explicit modeling and engineering, and so freeing the actors of all this *social* burden [16]. Artifacts are widespread in human society: the language can be considered an artifact, as well as the writing, blackboards, maps, post-its, traffic signs such as semaphores, electoral cards or the signature on a document.

Based on this background, recently *coordination artifacts* have been introduced as a conceptual and engineering framework for MAS and agent societies [16], [12]. So the idea here is that coordination artifacts can play a primary role for engineering trust in MAS, providing an answer to the points remarked in Section III.

A. Coordination Artifact Model and Framework

Coordination artifacts have been defined as embodied[1] entities specialised to provide a coordination service in a MAS [12]. As *infrastructure* abstractions, they are meant to improve coordination activities automation; they can be considered then as basic building blocks for creating effective shared collaborative working environments, alleviating the coordination burden for the involved agents.

As remarked for artifacts in general, human society is full of entities like coordination artifacts, engineered by humans in order to support and automate coordination activities: well-known examples are street semaphores, blackboards, queuing tools at the super-markets, maps, synchronisers and so on.

Basically, a coordination artifact *(i)* entails a form of mediation among the agents using it, and *(ii)* embeds and enact effectively some coordination policy. Accordingly, two basic aims can be identified: *(i) constructive*, as an abstraction essential for creating/composing social activities, *(ii) normative*, as an abstraction essential for ruling social activities.

From a constitutive point of view, a coordination artifact is characterised by:

- a *usage interface*, defined in terms of a set of *operations* which agents can execute in order to use the artifacts.
- a set of *operating instructions*, which formally describe how to use the artifact in order to exploit its coordination service.
- a *coordinating behaviour*, which formally describe the coordination enacted by the artifact.

[1]The term embodied is used here to remark their independent existence from the agents using them.

Then, taking the agent viewpoint, to exploit a coordination artifact simply means to follow its operating instructions, on a step-by-step basis.

Among the main properties which exhibit coordination artifacts (and which differentiate them from the agent abstraction) we have:

- *Specialisation* – Coordination artifacts are specialised in automating coordination activities. For this purpose, they typically adopt a computational model suitable for effective and efficient interaction management, whose semantics can be easily expressed with concurrency frameworks such as process algebras, Petri nets, or Event-Condition-Reaction rules.
- *Encapsulation: Abstraction and Reuse* – Coordination artifacts encapsulate a coordination service, allowing user agents to abstract from how the service is implemented. As such, a coordination artifact is perceived as an individual entity, but actually it can be distributed on several nodes of the MAS infrastructure, depending on its specific model and implementation.
- *Malleability* – Coordination artifacts are meant to support coordination in open agent systems, characterised by unpredictable events and dynamism. For this purpsose, their coordination behaviour can be adapted and changed dynamically, either *(i)* by engineers (humans) willing to sustain the MAS behaviour, or *(ii)* by agents responsible of managing the coordination artifact, with the goal of flexibly facing possible coordination breakdowns or evolving/improving the coordination service provided.
- *Inspectability and controllability* – A coordination artifact typically supports different levels of inspectability: *(i)* inspectability of its operating instructions and coordination behaviour specification, in order to let user agents to be aware of how to use it or what coordination service it provides; *(ii)* inspectability of its dynamic state and coordination behaviour, in order to support testing and diagnosing (debugging) stages for the engineers and agents responsible of its management.
- *Predictability and formalisability* – The coordinating behaviour of an artifact strictly follows the specification/service for which it has been forged: given that specification and the agent interaction history, the dynamic behaviour of the artifact can be fully predicted.

TuCSoN [13] is an example of agent coordination infrastructure supporting this framework: TuCSoN coordination artifacts are called *tuple centres* [9], spread over the network, collected in the infrastructure nodes. Tuple centres technically are *programmable* tuple spaces, i.e. tuple spaces [9] whose behaviour in reaction to communicating event – the insertion, removal, read of tuples from the spaces – can be suitably programmed so as to realise coordination laws managing interactions (ReSpecT is the language adopted for the purpose). Tuple centres can be framed as general purpose coordination artifacts, whose coordinating behaviour can be dynamically customised and adapted to provide a specific coordination

service.

B. Trust through Coordination Artifacts

The notion of coordination artifacts can be useful to model trust issues as discussed in Section III.

As far as social trust is concerned, coordination artifacts can play the role of the abstractions provided by the infrastructure with suitable expressiveness and effectiveness to construct trust articulated strategies. For instance, coordination artifacts can be used as embodiment of the rules of encounter, being concrete shared tools which are used by the agents to interact according a specified protocol. Operating instructions in this case describe what agents are meant to do in order to participate to the protocols (according to their role), artifact state keeps track of the state of the interactions, and artifact behaviour is concerned with the management of the interaction according to the coordinating behaviour described by the protocol.

More generally, as mediating abstractions, coordination artifacts can be used for supporting the *observation* and *traceability* of agent actions and interactions. They can be designed so as to log / trace all the interactions of interest and related events occuring during its usage, in order to be inspected / observed as interaction history concerning not only a specific agent but also the agent society itself. Actions and interactions history can be useful then to build trust models concerning both the overall society, and the individual participating agents. Such trust models could be created both by humans and agents by inspecting and reasoning about the information reified in the artifact interaction history, made available by suitable infrastructure services. From this point of view then, coordination artifacts can provide a useful support for constructing trust model for individual-level approaches based both on socio-cognitive capabilities and on quantitative formulations: heterogeneous agents could exploit the same information to build different kind of models.

Then, the basic properties characterising coordination artifacts impact on modelling both trust in societies and constructive trust. In this case modelling trust toward a system or a society in charge of a specific social task exploiting a specific coordination artifact accounts for two aspects: *(i)* trusting the effectiveness of the coordination artifact for achieving the objective of the social task; *(ii)* trusting agents in being able to use effectively the coordination artifact. Artifact basic properties – concerning inspectability, predictability, etc. – along with the fact that the correctness of artifact behaviour could be formally verifiable and then "certifiable", with the availability of operating instructions and of a clear interface – could impact effectively in both previous points. It is worth remarking that this introduces a relatively new ontological framework on which formulating trust, introducing new notions such as *usability* of the artifact, the *complexity* of their operating instructions, and so on. This could change and enrich the cognitive model adopted by socio-cognitive approach to model trust of agents towards the environment.

Finally, from an engineering point of view, inspectability and controllability properties of artifacts could impact significantly on the trust toward a system engineered in terms of coordination artifacts, both for a designer and for a user of the system. In particular, *controllability* – which includes also the possibility of making online tests and diagnosis of artifact behaviour and then of the social core of the system, despite of its openness – is an aspect that heavily contributes to determine trust in the system.

V. Contexts for Trust

The notion of *agent coordintation context* (ACC) has been introduced in [8] as infrastructural abstraction modelling the presence of an agent inside its (organisational) environment. As for coordination artifacts, ACCs have been brought into practice within the TuCSoN infrastructure [17]. Here we show their relevance for modelling and engineering the last aspects of trust mentioned in previous chapter, i.e. trust related to organisation and security.

A. The Agent Coordination Context Abstraction

The ACC abstraction can be framed as the conceptual place where to set the boundary between the agent and the environment, so as to encapsulate the *interface* that enables agent actions and perceptions inside the environment. A useful metaphor for understanding ACCs is the *control room* [8]. According to this metaphor, an agent entering a new environment is assigned its own control room, which is the only way in which it can perceive the environment, as well as the only way in which it can interact. The control room offers the agent a set of admissible inputs (lights, screens,...), admissible outputs (buttons, cameras,...). How many input and output devices are available to agents, of what sort, and for how much time is what defines the control room *configuration*, that is the specific ACC. So, the ACC abstraction can be fruitfully exploited to model the *presence* or position of an agent within an organisation, in terms of its admissible actions with respect to organisation resources and its admissible communications toward the other agents belonging to the organisation.

ACCs are meant to be inspectable: it must be possible for an agent to know what kind of ACC it can obtain from an organisation – and so what roles and related actions it is allowed to do.

Two basic stages characterise the ACC dynamics: *ACC negotiation* and *ACC use*. An ACC is meant to be negotiated by the agents with the MAS infrastructure, in order to start a *working session* inside an organisation. The agent requests an ACC specifying which roles to activate inside the organisation. The request can fail for instance if an agent requests to play a role for which he is not allowed, or which is not compatible with the other roles currently actively played by the agent inside the organisation. If the agent request is compatible with (current) organisation rules, a new ACC is created, configured according to the characteristics of the specified roles, and then released to the agent for active playing. The agent then can

use the ACC to interact with the organisation environment, by exploiting the actions/perceptions enabled by the ACC.

The ACC framework has been used to model and implement Role-Based Access Control architecture on top of TuCSoN infrastructure [17].

B. Trust through Agent Coordination Contexts

ACCs – supported by suitable infrastructures – guarantee the enforcement of organisational rules and related security policy inside a social environment: they can act as a barriers for agents, filtering only the patterns of actions and perceptions allowed according to their roles. This clearly impacts on the trust that we can have on the systems, providing a generalisation of the security mechanism mentioned for system-level trust. In particular ACC abstraction makes it possible to link trust with the organisational model adopted: agents can participate to activities only by playing some roles through dynamically requested ACC enabling and ruling their actions. In the overall, we can frame an ACC as the embodiment of a *contract* established between a specific agent and the system (organisation) where he is actively playing.

Each organisation can define (and change dynamically) the set of available roles and rules, and then the set of ACCs which can be released to agents. This information can be then made available – by means of suitable infrastructure services – for creating trust in agents and users aiming at participating at or using the systems.

VI. Conclusion

The notion of trust has a deep impact on the future of artificial systems. How trust is modelled, how it is engineered – that is, how it is actually built into artificial systems – are then crucial issues that are already discussed in literature, and in particular in MAS literature. In this paper, we first shortly summarised the many different acceptations of the trust notion, then we pointed out some fundamental open issues that seem to be of particular relevance to the modelling and engineering of trust in the context of complex artificial systems, in general, and of MAS, in particular.

As the main contribution of this seminal paper, we adopted the viewpoint of MAS infrastructures (as the most natural *loci* where to embed trust in MAS) and showed how two different infrastructural abstractions recently introduced (coordination artifacts and agent coordination contexts) can be exploited for modelling and engineering trust within MAS.

References

[1] P. Dasgupta. Trust as a commodity. In D. Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*, pages 49–72. Blackwell, 1998.

[2] Y. Engeström, R. Miettinen, and R.-L. Punamaki, editors. *Perspectives on Activity Theory*. Cambridge University Press, 1999.

[3] R. Falcone and C. Castelfranchi. Social trust: a cognitive approach. In R. Falcone, M. P. Singh, and Y. Tan, editors, *Trust in Cyber-Societies, Integrating the Human and Artificial Perspectives*, volume 2246 of *LNCS*. Springer-Verlag, 2001.

[4] R. Falcone, M. P. Singh, and Y. Tan, editors. *Trust in Cyber-Societies, Integrating the Human and Artificial Perspectives*, volume 2246 of *LNCS*. Springer-Verlag, 2001.

[5] Y. Mass and O. Shehory. Distributed trust in open multi-agent systems. In R. Falcone, M. P. Singh, and Y. Tan, editors, *Trust in Cyber-Societies, Integrating the Human and Artificial Perspectives*, volume 2246 of *LNCS*. Springer-Verlag, 2001.

[6] B. Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996.

[7] B. Nardi. Studying contexts: A comparison of activity theory, situated action models and distributed cognition. In B. Nardi, editor, *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996.

[8] A. Omicini. Towards a notion of agent coordination context. In D. Marinescu and C. Lee, editors, *Process Coordination and Ubiquitous Computing*, pages 187–200. CRC Press, 2002.

[9] A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, Nov. 2001.

[10] A. Omicini and S. Ossowski. Objective versus subjective coordination in the engineering of agent systems. In M. Klusch, S. Bergamaschi, P. Edwards, and P. Petta, editors, *Intelligent Information Agents: An AgentLink Perspective*, volume 2586 of *LNAI: State-of-the-Art Survey*, pages 179–202. Springer-Verlag, Mar. 2003.

[11] A. Omicini, A. Ricci, and M. Viroli. Formal specification and enactment of security policies through Agent Coordination Contexts. *Electronic Notes in Theoretical Computer Science*, 85(3), Aug. 2003.

[12] A. Omicini, A. Ricci, M. Viroli, and C. Castelfranchi. Coordination artifacts: Environment-based coordination for intelligent agents. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, New York, USA, 2004. ACM Press.

[13] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999. Special Issue: Coordination Mechanisms for Web Agents.

[14] S. Poslad, M. Calisti, and P. Charlton. Specifying standard security mechanisms in multi-agent systems. In *Workshop on Deception, Fraud and Trust in Agent Societies*, pages 122–127, Bologna, Italy, 2002. AAMAS 2002, Proceedings.

[15] S. D. Ramchurn, D. Hunyh, and N. R. Jennings. Trust in multi-agent systems. *Knowledge Engineering Review*, 2004. to appear.

[16] A. Ricci, A. Omicini, and E. Denti. Activity Theory as a framework for MAS coordination. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents World III*, volume 2577 of *LNCS*, pages 96–110. Springer-Verlag, Apr. 2003. 3rd International Workshop (ESAW 2002), Madrid, Spain, 16–17 Sept. 2002. Revised Papers.

[17] A. Ricci, M. Viroli, and A. Omicini. Agent coordination contexts: From theory to practice. In R. Trappl, editor, *Cybernetics and Systems 2004*, Vienna, Austria, 2004. Austrian Society for Cybernetic Studies. 17th European Meeting on Cybernetics and System Research (EMCSR 2004), Vienna, Austria, 2004. Proceedings.

[18] J. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.

[19] J. Sabater and C. Sierra. REGRET: A reputational model for gregarious societies. In *1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 475–482, Bologna, Italy, 2002. ACM Press. Proceedings.

[20] T. Sandholm. Distributed rational decision making. In G. Weiss and S. Sen, editors, *Multi-Agent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 299–330. AAAI/MIT Press, 1999.

[21] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[22] M. Viroli and A. Omicini. Coordination as a service: Ontological and formal foundation. *Electronic Notes in Theoretical Computer Science*, 68(3), Mar. 2003. 1st International Workshop "Foundations of Coordination Languages and Software Architecture" (FOCLASA 2002), Brno, Czech Republic, 24 Aug. 2002. Proceedings.

[23] P. Wegner. Why interaction is more powerful than algorithms. *Communication of ACM*, 40(5):80–91, May 1997.

[24] G. Zacharia and P. Maes. Trust through reputation mechanisms. *Applied Artificial Intelligence*, (14):881–907, 2000.

OWLBeans
From ontologies to Java classes

Michele Tomaiuolo, Federico Bergenti, Agostino Poggi, Paola Turci

***Abstract* — The Semantic Web is an effort to build a global network of machine-understandable information. Software agents should be enhanced with tools and mechanisms to autonomously access this information. The objective of this paper is to present a toolkit for extracting a subset of the relations expressed in an OWL document. It generates data structures and artifacts that can be handy for autonomous software agents to access semantically annotated information provided on the web.**

***Index Terms* — Semantic web, ontology, object-oriented systems, autonomous agents, multi-agent systems.**

I. Introduction

Semantic web promises to build a network of machine understandable information [4],[5],[9]. But to become a widespread reality, this vision has to demonstrate innovative applications, and so it is fundamental for its success to have software libraries and toolkits, enabling autonomous software agents to interface this huge source of information.

The OWLBeans toolkit, which is going to be presented in this paper, does not deal with the whole complexity of a semantically annotated web. Instead, its purpose is precisely to cut off this complexity, and provide simple artefacts to access structured information.

In general, interfacing agents with the Semantic Web implies the deployment of an inference engine or a theorem prover. In fact, this is the approach we're currently following to implement an agent-based server to manage OWL ontologies [15].

Instead, in many cases, autonomous software agents cannot, or don't need to, face the computational complexity of performing inferences on large, distributed information sources. The OWLBeans toolkit is mainly thought for these agents, for which an object-oriented view of the application domain is enough to complete their tasks.

The software artefacts produced by the toolkit, i.e., mainly JavaBeans [12] and simple metadata representations used by JADE [10], are not able to express all the relationships that are present in the source. But in some context this is not required. Conversely, especially if software and hardware resources are very limited, it is often preferable to deal only with common Java interfaces, classes, properties and objects.

The main functionality of the presented toolkit is to extract a subset of the relations expressed in an OWL document for generating a hierarchy of JavaBeans reflecting them, and possibly an associated JADE ontology to represent metadata. But, given its modular architecture, it also allows other kinds of conversions, for example to save a JADE ontology into an OWL file, or to generate a package of JavaBeans from the description provided by a JADE ontology.

II. Intermediate model

The main objective of the OWLBeans toolkit is to extract JavaBeans from an OWL ontology. But to keep the code maintainable and modular, we decided to create first an internal, intermediate representation of the ontology. In fact our tool, translating OWL ontologies to JavaBeans or vice-versa, can be viewed as a sort of compiler, and virtually every compiler builds its own intermediate model before producing the desired output. In compilers, this helps to separate the problems of the parser from those of the lexical analyzer and moreover, the same internal representation can so be used to produce different outputs. In the case of the OWLBeans toolkit, the intermediate model can be used to generate the sources of some Java classes, a JADE ontology, or an OWL file. And the intermediate model itself can be filled with data coming from different sources, obtained, for example, by reading an OWL file or by inspecting a JADE ontology.

A. Requirements

The main features we wanted for the internal ontology representation were:

- *Simplicity*: it had to include only few simple classes, to allow a fast and easy traversal of the ontology. The model had to be simple enough to be managed in scripts and templates; in fact, one of the main design goals was to have a model to be passed to a template engine, for generating the code directly from it.

Manuscript received September 27, 2004.

M. Tomaiuolo is with DII, University of Parma, Parco Area delle Scienze 181A, 43100, Parma, Italy (phone: +39 0521 905712; e-mail: tomamic@ce.unipr.it).

F. Bergenti is with DII, University of Parma, Parco Area delle Scienze 181A, 43100, Parma, Italy (phone: +39 0521 905708; e-mail: bergenti@ce.unipr.it).

A. Poggi is with DII, University of Parma, Parco Area delle Scienze 181A, 43100, Parma, Italy (phone: +39 0521 905728; e-mail: poggi@ce.unipr.it).

P. Turci is with DII, University of Parma, Parco Area delle Scienze 181A, 43100, Parma, Italy (phone: +39 0521 905708; e-mail: turci@ce.unipr.it).

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

- *Richness*: it had to include the information needed to generate JavaBeans and all other wanted artefacts. The main guideline in the whole design was to avoid limiting the translation process. The intermediate model had to be as simple as possible, though not creating a *metadata bottleneck* in the translation of an OWL ontology to JavaBeans. All metadata needed in the following steps of the translation pipeline had to be represented in the intermediate model. Moreover, though it had to be used mainly by template engines to generate JavaBeans, it had to be general enough to allow other uses, too.
- *Primitive data-types*: it had to handle not only classes, but even primitive data-types, as both Java and OWL classes can have properties with primitive data-types as their range.
- *External references*: often ontologies are built extending more general classifications an taxonomies, for example to detail the description of some products in the context of a more general trade ontology. We wanted our model not to be limited to single ontologies, but to allow the representation of external entities, too: classes had to extend other classes, defined locally or in other ontologies, and property ranges had to allow not only primitive data-types and internal classes, but even classes defined in external ontologies.

