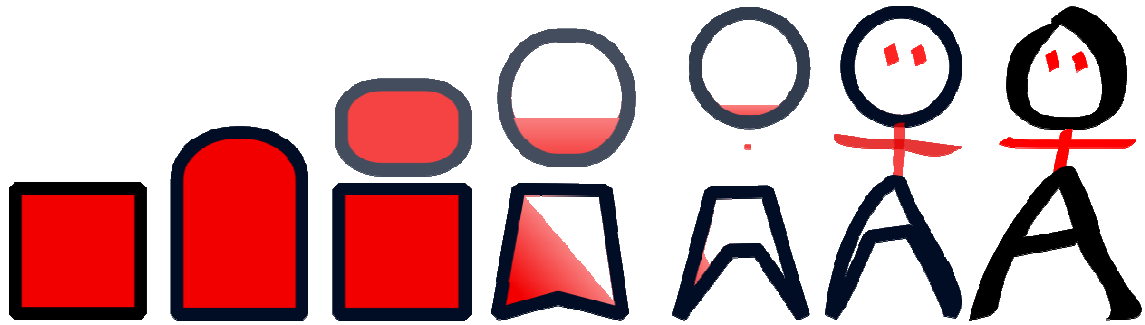
WOA'08

Dagli oggetti agli agenti



Evoluzione dell'agent development: metodologie, tool, piattaforme e linguaggi

Nona Edizione, WOA 2008
Palermo, 17 – 18 novembre 2008
Atti del Convegno
A cura di: Matteo Baldoni, Massimo Cossentino, Flavio De Paoli, Valeria Seidita.

Seneca Edizioni, ISBN 978-88-6122-122-2

WOA 2008 Home Page: http://www.pa.icar.cnr.it/woa08/

Editori

Matteo Baldoni
Universitá degli Studi di Torino
C.so Svizzera, 185
10149 Torino, Italy
baldoni@di.unito.it

Massimo Cossentino
ICAR-CNR
Viale delle Scienze, ed. 11
90128 - Palermo, Italy
cossentino@pa.icar.cnr.it

Flavio DePaoli
Universitá degli Studi di Milano - Bicocca
Viale Sarca, 336/14
20126 Milano, Italy
depaoli@disco.unimib.it

Valeria Seidita
Universitá degli Studi di Palermo
Viale delle Scienze, ed. 6
seidita@dinfo.unipa.it

Sponsors:

AIIA - Associazione Italiana per lIntelligenza Artificiale
TABOO - Associazione Italiana Tecnologie Avanzate Basate su concetti Orientati ad Oggetti
ICAR CNR - Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche, sede di Palermo
DINFO - Dipartimento di Ingegneria Informatica dellUniversit di Palermo
ENGINEERING INGEGNERIA INFORMATICA

Printed in November 2008
by Senecaedizioni, Strada del Drosso 22 - 10135 Torino

Prefazione

Il Workshop "WOA dagli Oggetti agli Agenti" é da alcuni anni ormai un importante momento d' incontro per favorire il confronto e lo scambio di idee su un argomento oggi molto rilevanti ed oggetto di tanta ricerca: i sistemi ad agenti. Le tecnologie ad agenti offrono modelli e tecniche per l'ingegnerizzazione di sistemi software complessi ed il loro uso sta assumendo un ruolo sempre piú rilevante in molti settori dall'Intelligenza Artificiale all'Ingegneria del Software.

L'edizione di quest'anno organizzata dal Gruppo di lavoro "Sistemi ad Agente e Multiagente" dell'Associazione Italiana per l'Intelligenza Artificiale (AI*IA) e dall'Associazione Italiana Tecnologie Avanzate Basate su concetti Orientati ad Oggetti (TABOO) in collaborazione con l'Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche, sede di Palermo, ed il Dipartimento di Ingegneria Informatica dell'Universitá di Palermo, ha come tema principale "Evoluzione dell'agent development: metodologie, tool, piattaforme e linguaggi".

L'obiettivo dell'evento é quindi esplorare i piú recenti risultati nel campo dello sviluppo di sistemi ad agenti. Questo include lo studio delle tecniche di progettazione, l'evoluzione delle piattaforme di sviluppo ed i linguaggi di codifica orientati agli agenti.

In questo volume sono raccolti diciassette articoli scelti dal Comitato di Programma per essere presentati al workshop. I contributi degli articoli coprono argomenti estremamente attuali ed interessanti nell'area di ricerca dei sistemi ad agenti come per esempio agenti e tecnologie di supporto alla cooperazione, ingegneria del software orientata agli agenti, metodologie e strumenti di sviluppo, simulazione orientata agli agenti.

Anche quest'anno é stata organizzata una miniscuola per studenti di dottorato e laureandi di secondo livello per discutere ed illustrare alcune tematiche fondamentali nel campo dei sistemi ad agenti.

Il Comitato Scientifico Organizzatore esprime un vivo ringraziamento a tutti coloro che hanno reso possibile la realizzazione dell'edizione di quest'anno: i componenti del Comitato di Programma, l'Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche, il Dipartimento di Informatica dell'Universitá degli Studi di Palermo, gli organizzatori locali, gli organizzatori della sessione demo e della miniscuola, gli sponsor e tutti collaboratori che hanno partecipato all'organizzazione.

Matteo Baldoni
Massimo Cossentino
Flavio De Paoli
Valeria Seidita

Comitato Scientifico Organizzatore

Matteo Baldoni
Massimo Cossentino
Flavio DePaoli
Valeria Seidita

Organizzatore Sessione Demo

Alfredo Garro

Cordinatori Miniscuola

Matteo Baldoni
Massimo Cossentino

Comitato Organizzatore Locale

Massimo Cossentino
Fabio Ferrara
Salvatore Gaglio
Ignazio Infantino
Roberto Pirrone
Riccardo Rizzo
Valeria Seidita
Pietro Storniolo
Alfonso Urso

Comitato di Programma

Stefania Bandini
Cristina Baroglio
Pietro Baroni
Federico Bergenti
Lorenzo Bettini
Enrico Blanzieri
Paolo Bouquet
Giacomo Cabri
Nicola Cannata
Francesco Donini
Rino Falcone
Giancarlo Fortino

Salvatore Gaglio
Alfredo Garro
Laura Giordano
Paolo Giorgini
Letizia Leonardi
Marco Mamei
Sara Manzoni
Viviana Mascardi
Ambra Molesini
Rebecca Montanari
Vito Morreale
Massimo Paolucci
Paola Quaglia
Alessandro Ricci
Giovanni Rimassa
Luca Sabatucci
Carla Simone
Emilio Tuosto
Eloisa Vargiu
Mario Verdicchio
Mirko Viroli
Giuseppe Vizzari

External Reviewers

Maciej Gawinecki

Direttivo WOA

Giuliano Armano
Matteo Baldoni
Antonio Corradi
Flavio De Paoli
Emanuela Merelli
Andrea Omicini
Agostino Poggi
Franco Zambonelli

Sponsors:

AIIA - Associazione Italiana per lIntelligenza Artificiale

TABOO - Associazione Italiana Tecnologie Avanzate Basate su concetti Orientati ad Oggetti

ICAR CNR - Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche, sede di Palermo

DINFO - Dipartimento di Ingegneria Informatica dellUniversit di Palermo

ENGINEERING INGEGNERIA INFORMATICA

Indice dei Contenuti

Agenti e Tecnologie di Sviluppo.

Fondamenti Teorici e Linguaggi.

Ingegneria del Software Orientata agli Agenti: Metodologie e Strumenti.

Multi-Agent Systems for e-Health and the CASCOM Project

Federico Bergenti and Agostino Poggi

***Abstract*—e-Health services and applications are probably one of the notable application fields where agent technology might act as a main actor in the near future. Multi-agent systems have been devised to deal with classes of problems–e.g., remote and heterogeneous software integration, remote monitoring and assistance–that are very typical for the large part of e-health services and applications. This paper describes some of the main reasons why multi-agent systems are now considered one of the best solutions for the realization and deployment of advances e-health software. The paper motivates this claim by addressing very general issues that have been previously identified as key problems of e-health. The paper is structured in two main parts. The first introduces the technological problems that characterize e-health and it shows how multi-agent systems tackle them. The second part describes an important European scale project that has recently adopted multi-agent systems to realize an e-health application scenario, i.e., decentralized emergency assistance.**

***Index Terms*—Agent-oriented software engineering, e-Health systems, m-Health systems, Cooperative systems.**

I. Introduction

MULTI-agent systems (MASs) are one of the most interesting fields in software research that have been contributing significantly in the past few years to the development of the theory and practice of complex distributed systems. Healthcare applications and services seem to be very suitable for taking advantage of MASs: *(i)* they are composed of loosely coupled, complex, heterogeneous, legacy systems; *(ii)* they manage data and resources that are inherently distributed; and *(iii)* they are often used by disperse users in (synchronous) collaboration.

The goal of this paper is to describe the main reasons why MASs should be considered one of the most interesting technologies for the development of healthcare systems. The paper also provides some guidelines to identify which kinds of healthcare applications can truly take advantage of MASs features. Then, the paper presents CASCOM [13], a recent European-scale project that adopted MASs to realize and field-trial a decentralized emergency assistance scenario.

Federico Bergenti is with Dipartimento di Matematica, Università degli Studi di Parma, Viale G.P.Usberti, 53/A, 43100 Parma, Italy (phone: +39-0521-096929; fax: +39-0521-906950; e-mail: federico.bergenti@ unipr.it).

Agostino Poggi is with Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Parma, Viale G.P.Usberti, 181/A, 43100 Parma, Italy (e-mail: agostino.poggi@unipr.it).

II. Agent Technology and e-Health

There is common agreement in the field that the buzzword *e-health* was introduced in 1999 as a consequence of the *e-** mania to talk about the provision of healthcare services through the Internet [11]. Notably, this buzzword was heavily promoted by the industry and by application and service vendors and soon the academic community started using it instead of the over-abused telemedicine. Such a widespread adoption of this new word was so wide and deep that anything that had to do with technology and health was quickly included. In order to clarify the obvious misunderstandings that immediately arose and to support such an important idea, the European Commission itself felt the urge to provide a common and generally acceptable definition of the word e-health as: *"the use of modern information and communication technologies to meet needs of citizens, patients, healthcare professionals, healthcare providers, as well as policy makers."* [7]

Besides the clarity–or possibly the confusion, someone may say–that the mentioned definition created, it is common understanding that e-health uses ICT for the provision of health-related services to sparse users, possibly on the move. This pushed interaction and communication as central to e-health and it immediately promoted agents-related technologies as ideal candidates to support next-generation e-health services and applications.

Similarly, e-health deals naturally with mobile users, e.g., in tele-assistance scenarios, and it is common understanding that e-health should accommodate transparently fixed and mobile users. So called *m-health* is another buzzword that has been recently proposed to stress this fact: m-health services should be accessible anyone, anywhere, anytime, anyhow, and *any-**. Fortunately, such characteristics are already common practice of agent technologies; e.g., JADE [4, 9] and JADE-LEAP [5] do take special care of transparently and dynamically allocating fixed and mobile users and agents.

Another important issue in e-health is about supporting the interoperability of (legacy) medical information systems in order to enable the integrated provision of advanced services capable of accessing information from different, remote sources. The dream of a single, universally accepted middleware supporting the development of new services and the renewal of legacy services was quickly abandoned and recent technologies that were originally intended to support the

(semantic) interoperation between heterogeneous services was quickly adopted and now is common practice. This, again, put agent technologies into the group of technologies capable of providing important contributions to e-health because of the inherent semantics-awareness of the interaction between agents that make them ready to deliver semantic interoperability.

Another important issue that most e-health services address regards the possibility of jointly supporting professionals in their highly specialized work. Computer-Supported Cooperative Work (CSCW) is already common practice in tele-surgery and tele-assistance and it seems an important ingredient of next-generation e-health services. Notably, the inherent cooperative nature of agents and the very fact that many CSCW technologies are already based on agents is another important contribution to the view of agent technologies as first-class citizens of e-health.

Similarly, another notable contribution of agent technologies in the development of next-generation e-health services and applications regards the central venue that security and privacy-awareness have in agent technology. In the agent realm, the issues of privacy-awareness are threaded under the umbrella of the more expressive notion of trust. Likewise, e-health strongly remarks the importance of preserving confidentiality and guaranteeing a high level of security for classified information about patients.

Even if the mentioned facts regarding the adoptability of agent technologies for next generation e-health services and applications can be convincing, we try to sustain our statement by adapting the well-known grouping criteria proposed in [3]. We say that agent-technology contributes to next-generation e-health from three points of view: *(i)* improving the quality of healthcare, *(ii)* facilitating the access to healthcare, and *(iii)* reducing costs.

The most important contribution that agent technology can provide to the overall quality of healthcare relies essentially on the possibility of feeding highly specialized healthcare professionals with the right information, at the right time, tailored to the patient. The proactive nature of agents and their semantic interoperability support such a need with the possibility of feeding users with information acquired from diverse sources and tailored to the concrete patient at hand. Thanks to the computerization of health records, that is now common practice in Western Countries, the transfer of complex health records globally and in real time increases the accessibility, unifies the information at every stage of complex healthcare processes and improves care continuity. Moreover, the longstanding tradition of expert systems that still lives behind the scenes of some agent technology can support healthcare professionals in using the provided information for taking the right decisions at the right time. Finally, the transparent integration of mobile terminals helps collecting data to quickly support contextualized decisions.

Notably, the scenario of allowing a quick and contextualized access to healthcare-related information from anywhere, at anytime and in the most convenient way can promote the long waited universality and equality of access to healthcare, especially for geographically or socially isolated patients. Such cases are extreme and they may seem visionary for the current lack of supporting infrastructures, e.g., communication networks and power supplies, but the inherent transparent decentralization that is always assumed in contemporary agent technologies is vital to facilitate the access to healthcare also in everyday scenarios. This is the case, e.g., of homecare to elderly, disabled or chronic patients. The widespread adoption of agent technologies at homes can help drastically reducing medical visits and related waitlists. Moreover, the proactive nature of agents helps creating a trusted link between agents and patients by having agents constantly pushing valuable information to patients with no need of explicit demands. Agents are good tools to help patients following preventive strategies and supporting self-care on a day-by-day basis.

The last grouping criterion that in [3] is about cost reduction of healthcare processes. This is an issue of notable importance for the inherent costs of quality healthcare and agent technologies are beneficial also from this point of view. The mentioned possibility of agents to provide the right information, at the right time, tailored for the patient supports efficiency the overall management of treatments. Moreover, the semantic interoperability of agents enables instant acquisition of information from its natural source, with minimal (if not null) pass of information along chains of intermediaries. Finally, the trusted and privacy-aware support that agents provide to healthcare processes is a valuable means to speedup and optimize many administrative procedures. All in all, we can summarize the contribution that agent technologies can provide to healthcare, from the point of view of cost reductions, with an earlier assistance and a structured prevention of the causes of further care.

III. The CASCOM Project

The EU funded project CASCOM [13] is one of the most recent attempts to bring the notable characteristics of agents to e-health. CASCOM is a technology-driven project that brings together three notable new technologies: MASs, semantic Web services and Peer-to-Peer (P2P). It finds its motivations in the following healthcare scenario, that was ran in many occasions throughout all Europe. The scenario involves Juha, a Finnish, who is abroad in Austria for business. Before leaving Finland, he loaded the CASCOM mobile agent suite on his mobile phone so that he can access CASCOM agents anywhere, anytime. Suddenly, he feels severe pain in his chest and he decides to call for help via his CASCOM personal agent in order to make his ignorance of German irrelevant. The agent prompts a few questions on Juha's phone screen and it forwards such information–in parallel–to the local emergency dispatch center and to the Finnish Emergency Medical Assistance service center (EMA). The agent contextualizes such information with Juha's current location and personal identification data. Such contextualized information allows the dispatch center immediately sending an ambulance down to

Fig. 1. Main parts of the CASCOM technology (from [13]).

Juha. On the road towards its destination, the CASCOM agent hosted on the mobile device of the physician on the ambulance selects and invokes the semantic Web services–physically located in Finland–that provide access to Juha's medical history. Thus, the physician on the ambulance can easily get an in-depth overview on Juha's health state on the way to the patient, without having seen him before. Once at destination, the ambulance takes Juha to the selected hospital and CASCOM agents are used to help the physician in acquiring the needed information to support his local supervision of Juha's health. Finally, just if needed, CASCOM agents are used to organize Juha's repatriation, by contacting needed semantic Web services for booking medically-equipped flights, and to exchange information between healthcare organizations in the process of his after-treatment.

From a functional point of view, CASCOM motivating scenario addresses well-known issues of emergency healthcare. Actually, a major challenge in emergency healthcare is to take the best decision on the treatment of the patient, with no background knowledge of the patient's medical history, e.g., known allergies and current medical treatments. CASCOM addresses such needs by providing physicians with contextualized information on the fly. Such information is acquired as needed directly from its source because CASCOM agents interact directly with the semantic Web services that organizations provide to access needed information in a secure and privacy-aware manner. This is by far a visionary scenario, because many organizations (see, e.g., [14]) are now in the process of opening their information systems via semantic Web services to allow foreign physicians to access patients' data, especially in emergency situations.

From a technological point of view, the distinguishing feature of CASCOM approach regards its openness and dynamism. Notably, no a-priori link is set between agents and/or semantic Web services and the pattern of communications is structured on the fly to satisfy the goals of agents. Similarly, the CASCOM service discovery agent identifies all relevant services and respective providers by means of the directory services hosted at EMA. Afterwards, the CASCOM planner agent creates an ad-hoc plan which composes the invocations to the services identified in the previous step. The CASCOM execution agent finally invokes all services specified in the plan and it applies failure handling mechanisms, just when needed. For the case of the motivating scenario, this includes accessing the healthcare information system that store Juha's medical history, which is located at EMA. As a result of this dynamically composed ad-hoc plan, information on Juha's medical history is collected–potentially from different sources–and transferred to the agent running on physician's device.

It is worth mentioning that CASCOM motivating scenario heavily uses one of the distinguishing features of agents that we mentioned in the previous section. Agents are flexibly deployed on fixed and mobile devices and they communicate using wired and/or wireless links on the sole basis of their current goals. For example, the physician's agent is initially started on the base station of the emergency center where a WLAN connection is available. As soon as the emergency car leaves the base station, it also leaves the connection range of the WLAN and the agent communication is seamlessly migrated to UMTS.

The main delivery of CASCOM project is a general-purpose, open-source middleware that implements a generic architecture for agent-based coordination and execution of semantic Web services in a so called Intelligent Peer-to-Peer (IP2P) network, i.e., a decentralized network of loosely coupled, proactive peers with no restriction on the actual means of connectivity. Such architecture transparently accommodates both mobile and fixed users into a seamless environment. In short, the CASCOM architecture provides easy, seamless and contextualized access to semantic Web services anytime, anywhere and using any device. The main ingredients of CASCOM architecture are depicted in Figure 1.

CASCOM architecture relies on a layered approach. The four main components of this architecture link the application layer with the underlying networks and are described in some detail below.

The Networking Layer provides a generic, secure, and open IP2P network infrastructure taking into account varying quality of service of wireless communication paths, limitations of resource-constrained mobile devices, and contextual variability of nomadic environments. In details, it provides the following functionality:

1) Efficient, secure, and reliable agent message transport communication over wireless (and wired) communication paths independently of the access technology;
2) Provision of network-related context information to the context subsystem;
3) Low-level service discovery; and
4) Agent execution environment for resource-constrained mobile devices.

Setting out from the services of the Networking Layer, and based on the functionalities offered by both the Context-Awareness and the Security and Privacy subsystems, the Service Coordination Layer takes an agent-based approach towards flexible semantic Web service discovery and coordination. Its main functionality is twofold:

1) Semantic service discovery, i.e., service discovery and semantic matchmaking; and
2) Service coordination, i.e., service composition, execution, and possible re-planning.

The Context subsystem, orthogonal to the layers described above, is in charge of acquiring, storing, and providing context information to both those layers.

The Security and Privacy subsystem, also orthogonal to the Networking and Service Coordination layers, is responsible for ensuring security and privacy of information throughout the different parts of the infrastructure. One of the main things we need to protect is the data that every node of the IP2P network maintains. In details, data confidentiality, integrity, and availability are topics of concern that any approach to security must address. The security and privacy functionality was considered at every level of the CASCOM architecture. This enables instant take-up of the CASCOM concepts for service-oriented business applications.

The use of agents to support nomadic computing has been intensively studied in the past few years and CASCOM builds on previous results by using a well-known and appreciated agent platform, i.e., JADE-LEAP [5], as the basis for the implementation of its architecture. Unfortunately, JADE-LEAP did not really take into account many P2P issues and the CASCOM project had to address them explicitly. This concerned the realization of a novel support for communication that does not make any assumption on the actual physical and logical topology of the network and it does not even require the availability of a special node acting as a bridge to the fixed network.

IV. Conclusions

According to Altman [1], one of the ten infrastructure challenges that Artificial Intelligence has to face to provide valuable contribution to healthcare regards having medical records *"based on semantically clean knowledge representation techniques."* Agents not only provide the needed tools to turn such a challenge into reality, but they also provide a clean way to make such records available anywhere, at any time. This is a notable improvement of the proposed challenge and agents are ideal means to achieve it.

Moreover, we agree with [2] and we believe that a key component of the *"smart use of computation"* that authors mention will be the use of agent technology. Agents will improve healthcare organizations and will also support doctors and caregivers. However, we also agree on the mentioned issues regarding the impact of the use of agent technology with patients, which will not only be an improvement but a radical change in how healthcare and assistance will be provided. Unfortunately, the adoption of agent technologies within e-health is taking place quite slowly and, despite the number of research projects on the topic, this by far an assessed practice. We believe that the main reasons for this are not in the agent technology itself, which is generally well accepted; rather they originate from the inherent difficulties of having ICT accepted in healthcare from the technical, social, political, legal and economical points of view. This is a well discussed topic in the literature and interested readers can refer to some notable works [3, 6, 8, 10, 12, 15].

Acknowledgment

This work is partially supported by project CASCOM (FP6-2003-IST-2/511632). The CASCOM consortium is formed by DFKI (Germany), TeliaSonera AB (Sweden), EPFL (Switzerland), ADETTI (Portugal), URJC (Spain), EMA (Finland), UMIT (Austria), and FRAMeTech (Italy). The authors would like to thank all partners for their contributions.

References

[1] R. B. Altman. AI in Medicine: The Spectrum of Challenges from Managed Care to molecular Medicine. AI Magazine, Vol. 20, No. 3, pp. 67-77, 1999.

[2] R. Annicchiarico, U. Cortés, C. Urdiales (eds.) Agent Technology and e-Health, Birkhäuser Basel, 2007.

[3] G.A. Barnes, M. Uncapher. *Getting to e-Health: The Opportunities for Using IT in the Health Care Industry*. Information Technology Association of America (ITAA), 2000.

[4] F.L. Bellifemine, G. Caire, D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley and Sons, 2007.

[5] F. Bergenti, A. Poggi, B. Burg, G. Caire. *Deploying FIPA-Compliant Systems on Handheld Devices*. IEEE Internet Computing 5(4):20-25 2001.

[6] Deloitte Center for Health Solutions. *Promoting Physician Adoption of Advanced Clinical Information Systems: A Deloitte Point of View*, 2006.

[7] European eHealth Ministerial Declaration, 22nd May 2003. Available at: http:/ec.europa.eu

[8] A.R. Jadad, V. Goel, C. Rizo, J. Hohenadel and A. Cortinois. *The Global e-Health Innovation Network – Building a Vehicle for the Transformation of the Health System in the Information Age*. Business Briefing: Next Generation Healthcare, 2000.

[9] JADE software Web site. Retrieved September 23rd, 2008 from http://jade.tilab.com.

[10] S. Laxminarayan and B.H. Stamm. *Technology, Telemedicine and Telehealth*, Business Briefing: Global Healthcare Issue 3, 2002.

[11] K. McLendon. *E-commerce and HIM: Ready or not, here it comes*. Journal of the American Health Information Management Association, 71(1):22-23, 2000.

[12] A. Ohinmaa, D. Hailey and R. Roine. *The Assessment of Telemedicine: General principles and a systematic review*. INAHTA Joint Project. Finnish Office for Health Care Technology Assessment and Alberta Heritage Foundation for Medical Research, 1999.

[13] M. Schumacher, H. Helin (eds.). *CASCOM: Intelligent Service Coordination in the Semantic Web*, Birkhauser Boston, 2008.

[14] M. Springmann, L. Bischofs, P.M. Fischer, H.-J. Schek, H. Schuldt, U. Steffens and R. Vogl. *Management of and Access to Virtual Electronic Health Records*, in Digital Libraries: Research and Development, Springer, Heidelberg, 2007

[15] P. Wilson, C. Leitner and A. Moussalli. *Mapping the Potential of eHealth: Empowering the Citizen through eHealth Tools and Services*, eHealth Conference, Cork, Ireland, 2004.

Using Agent Technology as a Support for an Enterprise Service Bus

Paola Mordacci, Agostino Poggi, Carmelo Giovanni Tiso, Paola Turci

***Abstract* — The research in distributed artificial intelligence has been addressing for several years the problem of designing and building coordinated and collaborative intelligent multi-agent systems. This interesting and advanced work can be fruitfully exploited in the area of service-oriented computing if agent technology is appropriately engineered and integrated with the key technologies in this field.**

To support this claim, in this paper we show how the agent technology integrated in an Enterprise Service Bus allows the conception and realization of real flexible, adaptive intelligent service-oriented systems.