One of the main issues regarded properties, as they are handled in different ways in description logics and in object oriented systems. While they are first level entities in Semantic Web languages, they are more strictly related to their "owner" class in the latter model. In particular, property names must be unique only in the scope of their own class in object-oriented systems, while the have global scope in description logics. Our choice was to have properties "owned" by classes. This allows an easier manipulation of the meta-objects while generating the code for the JavaBeans, and a more immediate mapping of internal description of classes to the desired output artefacts.

B. *Other models*

Before deciding to create a specific internal representation of the ontology, we evaluated two existing models: the one provided by Jena [13], which of course is very close to the Semantic Web model, and the one used internally by JADE, which instead is quite close to the object-oriented model.

The first one had the obvious advantage to be the most complete model of the ontology. According to Brooks, "the world is its own best model" [22]. Nevertheless it was too complex for our scopes. For example, we wanted it to be handled by template engines, to generate Java code directly from it.

The other one, used by JADE, had most of the features we desired. But it had some major disadvantages, too. First of all, it cannot easily manage external entities; though ontologies can be organized in hierarchies, it is not possible to define the namespace of classes. Another issue is that the classes of a JADE ontology are distinguished as predicates or concepts, and predicates for example cannot be used as range of properties; this matches the semantics of the FIPA SL language [6], but could be a problem for the representation of generic OWL ontologies, as such distinction does not exist in the language. The third, and perhaps most important, issue is it does not allow exploring the tree of classes and properties from the outside.

The internal field to store classes defined in the *Ontology* class, for example, is marked private; obviously, this is a good choice to encapsulate data, but no accessor methods are provided to get the names of all classes. Other problems regard the *ObjectSchema* class that does not provide a way to get all directly extended super-classes and all locally defined slots. Finally, the *CardinalityFacet* class does not expose minimum and maximum allowed values.

In fact, the JADE ontology model was designed to allow automatic marshalling and un-marshalling of objects from FIPA ACL messages [8], and not to reason about ontology elements.

Obviously, these limitations of the JADE ontology model, proved to be a serious problem when trying to save it in an OWL file, too. This facet will be discussed in more detail in the following sections.

C. *Core classes*

The intermediate model designed for the OWLBeans toolkit is made of just few, very simple classes. The simple UML class diagram shown in figure 1 describes the whole intermediate model package.

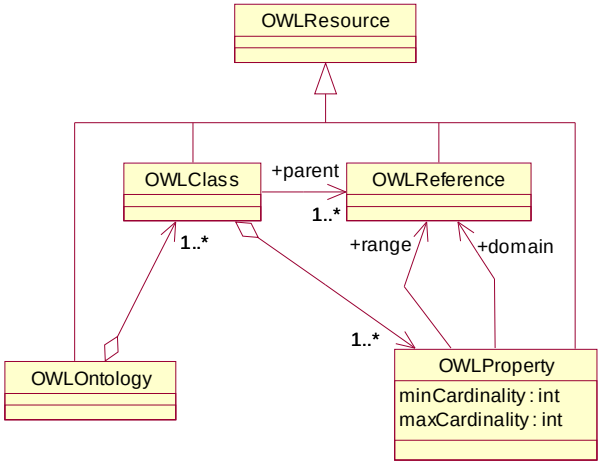


Fig. 1 - Class diagram of the intermediate model

The root class here is *OwlResource*, which is extended by all the others. It has just two fields: a local *name*, and a *namespace*, which are intended to store the same data as resources defined in OWL files. All the resources of the

intermediate model – refernces, ontologies, classes and properties – are implicitly *OwlResource* objects.

OwlReference is used as a simple reference, to point to super-classes and range types, and don't add anything to the *OwlResource* class definition. It is defined to underline the fact that classes cannot be used directly as ranges or parents.

OwlOntology is nothing more than a container for classes. In fact it owns a list of *OwlClass* objects. It inherits from *OwlResource* the name and namespace fields. In this case the namespace is mandatory and is supposed to be the namespace of all local resources, for which in fact it is optional.

OwlClass represents OWL classes. It points to a list of parents, or super-classes, and owns a list of properties. Each parent in the list is a *OwlReference* object, i.e. a name and a namespace, and not an *OwlClass* object. Its name must be searched in the owner ontology to get the real *OwlClass* object. Properties instead are owned by the *OwlClass* object, and are stored in the *properties* list as instances of the *OwlProperty* class.

OwlProperty is the class representing OWL properties. As in UML, their name is supposed to be unique only in the scope of their "owner" class. Each property points to a domain class and to a range class or data-type. Both these fields are simple *OwlReference* objects: while the first contains the name of the owner class, the latter can indicate an *OwlClass*, or an XML data-type, according to the namespace. Two more fields are present in this class: *minCardinality* and *maxCardinality*. They are used to store respectively the minimum and maximum allowed cardinality for the property values. Moreover, a *minCardinality = 0* has the implicit meaning of an optional property, while *maxCardinality = 1* has the implicit meaning of a functional property.

Probably you have already noticed the design choice to have indirect references to *OwlClass* objects in some places, in particular to point to super-classes and to allowed ranges. This decision has two main advantages over direct Java references to final objects: parsing an OWL file is a bit simpler, as references can point to classes that are not yet defined, and above all in this way super-classes and ranges are not forced to be local classes, but can be references to resources defined somewhere else.

III. Pluggable readers and writers

In our toolkit, the intermediate model is used as the glue to put together the various components needed to perform the desired, customizable task. These components are classes implementing one of the two interfaces (*OwlReader* and *OwlWriter*) representing ontology readers and writers, respectively. Not very surprisingly, readers can build an intermediate representation of the ontology, acquiring metadata from different kinds of sources, while writers can use this model to produce the desired artefacts.

The current version the toolkit provides readers to inspect OWL files and JADE ontologies, and writers to generate OWL files, source files of JavaBeans and JADE ontologies.

A very simple, yet handy application is provided, which can be customized with pluggable readers and writers, thus performing all the possible translations. While not pluggable into the main application, other components are implemented to provide additional features. For example, one of them allows to instantiate at runtime a JADE ontology and add classes to it from an intermediate ontology representation. Another component allows to load the generated code for JavaBeans directly into the Java Virtual Machine, using an embedded Java scripting engine. These components can be exploited, for example, by agent-based applications designed to be ontology agnostic, like some of those deployed in the Agentcities network [1],[2].

A. OWL files

Two classes are provided to manage OWL files. *OwlFileReader* allows reading an intermediate model from an OWL file, while *OwlFileWriter* allows saving an intermediate model to an OWL file. These two classes respectively implement the *OwlReader* and *OwlWriter* interfaces and are defined in the package confining all the dependencies from the Jena toolkit.

The latter process is quite straightforward, as all the information stored in the intermediate model can easily fit into an OWL ontology, in particular into a Jena *OntModel* object. But one particular point deserves attention. While the property names in the OWLBeans model are defined in the scope of their owner class, all OWL properties instead are first level elements and share the same namespace. This poses serious problems if two or more classes own properties with the same name, and above all if these properties have different ranges or cardinality restrictions.

In the first version of the OWLBeans toolkit, this issue is faced in two ways: if a property is defined by two or more classes then a complex domain is created in the OWL ontology for it; in particular the domain is defined as the union of all the classes that share the property, using an *owl:UnionClass* element. Cardinality restrictions are specific to classes in both models, and are not an issue. Currently, the range is instead assigned to the property by the first class that defines it, and is kept constant for the other classes in the domain. But this obviously could be incorrect in some cases. Using some class-scoped *owl:allValuesFrom* restrictions could solve most of the problems, but nevertheless difficulties would arise in the case of a property defined in some classes as a data-type property, and somewhere else as an object property.

Another mechanism allows to optionally use the class name as a prefix for the names of all its properties, hence automatically enforcing different names for properties defined in different classes. Obviously this solution is appropriate only for ontologies where names can be decided arbitrarily;

moreover it is appropriate when resulting OWL ontologies will be used only to generate JavaBeans and JADE ontologies, as in this case the leading class name would be automatically stripped off by the *OwlFileReader* class.

The inverse process, i.e. converting an OWL ontology into the intermediate representation, is instead possible only under very restrictive limitations, mainly caused by the rather strong differences between Semantic Web and object oriented languages. In fact, only few, basic features of the OWL language are currently supported.

Basically, the OWL file is first read into a Jena *OntModel* object and then all *classes* are analyzed. In this step all *anonymous classes* are just discarded. For each one of the remaining classes, a corresponding *OwlClass* object is created in the internal representation. Then all *properties* listing the class directly in their domain are considered and added to the intermediate model as *OwlProperty* objects. Here, each defined property points to a single class as domain and to a single class or data-type as range. Set of classes are not actually supported. Data-type properties are distinguished in our model by the namespace of their range, which is *http://www.w3.org/2001/XMLSchema#*. The only handled restrictions are *owl:cardinality*, *owl:minCardinality* and *owl:maxCardinality*, which are used to set the *minCardinality* and *maxCardinality* fields of the new *OwlProperty* object. The *rdfs:subClassOf* element is handled in a similar way: only parents being simple classes are taken into consideration, and added to the model.

All the rest of the information eventually being in the file is lost in the translation.

Inverse conversions are applied when writing an intermediate ontology model into an OWL file. Table 1 provides a synthetic view of these mappings.

OWL	**OWLBeans**
owl:Class	*OwlClass*
owl:ObjectProperty, *owl:DatatypeProperty*	*OwlProperty*
rdfs:range	*OwlProperty.range*
rdfs:domain	*OwlProperty.domain*
owl:FunctionalProperty	*OwlProperty.maxCardinality*
owl:minCardinality	*OwlProperty.minCardinality*
owl:maxCardinality	*OwlProperty.maxCardinality*
owl:cardinality	*OwlProperty.minCardinality*, *OwlProperty.maxCardinality*

Tab. 1 – Mappings between OWL/OWLBeans elements

B. Template engine

Rather than generating the source files of the desired JavaBeans directly from the application code, we decided to integrate a template engine in our project. This eventually helped to keep the templates out of the application code, and centralized in specific files, where they can be analyzed and debugged much more easily. Moreover, new templates can be added and existing ones can be customized without modifying the application code.

The chosen template engine was Velocity [19], distributed under LGPL licence from the Apache Group. It's an open source project enjoying widespread use. While its fame mainly comes from being integrated into the Turbine web framework, where it is often preferred to other available technologies, as JSP pages, it can be effortlessly integrated in custom applications, too.

Velocity template engine integration is performed through the *VelocityFormatter* class. This class hides all the implementation details of applying desired templates to an intermediate ontology and encapsulates all the dependencies from the Velocity engine. Two different types of templates are allowed, *ontology templates* and *class templates*. While the first ones only need an *OwlOntology* as parameter, the other ones also need an *OwlClass*. Ontology templates are used to generate as output the source code of JADE ontologies, for example. Class templates are instead applied to each *OwlClass* of the ontology to generate a Java interface and a corresponding implementation class, for example.

Currently, the OWLBeans toolkit provides templates to generate the source file for JavaBeans and JADE ontologies. JavaBeans are organized in a common package where, first of all, some interfaces mapping the classes defined in the ontology are written. Then, for each interface, a Java class is generated, implementing the interface and all accessor methods needed to get or set properties.

Creating an interface and then a separate implementing Java class for each ontology class is necessary to overcome the single-inheritance limitation that applies to Java classes. Each interface, instead, can extend an arbitrary number of parent interfaces. The corresponding class is eventually obliged to provide an implementation for all the methods defined by one of the directly or indirectly implemented interfaces.

The generated JADE ontology file can be compiled and used to import an OWL ontology into the JADE framework, thus allowing agents to communicate about the concepts defined in the ontology. The JavaBeans will be automatically marshalled and un-marshalled from ACL messages in a completely transparent way.

Translating an intermediate ontology to Java classes cuts off some details of the metadata level. In particular, no checks are imposed on the cardinality of property values, but only a rough distinction is made to associate non-functional properties (where *maxCardinality* is >1) with a Java *List*, to hold the sequence of values. Moreover, the class of the items of the list is not enforced, so the *range* information associated with the *OwlProperty* object is effectively lost. Instead, generating the JADE ontology does not impose the same loss of *range* and *cardinality* metadata. But nonetheless, the available set of primitive data-types is poor compared to the one of XML Schema, used in the intermediate model.

XSD	Java	JADE
xsd:Boolean	*boolean*	*BOOLEAN*
xsd:decimal, xsd:float, xsd:double	*double*	*FLOAT*
xsd:integer, xsd:nonNegativeInteger, xsd:positiveInteger, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte	*int*	*INTEGER*
xsd:base64Binary, xsd:hexBinary	*Object*	*BYTE_SEQUENCE*
xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:duration	*Date*	*DATE*
xsd:string, xsd:normalizedString, xsd:anyURI, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName	*String*	*STRING*

Tab. 2 – Mappings between XSD/Java types

The XML data-types supported by the OWL syntax are listed in [16]. For each of them a corresponding primitive Java or JADE type must be provided. Both these conversions are not zero-cost transformations, as the target types do not express the precise meaning of their corresponding XML Schema types. Table 2 shows these conversions, as they are defined in the default templates; the "*Java*" column indicates the Java data types, while in the "*JADE*" column indicates the name of the corresponding constants defined in the JADE *BasicOntology* class.

C. Scripting engine

An additional template is provided to put the source of all interfaces, classes and JADE ontologies together, into a single stream or file, where the *package* and *imports* statements are listed only once, at the beginning of the whole file. This proves useful to load generated classes directly into the Java Virtual Machine.

In fact, if a Java scripting engine like *Janino* [11] is embedded into the toolkit, it can be exploited as a special class-loader, to load classes directly from Java source files without first compiling them into byte-code. Source files don't even need to be written to the file system first. So, at the end, JavaBeans can be loaded into the Java Virtual Machine directly from an OWL file.

Obviously, pre-compiled application code will not be able to access newly loaded classes, which are not supposed to be known at compile time. But the same embedded scripting engine can be used to interpret some ontology specific code, which could be loaded at run time from the same source of the OWL ontology file, for example, or provided to the application in other ways.

Among the various existing Java scripting engines we tested for integration into the toolkit, currently Janino proves to be the best choice. It is developed as open source project and released under LGPL license. It is an embedded compiler that can read Java expressions, blocks, class bodies and sets of source files. The Java byte-code it generates can be loaded and executed directly into the Java Virtual Machine.

While other similar engines were not able to correctly read the source files produced by the template engine, Janino made its work promptly. For example, Beanshell was not able to parse source files of interfaces with multiple inheritance, which instead is an important feature required by the OWLBeans toolkit. Thanks to its features, and to its clean design, Janino is gaining popularity. Drools, a powerful rule engine for Java, uses Janino to interpret rule scripts, and even Ant and Tomcat can be configure to use Janino as their default compiler.

The possibilities open by embedding a scripting engine into an agent system are numerous. For example, software agents for e-commerce often need to trade goods and services described by a number of different, custom ontologies. This happens in the Agentcities network, where different basic services can be composed dynamically to create new compound services.

To increase adaptability, these agents should be able to load needed classes and code at runtime. The OWLBeans package allows them to load into the Java Virtual Machine some JavaBeans directly from an OWL file, together with the ontology-specific code needed to reason about the new concepts.

D. JADE ontologies

Probably one of most interesting application of the Semantic Web is its use by autonomous software agents, which could use ontologies to reason and manipulate their environment. Their world would be made of resources and services described in ontologies, which would not be supposed to be known a priori, at compile time. The OWLBeans toolkit provides software agents the ability to load ontologies and defined classes at run time, just when they're needed or when they're discovered.

Apart from using the embedded Velocity template engine and the embedded Janino scripting engine to load generated classes at run time into the Java Virtual Machine, another component is provided to instantiate an empty JADE ontology at run time, and populate it with classes and properties read from an OWL file, or from other supported sources.

This proves useful when the agent doesn't really need JavaBeans, but can use the internal ontology model of JADE to understand the content of received messages, and to write the content of messages to send to others. The generated JADE ontology is very similar to the one produced by the Velocity template, but it doesn't need to be compiled, as no source code is generated. Instead Java objects are manipulated to create a new instance of the *Ontology* class containing all the classes and properties of the intermediate model.

The class providing this functionality is defined in the *JadeOwlOntology* class. This class does not implement the *OwlWriter* interface, but extends the *Ontology* class of JADE, adding the ability to read classes from an OWLBeans intermediate model.

Table 3 shows how the entities of one model can be mapped to the other.

Creating and populating a JADE ontology from an intermediate model is quite a straightforward process. In fact an *OwlClass* can be mapped without particular difficulties into a JADE *Schema*, while an *OwlProperty* can easily fit into a JADE *SlotDescriptor* (a private inner class of *ObjectSchemaImpl*, which can be inspected through some public methods of the outer class). The only significant difference is JADE making explicit the *AggregateSchema*, for the range of slots with *maxCardinality* > *1*, and having a *TypedAggregateFacet* (i.e. a restriction) to enforce the schema of the single elements. Moreover, in a JADE ontology, *maxCardinality* and *minCardinality* are added to a slot through a *CardinalityFacet*, while in the OWLBeans model, for simplicity, they are two fields of the *OwlProperty* class.

JADE	OWLBeans
ObjectSchema	*OwlClass*
SlotDescriptor	*OwlProperty*
SlotDescriptor.schema	*OwlProperty.range*
SlotDescriptor.optionality	*OwlProperty.minCardinality*
CardinalityFacet.cardMin	*OwlProperty.minCardinality*
CardinalityFacet.cardMax	*OwlProperty.maxCardinality*
TypedAggregateFacet.type	*OwlProperty.range*

Tab. 3 – Mappings between JADE/OWLBeans elements

It is interesting to note that JADE defines facets, which are very similar to OWL restrictions, and which instead are missing in the OWLBeans model. This was a precise design choice to make traversing the model easier, without sacrificing needed metadata but probably loosing a bit of generality.

The *JadeReader* class encapsulates all the dependencies from the JADE framework This class does exactly what its name suggests: it "reads" an instance of a JADE ontology, and generates an intermediate model from it. Unfortunately, as we already underlined, JADE ontologies are not designed to be traversed from the outside. To be useful to inspect the content of an ontology, the model JADE uses internally lacks few accessor methods:

- it lacks a method, in the *Ontology* class, to obtain the name of all defined classes;
- it lacks a method in the *ObjectSchema* class to get the name of all defined properties;
- finally it lacks two methods to read minimum and maximum allowed cardinality, in *CardinalityFacet*.

In the implementation of the *JadeReader* class, these limitations are circumvented by using the reflection API of Java to access hidden fields and methods when necessary. Obviously, this solution can only be thought as a temporary, very limited and well documented, patch to allow JADE ontologies to be fully inspected from external code. In fact, since encapsulation is broken, even minimal modifications to the internal state representation of one of the three listed classes would stop *JadeReader* from working. We valued the possibility to export JADE ontologies to OWL files important enough to be released very soon, and thus creating such a patch proved necessary.

Anyway, the proposed modifications to the ontology API of JADE are going to be submitted to the JADE Board and to the JADE community for their introduction into the official distribution. They would make the API useful not only to extract the content of ACL messages, or to compose such messages, but even to inspect the described entities and discover some simple relationships among them. Moreover, they would not break backward compatibility, as just few methods need to be added or made public. Nothing else needs to be changed.

A particularity of the *JadeReader* class is that it silently adds some classes to the ontology it generates. These classes represent some basic FIPA types for ontology classes. FIPA SL in fact distinguishes ontology classes as *concepts*, representing objects of the model, or *predicates*, representing beliefs about the objects. Then there are more specific concepts representing *actions*, i.e. some tasks that agents can be requested to execute. The last basic class that's silently added is a concept for *agent identifiers*, or *AID*s, a class used for assigning unique names to FIPA agents [7]. Figure 2, captured from the Protégé ontology editor [17],[18], shows the hierarchy of the basic FIPA classes.

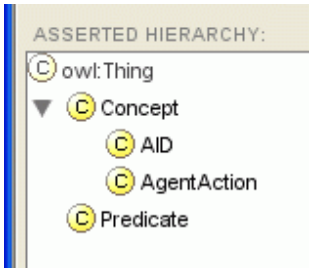


Fig. 2 – Basic FIPA classes

When the JADE ontology is traversed, each one of its schemas is checked for being an instance of a particular basic class and, accordingly, it is placed in the right branch of the generated hierarchy of classes. For example, a *ConceptSchema* class will be mapped into an *OwlClass* class having "*Concept*" among its ancestors, one of the classes added by default to the intermediate ontology soon after its creation. Similarly, a *PredicateSchema* class will instead have "*Predicate*" among its direct parents, or among its ancestors.

IV. USING THE TOOLKIT

A customizable Java application is distributed with the toolkit. Thanks to the modular design of the whole project, this application is very simple, yet allowing to exploit almost

all the functionalities of the toolkit. It simply takes the intermediate model produced by a pluggable reader, and feeds with it a pluggable writer. In this way, it can be used to realize all the format conversions made possible by combining available readers and writers. It can be used to generate Java classes from an OWL file, or to save a JADE ontology into an OWL file, or even to generate some JavaBeans adhering the descriptions provided by a JADE ontology.

The application can be execute from a shell, using the following syntax:

```
java it.unipr.aot.owl.Main [–input <input>] [-output <output>] [-package <package>] [-ontology <ontology>] [-imports (true|false)]
```

The optional arguments include the input file, the output folder for generated sources, the name of the package and the one of the ontology, and a flag to process imported ontologies. The last option is currently not yet implemented.

The following subsections show an example of execution. The first subsection shows the input ontology. The following one shows the source code generated by applying the default templates.

A. Input OWL ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about="">
    <rdfs:label>Test</rdfs:label>
  </owl:Ontology>
  <owl:Class rdf:ID="AID">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Concept"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Price">
    <rdfs:subClassOf rdf:resource="#Concept"/>
  </owl:Class>
  <owl:Class rdf:ID="Tradeable">
    <rdfs:subClassOf rdf:resource="#Concept"/>
  </owl:Class>
  <owl:Class rdf:ID="Predicate"/>
  <owl:Class rdf:ID="Book">
    <rdfs:subClassOf rdf:resource="#Tradeable"/>
  </owl:Class>
  <owl:Class rdf:ID="AgentAction">
    <rdfs:subClassOf rdf:resource="#Concept"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="price">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Tradeable"/>
    <rdfs:range rdf:resource="#Price"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="authors">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Book"/>
  </owl:DatatypeProperty>
  <owl:FunctionalProperty rdf:ID="currency">
    <rdfs:domain rdf:resource="#Price"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="value">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Price"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="title">
    <rdfs:domain rdf:resource="#Book"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