***Index Terms* — Multi-agent systems, service oriented architecture, ESB, rule engine.**

I. Introduction

Multi-agent systems and service-oriented computing are still evolving towards a complete maturity. The quite wide spreading of the service-orientation design paradigm and of the related technologies is having a twofold influence on the evolution of agent technology.

On the one hand, several researchers belonging to the agent community are convinced that this technical area is a natural environment in which the agent technology features can be leveraged to obtain significant advantages. It is plain, in fact, that service-oriented technologies cannot provide by themselves the autonomy and social and proactive capabilities of agents. Agents, taking advantage of their social ability, exhibit a flexible coordination that makes them able to both cooperate in the achievement of a global goal and compete in the distribution of resources and tasks.

On the other hand, one of the requirements for the success of multi-agent systems is that they have to guarantee an easy integration with other widely used industrial technologies.

Driven by such motivations, a number of research works have been undertaken with the aim of tackling the problem of integrating service-oriented technologies with multi-agent systems.

The aim of this paper is in a slight different direction, that is we try to cope with the issue of adding collaboration and coordination capabilities in a service-oriented architecture. In particular, we have considered an enterprise service bus (ESB) - a software infrastructure that facilitates the realization of SOA systems by acting as a middleware through which a set of services are made available - and we have shown how the integration of agent technology in an ESB may be considered very promising.

The brief survey of the literature in the area of service-oriented technologies, reported in the background section, has the aim of showing the scenario in which the agent possibly contribution should be set and at the same time to give a short preamble acting as a motivation and rationale of the research work that we have done. Section 3 gives an overview of the related research. Section 4 deals with the integration of JADE agents in ServiceMix, an open source ESB based on JBI specifications. Section 5 describes a simple but realistic application which shows how the powerful synergism between agent and service-oriented technologies could be very promising. The paper ends by drawing some conclusions around the results of the work done.

II. Background

The growing demand for high-levels of interoperability by organizations that want applications to have broader reach, have stimulated the rapid growth of novel standards, technologies and paradigms with the aim of giving an answer to such problem. The most appropriate response to this need seems a service-oriented architecture (SOA), i.e. a system assembled from a loosely coupled collection of services and in particular of Web services - the integration technology preferred by organizations implementing SOA (Dustdar & Schreiner, 2005),

For the sake of clarity, it is necessary to say that there is no one recognized definition of SOA, however a baseline of concepts and principles and a strategic vision have emerged and collectively characterize the service-oriented design paradigm as an approach to defining integration architectures based on the concept of service.

The last outcome of the SOA movement has been the ESB, an infrastructure that can be used as a backbone upon which to build service-oriented applications. As with SOA, there has been no industry agreed on the definition of ESB so far. It is

P. Mordacci is a student at DII, University of Parma, Viale Usberti 181A, 43100, Parma, Italy (e-mail: paola.mordacci@studenti.unipr.it).

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

C. G. Tiso is a student at DII, University of Parma, Viale Usberti 181A, 43100, Parma, Italy (e-mail: carmelo.giovanni.tiso @studenti.unipr.it).

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

still a controversial issue if it is a pattern, a product or an architectural component. According to the authoritative Gartner's definition: "an ESB is a new architecture that exploits Web services, messaging middleware, intelligent routing and transformation". Anyway, one thing seems to be unquestionable; using an ESB is the quickest and most cost-effective way to address the challenge of the enterprise application integration. Several middleware vendors provide or have on their roadmap an ESB.

However, a still open problem is that the information and the research activities in this area are quite fragmented (Papazoglou et al., 2006). What clearly emerges is that the subject of service-oriented applications turns out to be vast and enormously complex and more work needs to be done in order to realize real flexible, adaptive intelligent service-oriented systems. As a matter of fact, current SOA implementations are still restricted in their application context to being an in-house solution for companies (Domingue, 2008).

In the attempt to delineate an effective solution, some researchers have envisaged as strategic the integration of SOA with both semantic and Web technologies (Domingue, 2008). Others have turned their eyes towards the agent technology, as an interesting means for the realization of more effective and reliable service-oriented systems and for SOA to be successful on a worldwide scale. Clearly, such technologies are not competitive but complementary and therefore someone else has been developing systems which integrate SOA, semantic Web and multi-agent systems (Negri, 2006).

The agents ability of operating in dynamic and uncertain environments allows coping with the usual problems of failures or unavailability of services and the consequent need of finding substitute services and/or back tracking the system in a state where execute an alternative workflow. Moreover, the capabilities of some kinds of agent of learning from their experience make them able to improve their performance over the time avoiding untrusted and unreliable providers and reusing successful solutions. These remarkable agents' features have mainly driven the research activities of the agent community in the area of service-oriented computing so far, that is agents as a valuable support for realizing Web services composition.

Other interesting features, which can be the main ingredients for automatic cooperation between enterprise services, are the agents' capabilities of collaborating and coordinating themselves. In a business environment, an example would be a broker that has frequently to seek providers as well as buyers dynamically, to collaborate with them and finally to coordinate the interactions with and between them in order to achieve its goals. An intelligent service-oriented infrastructure could do it automatically or semi-automatically, within the defined constraints. These further agents' characteristics have been considered in the research work presented in this paper.

III. Related Work

The problem of realizing service-oriented applications exploiting agent technology has mainly concerned so far three fundamental issues: (i) the management of the interactions between agents and Web services; (ii) the execution of a workflow or more in general of a plan that describes how Web services interact; (iii) the discovery of the Web services that perform the tasks required in the plan.

In other words, to date the efforts have been mainly devoted to provide an effective solution to the problem of Web services composition.

In this context, the first issue that the agent community has had to cope with was the mapping between the different semantic levels of the two paradigms or patterns of communication (Greenwood & Calisti, 2004; Nguyen, 2005; Shafiq et al., 2005).

The next step has been the implementation of prototypes of agent based frameworks coping with the static and dynamic composition of Web services, through the use of workflow technologies (Buhler & Vidal, 2005). In fact, once the infrastructure, enabling a bi-directional connectivity between the two technologies, is in place an agent can play the role of the orchestrator of dynamic Web service compositions. Even if the current multi-agent solutions, aiming at realizing an effective agent-based service composition, are still in a preliminary phase and certainly need to be improved (Savarimuthu et al., 2005), a lot of researchers and software developers are really interested in giving a significant contribution in this direction.

As far as the automatic cooperation between enterprise services is concerned, to the best of the authors' knowledge, there are very few ongoing studies. The most significant is the one reported in (Paschke et al, 2007). In this work, the authors combine the ideas of multi-agent systems, distributed rule management systems and service-oriented and event-driven architectures. The work is mainly focused on the design and implementation of a pragmatic layer above the syntactic and semantic layers. Taking advantage of this layer, individual agents can form virtual organizations with common negotiation and coordination patterns. An enterprise service bus is integrated as a communication middleware platform and provides a highly scalable and flexible application messaging framework to communicate synchronously and also asynchronously with external services and internal agents. To date, the authors have developed an interesting methodology and an architectural design but they have only outlined a possible implementation of the system.

IV. Integration of JADE in ServiceMix

Apache ServiceMix (ServiceMix) is an open source ESB released under the Apache license, based on the Java Business Integration (JBI) standard (JBI, 2005). These two factors, open source and standard-based, allow for low entry cost, maximum flexibility, reuse and investment protection.

ServiceMix is lightweight and easily embeddable, integrates

Spring support and can be run at the edge of the network (inside a client or server) as a standalone ESB provider or as a service within another ESB.

ServiceMix includes a complete JBI container supporting all parts of the JBI specification: allows plug-in services (as configuration of JBI components) which can be combined to create a SOA; provides a Normalized Message Router (NMR), as the backbone of all communication (based on the exchange of normalized messages having an xml content) between services within the ESB; supports the four JBI message exchange pattern (i.e. protocols defining the messages exchanged between a service consumer and a service provider, involved in a service invocation); includes full support for the JBI deployment units with hot-deployment of JBI components.

ServiceMix distribution already includes many JBI components that provide support for some of the most common protocols and engines. JBI components are deployed to the JBI container and simply run in memory waiting for configuration. In fact, in order to make use of these components as an application developer, one has to provide a configuration for each component he/she intends to use.

ServiceMix therefore strongly encourages the "configure don't code" integration approach, that allows a faster and simpler (but less flexible..) integration.

Configurations are implementation specific but the packaging is defined by the JBI specifications. Each component configuration must be packaged as a Service Unit (SU) and each SU must be wrapped in a Service Assembly (SA). These are simply ZIP/JAR files that contain an XML descriptor.

According to the JBI specification, JBI components come in two flavours: Binding Component (BC) and Service Engine (SE).

ServiceMix BCs are used to communicate outside the JBI environment by means of a lot of supported remote protocols: HTTP/S, HTTP+SOAP, JMS, FTP, SMTP, XMPP, etc.

BCs are responsible for normalizing incoming (relative to JBI environment) messages and denormalizing outcoming messages.

ServiceMix SEs provide some type of logic inside the JBI environment. Some examples of SE components in ServiceMix include: rules engines, BPEL engines, XSLT engines, Plain Old Java Object (POJO), annotated POJO, schema validation of documents, support for Enterprise Integration Pattern, etc.

According to the "configure don't code" integration approach, to integrate JADE agents in an ESB is opportune to select the most appropriate ServiceMix component to configure, in order to obtain a service satisfying our goal. The basic idea is to build a service, acting as a proxy between the NMR and the agent community, that can interact with both services deployed in the JBI environment and JADE agents belonging to the agent community. By means of this proxy, services deployed in ServiceMix ESB can access capabilities offered by agents. Services can send normalized messages to the proxy service that can map them in ACL messages and forward them to a specific agent within the community. On the other hand, the proxy can receive requests from the agent community, map them in normalized messages and forward them to a specific service exposed in the bus. To make this possible, each proxy needs to be in relation to an associated agent that represents a proxy gate towards the agent community

A possible solution is based on the ServiceMix jsr181 component: a JBI SE exposing POJO as services on the bus.

When a normalized message, addressed to one of such services, arrives from the bus, an appropriate method of the POJO will be called, passing the service invocation parameters (specified as an xml content of the message) as method parameters. The request is forwarded in some way within such method to its associated agent (further details will be explained below). We discarded this option because the marshalling of the normalized message content is handled automatically and in a quite rigid manner

The chosen solution is based on the ServiceMix bean component, since this is a very flexible component that can receive messages from the NMR and process them in any way it likes. The bean component gives the developer the freedom to create any type of message handling but as a counterpart she/he has to hand coded all the way.

In order to configure this component one has to define a simple XML file, where the service name, and a Java class representing the bean are mainly defined. These two artefacts have to be packaged as a SU.

When a normalized message, addressed to such a service, arrives from the bus, an appropriate method of the bean will be called, passing the message as a method parameter. The message will be processed and the requested actions will be taken. Using JBI API the bean, in turn, can send the message towards other services.

The proxy service implementation is therefore accomplished by configuring the ServiceMix bean component, instantiating within the bean constructor its associated JADE agent that, using JADE API, will be executed within a new secondary container. It is therefore necessary that first JADE main container is executed, then the bean is initialized. The connection between the two entities is handled by means of references; the bean holds a reference to the agent and the agent holds a reference to the bean.

Within a bean method, invoked subsequently to the reception of a message by the proxy service, it is so possible to call an appropriate agent method that, in turn, can interact with whatever agent belonging to same or other FIPA communities.

Similarly, any agent can send an ACL message to the proxy agent that by means of the proxy service can interact with services exposed in the bus.

V. On-line Book Selling

The framework has been experimented in the realization of

an online book selling application where there are N book sellers and one broker. The broker is responsible for providing its users with elementary and aggregated information collected through the collaboration with sellers.

In Figure 1, the full system architecture (with N=2) is shown.

The sellers and the broker are distributed in the network. Each entity, i.e. the coupling agent-service, has an associated ServiceMix JBI container in which a number of JBI components are deployed. The blue rectangles represent services as configuration of a JBI component. The particular services, that is the sellers or the broker, act as a proxy between the NMR and the agent community. The agent associated to each proxy (i.e. respectively agent seller or agent broker) is linked to its proxy by a dotted line. A seller maintains book data within a relational database that is handled by an agent seller or possibly another agent of the community. Finally, in the figure it is also highlighted how all interactions between services within a JBI container are mediated by the NMR.

Each seller entity provides its users with the following features: (i) an easy support to obtain the complete book list offered by the seller; (ii) the possibility, if some constraints are met, to obtain a discounted price of a specified book; (iii) the opportunity to add a new book to the seller's book database.

Users can express a request for a seller book list by means of a web interface: the http request is received by the http-book-list service (a configuration of ServiceMix BC servicemix-http).

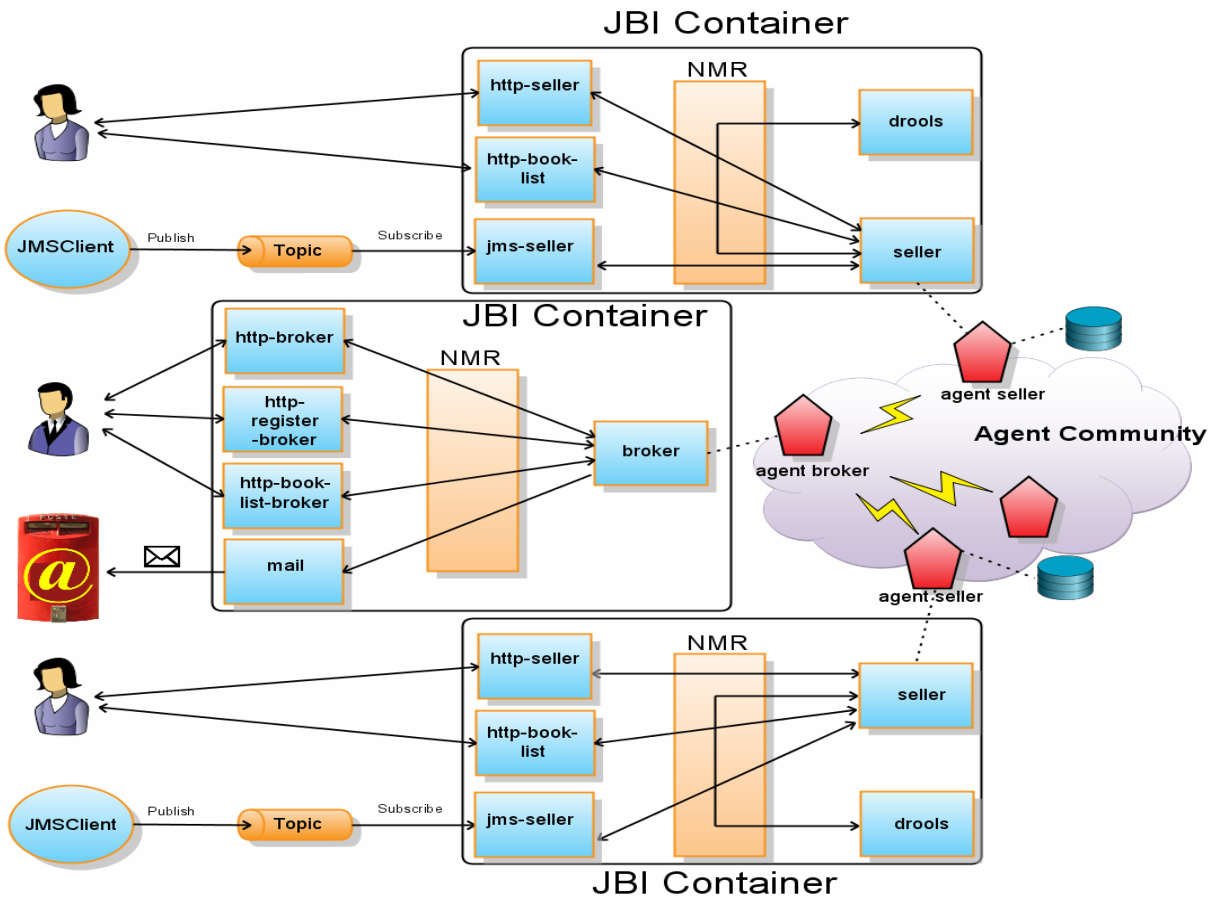


Figure 1- On-line book selling system architecture

This service maps the received http request in a message normalized content (an XML document), that is sent to the seller service (as already mentioned, a configuration of ServiceMix SE servicemix-bean).

This proxy forwards the request (mapped in an ACL message) to the agent seller, which fulfils the request or delegates the agent acting as a database handler.

The agent seller returns the response, an XML document containing the book list, to the proxy that, in turn, maps such response in a normalized message, then returned to the http-book-list service. At last, this service maps the received message in a proper HTTP response. Once the user has obtained the seller book list, he/she can request the final price (i.e. the price after the discount) of a selected book, by means of a web interface. It is necessary to specify, besides the book title, the user nickname and the desired quantity, since in this

simple application the final price of a book is determined by a set of business rules depending on this information.

The HTTP request, containing the user specified parameters, is received by the http-seller service (a configuration of ServiceMix BC servicemix-http).

Similarly to the http-book-list service, this service maps the received http request in a message normalized content, that is sent to the seller service. The seller service asks its linked agent seller the book base price that subsequently will be forwarded, together with user's data, to the drools service, a configuration of ServiceMix SE servicemix-drools.

Drools is a business rule management system with a forward chaining inference based rules engine, more correctly known as a production rule system, based on an enhanced implementation of the Rete algorithm. Drools is a JBoss open source project compliant to the JSR-94 standard for business rules engine.

A rule engine is mainly based on two concepts: a set of rules (the actual logic) and assertions (facts accessed by rules). Rules are coded in a drl file, following the Drools syntax. Assertions can be passed through Java.

To configure a servicemix-drools is necessary to define, besides an xbean.xml file, a drl file containing the business rules. Each normalized message received by a drools service is processed by the drools service as an assertion. A drl file can access JBIHelper class methods to interact with NMR (e.g. it is possible to create normalized message and send them to a specified destination).

In such way, the drools service can answer the request received by a seller service replying with a message containing the discount, determinate according to the codified business rules.

To add a new book to the seller's book database, an external application, that publishes a JMS message (containing new book data) on a specified JMS Topic, is used. Such message is withdrawn by jms-seller service (a configuration of ServiceMix BC servicemix-jms) representing a JMS TopicSubscriber. This service maps the received message in a new message addressed to the seller service that will forward it to the seller agent or through it indirectly to the agent acting as book database handler.

The broker entity provides his users with the following features: (a) a support to obtain the complete book list, as the union of seller book lists; (b) the possibility to know the seller that offers a selected book at the best final price; (c) subscription to broker's newsletters; (d) notification by e-mail of new books.

Cooperating with the agent sellers in accordance with the FIPA protocols, the agent broker can collect the information necessary to provide the aforementioned functionalities.

Users can request the complete book list by means of a web interface: the http request is received by the http-book-list-broker service (a configuration of ServiceMix BC servicemix-http). Similarly to the seller case, this service maps the received http request in a normalized message, that is sent to the broker service (a configuration of ServiceMix SE servicemix-bean: the broker proxy).

This proxy forwards the request to the broker agent, that will interact with every seller in order to collect the various lists. The broker agent builds the union of the seller book lists, returns it to the proxy and so on until a proper HTTP response is sent to the requester.

Once the user has obtained the complete list, he/she can ask the broker which seller offers the selected book at the best final price. For the same reasons explained above in the seller case, the user has to specify, besides the book title, the user nickname and the desired quantity.

The HTTP request is received by the http-broker service and then sent to the broker service. This proxy forwards the request to the agent broker, which collects information from every agent seller, offering the requested book. Once agent broker receives a response from each seller or a timeout is reached, returns the best received price, through the proxy, to the user.

An http-register-broker service has been made available to allow users to register with broker's newsletter. Users need to compile an html form specifying their name and e-mail.

As usual, this service maps the received http request in a normalized message, which is sent to the broker service. Such service will forward it to the agent broker, that consecutively will delegate the agent responsible for handling users' data to add the new user to the database.

Finally, when agent seller receives notification about the insertion of a new book, it sends an ACL message containing the new book data to the agent broker and to the agent in charge of handling the book database. By means of the broker service, the agent broker, once it has collected user data from the agent responsible for handling user database, sends a normalized message to the mail service (a configuration of ServiceMix BC servicemix-mail), responsible for sending an e-mail to each registered users

VI. Conclusion

In this paper, we have addressed the integration of multi-agent systems in an ESB, highlighting the benefits that both communities can achieve from this solution. On the one hand, multi-agent systems are able to interact with the key emerging technologies in the area of service-oriented computing. On the other hand, the interesting and advanced work carried out in several years by the agent community can be fruitfully exploited in the area of service-oriented computing.

Finally, we have tried to prove, by means of a simple but realistic application, how the powerful synergism between these technologies could be very promising.

Besides the significant role that collaboration and coordination capabilities play in our application, another important point that clearly emerges from our application is that the agent technology can make successful the exploitation of the ESB technology, based on JBI specification, on a worldwide scale. To date, the ESB technology does not provide a support for the federation of distributed JBI

container; each JBI container can interoperate with the others as with remote consumers or providers, that is by means of communication protocols supported by binding components.

REFERENCES

[1] Buhler P.A., Vidal, J.M. (2005). Towards Adaptive Workflow Enactment Using Multiagent Systems. Information Technology and Management, 6(1):61-87

[2] Greenwood D. & Calisti M. (2004). Engineering Web Service-Agent Integration. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pages 1918–1925. The Hague, Netherlands.

[3] Domingue J. & Fensel D. (2008) Toward a Service Web: Integrating the Semantic Web and Service Orientation, IEEE Intelligent Systems January/February, 2008.

[4] Dustdar, S., & Schreiner, W. (2005). A survey on web services composition, Int. J. Web and Grid Services, Vol. 1, No. 1, pp.1–30

[5] JBI Java Business Integration 1.0, Final Release August, 2005. available from http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html

[6] Negri A., Poggi A., Tomaiuolo M., Turci P. (2006). Agents for e-Business Applications, In AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. (pp. 907-914). Hakodate, Japan. ACM Press

[7] Nguyen X. T. (2005). Demonstration of WS2JADE. In Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pages 135–136. Utrecht, The Netherlands

[8] Papazoglou1 M., Traverso P., Dustdar S., Leymann F. (2006). Service-Oriented Computing Research Roadmap.

[9] Paschke, A., Boley, H., Kozlenkov, A., Craig, B. (2007). Rule Responder: RuleML Based Agents for Distributed Collaboration on the Pragmatic Web. 2nd International Conference on the Pragmatic Web Oct 22-23, 2007, Tilburg, The Netherlands.

[10] Savarimuthu B. T. R., Purvis M., Purvis M. & Cranefield S. (2005). Integrating Web Services with Agent Based Workflow Management System (WfMS). In WI '05: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence. (pp. 471 – 474). Washington, DC. IEEE Computer Society.

[11] Servicemix, available from http://servicemix.apache.org/

[12] Shafiq M. O., Ali A., Ahmad H. F., Suguri H. (2005). AgentWeb Gateway - a Middleware for Dynamic Integration of Multi Agent System and Web Services Framework. In Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, pages 267–270, Washington, DC. IEEE Computer Society

A Prolog-Based MAS for Railway Signalling Monitoring: Implementation and Experiments

Daniela Briola†, Viviana Mascardi†, Maurizio Martelli†,
Gabriele Arecco†, Riccardo Caccia‡, Carlo Milani‡
† DISI, Università degli Studi di Genova,
Via Dodecaneso 35, 16146, Genova, Italy
{Daniela.Briola, Viviana.Mascardi, Maurizio.Martelli}@unige.it, gabriele.arecco@gmail.com
‡ IAG/FSW, Ansaldo Segnalamento Ferroviario S.p.A., Italy
{Caccia.Riccardo, Milani.Carlo}@asf.ansaldo.it

***Abstract*—This paper describes the outcomes of a project that involved DISI, the Computer Science Department of Genoa University, and Ansaldo Segnalamento Ferroviario, the Italian leader in design and construction of signalling and automation systems for conventional and high speed railway lines. The result of the project, started in February 2008 and ended in September 2008, is an implemented MAS prototype that monitors processes running in a railway signalling plant, detects functioning anomalies, and provides support to the early notification of problems to the Command and Control System Assistance. The MAS has been implemented using DCaseLP, a multi-language prototyping environment developed at DISI, that provides libraries for integrating TuProlog agents into Jade. Due to the intrinsic rule-based nature of monitoring agents, Prolog has been proved extremely suitable for their implementation.**

I. Introduction

Distributed diagnosis and monitoring represent one of the oldest application fields of rule-based software agents.

ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems [12]) was Europe's largest ever project in the area of Distributed Artificial Intelligence. It was employed for monitoring and controlling the cycle of generating, transporting and distributing electrical energy to industrial and domestic customers, for the Iberdrola company, one of the world's leading private energy groups [8]. ARCHON's Planning and Coordination Module was implemented as a rule-based system.

In [25], Schroeder et al. describe a declarative and reactive diagnostic agent based on extended logic programming. Both the inference engine used for computing diagnoses and the reactive layer that implements a meta-interpreter for the agent were implemented in Prolog extended with communication facilities.