B. Generated Java source code

```
package bookstore;

import jade.util.leap.List;
import jade.content.onto.*;
import jade.content.schema.*;

public interface Price extends jade.content.Concept {

  public double getValue();
  public void setValue(double value);

  public String getCurrency();
  public void setCurrency(String currency);
}

public interface Tradeable extends jade.content.Concept {

  public Price getPrice();
  public void setPrice(Price price);
}

public interface Book extends Tradeable {

  public String getTitle();
  public void setTitle(String title);

  public List getAuthors();
  public void setAuthors(List authors);
}

public class PriceImpl implements Price {

  String currency;
  public String getCurrency() { return currency; }
  public void setCurrency(String currency) { this.currency = currency; }

  double value;
  public double getValue() { return value; }
  public void setValue(double value) { this.value = value; }
}

public class TradeableImpl implements Tradeable {

  Price price;
  public Price getPrice() { return price; }
  public void setPrice(Price price) { this.price = price; }
}

public class BookImpl implements Book {

  String title;
  public String getTitle() { return title; }
  public void setTitle(String title) { this.title = title; }


  List authors;
  public List getAuthors() { return authors; }
  public void setAuthors(List authors) { this.authors = authors; }

  Price price;
  public Price getPrice() { return price; }
  public void setPrice(Price price) { this.price = price; }
}

public class BookstoreOntology extends Ontology {
  public static final String ONTOLOGY_NAME = "Bookstore";

  // The singleton instance of this ontology
  private static Ontology theInstance = new BookstoreOntology();
  public static Ontology getInstance() { return theInstance; }

  // Vocabulary
  public static final String TRADEABLE = "Tradeable";
  public static final String TRADEABLE_PRICE = "price";
  public static final String PRICE = "Price";
  public static final String PRICE_VALUE = "value";
```

```
  public static final String PRICE_CURRENCY = "currency";
  public static final String BOOK = "Book";
  public static final String BOOK_TITLE = "title";
  public static final String BOOK_AUTHORS = "authors";

  public void addSlot(ConceptSchema schema, String slot, TermSchema type, int minCard, int maxCard) {
    int optionality = (minCard > 0) ? ObjectSchema.MANDATORY : ObjectSchema.OPTIONAL;
    if (maxCard == 1) schema.add(slot, type, optionality);
    else schema.add(slot, type, minCard, maxCard);
  }

  public void addSlot(PredicateSchema schema, String slot, TermSchema type, int minCard, int maxCard) {
    int optionality = (minCard > 0) ? ObjectSchema.MANDATORY : ObjectSchema.OPTIONAL;
    if (maxCard == 1) schema.add(slot, type, optionality);
    else schema.add(slot, type, minCard, maxCard);
  }

  public BookstoreOntology() {
    super(ONTOLOGY_NAME, BasicOntology.getInstance());

    try {
      PrimitiveSchema stringSchema = (PrimitiveSchema)getSchema(BasicOntology.STRING);
      PrimitiveSchema floatSchema = (PrimitiveSchema)getSchema(BasicOntology.FLOAT);
      PrimitiveSchema intSchema = (PrimitiveSchema)getSchema(BasicOntology.INTEGER);
      PrimitiveSchema booleanSchema = (PrimitiveSchema)getSchema(BasicOntology.BOOLEAN);
      PrimitiveSchema dateSchema = (PrimitiveSchema)getSchema(BasicOntology.DATE);
      ConceptSchema aidSchema = (ConceptSchema)getSchema(BasicOntology.AID);

      // Adding schemas
      ConceptSchema tradeableSchema = new ConceptSchema(TRADEABLE);
      add(tradeableSchema, Class.forName("bookstore.TradeableImpl"));

      ConceptSchema priceSchema = new ConceptSchema(PRICE);
      add(priceSchema, Class.forName("bookstore.PriceImpl"));

      ConceptSchema bookSchema = new ConceptSchema(BOOK);
      add(bookSchema, Class.forName("bookstore.BookImpl"));

      // Adding properties
      addSlot(priceSchema, PRICE_VALUE, doubleSchema, 0, 1);
      addSlot(priceSchema, PRICE_CURRENCY, stringSchema, 0, 1);

      addSlot(tradeableSchema, TRADEABLE_PRICE, priceSchema, 0, 1);

      addSlot(bookSchema, BOOK_TITLE, stringSchema, 0, 1);
      addSlot(bookSchema, BOOK_AUTHORS, stringSchema, 0, -1);

      // Adding parents
      bookSchema.addSuperSchema(tradeableSchema);

    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

V. Conclusions

The OWLBeans toolkit we presented in this paper ease the access to semantically annotated information by software agents. Its main functionality is to generate JavaBeans and other artefacts, that can be used by agents needing just an object-oriented model of their application domain.

Given its modular design, the toolkit is able to process various kinds of input and produce different outputs. So, while the main purpose is to extract relations from an OWL ontology and generate JavaBeans, it can also be used to perform all other conversions allowed by combining available readers and writers.

Possible improvements include a better management of name conflicts that can arise while converting properties from an object oriented system to an ontology, where their scope is not limited to a single class. A new reader should be added to build an ontology model, using Java reflection to analyze a package of Java classes and extract needed metedata.

Above all, some relations among ontologies should be recognized and handled. In fact, having a hierarchy of ontologies, with terms of an ontology referencing terms of parent ontologies, is quite common.

References

[1] *The Agentcities Network*. http://www.agentcities.net/

[2] *Agentcities.RTD*. http://www.agentcities.org/EURTD/

[3] *BeanShell*. http://www.beanshell.org

[4] Berners-Lee, Tim. Hendler, James, Lassila, Ora. *The Semantic Web*, Scientific American, May 2001.

[5] Berners-Lee, Tim *Semantic Web Road map*, September, 1998. Available from http://www.w3.org/DesignIssues/Semantic.html

[6] FIPA spec. XC00008. *FIPA SL Content Language Specification*. Available from http://www.fipa.org/specs/fipa00008/

[7] Available from FIPA spec. XC00023. *FIPA Agent Management Specification*. Available from http://www.fipa.org/specs/fipa00023/
[8] FIPA spec. XC00037. *FIPA Communicative Act Library Specification*. Available from http://www.fipa.org/specs/fipa00037/
[9] Hendler, James, Berners-Lee, Tim and Miller, Eric *Integrating Applications on the Semantic Web*, Journal of the Institute of Electrical Engineers of Japan, Vol 122(10): 676-680, 2002.
[10] *JADE*. Available from http://jade.tilab.it
[11] *Janino*. http://janino.net
[12] *JavaBeans*. http://java.sun.com/products/javabeans/
[13] *Jena Semantic Web Framework*. http://jena.sourceforge.net/
[14] *Java Server Pages*. http://java.sun.com/products/jsp/
[15] *OWL*. http://www.w3c.org/OWL
[16] http://www.w3.org/TR/2003/WD-owl-semantics-20030203/syntax.html
[17] *Protégé Ontology Editor and Knowledge Acquisition System.* http://protege.stanford.edu/
[18] *Protégé OWL Plugin - Ontology Editor for the Semantic Web.* http://protege.stanford.edu/plugins/owl/
[19] *Velocity* http://jakarta.apache.org/velocity/
[20] *W3C Web Pages* on *Semantic Web*. http://www.w3.org/2001/sw/
[21] *XML Schema* http://www.w3.org/XML/Schema
[22] Brooks, R.A., *How to build complete creatures rather than isolated cognitive simulators*, in K. VanLehn (ed.), *Architectures for Intelligence*, pp. 225-239, Lawrence Erlbaum Assosiates, Hillsdale, NJ, 1991.

Spatial Computing: the TOTA Approach

Marco Mamei, Franco Zambonelli

DISMI - Università di Modena e Reggio Emilia
Via Allegri 13, 42100 Reggio Emilia – ITALY
mamei.marco@unimore.it, franco.zambonelli@unimore.it

Abstract. *Spatial abstractions promise to be basic necessary ingredients for a novel "spatial computing" approach to distributed systems development and management, suitable to tackle the complexity of modern distributed computing scenarios and promoting self-organization and self-adaptation. In this paper, we analyze the key concepts underlying spatial computing and show how they can be organized around a sort of "spatial computing stack", in which a variety of apparently very diverse mechanisms and approaches can be properly framed. Following, we present our current research work on the TOTA middleware as a representative example of a general-purpose approach to spatial computing. In particular, we discuss how TOTA can be exploited to support the development and execution of self-organizing and self-adaptive spatial computing applications.*

1. Introduction

During the nineties, most researches in distributed computing have focused on the "network of workstations" scenario [CouDK94]. However, in the past few years, a number of novel scenarios have emerged including: *(i)* micro-networks, i.e., networks of low-end computing devices typically distributed over a geographically small area (e.g., sensor networks [Est02], smart dusts [Pis00] and spray computers [Zam04]); *(ii)* ubiquitous networks, i.e., networks of medium-end devices, distributed over a geographically bounded area, and typically interacting with each other via short/medium range wireless connections (pervasive computing systems and smart environments [GelSB02] and cooperative robot teams); *(iii)* global networks, characterized by high-end computing systems interacting at a world-wide scale (the physical Internet, the Web, P2P networks [RipIF02] and multiagent systems ecologies [Kep02].

Despite clear dissimilarities in structure and goals, one can recognize some key common characteristics distinguishing the above scenarios from more traditional ones:

- **Large Scale**: the number of nodes and, consequently, the number of components involved in a distributed application is typically very high and, due to decentralization, hardly controllable. It is not possible to enforce a strict control over their configuration (consider e.g., the nodes of a P2P network) or to directly control them during execution (consider e.g., the nodes of a sensor network distributed in a landscape).

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

- **Network dynamism**: the activities of components will take place in network whose structure derives from an almost random deployment process, likely to change over time with unpredictable dynamics. This may be due to factors such as environmental contingencies, failures (very likely e.g., in sensor networks and pervasive computing systems), and mobility of nodes (as e.g. in robot teams and in networks of smart appliances). In addition, at the application level, software components can be of an ephemeral or temporary nature (as e.g. the peers of a P2P network).
- **Situatedness:** The activities of components will be strongly related to their location in either a physical or a virtual environment. On the one hand, situatedness can be at the very core of the application goal (as e.g. in sensor networks and pervasive computing systems devoted to improve our interaction with the physical world). On the other hand, situatedness can relate to the fact that components can take advantage of the presence of a structured virtual environment to organize the access to distributed resources (as e.g., in P2P data sharing networks).

The first two characteristics compulsory require systems to exhibit – both at the network and at the application level – properties of self-organization and self-adaptation (or generally, "self-*" properties). In fact, if the dynamics of the network and of the environment compulsory require dynamic adaptation, the impossibility of enforcing a direct control over each component of the system implies that such adaptation must occur without any human intervention, in an autonomic way. The last characteristic calls for an approach that elects the environment, its spatial distribution, and its dynamics, to primary design dimensions. In any case, the three aspects are strictly inter-related, in that the enforcement of self-* properties cannot abstract from the capability of the system to become "context-aware", i.e., to have components perceive the local properties of the environment in which they are situated and adapt their behavior accordingly.

In the past few years, a variety of solutions exploiting specific self-* properties to solve specific application problems for large-scale systems in dynamic networks are being proposed [Dim04]. The question of whether it is possible to devise a single unifying conceptual approach, applicable with little or no adaptations to a variety of application problems and to scenarios as diverse as P2P networks and local networks of embedded sensors, is still open.

In this paper, we identify the important role that will likely be played in that process by spatial abstractions, and by their adoption as building blocks for a novel general-purpose "spatial computing" approach for distributed system development and management. A spatial computing approach – by abstracting the network as a continuum space and by having application level activities expressed in terms of sensing the properties of space and navigating in it – can effectively deal with network dynamics in large scale systems, can facilitate the integration of variety of self-* properties in distributed systems, and also suit systems whose activities are situated in an environment.

The remainder of this paper elaborates on spatial computing and is organized as follows. Section 2 introduces the basic concepts underlying spatial computing and discusses their relations with self-* properties. Section 3 proposes a framework around which to organize the basic abstractions and mechanisms involved in spatial computing. Section 4 presents our current research work on the TOTA middleware, as a representative example of a general-purpose approach to spatial computing. Section 5 concludes by sketching a rough research agenda in the area.

2. Spatial Computing

The key principles underlying spatial computing are that:

(i) the central role of the network – a discrete system of variously interconnected nodes – evolves into a concept of space – i.e., an abstraction of a metric continuum built over the network;

(ii) all application-level activities are abstracted as taking place in such space, and rely on the capability of application components of locally perceiving (and possibly influencing) the local properties of space;

In particular, in spatial computing, any type of networked environment is hidden below some of virtual metric n-dimensional space, mapped as an overlay over the physical network. The nodes of the network are assigned a specific area of the virtual space, and are logically connected to each other accordingly to the spatial neighborhood relations. Accordingly, each and every entity in the network, being allocated in some nodes of the network, is also automatically situated in a specific position in space.

In this way, components in the network become "space-aware". On the one hand, they perceive their local position in space as well as the local properties of space (e.g., the locally available data and services) and possibly change them. On the other hand, the activities of components in that space are related to some sort of "navigation" in that space, which may include moving themselves to a specific different position of space or moving data and events in space according to "geographical" routing algorithms. The primary way to refer to entities in the network is thus by "position", i.e., any entity is characterized by being situated in a specific position in the physical space.

The above characteristics notably distinguish spatial computing from traditional distributed computing models. In *transparent* distributed computing models [CouDK94, ChiC91], components are identified by logical names, applications abstract from the presence of a distributed environment, and only a priori known interaction patterns can be effectively supported. This makes them unable to deal with large-scale systems and with network dynamics. In network-aware models [Wal97], components are typically aware of executing in a network and are identified by their location in it (e.g.., the IP). This enables dealing also with applications executing in large-scale networks, but still call for an explicit and complex handling of dynamic changes in the network or in the position of components. Neither of the two promotes suitable abstractions of environment.

Spatial computing overcomes the above limitations in a very effective way:

- **Large scale**: the size of a network does not influence the models or the mechanisms, which are the same for a small network and for a dramatically large one.
- **Network dynamics**: the presence of a dynamic network is not directly perceived by components, being hidden behind a stable structure of space that is maintained despite network dynamism.
- **Situatedness**: the abstraction of space is a conceptually simple abstraction of environment, which also perfectly matches the needs of those systems whose activities are strictly intertwined with a physical or computational environment.

In addition, as discussed in the following sub-section, spatial computing promotes and support self-* computing.

3.1 Self-* Properties in Spatial Computing

Self-* properties, including the capability of a distributed system of self-configuring its activity, self-inspecting and self-tuning its behavior in response to changed conditions, or self-healing it in the presence of faults, are necessary for enabling spatial computing and, at the same time, are also promoted by the adoption of a spatial computing model.

On the one hand, to enable a spatial computing model, it is necessary to envision mechanisms to build the appropriate overlay spatial abstraction and to have such spatial abstraction be coherently preserved despite network dynamics. In other words, this requires the nodes of a network to be able to autonomously connect with each other, set up some sort of common coordinate systems, and self-position themselves in such space. In addition, this requires the nodes of the network to be able to self-reorganize their distribution in the virtual space so as to *(i)* make room for new nodes joining the network (i.e., allocate a portion of the virtual space to these nodes); *(ii)* fill the space left by nodes that for any reason leave the network; *(iii)* re-allocate the spatial distribution of nodes to react to node mobility. It is also worth outlining that, since the defined spatial structure completely shields the application from the network, it is also possible for a system to dynamically tune the structure of the space so as enforce some sorts of self-management of the network, transparently to the higher application levels. As an example, load unbalances in the network can be dynamically dealt, transparently from the application level, by simply re-organizing the spatial structure so as to have overloaded nodes occupy a more limited portion of the space.

On the other hand, the so defined spatial structure can be exploited by application level components to organize their activities in space in an autonomous and adaptive way. First of all, it is a rather assessed fact that “context-awareness” and “contextual activity”, i.e., the capabilities of a component to perceive the properties of the operational environment and of influencing them, respectively, are basic ingredients to enable any form of adaptive self-organization and to establish the necessary feedback promoting self-adaptation. In spatial computing, this simply translates in the capability of perceiving the local properties of space, which in the end reflect some specific characteristics of either the network or of some application-level characteristics and of changing them. Second, one should also recognize that the vast majority of known phenomena of self-organization and self-adaptation in nature (from ant-foraging to reaction-diffusion systems, just to mention two examples in biology and physics) are actually phenomena of self-organization in space, emerging from the related effect of some “component” reacting to some property of space and, by this reaction, influencing at its turn the properties of space. Clearly, a spatial computing model makes it rather trivial to reproduce in computational terms such types of self-organization phenomena, whenever they may be of some use in a distributed system.

1.1 Examples of Spatial Computing Approaches

The shift towards spatial computing is an emerging trend in diverse scenarios.

As an example, consider a sensor network scenario with a multitude of wireless sensors randomly deployed in a landscape to perform some monitoring of environmental conditions [Est02]. There, all activities of sensors are intrinsically of a spatial nature. First, each sensor is devoted to local monitoring a specific portion of the physical space (that it can reach with its sensing capabilities). Second, components must coordinate with

each other based on their local positions, rather than on their IDs, to perform activities such as detecting the presence and the size of pollution clouds, and the speed of their spreading in the landscape. All of this implies that components must be made aware of their relative positions in the spatial environment by self-constructing a virtual representation of the physical space [NagSB03]. Moreover, they can take advantage of "geographical" communication and routing protocols: messages and events flow towards specific position of the physical/virtual space rather than towards specific nodes, thus surviving in an self-adaptive way the possible dismissing of some nodes [RaoP03].

Another example in which spatial concepts appear in a less trivial way is world-wide P2P computing. In P2P computing, an overlay network of peers is built over the physical network and, in that networks, peers act cooperatively to search specific data and services. In first generation P2P systems (e.g., Gnutella [RipIF02]), the overlay network is totally unstructured, being built by having peers randomly connect to a limited number of other peers. Therefore, in these networks, the only effective way to search for information is message flooding. More recent proposals [Rat01] suggest structuring the network of acquaintances into specific regular "spatial shapes", e.g., a ring or an N-dimensional torus. When a peer connects to the networks, it occupies a portion of that spatial space, and networks with those other peers that are neighbors accordingly to the occupied position of space. Then, data and services are allocated in specific positions in the network (i.e., by those peers occupying that position) depending on their content/description (as can be provided by a function hashing the content into specific coordinates). In this way, by knowing the shape of the network and the content/description of what data/services one is looking for, it is possible to effectively navigate in the network to reach the required data/services. That is, P2P networks define a spatial computing scenario in which all activities of application components are strongly related to self-positioning themselves and navigating in an abstract metric space. It is also worth outlining that recent researches promote mapping such spatial abstractions over the physical Internet network so as to reflect the geographical distribution of Internet nodes (i.e., by mapping IP addressed into geographical physical coordinates [Row04]) and, therefore improve efficiency.

In addition to the above examples, other proposals in areas such as pervasive computing [Bor04] and self-assembly [MamVZ04] explicitly exploit spatial abstractions (and, therefore, a sort of spatial computing model) to organize distributed activities.

3. Framing Spatial Computing

Let us now have a more systematic look at the basic mechanisms that have been exploited so far in distributed computing to promote self-* properties in distributed systems. We will show that most of these mechanisms can be easily interpreted and mapped into very similar spatial concepts, and that they can be framed in a unifying flexible framework.

3.1. A Spatial Computing Stack

In this section, we introduce the "space-oriented" stack of levels (see Figure 1) as a framework for spatial computing mechanisms. In each level of the stack, by introducing a new paradigm rooted on spatial concepts, it is possible to interpret a lot of proposed self-*

approaches, in different scenarios, in terms of mechanisms to manage and exploit the space (see Table 1). On this basis, it is likely that a simply unifying model for self-* distributed computing – leading to a single programming model and methodology and – can be actually identified.

The "*physical level*" deals on how components start interacting – in a dynamic and spontaneous way – with other components in the systems. This is a very basic expression of self-organizing behavior which is a pre-requisite to support more complex forms of autonomy and of self-organization at higher levels. To this end, the basic mechanism exploited is broadcast (i.e. communicate with whoever is available). Radio broadcast is used in sensor networks and in pervasive computing systems, and different forms of TCP/IP broadcast (or of dynamic lookup) are used as a basis for the establishment of overlay networks in wide area P2P computing. Whatever the case, this physical level can be considered as in charge of enabling a component of a dynamic network application to get into existence and to start interacting with each other.

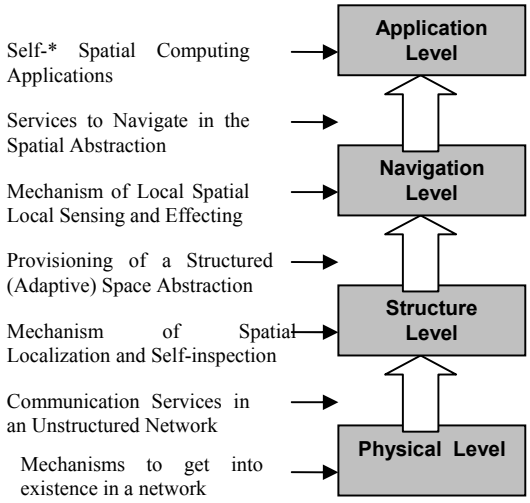


Figure 2. A Spatial Computing Stack.

The "*structure level*" is the level at which a spatial structure is built and maintained by components existing in the physical network. The fact that a system is able to create a stable spatial structure capable of surviving network dynamics and adapting the working conditions of the network is an important expression of self-organizing and self-adapting behavior *per se*. However, such spatial structure is not a goal for the application, and it is instead used as the basic spatial arena to support higher levels activities.

The various mechanisms that are used at the structure level in different scenarios are – again – very similar to each other. Sensor networks as well as self-assembly systems typically structure the space accordingly to their positions in the physical space, by exploiting mechanisms of geographical self-localization. Pervasive computing systems, in addition to mechanisms of geographical localization, often exploit logical spatial structures reflecting some sorts of abstract spatial relationships of the physical world (e.g.,

rooms in a building) [Bor04]. Global scale systems, as already anticipated, exploits overlay networks built over a physical communication network.

The "*navigation level*" regards to the basic mechanisms that components exploit to orient their activities in the spatial structure and to sense and affect the local properties of space. If the spatial structure has not any well-defined metric, the only navigation approaches are flooding and gossiping. However, if some sort of metric structure is defined at the structure level (as, e.g., in the geographical spatial structures of sensor networks or in metric overlay networks) navigation approaches relate in following the metrics defined at the structure level. For instance, navigation can imply the capability of components to reach specific points (or of directing messages and data) in the space based on simple geometric considerations as in, e.g., geographical routing [BosM01].

Starting from the basic navigation capability, is also possible to enrich the structure of the space by propagating additional information to describe "something" which is happening in that space, and to differentiate the properties of the space in different areas. One can say that the structure of space may be characterized by additional types of spatial structures propagating in it, and that components may direct their activities based on navigating these additional structures. In other words, the basic navigation capabilities can be used to build additional spatial structures with different navigation mechanisms. Typical mechanisms exploited at these additional levels are computational fields and pheromones. Despite the different inspiration of the two approaches (physical versus biological), we emphasize that they can be modeled in a uniform way, e.g., in terms of time-varying properties defined over a space [MamZ03]. The basic expression of self-organization that arises here derives from the fact that the structures propagated in the space – and thus the navigation activity of application components – are updated and maintained to continuously reflect the actual structure and situation of the space.

At the "*application level*", navigation mechanisms are exploited by application components to interact and organize their activities. Applications can be conveniently built on the following self-organizing feedback loop: *(i)* having components navigate in the space (i.e., discriminating their activities depending on the locally perceived structure and properties of the space) and *(ii)* having components, at the same time, modifying existing structure due to the evolution of their activities.

Depending on the types of structures propagated in the space, and on the way components react to them, different phenomena of self-organization can be achieved and modeled. For example, processes of morphogenesis (as needed in self-assembly, modular robots and mobile robotics), phenomena mimicking the behavior of ant-colonies and of flocks, phenomena mimicking the behavior of granular media and of weakly correlated particles, as well as a variety of social phenomena, can all be modeled in terms of:

- entities getting to existence in a space;
- having a position in a structured space and possibly influencing its structure;
- capable of perceiving properties spread in that space;
- capable of directing their actions based on perceived properties of such space and capable of acting in that space by influencing its properties at their turn.

	MICRO NETWORKS **Nano Networks, Sensor Networks, Smart Dust, Self-Assembly, Modular Robots**	UBIQUITOUS NETWORKS **Home Networks, MANETs, Pervasive Environments, Mobile Robotics**	GLOBAL NETWORKS **Internet, Web, P2P networks, multiagent systems**
"Application" Level (exploiting the spatial organization to achieve in a *self-organizing and adaptive way* specific app. goals)	Spatial Queries Spatial Self-Organization and Differentiation of Activities Spatial Displacement Motion Coordination & pattern formation DATA: environmental data	Discovery of Services Spatial Displacement Coordination and Distribution of Task and Activities Motion coordination & pattern formation DATA: local resources and environmental data	P2P Queries as Spatial Queries in the Overlay Motion Coordination on the Overlay Pattern formation (e.g., for network monitoring) DATA: files, services, knowledge
"Navigation" Level (dealing with the mechanism exploited by the entities living in the space to *direct activities and movements in that space*)	Flooding Gossiping (random navigation) Geographical Routing (selecting and reaching specific physical coordinates) Directed Diffusion (navigation following sorts of computational fields) Stigmergy (navigation following pheromone gradients)	Computational fields Multi-hop routing based on Spanning Trees Pattern-matching and Localized Tuple-based systems	Flooding Gossiping (random navigation) Metric-based (moving towards specific coordinates in the abstract space) Gossiping (random navigation) Stigmergy (navigation following pheromone gradients distributed in the overlay network)
"Structure" Level (dealing with mechanisms and policies to adaptively *shape a metric space* and let components find their position in that space)	Self-localization (beacon-based triangulation)	Self-localization (Wi-Fi or RFID triangulation) Definition and Maintenance of a Spanning Tree (as a sort of navigable overlay)	Establishment and Maintenance of an Overlay Network (for P2P systems) Referral Networks and e-Institutions (for multiagent systems)
"Physical" Level (dealing with the mechanism to *interact*)	Radio Broadcast Radar-like localization	Radio Broadcast RF-ID identification	TCP broadcast – IP identification Directed TCP/UDP messages Location-dependent Directory services

Table 1. Spatial Mechanisms in Modern Distributed Computing Scenarios

4.2 Multiple Spaces and Nested Spaces

In general, different scenarios and different application problems may require different perceptions of space and different spatial structures. For instance, a world-wide resource-sharing P2P network over the Internet may require – for efficiency reason – a 2-D spatial abstraction capable of reflecting the geographical distribution of Internet nodes over the earth surface. On the other hand, a P2P network for social interactions may require a spatial abstraction capable of aggregating in close regions of the virtual space users with similar interests. Also, one must consider that in the near future, the different network scenarios we have identified will be possibly part of a unique huge network (consider that IPv6 addressing will make it possible to assign an IP address to each and every square millimeter on the earth surface). Therefore, it is hard to imagine that a unique flat spatial abstraction can be effectively built over such a network and satisfy all possible management and application needs.

With this regard, the adoption of the spatial computing paradigm does not prescribe at all to adopt the same set of mechanisms and the same type of spatial structure for all networks and for applications. Instead, being the spatial structure a virtual one, it is possible to conceive both *(i)* the existence, over the same physical network, of multiple complimentary spatial abstraction independently used by different types of applications; and *(ii)* the existence of multiple layers of spatial abstractions, built one over the other in a multi-layered system.

With regard to the former point, in addition to the example of the different types of P2P networks calling for different types of spatial abstractions, one could also think at how different problems such as Internet routing, Web caching, virtual meeting points, introduce very different problems and may require the exploitation of very different spatial concepts.

With regard to the latter point, one can consider two different possibilities. Firstly, one can think at exploiting a first-level spatial abstractions (and the services it provides) to offer a second-level spatial abstraction enriching it with additional specific characteristics. For examples, one can consider that a spatial abstraction capable of mapping the nodes of the Internet into geographical coordinates can be exploited, within a campus, to build an additional overlay spatial abstraction mapping such coordinates into logical location (e.g., the library, the canteen, the Computer Science department and, within it, the office of Prof. Zambonelli). Such additional spatial abstraction could then be used to build semantically-enriched location dependent services. Secondly, one could think at conceiving a hierarchy of spatial abstractions that provides different levels of information about the space depending on the level at which they are observed, the same as the information we get on a geographical region are very different depending on the scaling of the map on which we study it. As an example, we can consider that the spatial abstraction of a wide-area network can map a sensor network – connected to the large network via a gateway – as a "point" in that space, and that the distributed nature of the sensor networks (with nodes having in turn a specific physical location in space) becomes apparent only when some activity takes place in that point of space (or very close to it).

4. TOTA: a Middleware Approach to Spatial Computing

The ambitious goal of a uniform modeling approach capable of effectively capturing the basic properties of self-organizing computing, and possibly leading to practical and useful general-purpose modeling and programming tools, is far from close. Earlier in this paper we have strongly advocated the generality, flexibility, and modularity of a spatial computing approach. Although we have do not have the ultimate proof that spatial computing can be effectively put to practice and fulfill all its promises, our experience in spatial computing with the TOTA [MamZ04] middleware can support in part our claims.

The TOTA middleware (short for "Tuples On The Air"), gathers concepts from both tuple space approaches [Cab03, MamZL04] and event-based ones [Car01, Jini] and extends them to provide applications with simple and flexible mechanisms to create, self-maintain, and exploit at the application level a variety of spatial structures, implemented by means of distributed tuples. Unlike traditional shared data space models, tuples are not associated to a specific node (or to a specific data space) of the network. Instead, tuples are injected in the network and can autonomously propagate and diffuse in the network

accordingly to a specified pattern.

To support this idea, the typical scenario of a TOTA application is that of a peer-to-peer network of possibly mobile nodes, each running a local version of the TOTA middleware. Each TOTA node holds references to a limited set of neighboring nodes and can communicate directly only with them.

Upon the distributed space identified by the dynamic network of TOTA nodes, each component is capable of locally storing tuples and letting them diffuse through the network. Tuples are injected in the system from a particular node, and spread hop-by-hop accordingly to their propagation rule. In fact, a TOTA tuple is defined in terms of a "content", and a "propagation rule". **T=(C,P)**. The content **C** is an ordered set of typed fields representing the information carried on by the tuple. The propagation rule **P** determines how the tuple should be distributed and propagated across the network. This includes determining the "scope" of the tuple (i.e. the distance at which such tuple should be propagated and possibly the spatial direction of propagation) and how such propagation can be affected by the presence or the absence of other tuples in the system. In addition, the propagation rules can determine how the content of a tuple should change while it is propagated. Tuples are not necessarily distributed replicas: by assuming different values in different nodes, tuples can be effectively used to build a distributed data structure expressing contextual and spatial information. So, unlike traditional event based models, propagation of tuples is not driven by a publish-subscribe schema, but it is encoded in tuples' propagation rule and, unlike an event, can change its content during propagation (see figure 3).

Distributed tuples must be maintained coherent despite network dynamism. To this end, the TOTA middleware supports tuples propagation actively and adaptively: by constantly monitoring the network local topology and the income of new tuples, the middleware automatically re-propagates tuples as soon as appropriate conditions occur. For instance, when new nodes get in touch with a network, TOTA automatically checks the propagation rules of the already stored tuples and eventually propagates the tuples to the new nodes. Similarly, when the topology changes due to nodes' movements, the distributed tuple structure automatically changes to reflect the new topology.

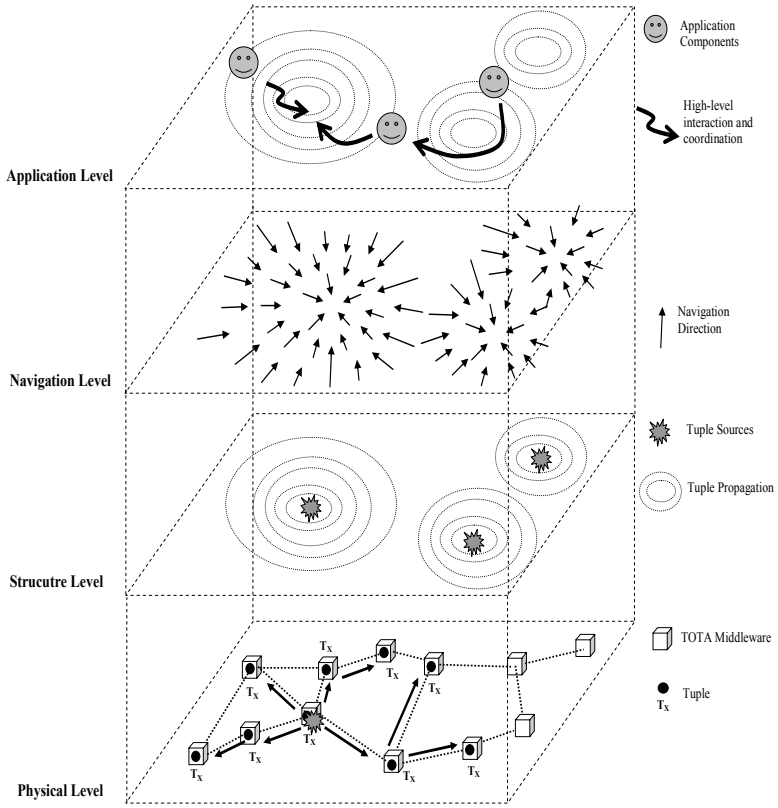


Figure 3: The General Scenario of TOTA in the spatial computing stack: at the physical level there is the network, communication is broadcast of messages encoding TOTA tuples. At the structure level, the space is represented by means of the TOTA distributed tuples. At the navigation level spatial structures can provide basic navigation directions. At the Application level complex coordination tasks can be achieved.

The TOTA middleware supports the spatial computing stack introduced in section 4. In fact, from the application components' point of view, executing and interacting basically reduces to create distributed spatial structures in the network (inject tuples), navigate such spatial structures (sense tuples in a neighborhood), and act accordingly to some application-specific policy.