Both ARCHON and Schroeder's agent systems date back to more than ten years ago. In the meanwhile, a large number of MASs for diagnosis and monitoring has been developed, many of them based on rule-based approaches.

There are many good reasons for choosing a MAS approach to process diagnosis and monitoring. Some of them had been clearly stated by N. Jennings[1]:

- *To permit reasoning based on information of different granularity*: The MAS may be organised in a hierarchy of agents with different competencies, starting from those at the lowest level, directly interfaced with the processes, and going up towards more and more sophisticated agents, equipped with expert system-like rules for devising problems according to the information coming from agents below in the hierarchy, and reporting aggregated information and diagnosis to the agents higher in the hierarchy.
- *To enable a number of different problem solving paradigms to be utilised*: Rephrasing Jennings' considerations,

  > *there is no universally best problem solving paradigm: procedural techniques may be required for algorithmic calculations, whereas symbolic reasoning based on heuristic search may be the best approach to diagnosis. A distributed approach enables each component to be encoded in the most appropriate method.*
- *To meet the application's performance criteria*: The distributed nature of a MAS makes it a suitable solution for monitoring different processes concurrently, thus gaining in performance and responsiveness.

The motivations for choosing a Distributed Artificial Intelligence approach given by [5], [1] also apply to the process diagnosis and monitoring domain: *economy*, *robustness*, *reliability*, *natural representation of the domain*.

Situational awareness, that is mandatory for the successful monitoring and decision-making in many scenarios, is one of the founding characteristics of intelligent software agents [13]. When combined with reactivity, situatedness may lead to the early detection of, and reaction to, anomalies.

Last but not least, an agent-based distributed infrastructure can be added to any existing system with *minimal or no impact* over it. Agents monitor processes, be them computer processes, business processes, chemical processes, by "looking over their shoulders" without interfering with their activities.

The simplest and most natural form of reasoning for producing diagnoses starting from observations is rule-based. Also,

[1]N. Jennings, *ARCHON: Cooperating Agents for Industrial Process Control*, http://users.ecs.soton.ac.uk/nrj/download-files/archon/arch10.html

a monitoring activity may be profitably modelled by means of reactive rules. If the agents employed in the MAS are implemented in a rule-based language, the implementation of a rule-driven reasoning mechanism is greatly simplified.

This paper describes the results of a joint academy-industry project started at the beginning of 2008. The project involves the Computer Science Department of Genoa University, Italy, and Ansaldo Segnalamento Ferroviario, a company of Ansaldo STS group controlled by Finmeccanica, the Italian leader in design and construction of railway signalling and automation systems.

The outcome of the project is a MAS prototype that monitors an Ansaldo process, which controls railway signalling, and reacts to anomalies either by interacting with other agents in the MAS or by killing the process that raised the anomaly. The MAS has been implemented in Jade [6] extended with TuProlog [9] by means of the DCaseLP libraries [18].

At this stage of the project, the MAS is running in an "off-line" modality: agents are not installed on machines in Ansaldo and the MAS tests have been carried out at DISI. Agents read original log-files provided by Ansaldo as if new lines were added by the monitored process once every m minutes. Agents act in accordance to the content of the last lines read, thus simulating an "on-line" reading phase. Also the "kill process" action is just simulated at the time of writing. When Ansaldo will fully integrate the MAS into its system, log-files will be read in real-time as they are produced by the monitored process. Furthermore, the MAS will be allowed to really kill processes and to contact the Assistance Centre of the Command and Control System for Railway Circulation to report the anomalies in an automatic way. When the MAS will be installed on the Ansaldo SCC system, other ways to manage processes, aside from the "kill process", will be studied.

The paper is structured in the following way: Section II describes the operating scenario and the MAS architecture, Section III describes the rule-based implementation of the agents, Section IV shows the potential of the system by discussing different execution runs. Section V overviews the related work and concludes.

II. Operating Scenario and MAS Architecture

The architecture of the MAS and its operating scenario have been extensively described in [17]. In this section we briefly recall them to allow the reader to understand the original contribution of this paper, namely the system implementation and execution described in Sections III and IV.

A. Operating Scenario

The Command and Control System for Railway Circulation ("Sistema di Comando e Controllo della Circolazione Ferroviaria", SCC) is a framework project for the technological development of the Italian Railways ("Ferrovie dello Stato", FS). It is based on the installation of centralised Traffic Command and Control Systems, able to remotely control the plants located in the railway stations, and to manage the movement of trains from the Central Plants (namely, the offices where instances of the SCC system are installed).

The SCC can be decomposed into five subsystems

- *Circulation*, for remote control of traffic and for making circulation as regular as possible;
- *Synoptic Frame*, for representing railway lines, nodes, and trains, in a summarised, easily understandable way;
- *Diagnosis and Upkeep*, for the diagnosis of plants and equipments of the SCC;
- *Information to Customers*, for providing information to the FS customers;
- *Remote surveillance, intrusion avoidance, fire detection, emergency management*, for dealing with all these situations efficiently.

The MAS we have implemented monitors and reacts to problems of one critical process belonging to the *Circulation* subsystem: *Path Selection*.

The Path Selection process is the front-end user interface for the activities concerned with railway regulation. There is one Path Selection process running on any workstation in the SCC and each operator interacts with one instance of this process. The Path Selection process visualises decisions made by the Planner process and allows the operator to either confirm or modify them.

The Planner process is the back-end elaboration process for the activities concerned with railway regulation. There is only *one* instance of the Planner process in the SCC, running on the server. It continuously receives information on the position of trains from sensors located in the stations along the railway lines, checks the timetable, and formulates a plan for ensuring that the train schedule is respected. Operators may modify the Planner's decisions thanks to the Path Selection process.

By integrating a monitoring MAS into the circulation subsystem, we equip any operator of the Central Plant (any workstation) with the means for early detecting anomalies that, if reported to the SCC Assistance Centre in a short time, and before their effects have propagated to the entire system, may allow the prevention of more serious problems.

To have an idea of the dimensions of an SCC and of the area it controls, the SCC of the node of Genoa, that we employed as a case-study for the implementation of our MAS, controls an area with 255 km of tracks, with 28 fully equipped stations plus 20 stops (Figure 1).

One of the 16 user workstations of Genoa's SCC is shown in Figure 2. The synoptic frame can be seen in the background.

It is worth noting that our MAS does not manage problems tightly connected with the railway domain. Indeed, it monitors parameters which are common to many processes in many domains, like the use of the cpu and the hard disk, the state of the connection to the network, etc.. The aim of our project was to develop a system able to monitor the execution of a process characterised by the above parameters. As a consequence, the architecture and the MAS developed are general and flexible enough for monitoring many different processes, and not only to the Path Selection one: our system could be easily adapted

Figure 1. Railway tracks controlled by Genoa's SCC.

Figure 2. Operator and synoptic frame in Genoa's SCC.

to monitor new processes without changing the architecture of the MAS but just creating specific reader agents and equipping the other agents with new rules.

B. MAS architecture

Our MAS consists of the four kinds of agent depicted in Figure 3.

Agents are organized in a hierarchy: Log Reader Agents are at the bottom of the hierarchy and interact with Process Monitoring Agents, which in turn interact with Computer Monitoring Agents. At the root of the hierarchy is the Plant Monitoring Agent, unique in each SCC. Agents live and act in the software Environment consisting of the already existing processes developed by Ansaldo, and interact with it in the limited way discussed below.

- **Log Reader Agent**. In our MAS, there is one Log Reader Agent (LRA) for each process that needs to be monitored. Thus, there may be many LRAs running on the same computer (if there are more processes to monitor; at the time of writing, only Path Selection is considered). Once every m minutes the LRA reads the log-file produced by the process P it monitors, extracts information from it, produces a symbolic representation of the extracted information in a format amenable of logic-based reasoning, and sends the symbolic representation to the Process Monitoring Agent in charge of monitoring P. Relevant information to be sent to the Process Monitoring Agent includes loss of connection to the net and life of the process. LRA is the only agent able to get information from the Environment where the MAS is situated.

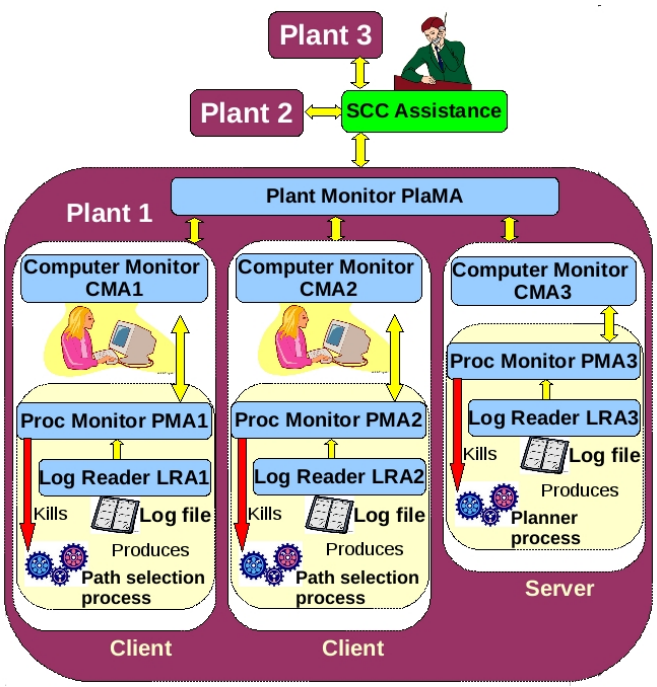


Figure 3. MAS architecture, from [17].

- **Process Monitoring Agent**. Process Monitoring Agents (PMAs) are in a one-to-one correspondence with LRAs: the PMA associated with process P receives the information sent by the LRA associated with P, looks for anomalies in the functioning of P, reports them to the Computer Monitoring Agent (CMA) and asks it for more information, and in case kills and restarts P if necessary. It implements a sort of social, context-aware, reactive and proactive expert system. PMA can interact with the Environment by killing and restarting the process it monitors.
- **Computer Monitoring Agent**. The CMA receives all the messages arriving from the PMAs that run on that computer, and monitors parameters like network availability, CPU usage, memory usage, hard disk usage. The messages received from PMAs together with the values of the monitored parameters allow the CMA to make hypotheses on the functioning of the computer where it is running. If necessary, the CMA may ask the PlaMA for more information, to know about the state of the entire plant and to act consequently.
- **Plant Monitoring Agent.** There is one Plant Monitoring Agent (PlaMA) for each plant. The PlaMA receives messages from all the CMAs in the plant and in case alerts the SCC Assistance Centre. It interacts with the Environment by alerting the remote assistance centre.

III. IMPLEMENTATION

All the agents of the MAS, apart from LRA that is a pure Jade [6] agent, have been implemented in TuProlog [9] integrated into Jade by means of an extended version of DCaseLP libraries [18]. The extension consists in making a *blocking_selective_receive* predicate available to the agents, which takes three arguments: *Performative*, *Content*, *Sender*. It was motivated by the need to allow our agents to retrieve only messages respecting a given pattern (in particular, messages arriving from a given *Sender*) from their message queue. Jade offers the *MessageTemplate* class that provides static methods to create filters for each attribute of the *ACLMessage*. The Jade *blockingReceive* method can accept a message template as argument, and retrieve only those messages that match the template. The DCaseLP *blocking_selective_receive* predicate creates a template that filters on the name of the *Sender*, and then calls the Jade *blockingReceive(mt)* to return the value for *Performative* and *Content*.

LRAs have been designed and developed as agents for clearly separating what has been developed as part of this project ("agents") from what already existed ("non agents"). We also wanted to emphasise their autonomy (although very limited) and to separate the functionality of parsing the log-file from the one of reasoning over facts. However, LRAs are very trivial agents and we could have designed and implemented them as "Artifacts" in the A&A metamodel [23] or as "Touchpoints" in the Autonomic computing terminology [2] as well.

The CMA, PMA and PlaMA have a cyclic "observe-think-act" behaviour [14] (and a "cyclic behaviour" in Jade) where they

- look if a new message matching a given template has been received;
- retrieve the message from their message queue and store it in their history;
- manage the message according to the rules in their program, and to their knowledge base (that includes all the messages received in the past);
- answer to the agent that has sent the message, and, in case, send messages to other agents in the MAS.

The architecture of each agent, apart LRA ones, is a declarative architecture where the knowledge base is modeled as a set of Prolog facts, the behavior is determined by Prolog rules, reactivity is implemented by allowing agents to look at their message box and to react to incoming messages. Messages arrive from the LRA to the PMA every m seconds (where m is a configuration parameter of the MAS), and the PMA looks for anomalies and starts the managing process if necessary.

Agents are equipped with different rules dealing with the different parameters to be monitored, namely:

1) parameters tightly connected to the process monitored by the PMA; these parameters include "cpu_usage" and "errors" and are not influenced by the state of the network or by other processes;
2) parameters influenced either by the state of the network, or by the behaviour of other processes as those running on the server (for example, "connection_to_server" and "view").

Parameters of the first type are treated locally by the PMA. Parameters of the second type are dealt with by PMA asking the CMA, which can ask the PlaMA, for more information, since they may involve non-local problems.

An example of message sent by the LRA to the PMA is: `log(time(''Mon Feb 11 21:30:43 CET 2008''), [view(normal), cpu_usage(normal), connection_to_server(active), disk_usage(normal), answer_to_life(slow), errors(absent), memory_usage(normal)])`, whose meaning is easy to understand.

Currently the agents do not use a common ontology, that is implicitly known as the set of the monitored parameters and their possible values.

The state of an agent consists of a set of facts representing what happened in the past. Different agents store different facts: PMAs store information about what local problems have been found and when (facts reporting a timestamp and what the problem is), CMAs keep information about the problems of all its PMAs and the notifications of a process killing (facts reporting the name of the process, a timestamp and what the problem is and facts reporting why and when a process have been killed), whereas PlaMA records facts about problems in the network (facts reporting the name of the machine and the process, a timestamp and what the problem is), but nothing about the solutions that have been taken (because they are local solutions). Messages received in previous interactions are also stored by agents in their knowledge bases, since agents may act in different ways if some problem is reported for the first time or if the problem is common to other agents that recently reported it.

This structure allows us to leave the rules that establish how to manage a problem (kill o not, according to the CMA advice) in the PMA, to store the intelligence to monitor a computer and decide when more information is needed in the CMA, and to have the PlaMA look over the whole network and answer CMAs' requests, but without intruding in the local management.

In the sequel we show some schemes of interaction protocols among agents aimed at managing some parameters. The translation from these schemes into Prolog code has been done creating and distributing rules among the agents involved in the interaction. We did not use any standard interaction protocol: indeed, we designed the needed interaction protocols on an application-driven basis.

For example, in order to manage the "connection_to_server" parameter, the MAS acts in the following way:

1) If the value of the "connection_to_server" parameter received by the PMA from the LRA is "active", no action has to be taken; instead
2) if the value received by the PMA from the LRA is "lost", the PMA asks the CMA to know if this problem is

common to other processes or not.

a) If the CMA has no recent information[2] about this problem in its history, it notifies the PMA that the problem is not a "net problem" (that is, it is not common to other processes); in this case, the PMA kills and restarts the process, and informs CMA of this.
b) If the CMA has other (one or more) recent notifications of the problem, before answering to the PMA, it asks the PlaMA to know if the problem is local to the machine where the CMA runs, or has been reported also on other computers.
   i) If the PlaMA received no notifications of this problem from other CMAs recently, it answers that the problem is not common to the MAS; in this case, the CMA answers the PMA that there are no net problems, and the PMA kills and restarts the process and informs CMA of this. Otherwise,
   ii) if the PlaMA already received notifications of the same problem in the last M minutes, it answers the CMA that there are network problems; the CMA forwards this answer to the PMA, which, in this case, does not kill the process.

There are also situations where the PMA waits for two (or more) consecutive messages from the LRA notifying the same problem, before reporting it to the CMA. This situation is shown below, for the parameter "answer_to_life":

1) If the value of the "answer_to_life" parameter received by the PMA from the LRA is "ready", no action has to be taken.
2) If the value received by the PMA from the LRA is "absent", the PMA kills and restarts the process (and informs CMA).
3) If the value received by the PMA is "slow", the PMA must wait for the successive message arriving from the LRA. If the successive message reports again a "slow" answer to life, then the PMA must ask the CMA to know if this problem is common to other processes or not.
   a) If the CMA received no recent notifications of this problem, it notifies the PMA that the problem is not a "net problem" (that is, it is not common to other processes). In this case, the PMA waits for two more messages from the LRA and, if the problem persists, kills and restarts the process and notifies it to CMA.
   b) If the CMA received recent notifications of the same problem, it asks the PlaMA if the problem is local to the machine or had also been reported on other computers.
      i) If the PlaMA answers that the problem is not common to the MAS, since no notifications of the same problem have been reported in the last M minutes, the CMA sends a "no net problem". The PMA waits for two next messages and kills and restarts the process (and notifies CMA), if the problem persists.
      ii) If the PlaMA was recently notified of the same problem, it answers the CMA that there are network problems; this answer is forwarded by the CMA to the PMA, that does not kill the process.

The hierarchical structure of the system ensures scalability: there will always be only one PlaMA in the MAS, but many CMAs may be connected to it, and many PMAs can be started on a machine and controlled by the local CMA. In case other PMAs should be developed in the future, with rules ad hoc for different processes, only the new PMAs and their associated LRAs should be developed from scratch, with no impact on the entire system.

IV. Running the system

In order to run the developed system, Jade and tuProlog (version 1.3, in order to be compliant with the DCaseLP libraries) need to be installed on the machine, as well as the extended DCaseLP libraries. The simplest configuration of the MAS includes

- one PlaMA
- one CMA
- one PMA

but usually the MAS will consist of at least two CMAs controlling different PMAs. At this stage of the project we use more PMAs of the same type, which is not a problem because the rationale is to simulate the behaviour of the CMA with more processes, regardless of their type. The PlaMA is one for each MAS. In the sequel we show the behaviour of the MAS concerning the management of different parameters, and with different configurations and history. Some figures will not show the LRA to let the reader better understand the interactions among the other agents.

The first example shows the behaviour of the MAS when the value "high" of the "cpu_usage" is reported by the LRA to the PMA, with the simplest MAS configuration consisting of just one agent of any kind.

When the PMA receives a message from the LRA:

1) If the value of the "cpu_usage" parameter is "normal", no action needs to be taken.
2) If the value of the parameter is "high", and it remains high in the successive message sent by the LRA, the PMA kills and restarts the process, and informs the CMA.

The simplest MAS configuration works well enough to demonstrate this behaviour, because it does not depend on how many PMAs encountered the same problem. As shown in Figure 4, the first message notifying a high cpu usage from the LRA does not cause the delivery of message from the PMA. The second message with the same content, instead, causes the PMA to send a message to the CMA, with the content "process killed".

Figure 5 depicts the behaviour of the PMA with respect to the "answer_to_life" parameter. In this configuration we have only one CMA. The PMA will wait for two messages

[2]By recent information, we mean information stored no later than M minutes ago, where M is a parameter that can be set by the person in charge of configuring the MAS.

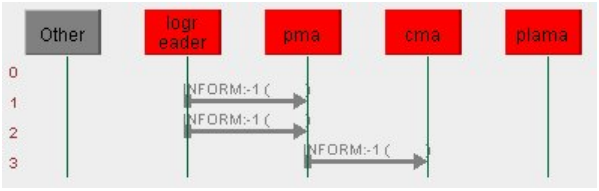


Figure 4. Execution run concerning the “cpu_usage” parameter.

from the LRA with the value “slow” for the parameter, then it contacts the CMA for further information: CMA received no recent information from other PMAs, so PMA kills (after two messages from LRA with the same problem) the monitored process and informs of this the CMA, as described in the previous section.

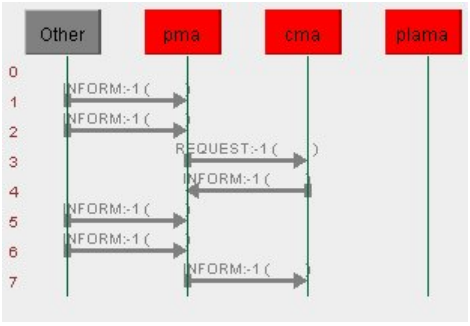


Figure 5. Execution run concerning the “answer_to_life” parameter.

The third example shows the behaviour of the system for the management of the “connection_to_server” parameter. The behaviour has been illustrated in Section III and is more complex than the one dealing with the “cpu_usage”. To allow a good understanding of how it works, we will use two different configurations and histories.

The first configuration, shown in Figure 6, involves one PlaMA, one CMA and two PMAs, named Pma1 and Pma2. Pma1 receives a message from its LRA with “connection_to_server(lost)”: Pma1 asks for more information to CMA, that has no recent notifications of this problem from other PMAs, and answers “no_network_problem” to Pma1. Pma1 kills and restarts the process and informs CMA of this. Later, also Pma2 receives the same message from its LRA, and, in the same way as Pma1, asks to CMA if the same problem has already been reported. CMA, which had registered the problem of Pma1 in its history, needs to verify if this is a local problem or a problem involving the entire network. Thus, it asks the PlaMA if it is aware of other CMAs with the same problem. For the PlaMA, this is the first notification of the problem so it registers it into its history and answers “no_network_problem”. The CMA forwards the message to Pma2 which kills and restarts the process, and informs CMA of it.

If we make the configuration even more complex (Figure 7), the behaviour of the MAS changes. We add another CMA named Cma2, controlling two PMAs (Pma3 and Pma4). The

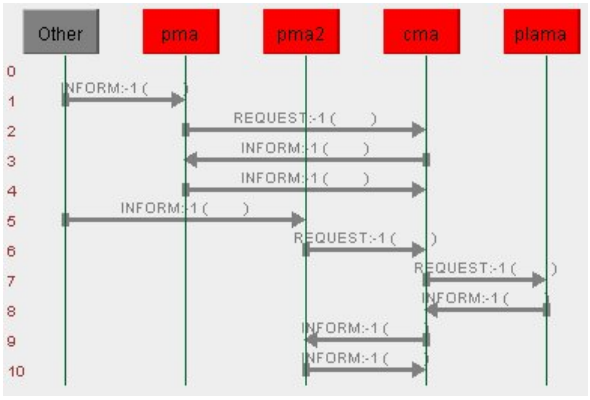


Figure 6. Execution run concerning the “connection_to_server” parameter.

agents shown in Figure 6 are still alive and their history includes the events discussed before. If Pma3 receives the notification of the “connection_to_server(lost)” problem, it reacts exactly as Pma1, and Cma2 acts as Cma1. That is, Cma2 answers to Pma3, without asking the PlaMA, that there are no network problems. But if also Pma4 receives the “connection_to_server(lost)” message from its LRA, then Cma2 must ask the PlaMA if there are network problems. The PlaMA’s history contains the fact that Cma1 reported the same problem a short while ago, so PlaMA sends a message with content “network_problem” to Cma2. This answer is propagated to Pma4 by Cma2, and, as a consequence, Pma4 does not kill the process because the problem cannot be managed locally.

V. Related Work and Conclusions

The exploitation of intelligent agents for monitoring and diagnosing distributed processes has a long and successful history dating back to the early and mid nineties. Before that, Distributed Artificial Intelligence (DAI) techniques were adopted. Even if the first DAI systems did not integrate “agents” as we intend them today, they were the ancestors of MASs and deserve to be shortly mentioned in this section.

In 1990, the “Large-internetwork Observation and Diagnosis Expert System”, LODES [29], was implemented. It represents an interesting example of application of DAI to diagnosis. The diagnostic system was created by reusing and unifying pre-existing network diagnosis expert systems. Each sub-LAN had its own LODES system, and problems were solved by their co-operative work. In the same year, Weihmayer and Brandau developed TEAM-CPS [30], a test bed for introducing DAI to control and manage customer networks: in TEAM-CPS the customers’ virtual private networks were automatically reconfigured using links from the public network. In 1992, the “Distributed Big Brother” was one of the earliest works where DAI was adopted for monitoring purposes in the telecommunications area [28]. The project applied DAI techniques to Local-Area Networks, to make their management more robust and faster.