To clarify and ground the discussion we introduce the following exemplary pervasive computing case study application: tourists with wireless PDAs visit a museum provided with an embedded computer network. We suppose that the PDAs and the embedded devices run the TOTA middleware and that they connect with each other forming a multi-hop mobile wireless network. In the following subsections, working on this case study application, we will detail how TOTA deals with all the levels in the spatial computing stack.

4.1. Physical Level

The physical level deals with how components find and start communicating with each other. At this level, the specific nature of the network scenario has an important role. Since our primary focus is pervasive computing, we mainly consider a wireless network scenario without long-range routing protocols available (like in a “bare” mobile ad-hoc network). In such scenario, it is easy to identify the node's neighborhood with the network local topology (e.g. all the nodes within 10m, for a Bluetooth network). In this case, a TOTA node detects in-range nodes via one-hop message broadcast.

Turning the attention to the case study, each PDA detects neighbor devices, by broadcasting and receiving “here I am” messages. Such discovery operations is executed periodically to take into account the possible movements of users. Upon injecting a tuple, the TOTA middleware broadcasts the tuple to its current neighbors. There, the tuple will be recursively broadcasted hop-by-hop to travel across the network, accordingly to its propagation rule.

To support our experiments, we developed a first prototype of TOTA running on HP IPAQs 36xx equipped with 802.11b wireless card, Familiar LINUX and J2ME-CDC (Personal Profile). IPAQs connect locally in the MANET mode (i.e. without requiring access points) creating the skeleton of the TOTA network. Tuples are being propagated through multicast sockets to all the nodes in the one-hop neighborhood. The use of multicast sockets has been chosen to improve the communication speed by avoiding 802.11b unicast handshake. By considering the way in which tuples are propagated, TOTA is very well suited for this kind of broadcast communication. We think that this is a very important feature, because it will allow in the future implementing TOTA also on really simple devices (e.g. micro mote sensors [Pis00]) that cannot be provided with sophisticate communication mechanisms.

It is important to remark that, despite our focus to wireless networks and pervasive computing, the TOTA mechanisms are general and independent from the underlying physical network. For example, in an Internet scenario (where a long-range routing protocol is available), TOTA identifies the neighborhood of a node with the nodes whose IP address is known (a node can communicate directly with another, only if it knows the other node's address). To realize neighbors discovery, TOTA can either download from a well-known server the list addresses representing its neighbors or it can start an expanding-ring search to detect close nodes [RipIF02]). Given that, the multi-hop propagation of a tuple proceeds as previously described.

4.2. Structure Level

TOTA tuples create a “structure of space” in the network. At the basic level, once a tuple is injected from a node and propagates across the network, it creates a source-centered spatial structure identifying some spatial features relative to the source.

For example, a tuple incrementing one of its fields as it gets propagated identifies a spatial structure defining the network distances from the source. This kind of structure of space provides spatial awareness to application agents. In fact, an agent is both able to infer its approximate distance from the source (in terms of hops – i.e. network link range), and the direction of the source by looking at where the gradient of the tuple descends.

Moreover, TOTA allows to combine different tuples to create more complex spatial representations. A particularly significant example of these mechanisms is the creation of

shared coordinate systems in the network on the basis of mere connectivity. Localization, in general, can rely on the (geometrically intuitive) fact that the position of a point on a surface can be uniquely determined by measuring its distance from at least three non-aligned reference points ("beacons"), via a process of "triangulation" [NagSB03]. Implementing such localization mechanism in TOTA is rather easy. *(i)* A leader election algorithm can elect three beacons nodes. *(ii)* Each beacon "arbitrarily" locates at specific coordinates (without external location information the coordinate system can only be internally coherent [NagSB03]). *(iii)* Each beacon injects a TOTA tuple, increasing its content hop-by-hop and marked with the beacon coordinates. As previously pointed out, this tuple allows other nodes to estimate their distance from the beacon. *(iv)* After at least three beacons had propagated their ranging tuples, nodes can apply a triangulation algorithm to infer their coordinates. Moreover, since TOTA tuples self-maintain, the coordinate system remains up to date and coherent despite network dynamism. If upon a node movement the topology of the network changes, the tuples maintenance triggers an update in the coordinate system, making the latter robust.

A shared coordinate system provides a powerful spatial structure in a network and allows to realize complex navigation and coordination tasks (see later).

In addition, although at the primitive level the space is the network space and distances are measured in terms of hops between nodes, TOTA allows to exploit a much more physically-grounded concept of space.

This may be required by several pervasive computing scenarios in which application agents need to interact with and acquire awareness of the physical space. For instance, one can bound the propagation of a tuple to a portion of physical space by having the propagation procedure - as the tuple propagates from node to node - to check the local spatial coordinates, so as to decide whether to further propagate the tuple or not. In order to bound agents' and tuples' behavior to the physical space, nodes must be provided with some kind of localization mechanism [HigB01]. From our perspective, such mechanisms can be roughly divided into two categories:

- A GPS-like localization mechanism provides absolute spatial information (e.g. it provides latitude and longitude of a node in the network). An actual GPS (Global Positioning System) getting spatial coordinates from satellites naturally belongs to this category. Beacon-based signal triangulation (coupled with beacons actual physical location) is anther example of this category (nodes get their coordinates in an absolute coordinate-frame defined by the beacons [NagSB03])
- A RADAR-like localization mechanism provides local information (e.g. relative distances and orientations between nodes). An actual radar or sonar device belongs to this category (radio and sound waves reflected by neighbor devices enable to infer their distance and orientation). A videocamera installed on a node can serve the same purpose (processing the image coming from the camera, a node can infer where other nodes are). Also network roundtrip-time and signal-strength attenuation may serve this purpose.

The kind of localization mechanism being available strongly influences how nodes can express and use spatial information. GPS-like mechanism are more suitable at defining "absolute" regions. For example, they allow to easily create tuples that propagate across a region defined by means of the coordinates of its corners (e.g. propagate in the square area defined by (0,0) and (100,100)). RADAR-like mechanism are more suitable at

defining “relative” regions, where for example tuples are constrained to travel north form the source or within a specified distance.

It is fair to report that a similar idea has been developed and exploited in the context of a recently proposed language to program a vast number of devices dispersed in an environment [Bor04]. The idea of this programming language is to identify a number of spatial regions relevant for a given application and to access the devices through the mediation of these regions (e.g. for all the devices on the “hill” do that). In [Bor04], the definition of the regions is performed adopting GPS devices and distributed data structures similar to TOTA tuples.

Other than the network and the physical space, one could think at mapping the peers of a TOTA network in any sort of virtual space. This space must be supported by an appropriate routing mechanism allowing distant peers to be neighbors in the virtual space. Such virtual spaces are particularly useful and enable the definition of advanced application such as content-based routing, as in CAN [Rat01]. TOTA concretely supports the definition of these kinds of applications. Also in this case it is fair to report that similar principles have been used in the Multilayered Multi Agent Situated System (MMASS) model [BanMV04]. In MMASS agents' actions take place in a multilayered environment. Each layer provides agents with some contextual information supporting agents' activities. The MMASS environment is thus a hierarchy of virtual spaces built upon one another, where lower layers provide the routing infrastructure for upper ones.

4.3. Navigation Level

TOTA defines a set of API to allow application components to sense TOTA tuples in their one-hop neighborhood and to locally perceive the space defined by them. Navigation in the space consists in having agents act on the basis of the local shape of specific tuples.

As a first simple example we can consider physical navigation. Turning the attention to our case study, it is clear that a PDA injecting a hop-increasing tuple in the network, becomes immediately reachable by other users. Users, in fact, can move following the gradient of the tuple downhill, to reach the tuple source. Moreover, since the tuple shape is maintained despite network dynamism, users can reach the source of a tuple even if it moves.

Navigation is not related to physical movement only. TOTA allows to relate the propagation of a tuple to other tuples already propagated (e.g. a tuple can propagate following another tuple). This can be at the basis of the routing algorithm detailed in the following [Poo00]. In very general terms, when a node “A” wants to send a message to a node “B”, it actually injects the network with a TOTA tuple, that holds: the source identifier i.e. “A”, the message, and the number of hops from the source of the message to the current node. Such structure not only trivially hand-off the message to “B”, but creates a path leading to “A” that can be exploited for further uses. If node “B” wants to reply, it can just send a message that follows the “A”-field downhill towards node “A”. In this case no flooding is involved. The field-like distributed data structures created in this process, can be used further also by other peers to communicate.

Complex spaces enable advanced navigation strategies. A shared coordinate system, like the one described in the previous section, allows, for example, to set-up geographic routing algorithm [BosM01]. A geographic routing algorithm is a mechanism that takes

advantage of the established coordinate frame to send messages to the node closer to a specific location. Such algorithm is suitable in a lot of application scenarios because it inherently supports communication decoupling in that senders and receivers are decoupled by the coordinate frame. For example, a sender can send a message to an unknown receiver located at a specific location and the message will be received by whoever is closer to that location.

4.4. Application Level

The spatial abstractions and tools promoted by TOTA enable to easily realize complex coordination tasks in a robust and flexible way.

Our research, up to now, has mainly focused on the problem of enabling a group of agents to coordinate their respective movements (i.e. distributed motion coordination). Specifically, considering our case study, we focus on how tourists can be supported in planning their movements across a possibly large and unfamiliar museum and in coordinating such movements with other, possible unknown, tourists. Such coordination activities may include scheduling attendance at specific exhibitions occurring at specific times, having a group of students split in the museum according to teacher-specific laws, helping a tourist to avoid crowd or queues, letting a group of tourist to meet together at a suitable location, and even helping to escape accordingly to specific evacuation plans.

An intriguing possibility to realize motion coordination is to take inspiration from the physical world, and in particular from the way masses in our universe move accordingly to the gravitational field. By interpreting (rather roughly) the General Relativity Theory, we can say that the gravitational field actually changes the structure of the space letting particles to globally self-organize their movements. Under this interpretation, particles achieve their "tasks" by simply following the structure of the space.

Realizing this kind of idea with the spatial abstraction promoted by TOTA is rather easy. Under the assumption that users spread hop-counting tuples in the network, it is possible to realize several coordination tasks. A group of tourist following downhill each other tuples will collapse in a single location allowing the tourists to meet somewhere in the building. Analogously, museum's guides could decide to sense each other's tuples (i.e. spaces) so as to maintain a certain distance from each other to improve their reachability by tourists. If a guide has to go away, the same tuples would allows the others to automatically and adaptively re-shape their formation.

Following this approach, agents achieve their goals not because of their capabilities as single individuals, but because they are part of an auto-organized system that leads them to the goal achievement. Such characteristics also imply that the agents' activities are automatically adapted to the environmental dynamism, which is reflected in a changing spatial representation, without forcing agents to re-adapt themselves.

Motion coordination with spatial abstractions is by no means limited to the presented case study. It can be applied to a wide range of scenarios ranging from urban traffic management, mobile software agents on Internet and even self-assembly in modular robots (detailed in the following). A modular or self-reconfigurable robot is a collection of simple autonomous mobile robots with few degrees of freedom. A distributed control algorithm is executed by all the robots that coordinate their respective positions to let the robot assume a global coherent shape or a global coherent motion pattern (i.e. gait).

From a methodological viewpoint, robots can exploit spatial abstraction and TOTA

tuples to self-organize their respective positions in space. In particular, starting from any spatial configuration of robots: *(i)* robots start diffusing specific types TOTA tuples; *(ii)* robots react to locally perceived tuples by trying to follow them downhill/uphill, or by changing their activity state possibly depending on the perceived values of the tuples (i.e. depending on their position in some abstract space); *(iii)* changes in the activity state of robots can lead to inhibiting the propagation of some tuples and/or to the diffusion of new types of tuples in the system, leading back to point *(i)*. One can then apply this process several times, with new types of tuples being propagated in different phases, so as to incrementally have robots self-organize into the required shape [MamVZ04].

In all these application scenario, we verified that the spatial abstractions promoted by TOTA effectively support robust and flexible self-organizing behaviors.

5. Conclusions

By abstracting the execution of distributed applications around spatial concepts, spatial computing promises to be an effective approach towards the identification of general and widely applicable self-* approaches to distributed systems development and management. Our experiences with the TOTA middleware confirm the effectiveness of the approach.

However, besides the claims of this paper and our personal experience, much work is needed to asses the potentials of spatial abstractions in distributed computing, and to verify whether they can actually pave the way to a sound and general-purpose approach to self*- computing. In particular:

- Is the spatial computing stack depicted in Table 1 meaningful and useful, or a better and more practical framing can be proposed?
- If and when such a unifying model will be found, will it be possible to translate it into a limited set of programming abstractions and lead to the identification of a practical methodology for developing self-organizing distributed computing systems?
- Is a middleware-centered approach like that of TOTA the best direction to follow?
- Several self-organization phenomena disregarded by this paper, deals with concepts that can be hardly intuitively mapped into spatial concepts. Would exploring some sorts of spatial mapping be still useful and practical? Would it carry advantages?
- Possibly most important of all questions: is the search for a unifying model fueled by enough applications? Or it is rather the search for specific solutions to specific problems the best direction to follow?

In our hope, further researches and a larger variety of studies about self-* properties in distributed systems will soon provide the correct answers to the above questions.

References

[BanMV04] S. Bandini, S. Manzoni, G. Vizzari, “Towards a Specification and Execution Environment for Simulations based on MMASS: Managing at-a-distance Interaction”, *Fourth International Symposium From Agent Theory to Agent Implementation (AT2AI'04)*, Vienna, Austria, 2004.

[Bor04] C. Borcea, “Spatial Programming Using Smart Messages: Design and Implementation”, *24th Int.l Conference on Distributed Computing Systems*, Tokio (J), May 2004.

[BosM01] P. Bose, P. Morin, I. Stojmenovic, J. Urrutia, “Routing with Guaranteed Delivery in Ad Hoc Wireless Networks”, Wirleless Networks 7:609-616, Kluwer Academic Publisher, 2001.

[Cab03] G. Cabri, L. Leonardi, M. Mamei, F. Zambonelli, Location-dependent Services for Mobile Users, *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems And Humans*, Vol. 33, No. 6, pp. 667-681, November 2003

[Car01] A. Carzaniga, D. Rosenblum, A. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service", ACM Transaction on Computer System, 19(3):332-383.

[ChiC91] R. S. Chin, S. T. Chanson, "Distributed Object-Based Programming Systems", *ACM Computing Surveys*, 23(1), March 1991.

[CouDK94] G.Coulouris, J. Dollimore, T. Kindberg, *Distributed Systems. Concepts and Design* Addison-Wesley, second edition, 1994.

[Dim04] G. Di Marzo, A. Karageorgos, O. Rana, F. Zambonelli (Eds.), *Engineering Self-organizing Systems: Nature Inspired Approaches to Software Engineering*, LNCS No. 2977, Springer Verlag, May 2004.

[Est02] D. Estrin, D. Culler, K. Pister, G. Sukjatme, "Connecting the Physical World with Pervasive Networks", *IEEE Pervasive Computing*, 1(1):59-69, 2002.

[GelSB02] H.W. Gellersen, A. Schmidt, M. Beigl, "Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts", *Mobile Networks and Applications*, 7(5): 341-351, Oct. 2002.

[HigB01] Hightower and G. Borriello, "Location Systems for Ubiquitous Computing," *Computer*, vol. 34, no. 8, Aug. 2001, pp. 57–66.

[Jini] JINI, http://www.jini.org

[Kep02] J. Kephart, "Software Agents and the Route to the Information Economy", *Proceedings of the National Academy of Science*, 99(3):7207-7213, May 2002.

[MamVZ04] M. Mamei, M. Vasirani, F. Zambonelli, "Experiments of Morphogenesis in Swarm os Simple Mobile Robots*", Journal of Applied Artificial Intelligence* (to appear) 2004.

[MamZ03] M. Mamei, F. Zambonelli, "Co-Fields: a Unifying Approach to Swarm Intelligence", *3rd Workshop on Engineering Societies in the Agents' Word*, LNCS No. 2677, April 2003.

[MamZ04] M. Mamei, F. Zambonelli, "Programming Pervasive and Mobile Computing Applications with the TOTA Middleware", *2nd IEEE Conference on Pervasive Computing and Communications*, Orlando (FL), IEEE CS Press, March 2004.

[MamZL04] Mamei, M., and F. Zambonelli. 2004b. Co-Fields: a Physically Inspired Approach to Distributed Motion Coordination. *IEEE Pervasive Computing*, 3(2):52-60.

[NagSB03] R. Nagpal, H. Shrobe, J. Bachrach, "Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network", *2nd International Workshop on Information Processing in Sensor Networks*, Palo Alto (CA), April, 2003.

[Pis00] K. Pister, "On the Limits and Applicability of MEMS Technology", *Defense Science Study Group Report*, Institute for Defense Analysis, Alexandria (VA), 2000.

[Poo00] R. Poor, Embedded Networks: Pervasive, Low-Power, Wireless Connectivity, *PhD Thesis, Massachusstes Institute of Technology*, 2001.

[RaoP03] A. Rao, C. Papadimitriou, S. Ratnasamy, S. Shenker, I. Stoica. "Geographic Routing Without Location Information". *ACM Mobicom Conference*. San Diego (CA), USA, 2003.

[Rat01] S. Ratsanamy,, P. Francis, M. Handley, R. Karp, "A Scalable Content-Addressable Network", *ACM SIGCOMM Conference 2001*, Aug. 2001.

[RipIF02] M. Ripeani, A. Iamnitchi, I. Foster, "Mapping the Gnutella Network", *IEEE Internet Computing*, 6(1):50-57, Jan.-Feb. 2002.

[Row04] A. Rowstron et al., "PIC: Practical Internet Coordinates", *24th International Conference on Distributed Computing Systems*, IEEE CS Press, Tokyo (J), May 2004.

[Wal97] J. Waldo et al., "A Note on Distributed Computing", *Mobile Object Systems*, LNCS No. 1222, Feb. 1997.

[Zam04] F. Zambonelli, M.P. Gleizes, R. Tolksdorf, M. Mamei, Spray Computers: Frontiers of Self-organization", *1st International Conference on Autonomic Computing*, IEEE CS Press, New York (I), May 2004.

SIMULATION IN THE TEXTILE INDUSTRY: PRODUCTION PLANNING OPTIMIZATION

Gianluigi Ferraris
University of Turin
Email: ferraris@econ.unito.it

Matteo Morini, corresponding author
University of Turin and LABORatorio 'R. Revelli'
Email: matteo.morini@unito.it

Abstract— **The work being introduced is aimed at supporting the crucial activity of deciding what is to be done, and when, within an industrial, applied, real-world situation. More specifically: matching assorted tasks to applicable production units, and deciding the priority every job is to be given. The problem, common to many different industries, arises when a considerable amount of different articles must be produced on a relatively small number of reconfigurable units. Similar issues have a strong impact on an essential concern, eminently in the textile industrial domain: satisfying the always-in-a-rush customers, while keeping accessory production costs (set-up costs, machinery cleaning costs, ...) under control, keeping at a minimum the losses related to wasteful resource-management practices, due to "under pressure" decision making.**

Given the real-world situation, where human planners tend to be the only ones considered able to tackle such a problem, the innovation hereby suggested consists of an automated, artificial intelligence based, system capable of objectively driving the search and implementation of good solutions, without being influenced by pre-existing knowledge, mimicking a powerful lateral-thinking approach, so difficult to accomplish when management pressure impedes and daunting tasks bound the human rationality.

Ranking the effectiveness of a candidate solution, where path-dependency and unexpected complex effects may bias the final outcome, is not a matter trivially manageable by traditional operational research-style systems where no dynamics (recursive phenomena, feedbacks, non-linearity) appear. In order to overcome the limitations that an analytical specification of the problem imposes, the Agent-Based Modelling paradigm had to be taken into consideration.

Thanks to ABM we're provided with the opportunity of "in-silico" experimenting every imaginable scenario, by executing the planning in a virtual lab, where the production events happen instead of simplistically being computed. In this way we avoid following a reductionist approach, clumsily based on the usage of a static representation of the enterprise world, squashed into a cumbersome system of equations.

The model have been built resorting to the Swarm toolkit (see [Bur94], [JLS99], [MBLA96]); the underlying programming language (Objective-C) made the procedure of mapping the agents involved in the process onto software objects a plain and consistent task.

The problem presented belongs to the "shop problems" family in general, although many peculiarities make it an unconventional and distinguished one. When referring to "production planning", the authors have in mind the scheduling problem rather than ERP/MRP issues. In fact, the stage of the production on which the work is focused gives the availability of raw and semi-finished materials for granted. The up- and down-streams of the supply chain are normally performed by significantly oversized equipment, in the textile industry. On the other side, "core" processes, spinning and weaving in particular, require peak exploitation of the available production units.

KEYWORDS:

Production, Scheduling, Optimization, Industrial Processes, Manufacturing

I. THE PROBLEM

Matching tasks to units, under additional constraints, is the key issue. While certain constraints are to be regarded as "hard" (let's think of a technical issue rendering some of the production units useless in working on particular a (sub)task, thus reducing the set of available units), others are "soft" constraints: different units perform better on certain tasks, whereas others can suboptimally do, maybe with worse (yet acceptable) results, or take a longer time.

The sequencing of tasks is, on the other hand, one of the degrees of freedom of the problem, being the choice of giving priority to one task driven by timely delivery constraints.

For the sake of readability in this paper the words "order", "task", "job" will be used interchangeably.

A. Minimizing the production overall cost

Different production plans result in varying (aggregate) production costs. Each evaluation in terms of costs is made by adding several components: some of them are costs in a proper sense, others are more like abstract values by which we try to capture the economical impact of undesirable situations. Examples of the first kind are the setup costs; on the other hand delayed deliveries are certainly unwanted, even if not directly expressible as economical losses. Being considered an unreliable supplier because of repeated delays, in the long run, leads to unsatisfied customers being lost. This is, of course, an hardly economically quantifiable loss: it depends on how the firm's management perceives the importance of reliability, and how strongly is feared the risk of losing a repeatedly "deluded" customer.

B. Textile technicalities explained

The simulation is performed on and limited to, for the sake of simplicity, one of the production chain tasks only: proper spinning. Previous and successive operations can be overlooked, since they normally take place in oversized departments. Warping and combing, for instance, require relatively inexpensive machinery to be completed: it is common practice

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

to buy extra units 'just in case', since most of the plants value comes from spinners. The department where extreme care must be taken in avoiding any bottleneck effect is the spinning room.

We may confidently say that, should a good production plan be found for the spinning, the raw materials availability could be taken for granted, and the operations due to be performed up- and downwards the production chain could be arranged easily, not acting as constraints.

Finding a good production plan often implies dealing with mutually exclusive goals, in situations ridden with trade-offs. The only reasonable way to manage so many different aspects simultaneously is to reduce everything to its economical meaning, and it is hardly a straightforward task.

1) Production units setup: Spinners are complicated machines that can be adapted to produce many different kinds of yarns: apart from technical-mechanical parameters that can be tuned (speed, crossing angle, twisting…), each head (*see Glossary*) can be set up, by physically substituting some parts, to make for a wide range of technical specifications. Every kind of yarn features specific technical parameters and may require different parts to be mounted. At least three families (each one made by three types or more) of mechanical parts must be kept into account: cards, rotors, nozzles (*see Glossary*).

The act of setting up a spinning unit in order to have it ready to produce a certain kind of yarn may take a considerable amount of time: up to three hours may be spent removing and re-inserting a big amount of different mechanical parts, apart from trimming the appropriate software controls.

Of course putting similar products in sequence saves setup time: the least different two lots put in sequence are, the simplest and quicker the setup operations will be.

$$SC_{i,j} = f(\dot{p}_{1,i,j}, \dot{p}_{2,i,j}, \ldots, \dot{p}_{N,i,j})$$

The setup cost SC for order i placed after order j (or on a stopped production unit) depends on the dummy variables $p_{\{1<n<N\},i,j}$: each of them expressing the fact that the spinner part enumerated as the n-th (out of N) needs being exchanged when order j comes after order i (regardless of the spinner involved).

This seems a good enough reason to keep similar, if not identical lots, together, sticking them one after another. We'll see later why it's not that simple.

Nevertheless, the cost of setup can simply and accurately be accounted for in terms of man/hours spent performing the operation: after all, it consists of a sort of opportunity cost.

2) Timely delivery: Each order the firm is asked to produce is labelled with an "expected delivery date": customers are promised their yarn will be ready to ship by an approved calendar date (sometimes stringent conditions are imposed by "big" buyers), which they expect to be reliable. Should the delivery constraints be missed, a disappointed customer would, to say the least, complain bitterly. We have a situation which is very difficult to express in economical terms; very seldom a penalty is contractually established, rather the firm reputation is at stake, and the risk is to lose customers.

In order to keep into account, besides of the setup constraint ("less is better"), this additional constraint, a figurative cost has been introduced. It consists of an amount of money associated with the delay and the importance, positively correlated with both: the longer the delay and the bigger the order, the higher the (not-so-metaphorical) cost to be charged. Expressed in symbols:

$$DC_i = f(d_i^+, w_i^+)$$

where the delay cost DC for order i grows as the delay d and weight w (in kilograms) grow.

It becomes clear that sequencing similar orders on the same spinner is not an option: the freedom to save setup costs is at odds with the need to satisfy the timely delivery condition. A simplified example is presented (see Appendix, gantt sample).

3) Simultaneous setups, patrolling: To make things even worse (and almost impossible to deal with "by hand", which is nowadays the only viable way available to enterprises) further constraints are to be kept into account.

Production units setups, for instance, are performed by specialized workers; the number of setup teams available is limited, thus limiting the amount of setup operations which can happen at the same time. The effect of a missed setup (because of the unavailability of a team) on the production is simply a delay in the production of the order: no setup can be performed until one of the busy setup teams is available again. The total production time, and the time the order will be ready to ship, will be determined by the actual production time plus the initial delay.

Other employees are committed to the so-called spinners "patrolling": they are required to follow the ongoing production, ready to fix any problem should occur. A patroller is normally assigned 4 to 6 spinners to watch; the complication here arises from heterogeneity in the behaviour of different spinners: every different yarn features a specific likelihood to create (generically speaking) problems, that is to draw more or less attention from the patrollers. A patroller will be able to follow productions that are problematic up to a certain point: the average must be kept below this critical point. Above the limit, production times will grow (in a more or less foreseeable way) for all of the spinners under the overloaded patroller.

An index of "problematicity" is needed in order to manage such a subtle issue. The patroller load PL corresponds (for the n-th patroller) to the sum of the "problematicity index" p for each order i multiplied by the number of heads, h, available on the spinner j.

$$PL_n = \sum_{i=1}^{S} h_i p_j$$

Index PL is normalized in order to have 1 as the maximum tolerable patrolling load. Above this load, orders production times increase by empirically determined amounts:

PL	ΔPT	
$0 < PL \leq 1$	0	normal load
$1 < PL \leq 1.2$	$+10\%$	slight overload
$1.2 < PL \leq 1.5$	$+25\%$	severe overload
$PL > 1.5$	$> +25\%$	unacceptable overload

$$PT = production\,time$$

C. Evaluation by simulation

In order to evaluate the alternative candidate production plans, being able to rank them by "goodness", it takes a metric: a measurement of their own figurative cost. Such an operation needs to take into account the intrinsic complexity of executing the plan: each decision taken with regard to the assignment of a certain task conditions the subsequent decisions. While executing a plan two dimensions come into place: time and space; its evaluation cannot overlook this crucial assumption. Setup teams, for instance, may grant a total availability, compatibly with daily timetables, yet this can be suboptimal if compressed in a limited amount of time: queues tend to form.

A simulation was introduced, based on the enterprise design, which let us overcome the hard - if not impossible - problem of keeping track of such effects in the accounting. By simulating, all the production events are "made happen": formation of queues, delays, interaction among entities emerge spontaneously and are accounted for, when evaluating the total cost. This way avoids introducing tricks and approximations such as assigning pre-digested costs to unforeseeable events, using average (yet reliable?) values that render the accounting less accurate.

Exploiting a simulation also gives the advantageous chance to experiment with unlikely settings, or hard to observe in real-world situations. The need to evaluate by traditional computational techniques a production unit breakdown, for instance, one would be compelled to resort to an average "expected time between failures": this implies accepting two unrealistic assumptions, that we deal with a continuous phenomenon, and that the events are evenly distributed. By simulating, randomly occurring (and randomly lasting) events can be generated, while keeping probabilities within a pre-defined range: instead of an unrealistic continuous distribution we are correctly working on discrete events, with different durations.

An accurate cost tracking and accounting is instrumental to a good final result: the figurative cost of each plan enters the solutions generator (the genetic algorithm), where it is used to evolve subsequent generations of solutions. Even small distortions may disrupt the search process towards inefficient regions of the solutions space, prolonging computational times and considerably worsening the quality and reliability produced solutions.

II. Experimenting solutions

A. Agents: a local definition for an umbrella-term

The wealth of definitions and interpretations that coexist when "agents" come into play calls for a clarification: the agents hereby presented are to be intended as interacting no-minded software objects (in the Object-Oriented programming sense), whose main role is to encapsulate data, to make (mostly basic) computations and to pass informations back and forth. There is no communication protocol specification apart from the well-known getters/setters; the Swarm toolkit is used as an useful framework (see also [LS00a], [Ter98]) where software agents perform actions in a (perhaps sophisticated) time sequence by means of a scheduler triggering events, in this specific case in a deterministic way.

B. An Enterprise to experiment upon

The Enterprise Simulator is the module where solutions are experimented, that is where the simulation takes place. A model of the supply chain under scrutiny is used in order to watch candidate plans 'happen': the production process is represented in abstract, resorting to representative agents. Production units agents, setup agents, patrollers agents have been developed with the aim of giving simple yet exhaustive representations of their respective roles. Even production orders are embodied by dumb agents: objects encapsulating all the informations pertaining to the tasks to be performed, which are bounced between proper agents that act based on the informations they achieve from the orders themselves.

Presenting how the process takes place in the model is out of the scope of this paper; the steps - in a way absolutely adherent to the real process - implemented are: orders reception (in batch), orders dispatching to production units (filling queues), PUs setups, involving setup time computation after setup teams gathering, patrollers capacity reservation, production, repeat.

Ongoing and predetermined orders, already loaded on PUs and/or already due, are completed before initiating the candidate plan evaluation.

III. Inventing solutions (enter the Golem)

To find a good planning solution, given the enormous[1] set presenting itself, a Genetic Algorithm has been implemented, based on the well known AI paradigm first introduced by J. Holland (in [Hol75]).

The idea was to emulate the natural evolutionary process performing reproduction and death of structures that are representing a strategy. Provided that a whole set of structures is normally called "the population" of the GA, each of them is analogously named "an individual"; each one encodes a strategy into a binary string called "a genome". After having created an initial random set of structures, each of them is evaluated, one item at a time, by performing the strategy it represents, encoded, into an appropriate simulated environment. In this way a serial[2] evaluation of each structure can be

[1] An average spinning mill needs to plan about 50 jobs onto about 15 spinners at a time, which results in circa 10^{67} different feasible schedules. The weaving industry involves even bigger figures: up to 100-120 jobs to plan on 50-60 weavers, giving 10^{120} schedules.

[2] The process of evaluating populations is intrinsically parallel, being the population refresh step the only "pivot" operation which needs to wait for the completion of the individual-by-individual fitness assignments. For an in-depth presentation of the authors' works in this direction, see [Mor04], and thereafter in this article.

performed, in order to assign every strategy a value measuring its goodness: the so-called fitness of the individual. When the whole set has been evaluated, an evolution step can be taken: each individual is assigned a probability to reproduce itself (give birth to "offspring") and a probability to die, according to its fitness value: better-fitted genomes are assigned a higher probability to reproduce and a lower probability to die, and vice versa. Reproduction is made by copying and crossing two individual's genomes to obtain a couple of new structures to put into two new individuals; these newborn individuals will replace two old structures selected - from the previous generation - to die. By performing this algorithm in a loop the population becomes more and more fitted and the better types tend to spread into the population. The GA method is very useful when a wide set of alternatives has to be explored: it is general-purpose, it does not require any previous coded knowledge about the problem and it allows finding reasonable solutions in a short time.

To face the scheduling problem a special, but general, implementation of a GA has been employed. The goal was to set up a boosted GA, able to handle individuals composed by more than one structure, and structures defined on a very large alphabet. Another requirement was that this special implementation of a GA, the Golem, needed to handle special structures where all the alphabet symbols appeared only once[3].

The decision to write a special GA was due to the peculiarities of the problem to tackle. Each candidate strategy aimed to solve it can be split into two parts:

1) which machine will have to make an order
2) which priority will be assigned to each order

The two parts interact between each other in a complex way so the goodness of a solution depends on the goodness of each of them, but it is not possible to determine the contribution of each part to the performance of the solution. Both have to be evaluated simultaneously. Unless that, the contents of each part are very different and they could be coded in a highly different way. The first part could be expressed by a sequence of numbers, each of them identifies a unit, whereas the position of each code number identifies the order to be made. Adopting the same structure for the priorities the problem to assign univocal values to each order has to be faced. In addition the code numbers are defined on a set which cardinality is given by the number of machines the enterprise owns, while the cardinality of the priority set is defined by the number of orders the enterprise is going to plan. Resorting to the standard two-symbol $(0, 1)$ alphabet would have caused an ineffective representation of the solutions space, given the problem to represent each number in binary code every time the number of orders, or the number of machines, is not a power of two.

The Golem tackles the aforementioned issues by allowing the user:

1) to decide independently for each genome how many symbols need to be used by the coding alphabet, i.e. how many different values will be used in it
2) to decide a different length for each genome, i.e. how many positions it will include
3) to handle genomes where each symbol of the related alphabet will appear only once.

In addition the Golem was written taking into consideration:

1) the robustness of the methods exposed to the user, who can hardly misuse them
2) the efficiency (performance-wise) of the program

The Golem features methods to let the users' applications smoothly handle and control the search process. The user has simply to define the structure of the strings/individuals by coding the number and specific parameters for each of them: type (univocal or random), length, alphabet cardinality. The application (the *Enterprise Simulator* in this case) can conveniently interact with the Golem, demanding for an individual to evaluate and, after having performed the evaluation, returning the fitness value to the Golem. When all the population's individuals have been evaluated, the Golem automatically performs the evolutionary step. The Golem code has been optimized to ensure a high performance level, and has been regression–tested versus the earlier, more readable versions.

IV. Experimenting invented solutions, ergo suggesting the good ones

The evolution process performed by the Golem is driven by evaluating each single candidate solution appearing in the GA population. The production plans require an estimation as accurate as possible, incorporating every element of the dynamic interaction characteristic of the enterprise operations. It is the existence of such relationships among the intervening parts which distinguishes the problem as one of a complex kind: the aggregated outcome differs from what is obtained by the single components.

Keeping in mind the facts mentioned above, the unfeasibility of operating by decomposing the problem in parts is self-evident: the interactional effect would be totally missed; likewise, resorting to mathematical functions, static by their own nature, would imply neglecting all the time-related features, which are fundamental when it comes to plan actions intended to happen over time, being themselves subject to scheduling.

Computer simulation, by allowing management facts to happen in an artificial laboratory (the enterprise model), permits to quantify and express costs, whether figurated or not, generated by each candidate schedule, accurately and significantly, in order to promote the search for the best solution to the given problem.

The very same tool can be exploited in performing what-if analyses driven by human decisions, in order to rank GA-made solutions; this allows comparing what's produced by the human heuristics versus what's suggested by AI techniques, in a straightforward way. Plausibly it's the only viable method to provide a shared metric which permits, given the amplitude

[3]The so-called "univocal" genomes, where every symbol representing a job must not be repeated nor left out of every perspective solultion.

of the problem, to decide whether the search direction is a productive one or not.

In order to exploit the enterprise simulation to these purposes, the modelled objects are required to act as a bridge between the (scheduling) plan from the inferential method (the Golem) to the enterpreneurial metric.

In designing the Golem, that concerning this activity is just one of the advantages aimed at: the chance to use an extended (symbolic) alphabet solved some coding issues that during the first trials performed by standard AGs hindered the search process. An alphabet restricted to binary digits forces production units and orders number to be expressed by grouped symbols (as many as needed in order to the maximum value in the definition domain to fit); wherever the defined domain is less dense than the set of the natural numbers (when dealing with orders classified by differentiating their number by thousands or tenths of thousands, for instance), several non-significant solutions may appear. In such circumstances translation algorithms need to be employed, which, keeping such unfavorable factors into account, operate extraneous transformations (i.e. back-and-forth remapping) unknown to the AG; in the worst cases the same value gets assigned to formally different structures. Such behaviours can sensibly mislead the solutions learning and refinement process, keeping effective results from being efficiently achieved: execution times may stretch considerably.

A further issue emerged from the orders execution priorities. A standard GA in this case tended to produce non-univocal outcomes: the same priority may have been assigned to several different orders. Artificially differentiating equal values, based on the position within the structure for instance, might have impaired the GA abilities also in this situation. The system would have somehow been "deceived" by such artifacts. Providing the ability to opt between different operators, applicable to different kinds of genomes, the Golem could solve this issue too.

Achieving reasonable solutions quickly is fundamental to the enterprise: by analysing the experimental results a logarithmic trend of the solutions goodness have emerged clearly, functional to the number of evolutions performed. Practically speaking, the Golem is able to rapidly improve the solutions during the early stages of learning, while its productivity decreases as the optimum is approached. Going for population convergence appeared a suboptimal behaviour: halting the system after a certain number of evolutions seems way better than comsuming a long time in exchange for marginal improvements.

V. Preliminary Results

Although the system hereby presented is, from a developement standpoint, mature, its adoption at a production stage by the pilot plants involved is at its early phase. Nevertheless, batteries of tests on real-world data have been thoroughly performed.

The typical set-up involved sampling batches of orders from a random date in the past (picking up real-time fresh data very seldom, for reasons to be explain afterward). The results obtained were measured against random plans (averaged over multiple runs with different random seeds), against an ingenuous strategy[4], against human solutions.

The system has been evaluated at various stages of the search: although only after a 5-minutes run on an ordinary desktop PC (details in Appendix) the proposed solution is already better then the human-made, the performance level (the costs saved) improves quickly, yet asymptotically (see fig. 1).

A systematic comparison between results is hard to perform: historical data on previous production plans isn't always available; asking human planners to re-evaluate prior data sets is very likely to lead to biased solutions; the same happens with "live" data. Additional problems associated with hard-to-extract implicit knowledge are very likely to arise, when dealing with real-world situations. This has been kept into account, and comparisons have been performed both against ad-hoc solutions on datasets expressly and silently submitted to human planners and historical data, when available, hoping to level out bias.

Early - yet consistent - results have been presented and discussed with managers and experts, and they clearly show the superiority of the system presented. In the following table, results are shown as an indicator normalized versus the human performance (made equal a hundred), and represent the overall costs kept into account by the system, which of course neglects exogenous costs.

random	100.00
pseudo-FIFO (see note 4)	92.05
human	82.27
5' run	68.75
30' run	62.25
6h run	60.62

VI. Conclusions

Production planning constitutes a typically complex problem: the interacting parts taking part in the process makes impossible the application of traditional search procedures, based for most part on the decomposability of the problem as a prerequisite.

Given an (although limited) number of tasks to schedule, even the plain enumeration of the possible solution becomes practically unfeasible, given the combinatorial explosion implied. In this scenario the limits of applying heuristics based on human experience have appeared: the human mind attempts to solve the problem operating on limited subsets at a time, implicitly decomposing the complex problem, thus missing an

[4]The strategy, which is an oversimplification of the human way of scheduling jobs, consists of a sort of modified and refined "First-In-First-Out"method: jobs are appended to jobs with similar set-ups requirements that are already in queue on a given production unit; jobs with different set-up requirements are scheduled either on free PU's, if any, or the first PU expected to become available.

overall view on it. Every single decision taken on the assignment of a task onto a production unit constitutes a sensible "cut-off" on the solutions space, resulting in neglecting the exploration of large areas.

Implementing GAs let us exploit their implicit parallelism, both from a computational and an investigative point of view: starting from randomly generated solutions, avoiding predigested strategies, the GA also considers solutions that would be rejected by a human solver as absurd ones; not seldom innovative ideas are found among such apparently suboptimal candidates, and they are the ones that give superior results.

Apparently this is the main reason for the superiority of the system with respect to the human approach. It demonstrates itself far superior both in computation duration - efficiency - and final results - efficacy.

The system put into place constitutes, though, just a starting point: ways to improve the efficiency are being investigated and experimented, by distributing the "thinking" part of the work, the simulation, on several distributed nodes of a computer network, drastically incrementing the degree of parallelism of the computational process. At the same time work is being done on making the inferential engine (the Golem) more powerful, by introducing even more dramatic variations with respect to the standard GA's. The ongoing tests concern: clustered, cooperating GA's, and GA's featuring varying populations and variable-length individuals.

GLOSSARY

A brief list of technical terms relevant to the textile industry.

head: one of the (tenths to hundreds of) elements working on a single thread, constituting a spinning mill.

card: a toothed brush used to disentangle fibers.

rotor: a rotating device used in transporting fibers.

nozzle: a v-shaped element through which air flows.

APPENDIX

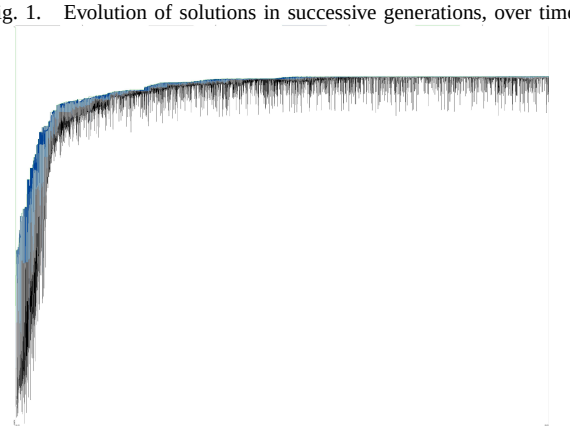
- Gantt example:

Customers *a* and *b* demanded, respectively, for [A1, A2] and [B1, B2]. Orders A1 and A2 are, from a technical standpoint, similar, and require a negligible setup time between them. B1 and B2 are also very similar. Ignoring (by now) the delivery constraints the obvious plan is to sequence similar orders on the same spinner (solution i):

spinner #	t_0	t_1	...	t_n
1	A1	q-A2		
2	B1	q-B2		
...				

The two customers, on the other hand, have different timing requests: a needs A1 and A2 as soon as possible; b is not pressing very much for a quick delivery and is fine for him to receive B1 and B2 by a later date. The most appropriate plan in this case would appear as follows (solution ii, grid entries changed from solution i have been italicized in order to let them stand out):

Fig. 1. Evolution of solutions in successive generations, over time



spinner #	t_0	t_1	...	t_n
1	A1	*l-B1*		
2	*A2*	l-B2		
...				

The small letters preceding the second orders are meant to show the different setup times required in both situations: as expected, q stands for 'quick' setup, l for 'long' setup.

Even in an oversimplified situation like the one described above, the complicated management of incompatible constraints appears; what makes solution i preferable over ii are the actual setup and delivery "costs", which must be accounted for as accurately as possible.

- Experimental set-up: technical details

The experimental gear used consisted of a rather aged desktop PC equipped with a single 800-Mhz Pentium-III CPU and 256 MB RAM. The amount of available memory becomes relevant when the GenomaBucket solutions caching system comes into play. It is beyond the scope of this article to present it; refer to [Mor03] for details.

REFERENCES

[AE94] R. L. Axtell and J. M. Epstein. Agent-based modelling: Understanding our creations. Bulletin of the Santa Fe Institute, 9(2), 1994.

[Axt99] R. Axtell. The Emergence of Firms in a Population of Agents. Brookings Institution, Washington, 1999.

[Axt00] R. Axtell. Why Agents? On the Varied Motivations for Agent Computing in the Social Sciences. Center on Social and Economic Dynamics, November 2000. Working Paper No. 17.

[BMVf] S. Bandini, S. Manzoni and G. Vizzari. Multi Agent Systems in Computer Science: Focusing on MAS Based Modelling and Agent Interaction, EXYSTENCE Thematic Institute for Complexity and Innovation, forthcoming.

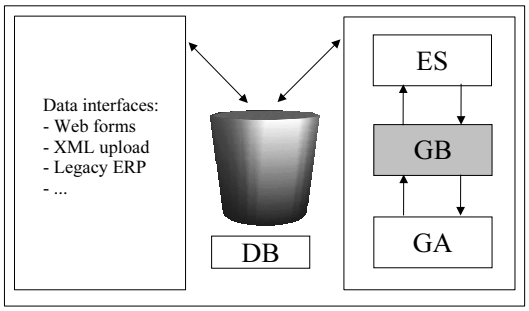
[Bur94] R. Burkhart. The Swarm Multi-Agent Simulation System, Position Paper for OOPSLA '94 Workshop on "The Object Engine", `http://www.swarm.org/archive/oopsla94.html`

[DG88] J.H. Holland D.E. Goldberg. Genetic algorithms and machine learning. Machine Learning, 3:95 104, 1988.

[Eps96] J. M. Epstein. Growing Artificial Societies. Brookings Institution Press,Washington, D. C., 1996.

[Eps99a] J. M. Epstein. Agent-based computational models and generative social science. Complexity, 4(5):41 60, 1999.

Fig. 2. Architectural overview



[Eps99b] J. M. Epstein. Learning To Be Thoughtless: Social Norms and Individual Computation. Center on Social and Economic Dynamics, September 1999. Working Paper No. 6.

[Fer01] G. Ferraris. GAMES: Algoritmi Genetici per l'Economia. Number 51 in Quaderni del Dipartimento di Scienze Economiche e Finanziarie G. Prato . Universit degli studi di Torino, Facolt di Economia, March 2001.

[GT00] N. Gilbert and P. Terna. How to build and use agent-based models in social science. Mind & society, 1(1), 2000.

[Hol75] J. H. Holland. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA, 1975.

[Hol98] B. Holmstrom. The firm as a subeconomy. In Bureaucracy: Issues and Apparatus, October 1998.

[HR99] M. Harris and A. Raviv. Organization Design. University of Chicago, July 1999.

[JLS99] P. Johnson, A. Lancaster, and B. Stefansson. Swarm User Guide. Swarm Development Group, November 1999.

[LS00a] Francesco Luna and Benedikt Stefansson, editors. Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming. Kluwer Academic Publishers, 2000.

[LS00b] Francesco Luna and Benedikt Stefansson, editors. Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming, chapter 9. Kluwer Academic Publishers, 2000. F.-R. Lin, T. J. Strader, M. J. Shaw, Using Swarm for Simulation the Order Fulfillment Process in Divergent Assembly Supply Chains.

[LS00c] Francesco Luna and Benedikt Stefansson, editors. Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming, chapter 10. Kluwer Academic Publishers, 2000. C. Schlueter-Langdon, P. Bruhn, M. J. Shaw, Online Supply Chain Modelling and Simulation.

[LTS96] F.-R. Lin, G.W. Tan, and M. J. Shaw. Multi-Agent Enterprise Modelling. University of Illinois at Urbana-Champaign, October 1996. Office of Research Working Paper 96-0134.

[MBLA96] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations. Santa Fe Institute, June 1996. `http://www.swarm.org/`

[MT00a] J. P. Marney and H. F. E. Tarbert. Why do simulation? toward a working epistemology for practitioners of the dark arts. Journal of Artificial Societies and Social Simulation, 3(4), October 2000.

[Mor03] M. Morini, *Penelope Project: Web-Based Textile Production Planning*, SwarmFest 2003, University of Notre Dame, IN. `http://www.nd.edu/swarm03/`

[Mor04] M.Morini, *Penelope Meets NEMOTE: Distributed Production Planning Optimization*, SwarmFest 2004, University of Michigan, Ann Arbor, MI. `http://cscs.umich.edu/swarmfest04/`

[PCG99] M. J. Prietula, K. M. Carley, and L. Gasser, editors. Simulating Organizations, Computational Models of Institutions and Groups. AAAI Press The MIT Press, 1999.

[SLS96] T. J. Strader, F.-R. Lin, and M. J. Shaw. Information infrastructure for electronic virtual organization management. University of Illinois at Urbana-Champaign, October 1996. Office of Research Working Paper 96-0135.

[SLS98] T. J. Strader, F.-R. Lin, and M. J. Shaw. Simulation of order fulfillment in divergent assembly supply chains. Journal of Artificial Societies and Social Simulation, 1(2), March 1998.

[Ter98] P. Terna. Simulation tools for social scientists: Building agent based models with swarm. Journal of Artificial Societies and Social Simulation, 1(2), 1998. `http://www.soc.surrey.ac.uk/JASSS/1/2/4.html`

An Agent-based Matchmaker

(*A case study in biomedical services discovery*)

Flavio Corradini, Chiara Ercoli, Emanuela Merelli and Barbara Re Dipartimento di Matematica e Informatica,
Università di Camerino
62032 Camerino, Italy
Email: {flavio.corradini,emanuela.merelli}@unicam.it, {chiara.ercoli,barbara.re}@studenti.unicam.it

***Abstract*— Service discovery is the process of localizing resources and services available in large scale open and distributed systems. In a distributed and redundant system as the Web, it is necessary, beside localizing services, to filter them in order to obtain those which are best for the activities for which they have been requested. By the term *matchmaker* we mean a software entity which monitors services availability, maintains an updated file of all useful information for using services and possibly ensures a quality choice of them. In this paper we propose an architecture for an agent-based *matchmaker*. The matchmaker that takes part in the request process has been developed by using the potential of a *quality model* based on suitable parameters to ensure the proper choice of a service to be consumed in a specific application domain. A case study in biomedical domain is presented. This case study is concerned with the development of a multi-agent system including a Bio-certifier in support of service discovery activity.**

I. Introduction

Service discovery is the process of localizing services and resources in the Web that best fit the requests of potential users.

The Web main feature is the interconnection of an ever increasing number of open, dynamic and geographically distributed systems which have an high heterogeneity of resources, information systems and tools for specific application domains. Hence the Web is a rather complex environment for service discovery activities as can be seen, for example, in the biomedical domain.

Biological and medical research is characterized by a global distribution of information and by an almost complete autonomy of research groups, from which an heterogeneous, redundant, incomplete and rapidly aging access to resources derives. Hence the choice of what could be the most suitable tool or service for biomedical work activities is often difficult and time consuming. From these considerations there follows the need of building a quality model to support the discovery process which will be based in symbolic descriptions of relations among concepts of one or more domains of interest allowing classification of services which are functionally similar [1].

Quality can be defined as all the features of an entity (resource, service, tool) that influence its capability to satisfy declared or implicit needs [2]. From this definition it is clear that it is difficult if not impossible, until today, to define a specific metrics capable of measuring the quality of resources available through the Web. Although there exists several criteria to evaluate consistency and internal correctness of a resource, true evaluation of its quality, that of interest to users, relies on the effectiveness of the resource itself. In other words, one has to ascertain if a specific group of users considers the use of that resource satisfactory for its information needs. In fact, before finding the ideal requirements for the quality model, it is necessary to carefully analyze the application domain in which the quality model has to be used. Hence the quality model has two main components, the general one which describes the quality aspects of the distributed system, e.g. the Web, and the other which describes the specific quality aspects of the application domain; the biomedical in our case study. The established quality model then becomes a tool of consultation for the software entity in charge of service discovery.

In this paper an architecture for a quality of service (QoS) agent-based matchmaker is presented. The term *matchmaker* [3], [4] means a software entity capable of monitoring the availability of services, maintaining an updated file of all information on service use and, we add, of providing a quality choice of service. The matchmaker is an agent contacted by other agents wanting to obtain a quality service with respect to the activity where the service will be used. In order to ensure a choice of quality of a requested service, the matchmaker communicates with the QoS *certification authority*, i.e. an agent capable of implementing the established quality model.

Briefly, an agent is a software system capable of acting with the aim of solving a problem in a flexible and autonomous way and in an open, dynamic, unpredictable environment which is typically populated by other agents. Often agents are created to interact and cooperate with each other. The need of making an agent interacting and communicating with other agents leads to the need of coordinating the activities of the agents involved in a system [5], [6]. In order to coordinate a pool of agents (MAS: Multi-Agent System) it is necessary and fundamental to understand which are the actors involved in the system, their roles and which information are more important. In so doing, we also achieve the result of specifying the importance and true value of the parameters that characterize the quality model.

The coordination model we have followed is the matchmaking model presented in [7] which is based on a process of mediation that implements direct communication among the providers and the consumers of services and resources.

This work was supported by the by the MURST strategic project 'Oncology Over Internet', by the CIPE project 'SICOM: Sistemi Coperativi Multi-agente' and the Center of Excellence for Research 'DEWS: Architectures and Design Methodologies for Embedded Controllers, Wireless Interconnect and System-on-chip'.

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

To the aim of showing the applicability of matchmaker architecture enriched with the QoS component, we have examined a case study in the biomedical domain, and developed a multi-agent system for the discovery of quality services based on JADE[1] platform.

The remaining of the paper is organized as follows. Section II is an introduction to the case study. Section III presents the architecture of the multi-agent system for service discovery and introduces the quality model. Section IV describes the architecture of the system, defines the quality model for biomedical domain and debates some experimental results. In section V different approaches proposed in the literature for the service discovery are analyzed and future extension of the paper are presented.

II. A Case Study in the Biomedical Domain

Health science is the applied science discipline that deals with human and animal health by means of study, research and application of knowledge with the aim of improving general health. Biomedicine is a branch of health science that applies biological and physiological principles to clinical practice. Support to biomedicine is given by the understanding of the way in which both human and animal biological systems work and by the analysis of the (sometimes hidden) existing relations between medical reports and results of performed therapies. In both cases, the use of computational tools allows us to find and analyse biological and clinical information in order to answer complex questions in biological domain. Moreover, appropriate computational models would also allow to simulate biological systems [8], [9] with the aim of verifying properties useful both for diagnosis and therapy.

The Web is an endless source of information of fundamental importance to increase knowledge in the biomedical domain, however it is often difficult and complex to retrieve this information.

In several disciplines, complex questions are stated by means of workflow of activities representing different instances of problems which are simpler to solve. Carrying out these activities implies the use of resources (tools and services) usually accessible through the Web [10]. Introducing quality in a workflow means giving a way of finding the most appropriate resource/service to effectively satisfy the requests of each activity. The following two scenarios are presented as examples in order to introduce the workflow concept in biomedicine:

"Let us assume that a biomedical researcher be an expert of a gene, of the corresponding protein, of the known mutations of that protein, and of consequent pathologies as well. This biomedical researcher wants to design a microarray[2] experiment to analyse the gene expression (i.e. how much the gene produces) in different normal and pathological tissues. This experiment allows him to also find out the genic expression of other genes in addition to the one of the gene being studied and hence he needs to have an updated list of the genes that could be involved in the same biochemical pathway (i.e. chain of biochemical reactions). For this purpose, the biomedical researcher decides to use the Gene Ontology (GO) annotation[3] to find out the relations among genes, biological processes and biochemical pathway." In this example GO is used as a domain-specific language to specify the request, thus GO terms will effect the domain specific quality aspects of the proposed model.

"A doctor is treating a patient that has some constant slight temperature (37,5 C). The temperature persists after antibiotics therapy and hence the doctor decides to control the protein level of the patient and prescribes some blood tests and urine test. The performed tests show that the level of some proteins is not normal but no sure conclusion can be drawn (no certain diagnosis). The doctor then decides to search the possible interactions among these proteins." Instead, this example does not use any domain specific language (ontology) to describe the service, thus the quality model consists only of the general quality aspects.

In these and similar situations the search for useful information with the aim of giving an answer to the questions being asked implies the choice and use of several resources. The discovery process should be capable of identifying the best service which will give the sought result in the shortest time, so making the system efficient and effective.

III. The Multi-Agent Architecture for Service Discovery

In this section we present the architecture of the multi-agent system and the quality model defined to support the service discovery in a distributed environment.

The system supporting service discovery has been designed using agent technology because the problem dealt with was suitable to be described in terms of autonomous, flexible actors which operate in a dynamic and unpredictable environment and are created to cooperate with each other. Our choice has also been affected by such parameters as development and administration costs of the discovery system, implementation of interoperability among the different active systems, and guarantee of an acceptable security level.

The proposed system architecture is an extension of the one defined in Retsina [11] infrastructure and its main feature is a group of three actors (agents) that communicate and exchange information among themselves with service discovery as their common goal.

The *service provider* supplies the services by which it is possible to find the required information or solve specific computational problems related to an application domain. The *service requester* is the user (or consumer) of the services offered by the *service provider*, and finally the *middle-agent* is the software entity that mediates between the previous two in order to find the sought services.

In the literature [11] *middle-agents* are classified according to their functionalities as mediators, brokers and matchmakers. In its original definition, given by Wiederhold in 1992 [12], a *mediator* is the active and dynamic interface by which a given

[1] http://jade.tilab.com

[2] A DNA microarray is a piece of slide with a microscopic array on which single DNA pieces are placed.

[3] www.geneontology.org/

user (*service requester*) access data and knowledge owned by itself.

A *broker*, sometimes also called a *facilitator*, is the existing interface between a *service requester* and a *service provider* which acts as a mediator for service requests. All communications between pairs of *service providers* and *service requesters* flow into the broker which typically contacts the more important *service providers* negotiating execution of and access control to the most appropriate service and returning the service result to the *service requester*.

On the contrary, the task of a *matchmaker* is to create a connection between the *service requester* and the *service provider*, matching the request of a given service requester to the offer of service of a *service provider*. In this case an autonomous interaction will take place. Unlike the functionalities of both the broker and the mediator, the functionality of the matchmaker is to return to the service requesting agent an ordered list of *service providers*. Consequently the *service requester* has to directly contact the *service provider* and negotiate with it in order to get the desired service.
One could then consider matchmaking as a subset of brokering but at the same time it can be seen as an extension of it because matchmaking allows the *service requester* a subsequent choice of *service provider* in a way independent of the match found by the matchmaker. The matchmaker has one weak point: each agent needs to be smart enough to form a query and evaluate how to choose among alternative *service providers*, this features being not always present in MAS systems.

The coordination model describing dependencies and interactions between the matchmaker and the other agents is called matchmaking [7]. When an agent publishes a service, the matchmaker records the name of the agent in his knowledge base together with the description of offered service according to the ontology used during the communication act. In this way, when an agent requests a service, the matchmaker looks in his knowledge base for a server capable of satisfying the request (*service matching*). Then the agent requesting service directly interacts with the chosen server in order to get the desired service and data (*service gathering*) so avoiding a possible bottleneck in data transmission or a possible interruption of the matchmaker activity, as described in Figure 1.

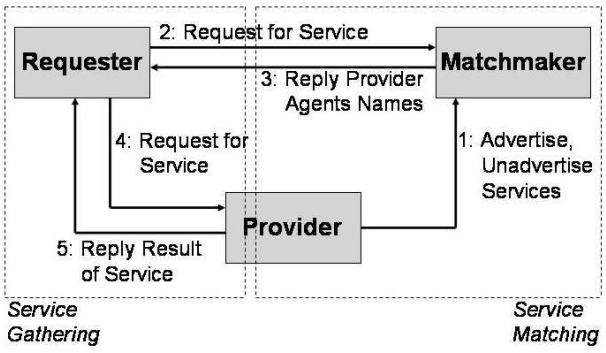


Fig. 1. Matchmaking coordination model [4]

A. The Proposed Agent-based Matchmaker

To the classical matchmaking model, as presented in [11], our architecture introduces a fourth kind of agent (see Figure 2) representing the QoS *certification authority*. This agent, through certification, ensures that resources, services and tools be consistent with the user request non-functional requirements. In the proposed extended model, the duties of the $matchmaker$ also include coordination of available services in accordance with specific protocols, agreements and policies, and mediation to obtain reliable services both in terms of quality and confidence, the latter being due to the multi-agent system.
Such a process of quality service discovery must include a component capable of analysing certain fundamental requirements that are made appropriate to the domain to which they belong. Verification of these requirements will allow the *QoS certification authority* to give a quality level to each registered services taken into consideration. In particular the general evaluation criteria of our authority include some macro-categories as aim of resource, user target, reliability, contents, privacy, updating of formal features and quantitative functions.

The main functionalities of each actor in the dynamics of mediation systems are as follows:

1) a *service provider* advertises its services to a *matchmaker* via WSDL-URI;
2) the *matchmaker* stores this information in an hash table and notifies the new services to the QoS certification authority;
3) the *QoS certification authority* contacts the *service provider* and verifies the quality service;
4) the *QoS certification authority* certifies the service to the *matchmaker* via an XML document;
5) a *service requester* asks *matchmaker* to find a *service provider* that provides the best services;
6) the *matchmaker* processes the request within his knowledge base (collection of information on services and *service providers*) and it yields either some information regarding the *service provider* or possibly the result of the application of the requested service;
7) the *service requester* send the request (service input) to the selected *service provider*;
8) after the executing of service, the *service provider* returns the result (service output) to the *service requester*.

This model, while looking extremely simple at first sight, is instead a rather complex one mainly because the Web is an open system, with plenty of information, subject to continuous changes of available resource location, heterogeneity and contents. The complexity of the model can increase when different user groups and different MAS come into play because each has its own goal which may be in conflict with those of the others.

The choice of integrating matchmaking with a QoS authorization component is a consequence of matchmaking being well suitable to the scenario proposed by the case study, whose features are distributed systems into which agents come in and out and the offering of multiple answers with the possibility

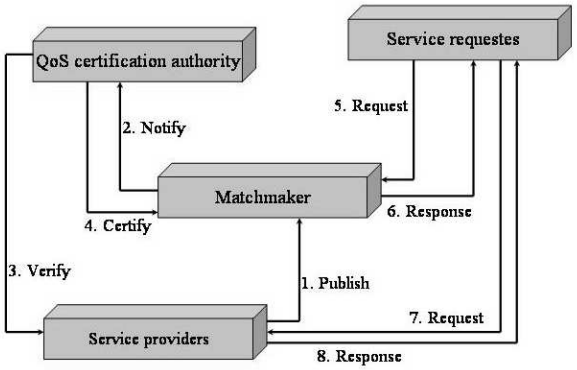


Fig. 2. The MAS architecture extended with the QoS certification authority

for each agent to keep control of its choices. The proposed matchmaker limits the choice among alternative.

B. The Proposed Quality Model

When a user looks for a service (resource, tool, etc), the system ideally should fetch the service exactly matching the one requested. It is practically unlikely that such a service be available and hence a service with "sufficiently similar features" is fetched.

What do we mean by "sufficiently similar"?
In its strongest meaning a service offered in the network and a requested service are "sufficiently similar" when they exactly contain the same functionalities. This is a too restrictive definition since the requesting user does not know in advance how a service is represented in the network and has an own idea of what the service should do. An acceptable definition of similarity can be one with less constraints so to accept a more flexible exactness degree.
Hence localizing services which can be used by the user despite the existing differences between request and offer represents a challenge for the system. Metrics measuring the distance between request and offer can be of help to the user in making a deliberate choice [13].

As above mentioned, the proposed quality model consists of two components, the one describing general quality aspects of the distributed computational environment where the service is offered, and the other including quality features of the application domain. In particular, the quality aspects chosen for the first component have been derived by analyzing the Web, and concluding that a qualitative web resource must provide information to satisfy the following requirements:

- *Aim* is the purpose for which the resource has been developed;
- *User target* is the list of hypothetical users;
- *Reliability* is the probability of successfully using a resource;
- *Feasibility* is the measurement of the easiness to access the resource;
- *Usability* is the measurement of the easiness to use the resource;
- *Originality* is the degree of correctness of the resource and its information;
- *Privacy* captures the legal conditions of using the resource;
- *Updating* is the attendance of the resource updating;
- *Uptiming* is the maximum length of time between two resource failures;
- *Timing* is the daily time of resource activity;
- *Speedy* is the measurement of the execution time;
- *Browsing* is the measurement of the human easiness to find a resource;
- *Popularity* is the number of active consumers;

Each quality aspect above defined is quantitative measured on the basis of several parameters not listed in this work, but available in [14], [15]. While the domain-dependent quality aspects are described in the Section IV-A dedicated to the case study quality model.

Our system draws a distinction among three matching levels:

Exact match is the highest degree of matching and takes place when requests are satisfied by the server with a percentage higher than 90%.

Plug-in match takes place when a service more general than the requested one is supplied but that can be used instead of the ideal requested service. This kind of matching happens when requests are satisfied with a percentage between 10 and 90%.

Relaxed match is the lowest degree of matching and takes place when requests are satisfied by the server with a percentage lower than 10%.

The matching algorithm measures the distance between the quality aspects and the user requirements for a request service. The matching algorithm developed in this work is carried out within the QoS *certification authority* to support the following actions:

- supporting the semantic matches in a flexible way on the basis of existing ontology;
- achieving matches with a minimum number of positive false matches and negative false ones.
- encouraging correct registrations and requests that take into account the cost of a mismatch due to false declarations;
- carrying out efficient matches that give results in a short time.

The main cycle of the matching algorithm is shown in the code below. It can be seen that the requests are compared with all parameters of the services which are stored in the knowledge base and that the coefficient measuring the degree of matching is evaluated for each service.