Among the oldest applications of rule-based intelligent

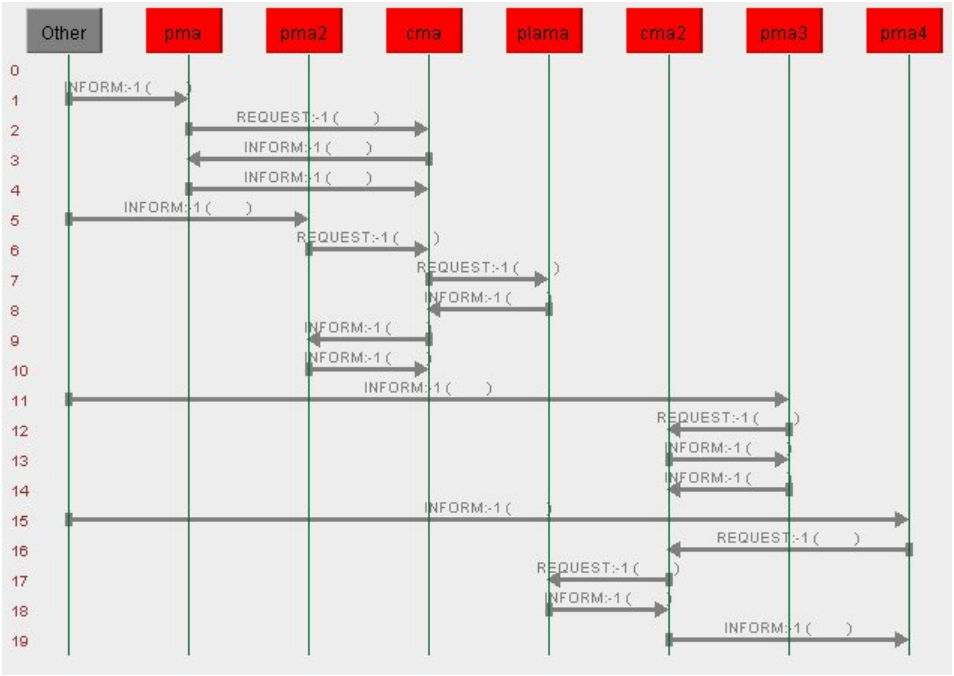


Figure 7. Execution run concerning the "connection_to_server" parameter, complex configuration.

agents in the monitoring and diagnosis domain, besides those already cited in Section I, we may mention a re-implementation of TEAM-CPS [31] where agents used the PRODIGY planning system [20] for local network planning, and the well-known Agent-Orientated Programming framework [27] for communication and control. In 1997, Leckie et al. [15] developed a prototype agent-based system for performance monitoring and fault diagnosis in a telecommunications network, where agents were implemented using C5 [24], based on the OPS5 rule language [11], and communicated using KQML [10].

An architecture for a software agent operating a physical device and capable of testing and repairing the device's components is described in [3]. In that work, the authors focus on modelling the agent's behaviour after the discovery of a fault in a circuit: the knowledge as well the behaviours of the agent are expressed in A-Prolog [4]. The life of the agent is an "observe-think-act" loop where actions are quite simple, but nevertheless able to modify the circuit in order to repair it. An industrial application of A-Prolog to a medium size knowledge-intensive application for controlling some functions of the Space Shuttle is described in [21]. However, no agents are used there.

Moving to nowadays, [26] describes Space Shuttle Ground Processing with Monitoring Agents. JESS is used to realize a system that helps the monitoring of all the processes, instrumentation and data flows of the Kennedy Space Center's Launch Processing System. The system, called NESTA, helps to monitor and above all to discover problems concerning the "ground process", i.e. the set of the operations carried out in the weeks before the Space Shuttle's launch. NESTA autonomously and continuously monitors shuttle telemetry data and automatically alerts NASA shuttle engineers if it discovers predefined situations. This system, developed and tested in a real, safety-critical scenario, shows that an agent-oriented solution implemented with a rule-based language may be employed to satisfy concrete industrial needs, and demonstrates the success of agents outside the boundaries of academia.

Other applications of agents for diagnosis and monitoring do not rely on a rule-based approach. For example, the paper [16] presents a technique for monitoring the start up sequences of gas turbine: the system uses a MAS where decisions are taken by combining partial information possessed by individual agents, thus obtaining a global view of the situation, and producing an automatic fault diagnosis for the engineers. The MAS is implemented with the ZEUS Agent Building Toolkit [22]. In 2006, the Rockwell Automation company applied agents to control manufacturing production [19]. The MAS is implemented with real-time control agents, and also the information transfer among the software agents takes place in real-time, using a Programmable Logic Controller. A MAS for the simulation of the environment for material handling systems has been implemented in Jade. Finally, [7] describes a model for managing faults in industrial processes. The model is based on a generic framework that uses MASs for distributed control systems; the system manages faults with feedback control process and decides about the scheduling of the preventive maintenance tasks, also running preventive and corrective specific maintenance tasks.

Our project, although similar in its purposes to other applications developed in the past, demonstrates an increased industrial interest and trust in both agent-based and rule-

based technologies. To the best of our understanding only few proposals of using rule-based agents led to the development of a MAS prototype used inside an industry ([12], [26]). The industrial strength system described in [19], despite not using rule-based technologies, shares with our project the choice of Jade as the agent middleware.

In 2004, the *Agent Technology Roadmap: Overview and Consultation Report* observed that "One of the most fundamental obstacles to the take-up of agent technology is the lack of mature software development methodologies for agent-based systems.". According to the experience of DISI and Ansaldo, agent tools, languages (rule-based ones in particular), and methodologies are today mature enough to be adopted by the industry. Although the competencies on *how* to exploit them are still missing in many companies, companies now know that agents exist, believe in their usefulness for coping with the complexity of open, distributed, dynamic applications, and are more and more keen on integrating them into their projects. The role of academia in providing a good support during the design and implementation of MASs for real applications is a key factor in the take-off of the agent technology, and the joint DISI-Ansaldo project discussed in this paper represents a success story in this direction.

ACKNOWLEDGMENTS

The authors acknowledge the "Iniziativa Software" CINI-FINMECCANICA project that partially funded this work.

REFERENCES

[1] E. Abel, I. Laresgoiti, J. Perez J., Corera, and J. Echavarri. A multi-agent approach to analyse disturbances in electrical networks. In *International Conference on Expert Systems Applications to Power Systems, ESAP'93, Proceedings*, 1993.

[2] B. A. Miller. http://www.ibm.com/developerworks/autonomic/library/ac-edge5/, 2005.

[3] M. Balduccini and M. Gelfond. Diagnostic reasoning with a-prolog. *Theory Pract. Log. Program.*, 3(4):425–461, 2003.

[4] M. Balduccini, M. Gelfond, and M. Nogueira. A-prolog as a tool for declarative programming. In *12th International Conference on Software Engineering and Knowledge Engineering, SEKE'00, Proceedings*, pages 63–72. Knowledge Systems Institute, 2000.

[5] J. Barandiaran, I. Laresgoiti, J. Perez, J. Corera, and J. Echavarri. Diagnosing faults in electrical networks. In S. Hashemi, J.P. Marciano, and J.G. Gouarderes, editors, *International Conference on Expert Systems Applications, EXPERSYS'91, Proceedings*. IITT Paris, 1991.

[6] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

[7] M. Cerrada, J. Cardillo, J. Aguilar, and R. Faneite. Agents-based design for fault management systems in industrial processes. *Computers in Industry*, 58(4):313–328, 2007.

[8] J. M. Corera, I. Laresgoiti, and N. R. Jennings. Using Archon, part 2: Electricity transportation management. *IEEE Expert*, 11(6):71–79, 1996.

[9] E. Denti, A. Omicini, and A. Ricci. tuProlog: A lightweight prolog for internet applications and infrastructures. In I. V. Ramakrishnan, editor, *3rd International Symposium on Practical Aspects of Declarative Languages, PADL'01, Proceedings*, pages 184–198. Springer, 2001.

[10] T. W. Finin, R. Fritzson, D. P. McKay, and R. McEntire. KQML as an agent communication language. In *3rd International Conference on Information and Knowledge Management, CIKM'94, Proceedings*, pages 456–463. ACM, 1994.

[11] C.L. Forgy. Ops5 user's manual. Technical Report CMU-CS-81-135, Carnegie-Mellon University, 1981.

[12] N. R. Jennings, E. H. Mamdani, J. M. Corera, I. Laresgoiti, F. Perriollat, P. Skarek, and L. Zsolt Varga. Using Archon to develop real-world DAI applications, part 1. *IEEE Expert*, 11(6):64–70, 1996.

[13] N. R. Jennings, K. P. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[14] R. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):391–419, 1999.

[15] C. Leckie, R. Senjen, B. Ward, and M. Zhao. Communication and coordination for intelligent fault diagnosis agents. In *8th IFIP/IEEE International Workshop for Distributed Systems Operations and Management, DSOM'97, Proceedings*, pages 280–291, 1997.

[16] E.E. Mangina, S.D.J McArthur, J.R. Mc Donald, and A. Moyes. A multi agent system for monitoring industrial gas turbine start-up sequences. *IEEE Transactions on Power Systems*, 16(3):396–401, 2001.

[17] V. Mascardi, D. Briola, M. Martelli, R. Caccia, and C. Milani. Monitoring and diagnosing railway signalling with logic-based distributed agents. In E. Corchado and R. Zunino, editors, *International Workshop on Computational Intelligence in Security for Information Systems, CISIS'08, Proceedings*, Advances in Soft Computing Series. Springer-Verlag, 2008.

[18] V. Mascardi, M. Martelli, and I. Gungui. DCaseLP: a prototyping environment for multi-language agent systems. In M. Dastani, A. El-Fallah Seghrouchni, J. Leite, and P. Torroni, editors, *In Proceedings of the First Workshop on LAnguages, methodologies and Development tools for multi-agent systemS, LADS'007 Post-proceedings*, volume 5118 of *LNCS*, pages 139–155. Springer-Verlag, 2008.

[19] V. Mařík, P. Vrba, K. H. Hall, and F. P. Maturana. Rockwell automation agents for manufacturing. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *4rd International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'05, Proceedings*, pages 107–113. ACM, 2005.

[20] S. Minton, C. A. Knoblock, D. R. Kuokka, Y. Gil, R. L. Joseph, and J. G. Carbonell. Prodigy 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, Carnegie-Mellon University, 1989.

[21] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An a-prolog decision support system for the space shuttle. In A. Provetti and T. Cao Son, editors, *1st International Workshop on Answer Set Programming, Towards Efficient and Scalable Knowledge Representation and Reasoning, ASP'01, Proceedings*, 2001.

[22] H. Nwana, D. Ndumu, L. Lee, and J. Collis. ZEUS: A tool-kit for building distributed multi-agent systems. *Applied Artifical Intelligence Journal*, 13(1):129–186, 1999.

[23] A. Ricci, M.o Viroli, and A. Omicini. The A&A programming model and technology for developing agent environments in MAS. In M. Dastani, A. El Fallah-Seghrouchni, A. Ricci, and M. Winikoff, editors, *Programming Multi-Agent Systems, 5th International Workshop, ProMAS 2007*, volume 4908 of *LNCS*. Springer, 2008.

[24] J.R. Roland, G.T. Vesonder, and J.M. Wilson. C5 user manual, release 2.1. Technical report, AT&T Bell Laboratories, 1990.

[25] M. Schroeder, I. de Almeida Móra, and L. Moniz Pereira. A deliberative and reactive diagnosis agent based on logic programming. In M. P. Singh, A. S. Rao, and M. Wooldridge, editors, *4th International Workshop on Agent Theories, Architectures, and Languages, ATAL'97, Proceedings*, volume 1365 of *LNCS*, pages 293–307. Springer, 1998.

[26] G. S. Semmel, S. R. Davis, K. W. Leucht, D. A. Rowe, K. E. Smith, and L. Boloni. Space shuttle ground processing with monitoring agents. *IEEE Intelligent Systems*, 21(1):68–73, 2006.

[27] Y. Shoham. Agent-orientated programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[28] Y. So and E. H. Durfee. A distributed problem-solving infrastructure for computer network management. *Int. J. Cooperative Inf. Syst.*, 2(2):363–392, 1992.

[29] T. Sugawara. A cooperative lan diagnostic and observation expert system. In *9th Annual International Phoenix Conference on Computers and Communications, PCCC'94, Proceedings*, pages 667–674. IEEE, 1990.

[30] R. Weihmayer and R. Brandau. A distributed ai architecture for customer network control. In *IEEE Global Telecommunications Conference, Globecom'90, Proceedings*, pages 656–662. IEEE, 1990.

[31] T. Weihmayer and M. Tan. Modeling cooperative agents for customer network control using planning and agent-oriented programming. In *IEEE Global Telecommunications Conference, Globecom'92, Proceedings*, pages 537–543. IEEE, 1992.

AgentService in a hand[1]

A. Passadore, A. Grosso, M. Coccoli, A. Boccalatte, *Università degli Studi di Genova, Dipartimento di Informatica, Sistemistica e Telematica, Via all'Opera Pia 13, 16145 Genova.*

Abstract—In this paper we present AgentService Mobile: an infrastructure aimed to the execution of agents on devices with limited resources. The mobile device plays the role of a client which consumes a set of services exposed by the AgentService platform. This is the entry of AgentService in a SOA context, with the main goal to open the multi-agent platform to the outside, by using the most recent service oriented technologies.

Keywords: mobile devices, SOA, Web Services, multi-agent systems.

I. Introduction

AGENTS are often in the common imaginary a sort of digital *alter-ego* of the human owners. They play the role of secretaries, avatars, and every other responsibility that can be delegated to a software entity running on a computer. If in the 70's and 80's a computer was a heavy box that rarely went out of offices and labs, since 90's the advent of notebooks allowed users to use their software more or less everywhere.

A new frontier of mobility has been reached with wireless networks that cut off the cables from portable computers. At now the mobility has reached incredible levels with concentrates of computer technology in the palm of a hand as PDAs (Personal Digital Assistants) and cellular phones are.

By using mobile devices we can easily access web pages and interact with remote services, from basilar mail servers to a plethora of web services that remotely expose functions and data. If we hold our all-day life and work activity in a hand, the possibility to host software agents, so close to us, is a real opportunity of releasing them from the closed environment of a multi-agent platform running on a motionless computer.

It is then our goal to expand the range of the agents that are in execution on the platform we are developing: *AgentService* [1].

For this reason, in this paper we present our on-going project with the aim of executing AgentService agents on devices having limited resources, in terms of CPU, RAM and display capabilities.

From a general point of view, this project introduces AgentService in a *Service Oriented Architecture* (SOA) context [2], where the platform exposes its services to the outside, in order to allow remote applications and users to exploit them. Therefore, by using a SOA interface, we do not open AgentService only to C# based agents on mobile devices, but also to whatever software entity able to manage web services. As we will see in section IV, the only other existing solution is based on an internal protocol which enables just compatible agents to contact the remote platform. Our contribution is then aimed to equip the .NET development community with a framework for integrating handheld devices in an agent based system, without renouncing to the flexibility and the opening to the outside.

Following the essence of AgentService, the infrastructure of the system in question is based on the latest technologies in the Microsoft .NET field (the framework is compatible with the Microsoft official releases, and the open source versions, as Mono), in particular the contracted version of the Framework .NET 3.5 (namely the *Compact Framework*) [3] and the *Windows Communication Foundation* (WCF) [4]: a communication infrastructure for building and running connected systems that offers a SOA implementation for Windows-based programming.

We consider AgentService as an alternative to the java based multi-agent platforms, taking into account the absence of solutions in the .NET community. AgentService is especially aimed to industrial applications, where the .NET Framework has gained a conspicuous space.

Considering the limited environment of a mobile device, the main issue of this project was the creation of a light infrastructure for executing agents and exploiting the services offered by a remote standard AgentService platform. AgentService is normally based on the Framework .NET 3.5 and exploits functionalities which have been removed in the Compact Framework version. For this reason, the challenge was the reduction and adaptation of the AgentService libraries to a minimal infrastructure, running on the mobile devices only the essential services and masking the others by using a proxy that contacts the main remote platform through web services. The final goal is to execute on a mobile device an agent that is developed for a standard AgentService application, furnishing a context that apparently is the same of a platform running on a heavy motionless computer.

In this paper we first introduce the main problems related to the development of SOA mobile applications, illustrating the state-of-the-art in relation with the Compact Framework, the Windows Communication Foundation and the other existing solution in the field of multi-agent systems.

[1] A special acknowledgement to Matteo Sommariva, for his precise work during his Master Thesis activity.

In the second part of this paper we show the architecture of the proposed system, focusing on the two main aspects: the runtime service proxies and the light execution environment.

Conclusions, impressions, and a brief comparison with the existing solution will follow.

II. MOBILE (AGENT) APPLICATIONS

A. The vivacious field of mobile applications

In the last years the software development for mobile devices has been subject to a sudden growth due to the massive diffusion of handheld computers. There exist different types of portable devices whose classification is now difficult because of the overlap of their features. However we can outline a brief classification of such devices, distinguishing:

- *Smartphones*: cellular phones with memory and computational resources, mainly aimed to communication features.
- *PDA* (Personal Digital Assistant): devices for the management of personal information as agenda, calendar, block-notes, etc.
- *Tablet PC*: denoting a touch-screen display, it is mainly used in industrial field, but also for gaming and multi-media player.

Smartphones and PDAs are often very similar and easily indiscernible; this fact proves that the trend is to converge to a multi-purpose mobile device with a heterogeneous set of features.

The development of multi-agent mobile applications cannot leave aside the hosting operating system. In the field of handheld devices, there are heterogeneous solutions that make this panorama more variegated than the desktop computers one.

The market is dominated by *Symbian OS* (65% of the market, in the fourth period of 2007), which outclasses *Windows Mobile* (12%), and *Blackberry OS* (11%). Other noteworthy solutions are based on *iPhone OS*, *Palm OS*, *JavaFX*, *OpenMoko Linux*, and the incoming *Google Android*. Some of them are open to third-part applications and provide a SDK for the development in proprietary languages, Java, C++, or C#. Several operating systems support Java libraries (as Blackberry, Android, JavaFx, OpenMoko, and Windows Mobile) and MIDP (Mobile Information Device Profile): a specification for the use of Java on mobile devices; it is part of J2ME: the micro edition of the Java Framework. The Microsoft's framework for handheld devices is the Compact Framework .NET 3.5 which at the present moment is supported only by the Windows Mobile family[2].

The choice of the Windows Mobile OS and the Compact Framework for our project is bind to the fact that AgentService is completely based on the Common Language Infrastructure (CLI) specification [5]. Although the most supported platform is Java, the .NET counter-party denotes features that are in step with the former one[3].

B. SOA Middleware

Mobile devices well represent the role of clients which exploit and consume services hosted on machines with more computational resources. In this sense, the power of a collaborative network involving mobile clients is just the possibility to access to a theoretically infinite set of services from a device with a small CPU and few RAM.

The SOA philosophy embodies the approach we want to apply to AgentService Mobile.

SOA [2] [6] is essentially a sort of style, paradigm, or concept aimed to support services on the web, in order to satisfy the user's requests and consider the single applications as coordinated components of a business process. SOA is based on few strong principles that are strongly relevant in agent based system too:

1) *loose coupling:* with a low intensity of connections among the different components, it is possible to easily update or modify a service without upset the rest of the system. This feature warrants the system scalability and decentralization.
2) *Heterogeneousness:* the different components of a system can be based on different platform and different implementation.
3) High interoperability: an easy communication among the components of the system is a basis of each implementation of both SOA and agent based systems.

The constitutional elements of a SOA are the *services*. A service can be described by three aspects: the *interface* that furnishes the signature of the exposed methods; the *contract*: a formal representation of the interface (expressed, for example, in WSDL); the *implementation* of the service, in strict accordance with the contract. In this sense, an AgentService platform will expose contracts for the usual FIPA services, allowing our mobile agents or other external applications (eventually based on different platforms) to consume them.

SOA is an abstract concept that must be implemented. At now, there exist different solutions which can be classified, considering the way followed by users in order to invoke services [7]: *remote procedure calls* (RPC) or *messages*. Regarding RPC, the client obtains the service interface and then contacts the service method by a (generally synchronous) parametric call. Example of RPC technologies that could be used to implement SOA are CORBA [8], Java RMI [9], and .NET Remoting [10].

[2] Even if there exists an open source version of the Framework named Mono, there is not yet an open source version of the Compact Framework. The developers of Mono ensures the compatibility only for those applications which have no GUI.

[3] There are several comparisons between J2ME and Compact Framework, but almost all are evidently prejudiced. For a equilibrated comparison see: http://www.must.edu.my/~dwong/resources/mobile_commerce_web/j2mevsnetcf.html

The second type is based on the exchange of few well-defined (generally asynchronous) messages. The frameworks based on messages are named *Message-Oriented Middleware* (MOM) and provides a complete management for message queuing, persistence, and security. MOMs support different platforms, protocols, standards, and languages. Because of their versatility and interoperability, they are closer to the SOA concepts. Several big enterprises invest in MOMs and deliver powerful frameworks: *IBM Websphere MQ*, *Sun Java Message Service*, *Microsoft Message Queue Server*, *BEA System Inc* (now Oracle) *MessageQ*, etc.

A technology which matches the two styles and represents a *de facto* standard for SOA system is the one offered by Web Service standards (WS-*) [2] [11]. A web service is based on SOAP: an XML protocol which supports both RPC and messages. Also a Web Service has a contract which is described by WSDL which, like SOAP, is an XML-based language. The combined use of SOAP and WSDL allows the definition of complex, dynamic, versatile systems, which can be easily considered platform-agnostic. Following this paradigm, AgentService Mobile is designed in order to exploit the powerfulness of Web Services. The implementation of the presented project is based on the new framework, introduced by Microsoft, for the development of connected services, named Windows Communication Foundation.

C. Exploiting Windows Communication Foundation in AgentService Mobile

With the release of the Framework 3.0, Windows applications can be based on four sub-systems that manage the different aspects of a software project: *Windows Presentation Foundation* (the graphical layout), *CardSpace* (management of digital identities), *Windows Workflow Foundation* (a complete API for workflow management), and *Windows Communication Foundation*.

The aim of WCF is to provide an environment for the development of distributed applications in Windows-based systems. It is a tool for implementing and hosting services and it supports several industrial standards that define interactions among services, type conversions, marshalling, and protocol management.

A classification of the WCF features [3], which demonstrates its adherence to SOA principles, is reported in the following points:

1) *Independent versioning*: adhering to the WS-* standards, WCF services can be developed with different timetables in respect with consumers.
2) *Asynchronous one-way messaging*: it allows an optimal use of the disposable computational resources.
3) *Platform consolidation*: all the different Microsoft communication technologies (RPC, ASMX, Remoting, COM+, and MSMQ) are now unified under WCF that collects all the features of the single solutions.
4) *Security*: WCF supports several security models.
5) *Reliability*: it warrants the safe delivery of messages, dividing the transport protocol from the delivery mechanism. This approach ensures a safe communication also on untrustworthy channels.
6) *Transaction support for atomic operations*.
7) *Interoperability*: WCF can interact with every single past Microsoft communication technology and other solutions that implement WS-* or REST, as Java.
8) *Performance*: different levels of interoperability and performance are supported.
9) *Extensibility*: every aspect of the platform can be customized, according to the application specification: channels, bindings, codings, transports, etc.
10) *Configurability*, with the support of XML configuration files.

Security and *reliability* are interesting features for the AgentService Mobile architecture: secure transactions among agents, over a reliable channel, are essential needs for trustworthy multi-agent applications.

Interoperability does not close the AgentService platform to contribute of external applications, eventually based on different platforms so, a Java version of mobile device-based agents could be developed in these languages and then, freely interact with the AgentService .NET platform (the Sun's Java project named *Tango* ensures a comfortable interoperability with WCF).

Even if WCF represents a good communication infrastructure, it denotes aspects that can be ameliorated. In the next sections we will show the architecture of the developed system, pointing out every difficulty that the WCF has risen. Especially in conjunction with the Compact Framework, from the client point of view, some issues have complicated the development of the mobile device architecture.

In general the combination of WCF and Compact Framework is not yet sufficiently tested and in the developers' community the experience about them is not so deep. For these reasons, the practice ripened during this project has been significant and appreciable.

D. The Compact Framework 3.5

A brief introduction of the Compact Framework is necessary to understand the technical choices that have guided the developers to the actual architecture of AgentService Mobile.

With a dramatic reduction to 30% of available classes in respect to the Framework .NET 3.5 and a physical size of 4 MB, the Compact Framework represents a typical bottleneck for development of complex applications, as the AgentService multi-agent platform is.

With the Compact framework, it is possible to develop applications in C# or Visual Basic .NET. A special high-performance, just-in-time compiler is provided with the framework.

Figure 1 shows which features are preserved in the Compact Framework. Server functionalities, ASP.NET, C++ and J# Development, and Remoting are not supported. In particular, Remoting would have been useful to create a communication channel between the mobile client and the

remote platform, simply calling the remote methods of the modules and services of AgentService.

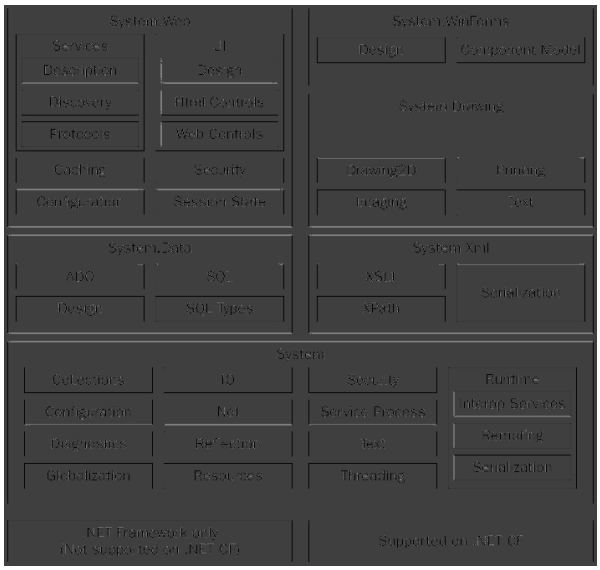


Figure 1: differences between .NET CF and .NET Framework.

Significant restrictions concern the serialization functions (both XML and binary), reflection, and threads. These restrictions have a certain impact during the porting of basic AgentService services from the usual Framework to the Compact one.

Appreciable is the support nearly complete to the Windows Communication Foundation that, in absence of Remoting (and thanks to other considerations that will be clear in the next subsection), is the chosen way to implement a communication channel between mobile agents and the resident platform or agents.

E. Agents on smart devices

In the multi-agent systems research and industry communities, different applications that consider agents on hand-held devices have been suggested. The application domains are quite different: the tourism industry [12], home care service [13], museum guides [14], educational [15], tracking of goods with RFID [16], ambient intelligence [17], industrial maintenance [18], etc.

Most of them exploit the well-known multi-agent framework called JADE [19] along with the LEAP extension [20], especially aimed to the execution of agents on mobile devices. The remaining solutions implement *ad-hoc* architectures essentially based on the Java framework and J2ME. These facts induce two considerations: JADE holds supremacy also in the management of agents on PDAs; if AgentService is one of the very little examples of MAS developed by using the .NET Framework, it is the only one that supports agents on the Compact Framework and then represents a different point of view for mobile agent programming.

Nonetheless, JADE represents once more the state of the art, considering the largeness of the project and the employed resources. For this reason we consider JADE LEAP as a basis for comparison and we report in the next paragraph a short introduction to the architecture.

F. JADE LEAP

LEAP means Lightweight and Extensible Agent Platform. The aim of this international project (involving some big enterprises as Motorola, Siemens, Broadcom, British Telecom, and TILAB) is to reduce the JADE framework in order to execute an agent container in a device with few resources and allow this agent to communicate with other agents running on different containers or platforms. LEAP is executable on every operating system supporting Java, from a powerful server with J2EE (Enterprise Edition), to a smart phone supporting MIDP.

A JADE platform is composed of containers: processes that are in charge to host and execute agents, providing runtime services. These containers must connect to a main container that represents the bootstrap point of the platform and hosts basic services as the *Directory Facilitator* and the *Agent Management System* (AMS). Agents running on different containers or even different platforms can communicate by using the MTP (Message Transport Protocol) furnished by JADE.

LEAP is totally inspired to the JADE architecture, therefore the idea is to execute a LEAP container over the handheld device, maintaining the same API of JADE. The container is connected by a main container residing in a desktop computer. Usually two JADE containers (which could be deployed over a network) communicate by using RMI; due to restrictions of J2ME this is not possible with LEAP, then a new proprietary protocol has been developed: the JICP (JADE inter-container protocol) which makes incompatible JADE and LEAP containers.

An appreciable feature of LEAP is the possibility to split the container into a front-end running on the device and a back-end running on a remote computer. The goal is to lighten the device, hosting almost all the runtime services on the back-end.

III. AGENTSERVICE MOBILE EDITION

A. Overview

Leaving aside a complete introduction of AgentService (for an exhaustive overview read [21]) we concentrate only on those elements that are essentials for the execution of mobile agents.

First, AgentService provides a particular model where an agent is essentially composed of *behaviours* and *knowledges*. Behaviours represent the business activities of an agent and generally are managed as threads executed concurrently. Behaviours of the same agent can share information by using knowledge objects: a sort of knowledge base containing data that can be persisted and that must be accessed concurrently. AgentService Mobile keeps the same model because is a specific requirement the possibility of running standard agents on a mobile device.

From the agent point of view, the AgentService platform is a sort of operating system that exposes services through a

runtime interface, dispatches messages, and schedules the agent behaviours.

The *runtime* interface is available from any agent behaviour. It exposes services useful during the execution time of a behaviour:

- *Yellow pages*: as suggested by FIPA, it is directory for publishing and advertising the services managed by the agent.
- *White pages service*: a sort of telephone directory for retrieving the agent addresses.
- *Console*: a service for monitoring the execution of an agent through text messages.
- *Context*: it returns the behaviour execution context, where it is possible to create and start new behaviours and create new knowledge objects.
- *Logging service* for the monitoring of behaviour execution.
- *Message service*: it is responsible for the forwarding of messages to the message module (and then to the recipient queue). Moreover, it supports the instantiation of conversations, namely preferential communication channels between two behaviours of different agents.
- *Mobility*: an infrastructure for the migration of an agent from a platform to another one.
- *Persistence*: in order to preserve the agent status following up a system crash or failure, this service freezes the agent and saves it in a storage facility (databases, xml file, etc...).

Considering the AgentService Mobile project, the majority of these services should reside on the remote platform. Agents that run on a mobile device access them through a proxy that hides the communication channel with the remote server. Other services, as persistence and logging, must be local, in order to maintain the information regarding agents on the local mobile machine. Finally, the mobility service is unsupported due to the purpose of these types of agent that are intimately tied with the mobile device.

Another fundamental aspect that AgentService Mobile must take into account is the *message dispatching*. Usually, an agent which wants to send a message to a peer, contacts, through the message client provided by the *runtime*, the messaging module which delivers the message to the message queue of the recipient agent. In case of the recipient is running on a federated platform, the messaging module directly contacts the other messaging module through the .NET Remoting. Unluckily, this simple mechanism is not allowed by the Compact Framework, therefore a mobile agent cannot directly interact with the remote central messaging module but must pass through a proxy, as shown in the following.

Incidentally, a recent improvement of the messaging module made it faster by suppressing every polling loop for checking the message queue, in waiting for a new message. Actually, every behaviour-thread that is waiting for a message, is put in *waiting* status and released only when an event, notified by the messaging module, occurred[4]. This enhancement increases the speed up to 350 times. The use of this approach allows also keeping down the resources consuming in the mobile version, where this requirement is fundamental.

Last, the scheduler is in charge of managing the agent behaviours running them as threads. In the standard AgentService version there are different policies to execute behaviours:

- *One behaviour, one thread*: this is the preferred way in term of performance.
- *Every behaviour in a single thread*: preferable in term of resource occupation.
- A *mixed approach* for scenarios where some behaviours have to be executed taking into account the performance and others, the resources.

Due to limitations of the Compact Framework in thread management, we chose to implement the simplest scheduler, namely the first. The tests we made, both on an emulator and on physical devices gave a good response in term of execution speed.

B. Architecture

Figure 2 shows the whole architecture involving the mobile sub-platform and the remote, motionless AgentService system.

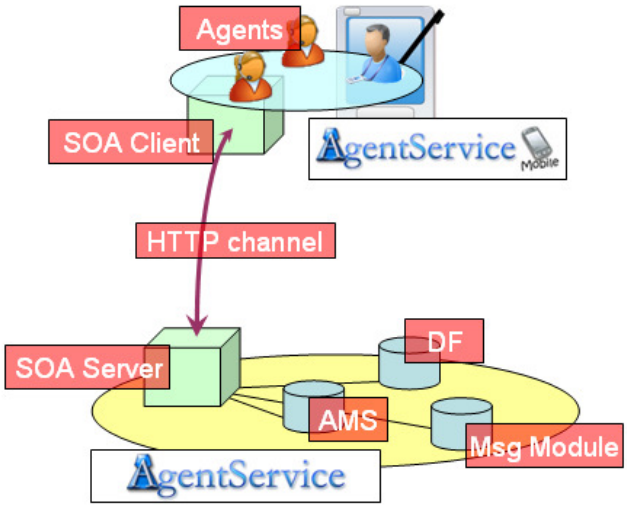


Figure 2: AgentService Mobile architecture.

From the server-side, AgentService deploys a SOA server which receives the requests from mobile agents. It is a sort of bridge between the mobile device and the basic services exposed by AgentService. At the communication level it creates an *http channel* (the only one supported by the Compact Framework) and sends SOAP messages. It

[4] for further information, see *AutoResetEvent* on http://www.developerfusion.co.uk/show/5184/3/

uses a bidirectional message exchange, in order to simulate a remote calling of a method (*invocation* and *return value*).

The AgentService SOA server has been implemented to reserve the channel for a short time, in order to avoid pending calls. For example, the waiting for a message: between the call of *WaitForMessage* method and its return, several minutes could pass; for this reason the SOA server does not wait for the message, leaving the call pending, but it is the mobile client that periodically contacts the server (See **Figure 4**).

Essentially, the SOA Server it is a sort of special runtime that is able to contact the messaging module, the AMS, and the DF, on mobile agents place.

Moreover, it manages the logging credentials for accessing the server from a mobile device. From a configuration file, it can be possible to set up a platform opened to everyone, or a protected one, based on a list of credentials.

Finally, the server periodically controls the connection state of the mobile devices. In case of timeout (customizable in the configuration file), it deregisters the lost mobile agents from the AMS and DF.

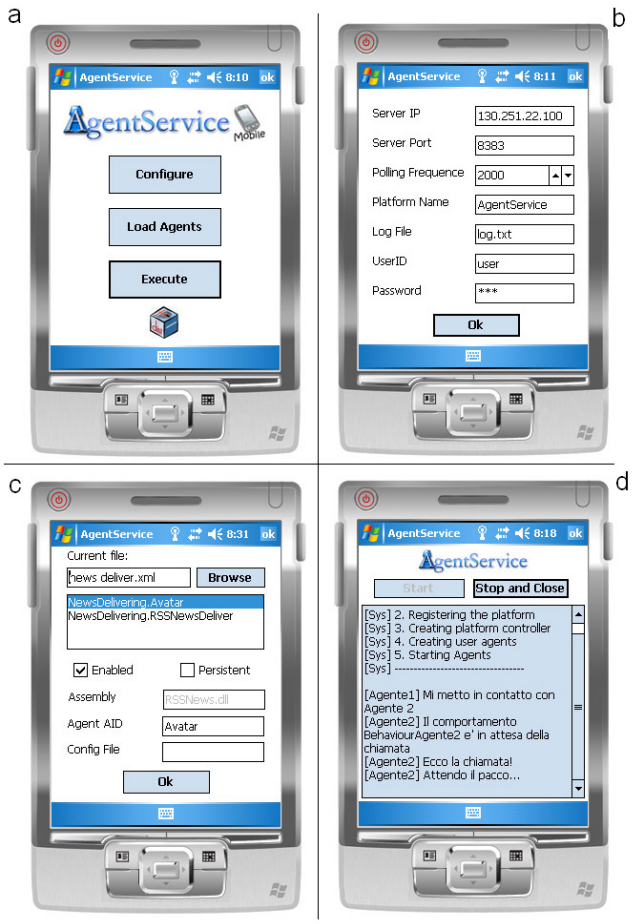


Figure 3: a) the main form of AgentService Mobile; b) general settings; c) batch for agent execution; d) the console.

On the client side, the architecture is implemented by an executable file which instantiates and executes the agents the user suggests in a configuration file or through the GUI.

Screenshots in **Figure 3** show the settings (**Figure 3**.b and **Figure 3**.c) the user must enter in order to connect to a remote platform (settings that can be stored in a configuration file). In particular, he must set the IP address and the port through which the SOA Server is listening and he must enter the username and password that allow him to gain the access to the platform. An interesting setting is the polling time which indicates how frequently the polling thread monitors events coming from the remote platform. These events regard, for example, the notification of a new incoming conversation request, the presence of a new message, etc. This polling loop is the only one running on the mobile platform: it is an important technical solution, because we avoid the saturation of the CPU due to a looping thread for each behaviour. Thanks to this solution, behaviours that are waiting for a message, notify their request to the polling loop manager and then put their thread in *waiting* status. As illustrated in **Figure 4**, when a notification of an incoming message arrives to the polling thread, the behaviour is awoken, the message is downloaded, and the behaviour execution can continue. It is important to mark that from the agent point of view, this complex procedure is hidden behind a simple method call.

C. AgentService Mobile at runtime

The mobile side of the presented solution consists in a *driver* object that first runs the SOA client, contacting the SOA server and presenting the user credentials. The SOA client also runs the aforementioned polling thread and creates a *runtime* object: a sort of proxy for the remote services hosted in the standard AgentService platform. Second, the *driver* executes the *mobile platform controller* which is in charge of creating agent instances, taking the information about the types of agent templates, behaviours and knowledge objects by using the .NET Reflection. The platform controller also binds the runtime object to the agent instances created by the SOA Client, granting a solid link to the services of the remote platform, in a totally transparent manner for the mobile agents. Finally, the platform controller starts the agents, registering them to the remote AMS and activating the behaviour schedulers.

Figure 5 describes in details the bootstrap process, from the start of the driver to the agent execution. It shows the sequences of steps that allow mobile agents to subscribe to the remote platform services. From the UML diagram transpires the most important feature of the *driver* which eases the agent activities furnishing a runtime environment identical to the desktop platform one.

IV. Conclusions

A. Implementing SOA with WCF

In the previous sections we shown the technical solutions adopted to implement AgentService Mobile, applying the SOA model to a multi-agent system infrastructure. WCF offers a particular point of view, largely diffused in the Windows-based programming.

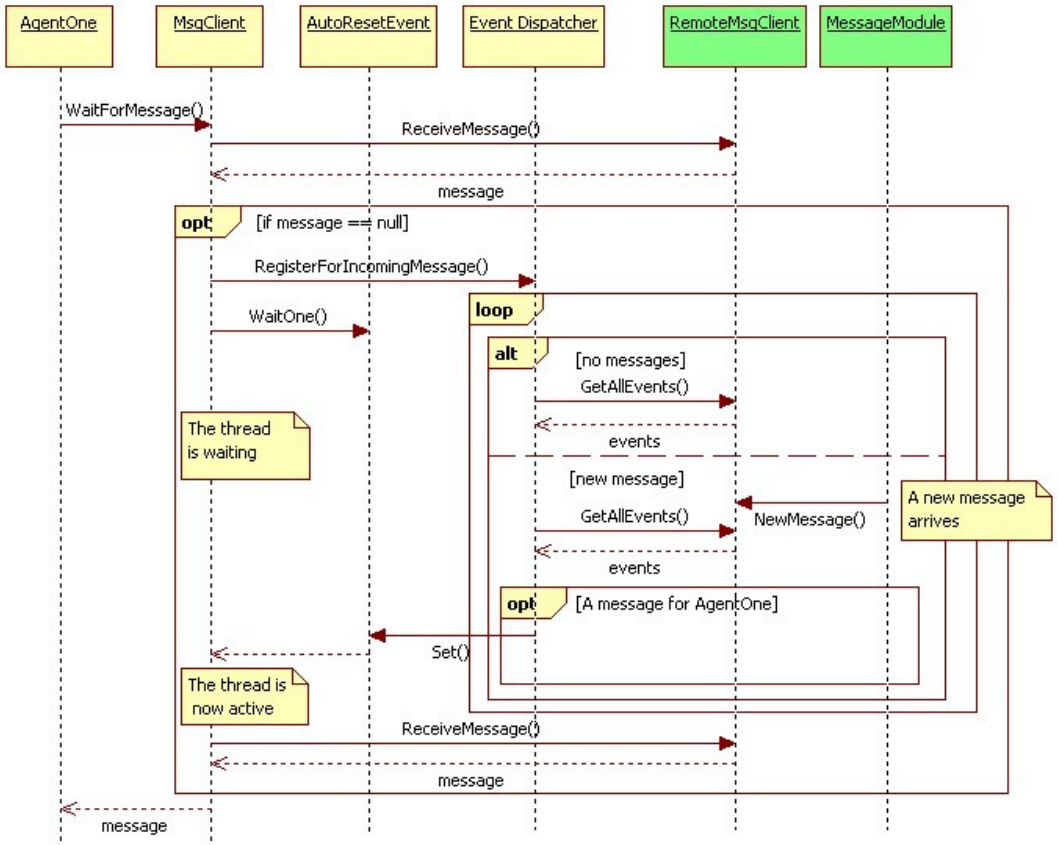


Figure 4: how an agent gets a message.

The good points appeared clearly in the previous sections, but during the implementation emerged some points which can be considered as faults. In particular, the integration of WCF and Compact Framework, although appreciable and powerful, represent a slight difficulty in order to exploit the real comfort of web services and WCF programming. In particular, due to the sensible reduction operated in the Compact Framework, the WCF support is surely complete but free from those classes that represent a comfortable surround. For this reason, because the WCF on the Compact Framework does not directly support the *Data Contract serialiazer* (the standard WCF serializer), we had to manually instruct the *basic xml serializer* in order to (de)serialize the messages used to invoke the WCF services.

A further issue regards not only WCF but also the SOA paradigm. It is a usual need, among agents, to send objects as message content. For this reason the message body is defined in AgentService as a box which can contain instances of every type targeting the .NET Framework (and then which derives from the *System.Object* class). The rigidity of the contract which characterizes every web service does not permit the invocation of a method passing a parameter which contains a generic object. This lack of flexibility forces us to convert such generic objects in strings containing their binary serialization, granting a solid contract which counts strings instead of generic objects.

B. JADE LEAP and AgentService Mobile

For the two projects, the main obstacle was the porting of the platform infrastructure on a little device. Curiously, for both the implementations, the principal cause was the message transport sub system. In JADE, agent containers exchange messages by using the Java RMI RPC technology. In AgentService does not exist a correspondence with containers, because we consider a platform the basic environment for circumscribed agent communities. In order to interconnect different agent societies we use .NET Remoting (which can be compared with Java RMI). Both Java RMI and .NET Remoting are not supported by J2ME and Compact Framework, here-hence the need to implement a new way for message exchanging. In JADE LEAP they opted for the creation of a proprietary protocol named JICP. In AgentService we opted for a service-oriented solution.

Continuing to analyze the differences between JADE LEAP and AgentService, because of the existence of the container concept in JADE, LEAP developers created a complete and autonomous infrastructure similar to a usual container to host agents on a mobile device. In AgentService we implement a minimal infrastructure which essentially warrants the execution of an agent and entrusts the implementation of each service to a remote standard

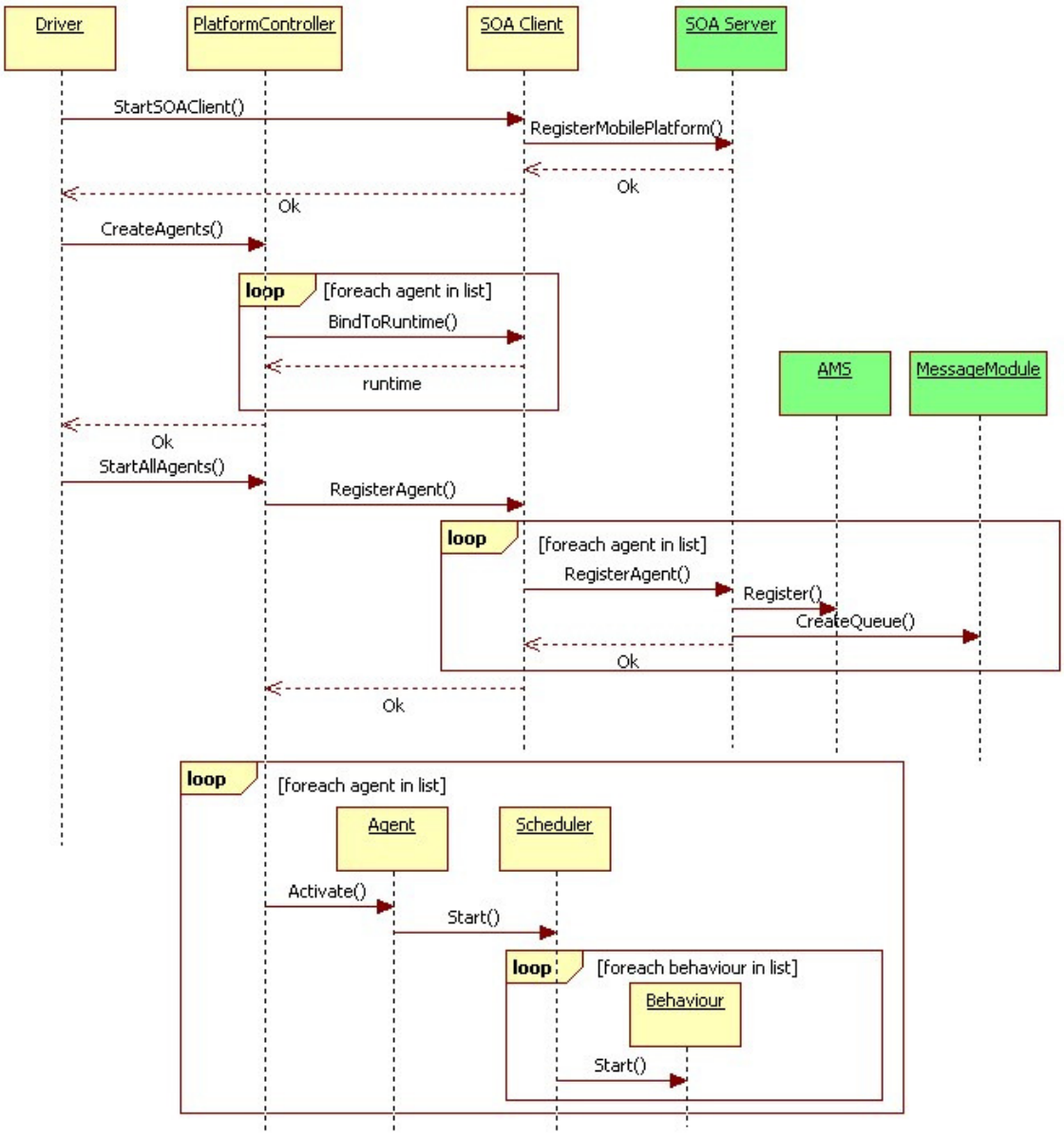


Figure 5: the bootstrap process.

AgentService platform. The JADE LEAP solution guarantees a certain independence for an agent container. On the other hand, the AgentService Mobile solution reduces the resources consuming of the infrastructure, leaving computational resources to the agent application. Our approach looks like the split container option in JADE LEAP but, while in LEAP it seems to be discouraged in the majority of cases, in AgentService Mobile gives good results in term of speed and performances.

A further difference between the two solutions is the fact that LEAP is an add-on, while AgentService Mobile is embedded in AgentService. In case the user want to add mobile agents on a preexistent platform, in AgentService it is sufficient to enable the SOA Server, while in JADE the platform must be replaced with the LEAP version, because a LEAP container and a JADE container must not live together (the cause is the different message transport sub system). From this point of view the AgentService solution appears to be more flexible.

Moreover, the service oriented interface of AgentService is not only aimed to agents on mobile devices but also to those external applications which want to communicate with agents and exploit the platform services.

We experienced that this requirement is fundamental for a lot of AgentService users.

C. Future works

AgentService Mobile is an ongoing project which can be improved in order to provide a more useful environment for our agents. Considering the compliancy to web standards, AgentService follows the *ws-** specification, but additional work as to be done for supporting *ws-security*, *ws-reliability*, and *ws-atomic transaction* that could be employed in the agents interaction protocol infrastructure

provided by AgentService in both the design and execution phases.

In addition, we will develop a light message dispatcher embedded in the mobile infrastructure, in order to avoid the employ of the remote message module for those conversations which involve agents running on the same device. This modification could distort the essence of our project, because every platform service is now considered as a web service. Nonetheless, for some application the rate of messages exchanged among agents hosted in the same device could justify this by pass. For this reason, the local message dispatcher will be optional.

An interesting evolution that confirms the trend to run agents on heterogeneous and limited devices is the development of an ultra-light infrastructure to host an agent on an embedded device. Recently, a .NET Micro Framework [21] has been delivered, in order to export the .NET programming also on basic devices with ARM processors, few RAM and no operating system. It could be interesting to run micro agents on these small devices and connect them to a standard AgentService platform. This improvement could consolidate AgentService applications in the industrial field, with agents deployed on sensors and actuators.

REFERENCES

[1] C. Vecchiola, A. Grosso, A. Boccalatte; "AgentService: a framework to develop distributed multi-agent systems", *Int. J. Agent-Oriented Software Engineering*, Vol. 2, No.3 pp. 290 – 323, 2008.

[2] N. Josuttis, "SOA in Practice, The Art of Distribuited System Design", O'Reilly, 2007.

[3] J. Smith, "Inside Microsoft Windows Communication Foundation",Microsoft Press 2007

[4] J. Löwy, "Programming WCF Services", O'Reilly Media, 2007

[5] Standard ISO/IEC 23271:2003: Common Language Infrastructure, March 28, 2003, ISO.

[6] OASIS, "Reference Architecture for Service Oriented Architecture 1.0", Public Review Draft 1, Apr. 23, 2008, http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf

[7] D. Melgar, "Message-Centric Web Services vs RPC-Style Invocations", *SOA World Magazine*, March 2003.

[8] Object Management Group, CORBA 3.0 Specification http://www.omg.org/technology/documents/formal/corba_2.htm

[9] Sun Developer Network, Remote Method Invocation (RMI) http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

[10] NET Framework Developer's Guide, .NET Remoting Overview http://msdn.microsoft.com/en-us/library/kwdt6w2k.aspx

[11] W3C Working Group, Web Services Glossary, 11 February 2004 http://www.w3.org/TR/ws-gloss/

[12] J. S. Lopez and F. A. Bustos, "An Agent Application on the Tourism Industry", In *Proceedings of the International Joint Conference IBERAMIA/SBIA/SBRN 2006 - 1st Workshop on Industrial Applications of Distributed Intelligent Systems (INADIS'2006)*, Ribeir˜ao Preto, Brazil, October 23–28, 2006

[13] G. Itabashi, M. Chiba, K. Takahashi, Y. Kato, "A Support System for Home Care Service Based on Multi-agent System",In *Proceedings of Fifth International Conference on Information, Communications and Signal Processing*, 2005

[14] M. Bombara and D. Cal and C. Santoro, "Kore: A multi-agent system to assist museum visitors", In *Proceedings of the Workshop on Objects and Agents (WOA2003)*, 2003

[15] E. McGovern , B. J. Roche1 , E. Mangina and R. Collier, "IUMELA: A Lightweight Multi-Agent Systems Based Mobile Learning Assistant Using the ABITS Messaging Service", Lecture *Notes in Computer Science*, Springer Berlin / Heidelberg, 2007

[16] Feng Li Ying Wei, "Tracking In-Transit RFID-Tagged Goods Using Multi-Agent Technology", In *Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing, WiCom 2007*.