```
match (request){
    recordMatch = empty list
    forall service in mirror do{
       recordMatch.addElement(service, coff)
    }
    return best(recordMatch);
}
```

Through our research we have found some general criteria for evaluating quality of resources, services or tools. These

criteria can be grouped in macro categories as: purpose, user target, reliability, privacy, updating, formal aspect adherence, interactivity, stability, ease of use and use of and access to established standards.

IV. The QoS Matchmaker in the Case Study

As a case study we have chosen the biomedical domain for its complexity and because from carried out researches come out that quality of Internet medical information is affected by heterogeneity and dispersion of resources and inaccuracy and incompleteness of available information. These factors are well suitable for the definition of a quality model to be integrated in a matchmaker architecture.

Defining a quality model means quantifying the parameters that are typical of the application domain and specifying the framework in which the model will be used. We assume to be in a simplified situation in which biomedical information and services are supplied by information repository which are distributed in the network, called MOBY-Central [16] in our case. Figure 3 shows the components of matchmaker architecture and their interactions.

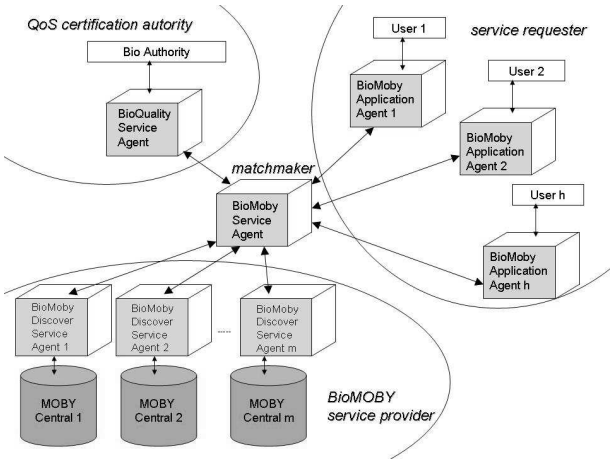


Fig. 3. The matchmaker architecture in the case study

The central element, that is the *matchmaker*, represents the knowledge base of the poll of agents involved in the biomedical system (BioMAS). A *matchmaker* interacts with three separate components (agents): the BioMOBY *service provider* through which increases the system knowledge base which is used in the discovery of new biomedical services, the QoS *certification authority* supporting the discovery service and the *service requester* that carries out the user side application for locating services that satisfy given properties.

In details, the BioMAS consists of four kinds of agents: the `BioMobyServiceAgent`, which is the system main actor because all other agents refer to it. This agent has three different roles: (i) coordinates all other agents; (ii) manages the system knowledge base, and (iii) carries out the discovery of a quality service. The `BioMobyDiscoverServiceAgent` is the interface between the biological information repository and the `BioMobyServiceAgent` with the aims of discovering the requested service on the one side and of advertising any newly offered service on the other side. The `BioMobyApplicationAgent` helps the user in the search of the best service among those advertised by the `BioMobyServiceAgent`. Finally, the `BioQualityServiceAgent` is the authority that certifies services according to the quality model defined for the biological domain.

A. The Quality Model for the Biomedical Domain

The *QoS certification authority* developed in this work used an instance of the quality model introduced in III-B. The first component is characterized by the quality aspects of a Web Services (i.e. `BioMOBY`), while the second is characterized by information introduced by the biomedical domain. The subset of quality aspects chosen for the first component are:

- the *Reliability* based on three parameters: the first one assigns a value to the author based on his professional competence, the second allows to find whether the author adheres to certified standards and the last allows to find out whether the supplier of service is profit oriented;
- the *Originality* based on two boolean parameters: publicity policy, that is whether there are sponsors and official agencies financing the resource and fidelity procedure, that is the monitoring of consumer surveys;
- the *Privacy* based on a boolean parameter that makes sure that privacy policies, data security, personal data processing (including that of unaware users) are in accordance with existing laws;
- the *Updating* based on a parameter that addresses the time period (daily, monthly, yearly) the resource is updated;
- the *Usability* based on a parameter that measures the easiness in using a resource:

Finally, by formal aspect concept we mean two strictly technical parameters which give a measure of the daily service performance:

- the *Timing* that is a measurement of the time period that a service is active;
- the *Speed* that is a measurement of the service execution time.

The matching algorithm, after having analysed the above listed information and after having made a classification of services, goes on to examining the information made available by biomedical domain.

In this second part of the model,

- *name*, represents the most important parameter because the knowledge of it by the user will cause the search necessarily returning the specified service (match weight=51);
- *description*, made of keywords which will be sought inside every individual service stored in the knowledge base (match weight=4);
- *type*, has little importance in the model because can only be one of seven kinds (service, retrieval, resolution, parsing, registration, analysis, NCBI_Blast) (match weight=2);
- *author*, it simply represents his name and does not carry his credentials with it (match weight=4);

- *input* and *output*, they are fundamental parameters because the user already knows what he has got and what he wants to get (match weight=17 and 22);

A more detailed description of the model of quality can be found in [14], [15].

B. Examples

As explained in Section II, as of today the huge amount of information and services in the biomedical domain which are in the Web makes rather difficult to the user to understand which is the best service for his needs. Our contribution to solving this problem has been the implementation of a model based on a qualitative matching algorithm by which it is possible to make the correct choice. Moreover, by the use of an agent based technology, waiting time and interaction time by the user with the system have been considerably reduced because of the presence of a software assistant. In order to show some preliminary results of the effectiveness of the proposed model we will consider the two simple examples previously shown.

In the first case "the biomedical researcher decides to use the Gene Ontology (GO) annotation to find out the relations among genes, biological processes and biochemical courses." The results we would get from BioMOBY by making the following query with keywords 'GO', 'Gene' and 'Ontology':