[17] D. I. Tapia, J. Bajo, J. M. Sánchez and J. M. Corchado, "An Ambient Intelligence Based Multi-Agent Architecture", *Developing Ambient Intelligence*, Springer Paris, 2008

[18] A. Passadore and G. Pezzuto, "A multi-agent platform supporting maintenance companies on the field", In *Proceedings of the Workshop on Objects and Agents (WOA2007)*, 2007

[19] F. Bellifemine, G. Caire, D. Greenwood, "Developing Multi-agent Systems with JADE", John Wiley & Sons, 2007.

[20] M. Berger, S. Rusitschka, D. Toropov, M. Watzke, M. Schlichte, "Porting Distributed Agent-Middleware to Small Mobile Devices", In *Proceedings of the Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices*, Bologna, 2002.

[21] D. Thompson, R. S. Miles, "Embedded Programming with the Microsoft .NET Micro Framework", Microsoft Press, 2007.

Conservative re-use ensuring matches for service selection

Matteo Baldoni, Cristina Baroglio, Viviana Patti, and Claudio Schifanella
Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
{baldoni,baroglio,patti,schi}@di.unito.it

***Abstract*—The greater and greater quantity of services that are available over the web causes a growing attention to techniques that facilitate their reuse. A web service specification can be quite complex, including various operations and message exchange patterns. In this work, we give a declarative representation of services, and in particular of WSDL operations, that enables the application of techniques for reasoning about actions and change. By means of these techniques it becomes possible to reason on the specification of choreography roles and on possible role players, as a basis for selecting services which match in a flexible way with the specifications. Flexible match is, indeed, fundamental in order to enable web service reuse but it does not guarantee the preservation of the goals, that can be proved over the role specification itself. We show how to enrich various kinds of match proposed in the literature so to produce substitutions that preserve goals.**

I. Introduction

One of the key ideas behind web services is that services should be amenable to automatic retrieval, in order to allow the direct invocation as well as the composition with other services in order to fulfill a query. Nowadays, however, retrieval cannot yet be accomplished automatically as well and precisely as desired because the representations used for web services and the discovery mechanisms are semantically poor. The need of adding a *semantic layer* to service descriptions brought to initiatives like the development of OWL-S [1] and the development of the Web Service Modeling Ontology (WSMO) [2]. In these approach a richer annotation, aimed at representing the so called IOPEs (inputs, outputs, preconditions and effects of the service), is used. Inputs and outputs are usually described in terms taken from a public ontology, while preconditions and effects are often expressed by means of logic representations. A similar representation, based on preconditions and postconditions, is also typical of *design by contract*, originally introduced by Meyer for the Eiffel^{TM} language [3]. Here preconditions are the part of the contract which is to be guaranteed by the client; if this condition is guaranteed in the execution context of a method, then the server commits to guaranteeing that the postcondition holds in the state reached by the execution.

Semantic annotation allows the discovery of services, whose descriptions *do not exactly match* with the corresponding queries (e.g. [4], [5], [2]). Semantic matchmaking focuses on the discovery of single services, in the sense that a service is considered as corresponding to a *single operation*. In general, however, the use of a web service implies the execution of a *sequence of operations* in a particular *order*, which might even involve other services [6]: for instance, the clients of a supplier web service have to identify themselves, request item prices and delivery time, and so on. In order for the interaction to be successful, the interaction must obey some constraints: if they are not satisfied the service will be unable to proceed and will return an error. To allow the interaction, web services exhibit *interfaces* (port-types) which gather various operations that are logically related.

On the other hand, the need of describing *compositions* of services, which have to interact according to (complex) patterns of interaction, ruled by conversation protocols, has lead to the development of choreography languages like WS-CDL [7]. WS-CDL is aimed at describing collaborations between any type of participant independently from the programming model used by its implementation. Also a WS-CDL specification can be seen as a sort of contract, that specifies the ordering conditions and constraints that rule the interaction. The description is done from a global point of view, encompassing the expected behavior of all the participants. Each participant is supposed to use the global definition to build and test solutions that conform to it.

In this work, we focus on the problem of selecting existing services that have to play the roles of a given choreography. This task implies verifying two things: the *conformance* of the service to the specification of a role of interest, and that the use of that service allows the achievement of the *goal*, that caused its search. *Conformance* guarantees the interoperability of the service with the players of the other roles [8], [9], [10] by guaranteeing that the message exchange will produce correct and accepted conversations. The *goal* that caused the search of a service is a condition that should hold after the whole interaction has taken place. It is not tied to the descriptions of some service operation but it is a *global condition* that should hold in the final state, obtained after the conclusion of the conversation/interaction. In a framework in which it is possible to reason on operation preconditions and effects, and where an appropriate specification of the choreography is given, it becomes possible to design services which have a much higher degree of autonomy w.r.t. existing ones and whose behavior resembles more closely the behavior of autonomous agents. In particular, a service can decide whether playing a role by reasoning on the effects of playing the role and see whether it

can achieve a *goal* of interest by doing so. The achievement of the goal depends on the operation sequence because each operation can influence the executability and the outcomes of the subsequent ones. Many of the operations, however, are offered by the partners in the interaction which, at the time when the service reasons about the choreography, they are still unknown. The reasoning process must, therefore, use the specifications of the operations that will be supplied by the partners, specifications which are included in the choreography (that we call unbound operations). A selection process will link unbound operations with operations offered by existing services, and it does so by applying some kind of (possibly flexible) match. In [11], however, we showed that performing a match operation by operation, by applying the definitions in [12], *does not* preserve the global goal. Therefore, the match-making process, that is applied to discover services, should not only focus on local properties of the single operations, e.g. IOPEs, but it should also consider constraints that derive from the global schema of execution, which is given by the choreography. In this paper we extend the results achieved in [11], limited to the so-called plugin match, to the class of re-use ensuring matches [13]. To this aim, inspired by [1], [2], [14], we exploit an action-based representation of the specifications of the operations of a service: each operation is described in terms of its preconditions and effects, as in [15], [16], without taking into account the ontology layer which is not functional to the aims of the work. This representation supplies the mechanisms and the tools for reasoning on compositions of services, as described in choreographies; in particular, it supplies a representation of states and an execution model that can be reasoned about.

The paper is organized as follows. Section II sets the representation of services and of choreographies that we adopt. Section III discusses various kinds of match and reports our proposal for producing conservative matches. A running example is distributed along the pages to better explain the proposed notions and mechanisms. Conclusions end the paper.

II. A THEORETICAL FRAMEWORK FOR REPRESENTING AND REASONING ABOUT SERVICES

In this section, we briefly summarize the notation that we use to represent services, introduced in [16], and we discuss the problem of verifying a global goal. The notation is based on a logical theory for reasoning about actions and change in a modal logic programming setting. In this perspective, the problem of reasoning amounts either to build or to traverse a sequence of transitions between *states*. A state is a set of *fluents*, i.e., properties whose truth value can change over time, due to the application of actions. In general, we cannot assume that the value of each fluent in a state is known: we want to have both the possibility of representing unknown fluents and the ability of reasoning about the execution of actions on incomplete states. To explicitly represent unknown fluents, we use an epistemic operator $\mathcal{B}$, to represent the beliefs an entity has about the world: $\mathcal{B}f$ means that the fluent f is known to be true, $\mathcal{B}\neg f$ means that the fluent f is known to be false. A fluent f is undefined when both $\neg\mathcal{B}f$ and $\neg\mathcal{B}\neg f$ hold ($\neg\mathcal{B}f \wedge \neg\mathcal{B}\neg f$). For expressing that a fluent f is undefined, we write $u(f)$. Thus each fluent in a state can have one of the three values: *true*, *false* or *unknown*.

Services exhibit interfaces, called port-types, which make a set of operations available to possible clients. In our proposal, a *service description* is defined as a pair $\langle \mathcal{S}, \mathcal{P} \rangle$, where $\mathcal{S}$ is a set of basic operations, and $\mathcal{P}$ (*policy*) is a description of the complex behavior of the service. Analogously to what happens for OWL-S composite processes, $\mathcal{P}$ is built upon basic operations and tests that control the flow of execution.

A. Basic Operations

The set $\mathcal{S}$ contains the descriptions of a set of service operations. According to the main languages for representing web services, like WSDL and OWL-S, there are four basic kinds of operations [6] (or atomic processes, when using OWL-S terminology [1]): *one-way*, *notify*, *request-response*, and *solicit-response*. The first two involve a single message exchange. In a one-way operation, a client invokes an operation by sending a message to the service, while by a notification the client receives a message from the service. The other two operations involve the exchange of two messages. Request-response operations are initiated by the invoker of the operation, which sends a message to the service and, after that, waits for a response. In solicit-response operations the order of the messages is inverted: first the invoker waits for a message from the service and then it sends an answer.

An operation is described in terms of its *executability preconditions* and *effects*, the former being a set of fluents (introduced by the keyword **possible if**) which must be contained in the service state in order for the operation to be applicable, the latter being a set of fluents (introduced by the keyword **causes**) which will be added to the service state after the operation execution. Formalized in these terms, operations, when executed, trigger a revision process on the actor's beliefs. Since we describe web services from a *subjective* point of view, we distinguish between the case when the service is either the initiator (the operation *invoker*) or the servant of an operation (the operation *supplier*) by further decorating the operation name with a notation inspired by [14]. With reference to a specific service, $operation^{\gg}$ denotes the operation from the point of view of the invoker, while $operation^{\ll}$ denotes the operation from the point of view of the supplier. The view of operations that is used by *invoker* is given in terms of the operation inputs, outputs, preconditions, and effects as usual for semantic web services [1]. In the next part of this section, inputs and outputs are represented as single messages for simplicity but the representation can easily be extended to sets of exchanged data, as in Example (1). Preconditions P_s and effects E_s are respectively the conditions required by the operation in order to be invoked, and the expected effects that result from the execution of the operation. For what concerns the view of the *supplier*, also in this case the operation is described in terms of its inputs and outputs. Moreover, we also represent a set of conditions that enable the executability

of the operation (R_s, *requirements*) and that constitute the *side effects*, S_s. For example, a *buy* operation of a selling service has as a precondition the fact that the invoker has a valid credit card, as inputs the credit card number of the buyer and its expiration date, as output it generates a receipt, and as effect the credit card is charged. From the point of view of the supplier, the requirement to the execution is to have an active connection to the bank, and the side effect is that the store availability is decreased while the service bank account is increased of the perceived amount.

Let us now introduce the formal representation of the four kinds of basic operations. For each operation we report both views.

One-Way, invoker point of view:
(a) $\mathsf{operation}^{\gg}_{ow}(m_{in})$ **possible if** $\mathbf{B}^{Invoker} m_{in} \wedge P_s$
(b) $\mathsf{operation}^{\gg}_{ow}(m_{in})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$
(c) $\mathsf{operation}^{\gg}_{ow}(m_{in})$ **causes** E_s

In one-way operations, the invoker requests an execution which involves sending an information m_{in} to the supplier; the invoker must obviously know the information to send before the invocation (a). The invoker can execute the operation only if the preconditions to the operations are satisfied in its current state (a). The execution of the invocation brings about the effects E_s of the operation (c), and the invoker will know that it has sent an information to the supplier (b). Using OWL-S terminology, m_{in} represents the *input* of the operation, while P_s and E_s are its preconditions and effects. One-way operations have no output.

One-Way, supplier point of view:
(a) $\mathsf{operation}^{\ll}_{ow}(m_{in})$ **possible if** R_s
(b) $\mathsf{operation}^{\ll}_{ow}(m_{in})$ **causes** $\mathbf{B}^{Offerer} m_{in}$
(c) $\mathsf{operation}^{\ll}_{ow}(m_{in})$ **causes** S_s

On the other hand, the supplier, which exhibits the one-way operation as one of the services that it can execute, has the requirements R_s (a). The execution of the operation causes the fact that the supplier will know the information sent by the invoker (b). We also allow the possibility of having some side effects on the supplier's state. These effects are not to be confused with the operation effects described by IOPE, and have been added for the sake of completeness.

Notify, invoker point of view:
(a) $\mathsf{operation}^{\gg}_{n}(m_{out})$ **possible if** P_s
(b) $\mathsf{operation}^{\gg}_{n}(m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$
(c) $\mathsf{operation}^{\gg}_{n}(m_{out})$ **causes** E_s

In notify operations, the invoker requests an execution which involves receiving an information m_{out} from the supplier. The invoker can invoke the execution of the operation only if the preconditions to the operations are satisfied in its current state (a). The execution of the invocation brings about the effects E_s of the operation (c), and the invoker will know the received information (b). Using OWL-S terminology, m_{out} represents the *output* of the operation, while P_s and E_s are its preconditions and effects. Notify operations have no input.

Notify, supplier point of view:
(a) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **possible if** $\mathbf{B}^{Offerer} m_{out} \wedge R_s$
(b) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **causes** $\mathbf{B}^{Offerer} sent(m_{out})$
(c) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **causes** S_s

The supplier must know the information to send and must meet the requirements R_s (a). The execution of the operation simply causes the fact that the supplier will know that it has sent some information to the invoker (b). As above, we allow the possibility of having some side effects on the supplier's state (c).

Request-response, invoker point of view:
(a) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **possible if** $\mathbf{B}^{Invoker} m_{in} \wedge P_s$
(b) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$
(c) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$
(d) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **causes** E_s

In request-response operations, the invoker requests an execution which involves sending an information m_{in} (the input, according to OWL-S terminology) and then receiving an answer m_{out} from the supplier (the output in OWL-S). The invoker can execute the operation only if the preconditions P_s are satisfied in its current state and if it owns the information to send (a). The execution of the invocation brings about the effects E_s (d), and the fact that the invoker knows that it has sent the input m_{in} to the supplier (b). One further effect of the execution is that the invoker knows the answer returned by the operation (c). This representation abstracts away from the actual message exchange mechanism, which is implemented. Our aim is to reason on the effects of the execution on the mental state of the parties [15].

Request-response, supplier point of view:
(a) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **possible if** R_s
(b) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} m_{in}$
(c) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} m_{out}$
(d) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} sent(m_{out})$
(e) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** S_s

As for one-way operations, the supplier has the requirements R_s to the operation execution (a). It receives an input m_{in} from the invoker (b). The execution of the operation produces an answer m_{out} (c), which is sent to the invoker (d). As usual, we allow the possibility of having some side effects on the supplier's state. On the supplier's side, we can notice more evidently the abstraction of the representation from the actual execution process. In fact, we do not model how the answer is produced but only the fact that it is produced.

Solicit-response, invoker point of view:
(a) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **possible if** P_s
(b) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$
(c) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{in}$
(d) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$
(e) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** E_s

In solicit-response operations, the invoker requests an execution which involves receiving an information m_{out} (the output,

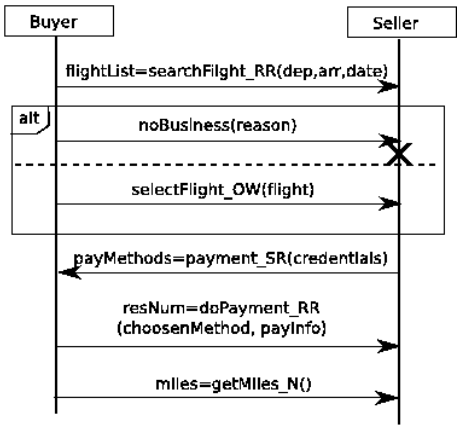


Fig. 1. An example of a simple interaction protocol, for reserving a flight, expressed as a UML sequence diagram.

according to OWL-S terminology) and then sending a message m_{in} to the supplier (the input in OWL-S). The invoker can execute the invocation only if the preconditions P_s are satisfied in its current state (a). The execution of the invocation brings about the effects E_s (e). The invoker receives a message m_{out} from the supplier (b) then, it produces the input information m_{in} which is sent to the supplier, see (c) and (d).

Solicit-response, supplier point of view:

(a) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **possible if** $\mathbf{B}^{Offerer} m_{out} \wedge R_s$
(b) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} sent(m_{out})$
(c) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} m_{in}$
(d) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** S_s

As for notify operations, the supplier must know the information to send and to fulfill the requirements R_s (a). The execution of the operation causes the fact that the supplier will know that it has sent some information to the invoker (b). Moreover, it produces also the knowledge of the information m_{in} received by the invoker (c). As above, we allow the possibility of having some side effects on the supplier's state (d).

Example 1: As an example, let's consider the searchFlight operation of the flight reservation protocol depicted in Figure 1, which is offered by a *seller* and can be invoked by a *buyer* to search information about flights with given departure (dep) and arrival locations (arr) plus the date of departure ($date$). From the point of view of the buyer, the operation, which is of kind request-response, is:

(a) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **possible if** $\mathbf{B}^{buyer} dep \wedge \mathbf{B}^{buyer} arr \wedge \mathbf{B}^{buyer} date \wedge \mathbf{B}^{\mathbf{buyer}} \neg \mathbf{sellingStarted}$
(b) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{buyer} sent(dep) \wedge \mathbf{B}^{buyer} sent(arr) \wedge \mathbf{B}^{buyer} sent(date)$
(d) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{buyer} flightList$
(c) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{\mathbf{buyer}} \mathbf{sellingStarted}$

The *inputs* of the operation are dep, arr, and $date$, while the *output* is $flightList$. In this case the set P_s contains only the belief $\mathbf{B}^{buyer} \neg sellingStarted$ (in bold text above) while the set E_s of effects contains the belief $\mathbf{B}^{buyer} sellingStarted$ (in bold text as well).

From the point of view of the supplier, instead, the operation is represented as:

(a) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **possible if** $true$
(b) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} dep \wedge \mathbf{B}^{seller} arr \wedge \mathbf{B}^{seller} date$
(c) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} flightList$
(d) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} sent(flightList)$

In this case the sets R_s and S_s of requirements and side effects are empty. The operation expects as input the departure and arrival locations and the date of the flight, and it produces a $flightList$, which it sends to its customer, so after the operation the belief $\mathbf{B}^{seller} sent(flightList)$ will be in its belief state. □

Last but not least, a service can also have internal operations, which can be included in its policy but are not visible from outside. Each operation is represented again as an atomic action, specified by its *preconditions* and its *effects*. Formally, it is defined as:

$\mathsf{operation}(content)$ **causes** E_s
$\mathsf{operation}(content)$ **possible if** P_s

where E_s and P_s, denote respectively the fluents, which are expected as effect of the execution of an operation and the precondition to its execution, while $content$ denotes possible additional data that is required by the operation. Notice that such operations can also be implemented as invocations to other services.

B. Composite operations

$\mathcal{P}$ encodes the complex behavior of the service; it is a collection of clauses of the kind:

p_0 **is** $p_1, \ldots, p_n$

where p_0 is the name of the procedure and p_i, $i = 1, \ldots, n$, is either an atomic action (operation), a test action (denoted by the symbol ?), or a procedure call. Procedures can be recursive and are executed in a goal-directed way, similarly to standard logic programs, and their definitions can be non-deterministic as in Prolog.

A *choreography* is made of a set of interacting *roles*, a role being a subjective view of the interaction that is encoded. When a service plays a role in a choreography, its policy will contain some operations which are not of the service itself but belong to some other role of the choreography, with which it interacts. In other words, $\mathcal{S}$ can be partitioned in two sets: a set of bound operations and a set of *unbound operations*, that must be supplied by some counterpart(s). Until the counterpart service(s) is (are) not defined, such operations will be those

specified in the choreography. Such operations will be offered by the interlocutors as $\gg$ operations. We assume that they are represented in a way that is homogeneous with the representation of operations, i.e. by means of *preconditions* and *effects*. The binding will be possible only when the partners in the interaction will be found.

The fact that a service is taking a given role in the choreography is due, in our proposal, to the fact that it knows that a certain goal condition will be true after the execution of the role. When a possible partner is identified for the latter role, after the binding has taken place, it is necessary to check if the goal condition is preserved. The reasons for which this could not happen are explained in the following section; hereafter, we formalize the notion of *substitution* that we interpret as the binding.

Let $S_d = \langle \mathcal{S}, \mathcal{P} \rangle$ be a service description, and let $\mathcal{S}_u$ be a subset of $\mathcal{S}$, containing unbound operations that are to be supplied by a same counterpart S_i. Let $\mathcal{S}_{S_i}$ be the set of operations in S_i that we want S_d to use, binding them to $\mathcal{S}_u$. We represent the binding by the substitution $\theta = [\mathcal{S}_{S_i}/\mathcal{S}_u]$ applied to S_d, i.e.: $S_d\theta = \langle \mathcal{S}\theta, \mathcal{P}\theta \rangle$, where every element of $\mathcal{S}_u$ is substituted by/bound to an element of $\mathcal{S}_{S_i}$. Notice that not all elements of $\mathcal{S}_{S_i}$ are, instead, necessarily bound. An example is reported in *Example* 2.

Example 2: As an instance, here we report the definition of the buyTicket procedure of the flight company example. The procedure will encode a role in a choreography if all of the involved operations are unbound. It will, instead, encode a service behavior when all of its operations are bound to those offered by one or more services that act as interlocutors.

(a) buyTicket **is**
searchFlight$^{\gg}_{rr}((dep, arr, date), flightList)$;
evaluateAndBuy
(b) evaluateAndBuy **is**
noBusiness$^{\gg}_{ow}(reason)$
(c) evaluateAndBuy **is**
selectFlight$^{\gg}_{ow}(flight)$;
payMethods$^{\ll}_{sr}(payMethods, credentials)$;
doPayment$^{\gg}_{rr}((chosenMethod, payInfo), resNum)$;
getMiles$^{\gg}_{n}(miles)$

This procedure encodes the behaviour of the buyer of a flight ticket, be it a role or a specific service. First, it invokes an operation for searchig a flight (searchFlight$^{\gg}_{rr}((dep, arr, date), flightList)$), and it evaluates the result (evaluateAndBuy). The evaluation can give either a negative outcome, hence the interaction is interrupted (noBusiness$^{\gg}_{ow}(reason)$) or the interaction continues with the flight selection (selectFlight$^{\gg}_{ow}(flight)$), the payment (payMethods$^{\ll}_{sr}(payMethods, credentials)$ and doPayment$^{\gg}_{rr}((chosenMethod, payInfo), resNum)$) and, at the end, the client is notified about the obtained miles (getMiles$^{\gg}_{n}(miles)$). □

C. Reasoning on goals

In the outlined framework, it is possible to reason about goals by means of queries of the form:

$$Fs \ \textbf{after} \ p$$

where Fs is the goal (represented as a conjunction of fluents), that we wish to hold after the execution of a policy p. Checking if a formula of this kind holds corresponds to answering the query: "Is it possible to execute p in such a way that the condition Fs is true in the final state?". When the answer is positive, the reasoning process returns a sequence of atomic actions that allows the achievement of the desired condition. This sequence corresponds to an execution trace of the procedure and can be seen as a plan to bring about the *goal* Fs. This form of reasoning is known as *temporal projection*. Temporal projection fits our needs because, as mentioned in the introduction, in order to perform the selection we need a mechanism that verifies if a goal condition holds after the interaction with the service has taken place. Fs is the set of facts that we would like to hold "after" p.

Let $S_d = \langle \mathcal{S}, \mathcal{P} \rangle$ be a service description. The application of temporal projection to $\mathcal{P}$ returns, if any, an execution trace, that makes a goal of interest become true. Let us, then, consider a procedure p belonging to $\mathcal{P}$, and denote by G the query Fs **after** p. Given a state S_0, containing all the fluents that we know as being true in the beginning, we denote the fact that G is successful in S_d by:

$$(\langle \mathcal{S}, \mathcal{P} \rangle, S_0) \vdash G$$

The execution of the above query returns as a side-effect an *execution trace* σ of p. The execution trace σ is *linear*, i.e. a terminating sequence $a_1, \ldots, a_n$ of atomic actions.

Example 3 (Flight-purchase, second part): Let us suppose that the initial state of the service $b1$ is $S_0 = \{\mathbf{B}^{buyer} dep, \mathbf{B}^{buyer} arr, \mathbf{B}^{buyer} date, \mathbf{B}^{buyer} deferredPaymentPossible, \mathbf{B}^{buyer} \neg sellingStarted\}$, (all the other fluents truth value is "unknown"). This means that $b1$ assumes a date, a departure location, an arrival location, the fact that it is possible to defer the payment to the departure (at a desk at the airport), and that no selling process has started yet. The goal of $b1$ is to achieve the following condition: $G = \{\mathbf{B}^{buyer} sellingComplete, \mathbf{B}^{buyer} resNum\}$ **after** buyTicket Intuitively, the buyer expects that, after the interaction, it will have a reservation number as a result.

By reasoning on its policy and by using the definitions of the unbound operations that are given by the choreography, $b1$ can identify an execution trace, that leads to a state where G holds:

$\sigma =$ searchFlight$^{\gg}_{rr}((dep, arr, date), flightList)$;
selectFlight$^{\gg}_{ow}(flight)$;
payMethods$^{\ll}_{sr}(payMethods, credentials)$;
doPayment$^{\gg}_{rr}((chosenMethod, payInfo), resNum)$;
getMiles$^{\gg}_{n}(miles)$

This is possible because in a declarative representation specifications are executable. Moreover notice that this execution

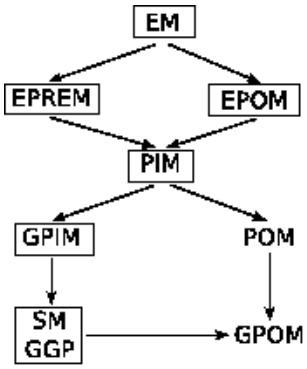


Fig. 2. The lattice of the local matches: on top the strongest; names in a box are re-use ensuring matches; SM and GGP are in same box because logically equivalent.

does not influence the belief about the deferred payment, which persists from the initial through the final state and is not contradicted. □

III. Conservative, re-use ensuring matches

When the matching process is applied for selecting a service that should play a role in a (partially instantiated) choreography, the desire is that the substitution (of the service operations to the specifications contained in the choreography) preserves the properties of interest. In [11] we have formalized this notion in the following way:

Definition 1 (Conservative substitution): Let us consider a service $S_i = \langle \mathcal{S}, \mathcal{P} \rangle$ playing a role R_i in a given choreography, and a query G such that, given an initial state S_0,

$$(\langle \mathcal{S}, \mathcal{P} \rangle, S_0) \vdash G \text{ w.a. } \sigma$$

Consider a substitution $\theta = [\mathcal{S}_{S_j}/\mathcal{S}^{\sigma}_{u(R_j)}]$, where $\mathcal{S}^{\sigma}_{u(R_j)} = \{o_u \in \mathcal{S} \mid o \text{ occurs in } \sigma\}$ is the set of all unbound operations that refer to another role R_j, $j \neq i$, of the same choreography, that are used in the execution trace σ. θ is conservative when the following holds:

$$(\langle \mathcal{S}\theta, \mathcal{P}\theta \rangle, S_0) \vdash G \text{ w.a. } \sigma\theta$$

□

In the above definition, θ can be any kind of association between the operations of a service with the unbound operations described in a choreography. In practice it is the result of a *matching process*. In the literature it is possible to find many match algorithms, mostly based on the seminal work by Zaremski and Wing [12] on software components, and surveyed in [17].

Given a software component S, with precondition S_{pre} and postcondition S_{post}, and a specification (or query, in the match-making community) Q, with precondition Q_{pre} and postcondition Q_{post}, the most important kinds of relaxed match between Q and S are:

- EM (*Exact Pre/Post Match*): $Q_{pre} \Leftrightarrow S_{pre} \wedge Q_{post} \Leftrightarrow S_{post}$
- EPREM (*Exact Pre Match*): $Q_{pre} \Leftrightarrow S_{pre} \wedge S_{post} \Rightarrow Q_{post}$
- EPOM (*Exact Post Match*): $Q_{pre} \Rightarrow S_{pre} \wedge Q_{post} \Leftrightarrow S_{post}$
- PIM (*Plugin Match*): $Q_{pre} \Rightarrow S_{pre} \wedge S_{post} \Rightarrow Q_{post}$
- POM (*Plugin Post Match*): $S_{post} \Rightarrow Q_{post}$
- GPIM (*Guarded Plugin Match*, a.k.a. Weak-Plugin [18]): $Q_{pre} \Rightarrow S_{pre} \wedge ((S_{pre} \wedge S_{post}) \Rightarrow Q_{post})$
- SM (*Satisfies Match*, a.k.a. relaxed plug-in in [13], plug-in compatibility [19]): $Q_{pre} \Rightarrow S_{pre} \wedge (Q_{pre} \wedge S_{post} \Rightarrow Q_{post})$
- GPOM (*Guarded Post Match*, a.k.a. Weak-Post [18]): $((S_{pre} \wedge S_{post}) \Rightarrow Q_{post})$
- GGP (*Guarded-Generalized Predicate*): $(Q_{pre} \Rightarrow S_{pre}) \wedge ((S_{pre} \Rightarrow S_{post}) \Rightarrow (Q_{pre} \Rightarrow Q_{post}))$

The different matches can be organized according to a lattice [17], that we have reported in Fig. 2. On top, there is *Exact pre/post match*, which states the equivalence of Q and S. Moving down in the lattice weaker and weaker match conditions are found. For instance, in *Plugin match* S must only be behaviorally equivalent to Q when plugged-in to replace Q. *Plugin post match* relaxes the former: only the postcondition is considered. *Guarded matches* focus on guaranteeing that the desired postcondition holds when the precondition of S holds, not necessarily in general.

In our application domain, Q is an *unbound operation*, while S is an *operation*. For short, we decorate substitutions with an acronym denoting the applied match (e.g. θ_{EM} is a substitution obtained by applying the exact match, θ_{PIM} by applying the plugin match, etc.). All these matches have been defined for the retrieval of single components, and have a *local* nature, i.e. they compare a requirement to a software specification (in our case, an unbound operation) independently of the *context of usage* (in our work, the choreography role). On the other hand, a choreography defines the *global execution context*, in which unbound operations are immersed.

In [11] we have proved that, in general, flexible matches do not satisfy Definition 1. In other words, it is *not guaranteed* that after the substitution of a set of operations, which were selected by applying one of the local flexible matches, to a set of unbound operations in a role, a goal that could previously be achieved is still achievable. In fact, besides a few special cases (EM and EPOM are the only matches which, by their own nature are conservative), the identified operation can produce additional effects w.r.t. Q_{post}. This is not a problem when the operation is to be used alone but when it is inserted in the context of a role execution, the additional effects may inhibit the preconditions of operations that follow. This is a problem because the choice of playing a role bases on the proof that the adoption of that role allows the achievement of a goal of interest for the player. The substitution is necessary in order to make the role executable but this transformation should not affect the possibility of reaching the goal, that was demonstrated for the role specification. In [11] we also showed how to enrich the plugin match so to guarantee that the built substitutions are conservative. This is done by taking into

account the *overall structure*, encoded by the choreography.

In the following we extend this result to a wider class of matches; in particular, we show that all matches which are *re-use ensuring*, according to the definition given by Chen and Cheng in [13], can be enriched in order to guarantee the production of conservative matches. Once again, we do this by exploiting only constraints that can be inferred from the choreography, without modifying the local nature of the considered matches.

Definition 2 (Re-use ensuring match [13]): A specification match M is re-use ensuring iff for any S and Q, $M(S,Q) \wedge \{S_{pre}\}S\{S_{post}\} \Rightarrow \{Q_{pre}\}S\{Q_{post}\}$.

In the above definition, $\{C_{pre}\}C\{C_{post}\}$ denotes a Hoare triple and is informally interpreted as the truth of "program C started with C_{pre} satisfied will terminate in a state such that C_{post} holds" [20]. Considering the lattice in Figure 2, re-use ensuring matches are all those in a box, while POM and GPOM are not re-use ensuring.

In order to extend the results in [11] to all re-use ensuring matches, we need to recall a few notions given in that paper. Intuitively, we take into account the *dependencies* between operations, which produce as effects fluents, that are used as preconditions by subsequent operations. The idea is to verify that the "causal chains" which allow the execution of the sequence of operations, are not broken after the substitution. The obvious hypothesis is that we have a choreography and that we know that it allows to achieve the goal of interest, i.e. that there is an execution σ, which allows the achievement of the goal. We will use this trace for defining a set of constraints that, whenever satisfied by a substitution obtained by a re-use ensuring match, guarantee that the substitution is also conservative. This is a "sufficient" condition because there might exist conservative substitutions that do not satisfy this set of constraints.

Let us start with the notions of dependencies between operations and dependency sets for fluents. Consider a service description $S = \langle \mathcal{S}, \mathcal{P} \rangle$, and suppose that, given the initial state S_0, the goal $G = Fs$ **after** p succeeds, thus obtaining as answer the successful sequence of operations $\sigma = a_1; a_2; \ldots; a_n$, which is an execution trace of p. We denote by $\overline{\sigma}$ the sequence of operations $a_0; a_1; a_2; \ldots; a_n; a_{n+1}$, where a_0 and a_{n+1} are two *fictitious* operations that will be used respectively to represent the initial state S_0 and the set of fluents Fs, which must hold after σ. That is, we assume a_0 has no precondition and $E_s(a_0) = S_0$, and that a_{n+1} has no effect but $P_s a_{n+1} = Fs$.

Consider two indexes i and j, such that $j < i$, $i, j = 0, \ldots, n+1$. We say that *in* $\overline{\sigma}$ the operation a_i *depends on* a_j *for the fluent* $\mathcal{B}l$, written $a_j \rightsquigarrow_{\langle \mathcal{B}l, \overline{\sigma} \rangle} a_i$, iff $\mathcal{B}l \in E_s(a_j)$, $\mathcal{B}l \in P_s a_i$, and there is not a k, $j < k < i$, such that $\mathcal{B}l \in E_s(a_k)$. Given a fluent $\mathcal{B}l$ and a sequence of operations σ, we can, therefore, define the *dependency set* of $\mathcal{B}l$ as $\mathsf{Deps}(\mathcal{B}l, \sigma) = \{(j,i) \mid a_j \rightsquigarrow_{\langle \mathcal{B}l, \overline{\sigma} \rangle} a_i\}$.

Let $[s/o_u]$ be a specific substitution of a service operation s to an unbound operation o_u, that is contained in θ, we say that a fluent $\mathcal{B}l \in E_s(s) - E_s(o_u)$ (i.e. an additional effect of s w.r.t. the effects of o_u) is an *uninfluential fluent* w.r.t. the sequence $\sigma\theta$ iff for all pairs $(j,i) \in \mathsf{Deps}(\mathcal{B}\neg l, \sigma)$, identifying by k the position of o_u in σ, we have that $k < j$ or $i \leq k$. Intuitively, this means that the fluent *will not break* any dependency between the operations which involve the inverse fluent because either it will be overwritten or it will appear after its inverse has already been used. Note that σ and $\sigma\theta$ have the same length and are identical as sequences of operations but for the fact that in the latter the selected service operations substitute unbound operations. For this reason, we can reduce to reasoning on σ for what concerns the operation positions.

Definition 3: A substitution θ is called *uninfluential* iff for any substitution $[s/o_u]$ in θ, all beliefs in $E_s(s) - E_s(o_u)$ are uninfluential fluents w.r.t. σ.

Proposition 1: Let M be a re-use ensuring match, any substitution θ_M that is uninfluential is also conservative.

Proof: The proof is by absurd and it uses the proof theory introduced in [21]. Let us assume that $(\langle \mathcal{S}, \mathcal{P} \rangle, S_0) \vdash G$ w.a. σ but $(\langle \mathcal{S}\theta_M, \theta_M, \mathcal{P}\theta_M \rangle, S_0) \nvdash G$ w.a. $\sigma\theta_M$. Therefore, there exists a fluent F such that $a_0, a_1, \ldots, a_{i-1} \vdash F$ but $(a_0, a_1, \ldots, a_{i-1})\theta_M \nvdash F$, where $\sigma = a_0, a_1, \ldots, a_{i-1}, a_i, \ldots, a_n$ and $F \in P_s(a_i)$, i.e. a_i is not executable because one of its preconditions does not hold. Now, since $a_0, a_1, \ldots, a_{i-1} \vdash F$, there exists $j \leq i-1$, such that $a_0, a_1, \ldots, a_j \vdash F$ and $F \in E_s(a_j)$ but $(a_0, a_1, \ldots, a_j)\theta_M \nvdash F$. Let us assume that j is the last operation to produce F before a_i. There are two possible cases, either $F \notin E_s(a_j\theta_M)$ or there is another operation $a_k\theta_M$, with $j < k < i$, such that $\neg F$ is one of its effects. The first case is absurd since by hypothesis the match is re-use ensuring, therefore $(a_0, a_1, \ldots, a_{i-1}, a_i)\theta_M \vdash F$, for any fluent F in $E_s(a_i)$. The second is absurd as well, since j is the last operation to produce F, the effect $\neg F$ of $a_k\theta_M$ should be one of its additional effects but this is absurd because by hypothesis θ_M is an uninfluential substitution. ■

From the above proposition and the construction of dependency sets, it is easy to show that the following theorem holds.

Theorem 1: Let M be a re-use ensuring match, $S_i = \langle \mathcal{S}, \mathcal{P} \rangle$ be a service which plays a role R_i in a given choreography, and G a query such that, $(\langle \mathcal{S}, \mathcal{P} \rangle, S_0) \vdash G$ w.a. σ, where S_O is the initial state. Let θ_M be a substitution for all unbound operations of S_i that refer to another role R_j played by the service S_j, $j \neq i$. The problem of determining whether θ_M is conservative w.r.t. G is decidable.

Example 4: Let us now consider the goal and the service description specified in the previous examples, and suppose that the operation payMethod is defined in this way:

(a) $\mathsf{payMethods}^{\ll}_{sr}(payMethods, credentials)$ **possible if**
$\mathbf{B}^{buyer} credentials \wedge$
$\mathbf{B}^{buyer} mustPay(flight)$
$\wedge \mathbf{B^{buyer}}\mathbf{deferredPaymentPossible}$

(b) $\mathsf{payMethods}^{\ll}_{sr}(payMethods, credentials)$ **causes**
$\mathbf{B}^{buyer} sent(credentials)$

(c) $\mathsf{payMethods}^{\ll}_{sr}(payMethods, credentials)$ **causes**
$\mathbf{B}^{buyer} payMethods$

(d) $\mathsf{payMethods}^{\ll}_{sr}(payMethods, credentials)$ **causes**

$\top$

In particular, the operation has, as a precondition, the possibility of pay the ticket directly at the airport ($\mathbf{B}^{buyer} deferredPaymentPossible$).

Let us now consider a service, that is a candidate to play the role of *Seller*, which is equivalent to the role specification but for the operation that implements searchFlight, which is specified as:

(a) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **possible if** $true$

(b) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} dep \wedge \mathbf{B}^{seller} arr \wedge \mathbf{B}^{seller} date$

(c) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} flightList$

(d) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} sent(flightList)$

(e) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{\neg B^{buyer} deferredPaymentPossible}$

This operation has an additional effect, w.r.t. to the corresponding unbound operation, that is it negates the possibility of paying the ticket at the airport ($\neg \mathbf{B}^{buyer} deferredPaymentPossible$). Despite this, the service matches with the unbound operation according to many of the re-use ensuring matches (e.g. EPREM, PIM, GPIM, SM). This additional effect is not uninfluential because it prevents the executability of the operation payMethod, as defined above.

If the additional effect were, for instance, that $\mathcal{B}veg\_meals$, supplying an additional information concerning the availability of vegetarian meals, the achievement of the goal would not be compromised and the selection would be allowed. □

IV. Conclusions

In this work we extended the proposal in [11] by proving that for any re-use ensuring match, as defined in [13], it is decidable to verify that the obtained substitutions are conservative w.r.t. a goal that is proved by using for the unbound operations the specifications provided by the choreogrpahy. This result allows the enrichment of the matches with a test that can be applied at the match execution time, locally, i.e. operation by operation. This is done by taking into account the execution context given by the choreography.

The literature related to matchmaking is wide and it is really difficult to be exhaustive. The matches proposed in [12] have inspired most of the semantic matches for web service discovery. Amongst them, Paolucci et al. [4] propose four degrees of match (exact, plugin, subsumes, and fail). Differently than in our proposal, these matches are computed on the ontological relations of the outputs of an advertisement for a service and a query and are orthogonal to our work. This approach tackles DAML-S representations, in which services are described by means of inputs and outputs. This approach is refined in [5], a work that describes a service matchmaking prototype, which uses a DAML-S based ontology and a Description Logic reasoner to compare ontology-based service descriptions, given in terms of input and output parameters. The matchmaking process, like in [4], produces a discrete scale of degrees of match (Exact, PlugIn, Subsume, Intersection, Disjoint).

WSMO (Web Service Modeling Ontology) [2] is an organizational framework for semantic web services. As such, it does not suggest a specific matching rule, which is up to the specific implementations. However, the authors propose in [22] an approach that is based on [12] and on [5], hence it suffers of the same limits that we have mentioned.

Works like [23], [24] propose approaches for goal-driven service composition based on planning. However, the task is accomplished without reference to any choreography. In particular, in [23] the composition and the semantic reasoning phases (carried on on inputs and outputs) are separated and the latter is performed on a local basis only. In [25], [26] web services are composed by composing their interaction protocols in a social framework, by means of a temporal logic.

References

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

[2] D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, and A. Polleres, *Enabling Semantic Web Services : The Web Service Modeling Ontology*. Springer.

[3] B. Meyer, "Applying "design by contract"," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[4] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *Proc. of ISWC '02*. Springer, 2002, pp. 333–347.

[5] L. Li and I. Horrocks, "A software framework for matchmaking based on semantic technology," in *Proc. of WWW Conference*. ACM Press, 2003.

[6] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services*. Springer, 2004.

[7] WS-CDL, "http://www.w3.org/tr/ws-cdl-10/."

[8] S. K. Rajamani and J. Rehof, "Conformance checking for models of asynchronous message passing software," in *Proc. of 14th International Conference on Computer Aided Verification, CAV 2002*, ser. LNCS, vol. 2404. Springer, 2002, pp. 166–179.

[9] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based analysis of obligations in web service choreography," in *Proc. of IEEE International Conference on Internet&Web Applications and Services 2006*, 2006.

[10] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro, "Choreography and orchestration: a synergic approach for system design," in *Proc. of ICSOC 2005*, 2005.

[11] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, "Service selection by choreography-driven matching," in *Emerging Web Services Technology*, ser. Whitestein Series in Software Agent Technologies and Autonomic Computing, T. Gschwind and C. Pautasso, Eds. Birkhäuser, September 2008, vol. II, ch. 1, pp. 5–22.

[12] A. M. Zaremski and J. M. Wing, "Specification matching of software components," *ACM Transactions on SEM*, vol. 6, no. 4, pp. 333–369, 1997.

[13] Y. Chen and B. H. C. Cheng, *Foundations of Component-Based Systems*. Cambridge Univ. Press, 2000, ch. A Semantic Foundation for Specification Matching, pp. 91–109.

[14] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Synthesis of Underspecified Composite e-Service bases on Automated Reasoning," in *Proc. of ICSOC04*. ACM, 2004, pp. 105–114.

[15] 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*, vol. 41, no. 2-4, pp. 207–257, 2004. [Online]. Available: http://www.kluweronline.com/issn/1012-2443

[16] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti, "Reasoning about interaction protocols for customizing web service selection and composition," *JLAP, special issue on Web Services and Formal Methods*, vol. 70, no. 1, pp. 53–73, 2007.

[17] H. Toth, "On theory and practice of assertion based software development," *Journal of Object Technology*, vol. 4, no. 2, pp. 109–129, 2005.
[18] J. Penix and P. Alexander, "Efficient specification-based component retrieval," *Automated Software Engg.*, vol. 6, no. 2, pp. 139–170, 1999.
[19] B. Fischer and G. Snelting, "Reuse by contract," in *ESEC/FSE-Workshop on Foundations of Component-Based Systems*, 1997.
[20] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, 1969.
[21] M. Baldoni, L. Giordano, A. Martelli, and V. Patti, "Programming Rational Agents in a Modal Action Logic," *AMAI*, vol. 41, no. 2-4, pp. 207–257, 2004. [Online]. Available: http://www.kluweronline.com/issn/1012-2443
[22] U. Keller, R. L. A. Polleres, I. Toma, M. Kifer, and D. Fensel, "D5.1 v0.1 wsmo web service discovery," WSML deliverable, Tech. Rep., 2004.
[23] M. Pistore, L. Spalazzi, and P. Traverso, "A minimalist approach to semantic annotations for web processes compositions." in *ESWC*, 2006, pp. 620–634.
[24] 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," in *Web Intelligence*. Springer, 2003.
[25] L. Giordano and A. Martelli, "Web Service Composition in a Temporal Action Logic," in *Proc. of 4th International Workshop on AI for Service Composition (held in conjunction with ECAI 2006)*, Riva del Garda, August 2006.
[26] A. Martelli and L. Giordano, "Reasoning About Web Services in a Temporal Action Logic," in *Reasoning, Action and Interaction in AI Theories and System*, ser. LNAI. Springer, 2006, no. 4155, pp. 229–246.

Design and Development of Intentional Systems with PRACTIONIST *Studio*

Angelo Marguglio*, Giuseppe Cammarata*, Susanna Bonura*, Giuseppe Francaviglia*
Michele Puccio*, and Vito Morreale*.
**Intelligent Systems* unit - R&D Laboratory
ENGINEERING Ingegneria Informatica S.p.A.

***Abstract*— In this paper we present PRACTIONIST *Studio*, which is an integrated design and development environment for BDI agent-based systems, providing facilities and tools to represent the concepts and intentional elements underlying such a model as well as several common features offered by UML-based tools.**

PRACTIONIST *Studio* aims at bridging the gap between the increasing trend of developing BDI-based multi-agent systems and the availability of tools for their design. It supports developers from early requirements analysis to automatic code generation.

More in detail, we first give an overview of the modelling editors provided with PRACTIONIST *Studio*. Then some fragments of the modelling and development approach when applied to a real-world implementation are presented. Such a complex system is the PSTS (PRACTIONIST Stock Trading System), which is aimed to monitor investors' stock portfolio by managing risk and profit and supporting decisions for on-line stock trading, on the basis of investors' trading rules and their risk attitude.

I. INTRODUCTION

Recently the increasing complexity and the introduction of new Web and networking technologies are making it difficult for designers to entirely model systems and for operators to handle effectively all unpredictable situations. The effort of scientific communities is towards the building of systems where interactions among components cannot be thoroughly planned and anticipated.

In other words an open issue is to investigate the modelling of systems where the collective behavior of their parts is related to the emergence of properties that can hardly, if not at all, be infered from properties of the parts. Aristotele stated that *"The whole is more than the sum of its parts"*; with this assertion he had already, more than two thousand years ago, defined what complex systems are.

Several authors (e.g. McCharty [1]) have argued that in certain situations, the so-called intentional stance [2] of systems can aid to efficiently predict, explain, or define their behaviour, without having to understand how they actually work. Therefore, some systems may be better explained in terms of mental qualities or attitudes, rather than in terms of conventional physical phenomena or design artifacts, i.e. by specifying the so-called intentional stance of systems.

In the context of the development of intentional systems, the agent-oriented approach plays a central role, due to the vast number of theories and models that have been developed for twenty years. Moreover, with regards to complex systems, Georgeff [3] asserts that *"the notions of complexity and change will have a major impact on the way we build computational systems, and that software agents - in particular BDI agents - provide the essential components necessary to cope with the real world"*.

The Belief-Desire-Intention (BDI) architecture [4] suggests that the development of agents should rely on the specification of some mental states, i.e. beliefs, desires, and intentions, which are very intuitive for people to understand. Indeed, beliefs represent information the agent has about the world; desires represent state of affairs the agent wishes to bring about and intentions are desires that it has committed to achieving.

Although BDI model has become a very attractive approach for dealing with the complexity of modern software applications, engineering such systems is still a challenge due to the lack of effective tools and actual implementations of very interesting and fascinating theories and models.

In past years we developed the PRACTONIST Framework [5], which is a set of Java libraries to develop agent-based systems according to the BDI model. PRACTIONIST adopts a goal-oriented approach and a clear separation between the deliberation and the means-ends reasoning, and consequently between the states of affairs to pursue and the way to do it. Moreover, PRACTIONIST allows developers to implement agents able to reason about their beliefs and the other agents' beliefs, expressed by modal logic formulas.

Due to the differences between the objects and agents [6], design tools used to model object-oriented systems do not represent the best way to design and develop agent-oriented softwares, especially BDI agent systems. In addition, several existing MAS modelling tools (e.g. INGENIAS Development Kit [7]) suffer a too strong tie-up with specific methodologies for the development of MAS. Moreover, some of them cover well only a subset of development phases (e.g. TROPOS Tool for Agent Oriented visual Modeling [8]). Other tools are simple prototypes and do provide a very limited assistance when developing agents and their components. In practice, none of them can be directly adopted (or extended) to design and develop multi-agent systems according to the PRACTIONIST model.

Thus we developed the PRACTIONIST *Studio*, which is the novel visual environment to model, design and develop PRACTIONIST-based systems. The PRACTIONIST *Studio*

has been developed by using several Eclipse[1] plug-ins, such as: UML2, Eclipse Modelling Framework (EMF), Graphical Editing Framework (GEF), Graphical Modeling Framework (GMF) and other Eclipse extensibility features. It supports the representation of the concepts underlying the BDI model and part of UML 2.0 meta-model as well as several features common to (commercial) well-known UML-based CASE tools, such as unified underlying model for all diagrams within an project, consistency check within diagrams, editing facilities (e.g. cut and paste, unlimited undo and redo, and so forth).

In this paper we present an overview of the modelling editors and facilities included in PRACTIONIST *Studio*, along with some fragments of modelling and development of a real system, i.e. the PSTS (PRACTIONIST Stock Trading System).