```
select servicename, url
from service_instance
where description like %Gene%
and description like %Ontology%
and description like %GO%
```

are:

```
servicename: getGoTerm
        url: http://mobycentral.cbr.nrc.ca
             /cgi-bin/Services/Services.cgi

servicename: getSHoundGODBGetParentOf
        url: http://mobycentral.cbr.nrc.ca
             /cgi-bin/Services/Services.cgi

servicename: getSHoundGODBGetChildrenOf
        url: http://mobycentral.cbr.nrc.ca
             /cgi-bin/Services/Services.cgi
```

While the result obtained by the QoS matchmaker is:

```
servicename: getGoTerm
        url: http://mobycentral.cbr.nrc.ca
             /cgi-bin/Services/Services.cgi
```

It can be noticed that in the first case the answer also contains addresses which are not meaningful for the made query forcing the user to a kind of classification or to repeated trials before singling out the service which best fits his needs. In the second case, the QoS machmaker, on the bases of the matching algorithm filters the best service.

Let us analyse the second example "the doctor then decides to search the possible interactions among these proteins." By making the following query to BioMOBY with keywords 'protein' and 'interact':

```
select servicename, url
from service_instance
where description like %interact%
and description like %protein%
```

we would get the following results:

```
servicename: getInteractions
        url: http://www.pdg.cnb.uam.es/moby
             /cgi-bin/mobyservice

servicename: getInteractionsXML
        url: http://www.pdg.cnb.uam.es/moby
             /cgi-bin/mobyservice

servicename: getInteractingMethods
        url: http://www.pdg.cnb.uam.es/moby
             /cgi-bin/mobyservice
```

While, through the middle-agent mediation and use of the model of quality, we obtain:

```
servicename: getInteractions
        url: http://www.pdg.cnb.uam.es/moby
             /cgi-bin/mobyservice
```

Also from this second case, it can be seen that the use of a quality model has the same effect of applying a filter to the set of possible answers.

V. Related and Future Work

Many works have been presented in the literature to support service discovery in the Web environment [3], [4]. Some use UDDI technology and Web Services, others use the agent technology, a few just use a mediator. None of these suggests the integration of a quality model within the matchmaker architecture in support to service discovery in a biomedical domain.

UDDI[4] (Universal Description, Discovery, and Integration) has become a de-facto standard for service discovery in the community of Web Service and it is commonly looked at as a "yellow pages" service. In the UDDI model services are localized through their description by the supplier or by the type of service and both ways of service discovery are built with a limited number of high level sentences that produce a rigid scheme. Although UDDI is a de-facto standard, it does not allow neither a quantitative nor a semantic discovery but only a keyword based search.

Retsina [3] is an open infrastructure for MAS which is capable of supporting communities (oppure populations) of

[4]http://www.uddi.org

heterogeneous agents. Service discovery is based on OWL-S[5] ontological language for service functionality description. The resulting matching process is only a semantic one and not necessarily of quality.

DiscoveryLink[6] is a product developed by IBM that allows a discovery process on many specialized heterogeneous databases by means of a single query that uses specialized wrappers. The resulting system is a rigid one again and forces the user to predefined and limited choices without offering either a semantic service discovery or one of quality.

MyGrid[7] is a pilot project of UK e-Science that provides a middleware open-source Grid developed to supply a virtual workbench in bioinformatics domain. Emphasis is placed on the workflow as an integration tool and on the customisation and source of data. Resources are considered as services that can be statically or dynamically combined within a given framework. However also in this product no quality of the offered service is guaranteed.

We plan to extend this work in the future by customizing requests to target, that is by including in our model of quality parameters which are proper of user profile (computer scientists, biological computer scientists, biologists, etc.). We also mean to add use of ontology in order to describe the user requests both to verify their validity and to correctly describe each service. In fact, GO could also be included in the quality model to describe the bio-domain. In our case study GO represents the service description language.
The introduction and quantification of additional certification parameters will help both the certifying agent and the middle-agent to keep their information updated and hence to answer even complex requests by giving a service workflow. Last but not least, we plan to develop the system in Hermes[8], a mobile agent middleware supporting distributed applications and mobile computing, in order to use mobility to optimize the cost of data transfer and evaluate the possibiltiy to improve the performance of the matchmaker.

ACKNOWLEDGEMENTS

We wish to thank Ing. Lucio Forastieri for the interesting discussions that have led us to the development of this work and Ing. Paolo Romano for having given us useful case studies for validating the proposed model.

REFERENCES

[1] G. Eysenbach and al., “Towards quality management of medical information on the internet: evaluation, labelling, and filtering of information,” 1998.
[2] T. C. I. . International organization for standardization, “Iso 8402: Quality management and quality assurance.” 1994, vocabulary. 2nd ed. Geneva: International organization for standardization.
[3] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa, “The RETSINA MAS infrastructure,” *Autonomous Agents and Multi-Agent Systems*, vol. 7, no. 1–2, pp. 29–48, July–Sept. 2003.
[4] K. Decker, M. Williams, and K. Sycara, “Matchmaking and brokering,” in *ICMAS-96*, May 1996.
[5] M. Wooldridge and N. R. Jennings, “Agent theories, architectures and languages: A survey,” in *Intelligent Agents, ECAI-94 Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag, 1994, pp. 1–39.
[6] N. Jennings and M. Wooldridge, “Application of intelligent agents,” in *Agent Technology: Foundations, Applications, and Markets*. Springer-Verlag, 1998.
[7] K. ycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, “Automated discovery, interaction and composition of semantic web services,” *Journal of Web Semantics*, vol. 1, no. 1, pp. 27–46, 2003.
[8] H. Kitano, “Systems biology: A brief overview,” *Science*, vol. 295, pp. 1662–1664, march 2002.
[9] N. Cannata, F. Corradini, E. Merelli, A. Omicini, and A. Ricci, “An agent-based conceptual framework for biological systems simulation,” in *Fourth International Workshop on Network Tools and Applications in Biology*, Camerino, 2004.
[10] F. Corradini, L. Mariani, and E. Merelli, “An agent-based approach to tool integration,” *Journal of Software Tools Technology Transfer*, 2004, to appear.
[11] M. Klusch and K. Sycara, “Brokering and matchmaking for coordination of agent societies: A survey,” in *Coordination of Internet Agents: Models, Technologies, and Applications*, A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, Eds. Springer-Verlag, Mar. 2001, ch. 8, pp. 197–224.
[12] G. Wiederhold, “Mediators in the architecture of future information systems,” *IEEE Computer System*, vol. 25, no. 3, pp. 38 – 49, March 1992.
[13] R. Culmone, G. Rossi, and E. Merelli, “An ontology similarity algorithm for bioagent,” in *NETTAB Workshop on Agents nd Bioinformatics*, Bologna, 2002.
[14] B. Re, “Un modello di qualità per la scelta di servizi web in ambito biologico - il ruolo del modello di coordinazione,” Master’s thesis, Laurea in Informatica, Università di Camerino, a.y. 2003-2004, http://dmi.unicam.it/merelli/tesicl26/re.pdf.
[15] C. Ercoli, “Un modello di qualità per la scelta di servizi web in ambito biologico - il ruolo del middleware,” Master’s thesis, Laurea in Informatica, Università di Camerino, a.y. 2003-2004, http://dmi.unicam.it/merelli/tesicl26/ercoli.pdf.
[16] M. Wilkinson, D. Gessler, A. Farmer, and L. Stein, “The biomoby project explores open-source, simple, extensible protocols for enabling biological database interoperability,” 2003, procedings of the virtual conference on genomics and bioinormatics. (3):17-27.

[5] http://www.owl-s.org
[6] http://www.discoveryLink.ibm.com
[7] http://www.mygrid.org
[8] http://hermes.cs.unicam.it