The paper is organized as follows: in section II we first present the PSTS as a running example. Then we give an overview of the PRACTIONIST suite (section III), while in section IV PRACTIONIST *Studio* is described in details; in section 4 some of the models of the PSTS developed with PRACTIONIST *Studio* are shown, while in section 5 we present how PRACTIONIST *Studio* has supported the implementation of the PSTS. Finally, we point out our intended future work and give some conclusions.

II. Running example

Systems supporting stock markets' operations and decisions are an example of systems with a high complexity. Here elementary building blocks can be individual traders, each making buying and selling decisions from his/her own perspective.

It should be noted that systems for stock trading management have been implemented by adopting agent technology and related approaches. Among them, Wang et al. [9] have presented a lightweight, distributed, intelligent agent-based financial monitoring system that monitors and reports on transactions within an organization. In such a prototype system, the intelligent agents are assisted by a formal conceptual model that makes up an unambiguous understanding of the institution, the transactions, the instruments involved, and the business processes.

In [10], Feng and Jo present a system, called AST (Agent-based Stock Trader), which is a stock-trading expert based on intelligent agents using the BDI model of agency. Finally, Davis et al. [11] have designed a system around portfolio management tasks that include eliciting user profiles, collecting information on the users portfolio position, monitoring the environment on behalf of the user, and making decision suggestions to meet the users investment goals.

The existence of such implementations confirm that agent-based systems can benefit the development of complex systems even in critical fields such as financial and stock trading. For this reason we chose to use such an application domain to test and evaluate the PRACTIONIST *Studio* by designing the PRACTIONIST Stock Trading System (PSTS), which is also used as a running examble throughout the paper.

[1]http://www.eclipse.org/

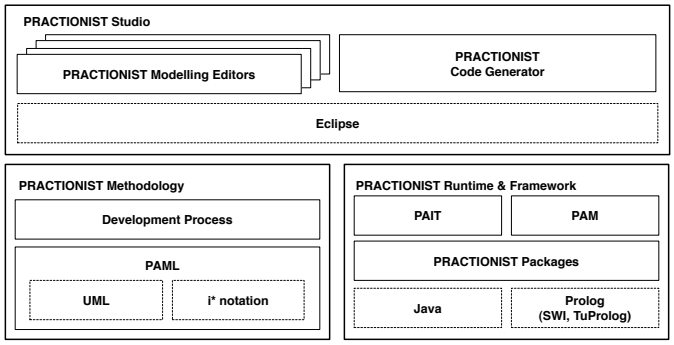


Fig. 1. PRACTIONIST Suite.

PSTS is a PRACTIONIST-based system, which is able to monitor investors' stocks portfolio, to monitor and manage risks, to manage and optimize profit and to support decisions regarding on-line stock trading, by taking into account investors' trading preferences and rules (i.e. stop loss, stop profit, profit target, tolerance, maximum budget to be inevested per week) and their degree of willingness to risk.

III. PRACTIONIST Suite

As stated above, in order to exploit the full potential offered by the agent-oriented paradigm, and particularly the BDI model, the support of efficient integrated development tools and methodologies is required to produce real-world (and sometime complex) software systems.

Our solution to this issue is PRACTIONIST (PRACTIcal reasONIng sySTem), which is an integrated suite providing the following tools (see Fig. 1):

- *PRACTIONIST Methodology*, including the *PRACTIONIST Agent Modelling Language (PAML)*, which is a UML-based modelling language, and an iterative development process;
- *PRACTIONIST runtime and framework (PRF)*, which provides the APIs to develop PRACTIONIST-based agent systems by defining the execution logic and providing the builtin components according to such a logic;
- *PRACTIONIST Studio*, a visual modelling, design and development environment supporting the representation and specification of the concepts underlying the BDI model as well as several features present in other UML-based tools.

The focus of this paper is on modelling facilities provided by PRACTIONIST *Studio*, which is described in details in the following sections, while in the remaining part of this section an overview of the other two components of PRACTIONIST is given.

A. Methodology

PRACTIONIST Methodology is based on an iterative and incremental development process supporting developers from early requirements analysis to coding, debugging and testing of agents and artefacts (according to the A2A approach [12]). It is the result of the following interacting tasks: (i) theoretical analysis of requirements that similar processes should meet,

(ii) theoretical analysis of novel features introduced with PRACTIONIST that need to be specified at the design time and (iii) practical application of PRACTIONIST in real cases as well as its integration with other technologies, such as Web services.

It should be noted that the development process is still a work in progress and our research is going towards the definition of a more general framework for process and software engineers, with the aim of providing tools to define/customize processes as well as full support to the usage of them during development phases.

As an important part of our methodology, PAML is a UML-based visual modelling language for specifying, modelling and documenting BDI multi-agent systems. Its meta-model contains general metaclasses to model intentional components of BDI agents, such as beliefs, goals and relations among them, plans and so forth. It also includes metaclasses specific to PRACTIONIST and the development of related systems.

PAML extends the Agent modelling Language (AML) [13], a semi-formal visual modelling language for specifying, modelling and documenting systems that incorporate general concepts drawn from the Multi-Agent Systems (MAS) theory. AML can be used to build models that consist of autonomous entities able to observe and interact with their environment using complex interactions and aggregated services.

Thus rather than extending the UML and building a new modelling language, PAML extends the AML, particularly for the concepts underlying the Belief-Desire-Intention (BDI) model. Indeed, AML already provides a Mental section, that lets the modelling of mental attitudes of autonomous entities having deliberative and motivational states. Moreover, PAML also extends the UML [14], in order to meet specific requirements of for developing PRACTIONIST systems.

The overall package structure of the PAML is depicted in the Figure 2.

The detailed description of PAML is out of the scope of this paper. In brief, the `Kernel` package defines the metaclasses to model artefacts, agents and their components as well as architectutal aspects of multi-agents systems. More in detail, the `Mental Attitudes` package defines the metaclasses to model intentional attitudes of PRACTIONIST agents (i.e. belief, desires, intentions, goals and plans), extending the `Mental` package of AML. The `Interactions` package defines the metaclasses to model ways and means agents use to interact with the environment where they live, including perceptors, to listen to relevant external stimuli (i.e. perceptions) and actions, to act over the environment and the effectors that actually execute such actions. The `Planning` package defines the metaclasses to model the body of plans; indeed the body represents the actual sequence of act being executed by the agent. The `BDIEntities` package defines the metaclasses to model artefects and agents, which are the building blocks of the system. Finally, the `Requirements` package defines the metaclasses to support the requirement analysis phase according to the i* notation [15] and use case model.

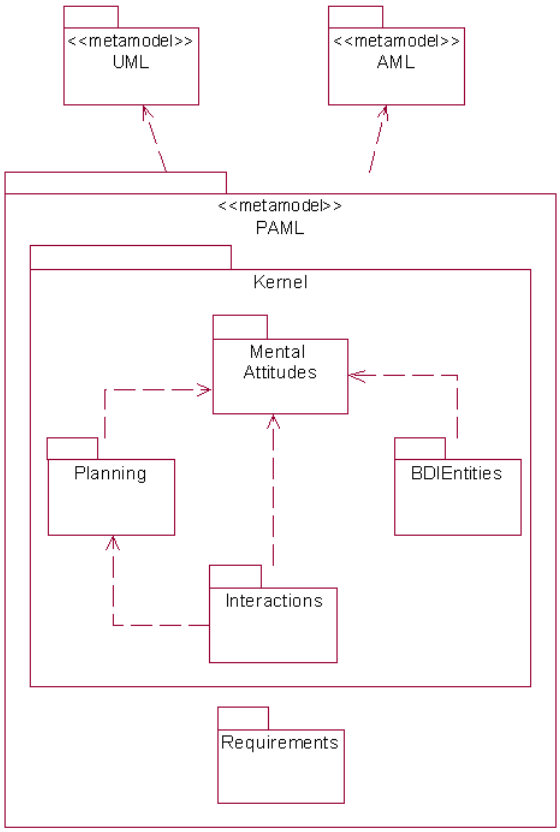


Fig. 2. Overall package structure of the PAML metamodel.

B. Runtime & Framework

As already mentioned, PRACTIONIST suite provides the framework and the runtime environment, respectively supporting the coding and the execution of BDI agents *(i)* endowed with a symbolic representation about their beliefs, *(ii)* able to proactively deliberate about their intentions, *(iii)* capable of performing reactive behaviours, and *(iv)* endowed with the ability to plan their activities in order to meet some objectives [16].

PRACTIONIST framework supplies the required built-in services that define the computational model of PRACTIONIST agents. This includes the *belief logic*, the *deliberation* mechanisms that produce agent intentions, the way the agent makes *means-ends reasoning* to figure out the means (i.e. plans) to achieve its intentions [17], and the support for the actual execution of such plans. Thus PRACTIONIST agents present a double-layered structure: the bottom layer represents the framework, which defines the execution logic and provides some built-in services implementing such a logic, while the top layer includes the specific agent components to be defined in order to satisfy specific application requirements [5].

More concretely, in order to design a PRACTIONIST agent developers shall specify the following components: *(i) Goal Model*, that is the set of *Goals* the agent could pursue and the relations among them; *(ii) Plan Library*, that is a set of

means, to pursue such goals or to react to the stimuli coming from the environment; *(iii) Perceptors* to receive such stimuli; *(iv) Actions* the agent could perform and the corresponding *Effectors*, and *(v) Belief Base*, that is a set of beliefs and rules on them to model the information about both its internal state and the external world.

Moreover, agents are endowed with the ability to dynamically build plans (i.e. *Planning*). Finally the management of perceptors and effectors is part of the agent *core services* infrastructure.

The framework also includes the PRACTIONIST Agent Introspection Tool (PAIT), a visual integrated monitoring and debugging tool, which supports the analysis of the agent's state during its execution. In particular, the PAIT can be suitable to display, test and debug the agents' mental attitudes (i.e. beliefs, desires, and intentions) and their execution flow, in terms of active behaviours. Each of these components can be observed at run-time through a set of specific tabs.

Furthermore, the runtime and framework supplies facilities and built-in components for autonomically manage PRACTIONIST-based applications and external resources.

Finally, it is worth mentioning that PRACTIONIST runtime and framework has been designed on top of JADE[2], a widespread platform compliant to the FIPA[3] specifications, that provides some core services, such as a communication support, interaction protocols, life-cycle management, and so forth.

IV. PRACTIONIST *Studio*

PRACTIONIST *Studio* is a modelling, design and development tool for BDI agent systems according to the PRACTIONIST mdoel. It includes a set of visual modelling editors, some of which are based on UML 2.0 metamodel, whereas others are based on PRACTIONIST Agent Modelling Language (PAML). More accurately, a brief description of such visual modelling editors follows:

- *i* [15] based editors*:
  - *Strategic Dependency (SD) editor*: to describe the dependency relationships among various actors in an organizational context;
  - *Strategic Rational (SR) editor*: to describe stakeholder interests and concerns and how they might be addressed by various configurations of systems and environments;
- *UML2.0 based editors*:
  - *Use Case editor*: to model use cases and system funcionalities from the actor's point of view;
  - *Class editor*: to model static structures of a system or of its parts;
- *PRACTIONIST specific editors*:
  - *Agent editor*: to model agents and specify their components;
  - *Domain editor*: to model facts about the world the agent can believe or not;
  - *Goal editor*: to model agent goals and the relationships among them;
  - *Effector/Action - Perceptor/Perception editor*: to model the means agents use to interact with their environment;
  - *Plan editor*: to model the features of plans agent can adopt to pursue their intention;
  - *Plan Body editor*: to model the body of plans, in terms of (simple or complex) flow of acts.

PRACTIONIST *Studio* aims at bridging the gap between the increasing need of development of multi-agent systems and the availability of tools for their design. Indeed, it can be used in the same way as other software modelling tools to develop multi-agent systems as it supports developers from the requirements analysis to the code generation of agents.

As many well-known CASE tools, PRACTIONIST *Studio* provide all the features that support the development of complete and consistent visual models, such as:

- *Unified model*: all diagrams created inside a PRACTIONIST project share the same model (i.e. an instance of the meta-model), whereas each generic GMF diagram file has usually its own model file.
  Sharing the same model file means sharing the same command stack, allowing us to execute cross-checks among elements and consequently model more complex and greater systems as a whole;
- *Drag and Drop* support: a PRACTIONIST project has its own model view, where the developed model is displayed as a tree. From this view it is possible to *drag and drop* the elements into diagrams, enabling us to use the same elements in different diagrams as well. Thus, if an element is modified in a diagram, it will be updated in all the other diagrams.
- *Delete from diagram* and *delete from model* actions: in a GMF diagram the *delete from model* action is enabled by default, so when an element is deleted in the diagram it is also automatically deleted from the model. Such a behaviour was modified in order to get the *delete from view* action as well, and thus have a more flexible model management.

For the development of PRACTIONIST *Studio*, the support provided by the Eclipse environment has been fully exploited. As a consequence:

- a PRACTIONIST project, which is a custom Eclipse Java project, provides several sections where developers can create their own diagrams and the source folder that will contain the generated source code;
- the model view of a PRACTIONIST project is a custom Eclipse view that displays the unified model underlying the project;
- the PRACTIONIST Java code can be generated starting from diagrams in a simple way.

The hierarchical representation of a PRACTIONIST *Studio* project is composed by several sections where developers can create their own diagrams and manage the source code generated; besides, the model view of the project is a custom Eclipse view that presents the unified model underlying the

[2]http:jade.tilab.com
[3]http://www.fipa.org

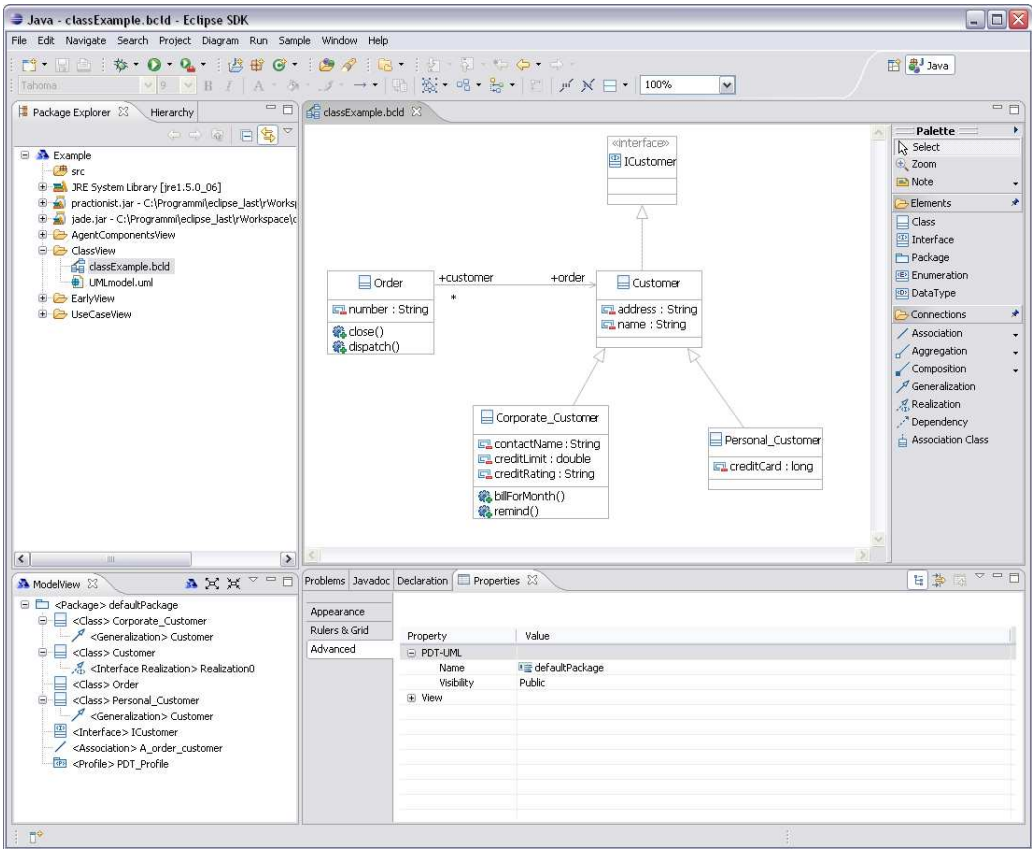


Fig. 3. A snapshot of the PRACTIONIST *Studio*.

project; finally, the Java code generation in a very simple process.

V. Modelling with PRACTIONIST *Studio*

Throughout this section we present an overview on how to model a PRACTIONIST system by using the facilities and functionalities offered by PRACTIONIST *Studio*. This is done by describing the design of some components of the PSTS and showing some snapshots of models developed with PRACTIONIST *Studio*.

As a complex system should be able to select at runtime the best behaviour on the basis of the current situation, we believe that in the requirements analysis phase, goals can be used as an abstraction to model the functions around which the systems can autonomously select the proper behaviour [17].

The requirements state that the goals of the PSTS (PRACTIONIST Stock Trading System) must be the monitoring of investors' stock portfolio in terms of risk and profit management and supplying a decision support for the on line stock trading, by considering investors' trading rules (i.e. stop loss, stop profit, profit target, tolerance, maximum budget to be inevested a week) and their degree of willingness to risk. Besides, if users so wish, the PSTS has to be able to replace orders (*system orders*) which are too risky or profitable, asking a broker to execute them. Moreover, through the PSTS users have to be allowed to place *market orders* (a market order is a buy or sell order to be executed by the broker immediately at current market prices) and *limit orders* (a limit order is an order to buy a stock at no more - or sell at no less - than a specific price).

A. Modelling the organizational environment of the PSTS

In order to provide a deeper level of understanding about how the PSTS can be embedded in the organizational environment, the relevant stakeholders of the application domain were modelled, where also the system-to-be (the PSTS) was introduced as another actor, along with the dependencies among them in terms of goals, tasks or resoures. In other words, it was created a Strategic Dependency (SD) diagram. Indeed the SD model focuses on the intentional relationships among organizational actors.

Referring to the PSTS case study, the SD diagram was modelled by using the SD editor of PRACTIONIST *Studio*, which is shown in the Figure 4, where actors, depicted as circles, are the `Investor`, the `PSTS`, `Yahoo` and the `Bank`; the dependencies among the actors are depicted as arrowed lines connected by a graphical symbol varying according to the dependum: a rectangle if the dependum is a resource, a rectangle with rounded corners if the dependum is a hard goal

More in detail, the `Investor` would like to have a system (the `PSTS`) that is able to provide information about stock market (`Get Stock Information`), to provide updated and detailed data about his/her Stock Portfolio (`Get Portfolio Information`), to manage risky and profitable stocks of the portfolio(`Default Manage Risky`

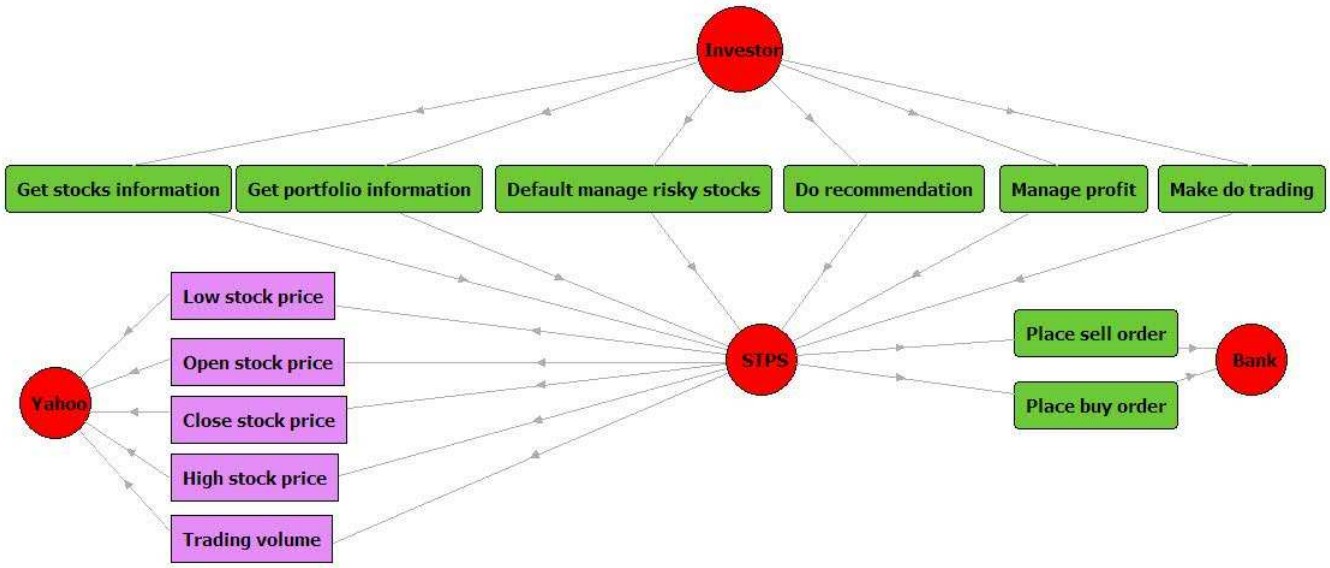


Fig. 4. Strategic Dependency model.

`Stocks` and `Manage Profit`), to give adivice on stocks to be bought or sold (`Do recommendation`) and allow to do trading autonomously (`Make do trading`). The `PSTS` depends on Internet (i.e. in particular `Yahoo`) to obtain every day the current data about stocks (opening and closing stock prices, highest and lowest stock prices and volume of trading). Finally, the PSTS depends on a broker (i.e. the `Bank`), to `Place sell order`s and `Place buy order`).

B. Modelling agents' goals

Architectural analysis of the PSTS produced the entities classified as agents. In the resulting design they are

- the `Trader`, the agent in charge of managing all kinds of order (i.e. market, limit and system orders) by asking the broker (the `Bank`) to place them;
- the `Analyst`, the agent liable for executing the market analysis;
- the `Advisor`, the agent which interprets the `Analyst`'s signals and does recommendations to investors;
- the `HoldingStockManager`, the agent which monitors investor's stock current prices and places sell order for that stocks resulting too profitable (if they have reached the profit target indicated by investor or the profit has descended below the stop profit of investors) or too risky (if their value has descended below the stop loss of investors).

It is worth noting that the goal turns out to be an interesting abstraction related to autonomous entities for the development of software systems whose requirements are not entirely known at design time. Thus, the explicit representation of goals and the ability to reason about them from agents, plays an important role in the modelling phase. By using PRACTIONIST *Studio*, a designer can specify for each agent, goals it could pursue and their properties, and all relations among such goals.

More in detail, for each goal it is possible to define the success condition, the applicability condition stating whether it is possible (given current conditions) to achieve that goal and the cancel condition stating in which situations the agent should give up to pursue a goal.

Regarding the relationships among goals, PRACTIONIST *Studio* allows to model *(i)* the inconsistency between two goals (if the designer want to declare that if a goal succeds, the other one fails), *(ii)* the entailment (if the designer want to declare that if a goal succeeds, then also the other one succeeds), *(iii)* the fact that a goal is a precondition of another goal (that is the fact that a goal must succeed in order to be possible to pursue another goal), *(iv)* the dependence (if the designer want to declare that a goal is precondition of another goal and must be successful while pursuing this last one). A formal definition of the goal relationships in PRACTIONIST can be found in [17].

For example, referring to the `HoldingStockMaganer`, the properties of goals and their relationships were modelled in PRACTIONIST `Studio` as in Fig. 5

The main objective of the `HoldingStockMaganer` is `ManageHoldingStock`: it is *applicable* if the agent believes that a new current price is available for one of the holding stocks (`NewStockPriceReceived` predicate), or that the investor has bought a new stock (`NewStockBought` predicate); indeed in the last case, the agent has manage such a new stock.

The agent monitors both the profit and the risk of a stock, so the `ManageHoldingStock` depends on the `ManageRiskyStock` and the `ManageProfitableStock` goals; the agent continues to pursue these goals until the price and the amount of a stock do not change, otherwise both goals have to be cancelled.

To manage the risk and the profit of a stock, the agent has to analyze its risk and profit on the basis of investors' rules, so the goals `ComputeRisk` and `ComputeProfit`

Fig. 5. Goal Diagram related to the *HoldingStocksManager* agent.

entail respectively the goals `ManageRiskyStock` and `ManageProfitableStock`.

Finally, the dependency relationships of the `ComputeProfit` and `ComputeRisk` goals were modelled; the `dependee` goals have to be achieved to compute the value of some investor's trading rules, that is the `profit target`, the `stop profit`, and the `stop loss`.

C. Modelling agents' plans

In the BDI agent model, another key element is the library of plans, as it represents the set of *recipe* to meet agent's intentions.

In PRACTIONIST *Studio* it is possible to declare a set of plans an agent has to own (the *plan library*), to specify the activities it should undertake in order to achieve its intentions, or handle incoming perceptions, or react to changes of its beliefs.

Each plan presents five slots: (i) *practical*, which defines the kind of events the plan is able to manage; (ii) *applicable*, to define the formula that has to be believed as true by the agent in order to actually adopt a practical plan; (iii) *invariant*, to define the condition to hold during the execution of the plan; (iv) *cancel*, to define when the plan has to be stopped with failure; (v) *success*, to define the formula has to believed as true by the agent then the plan ends with success.

But, the way a certain event is handled has to be specified in the body, which is an activity that can contain a set of *acts* ([17]), such as desiring to pursue some goal, adding or removing beliefs, sending ACL messages, doing an action and so forth.

Thus, in order to model the plan's body, a designer can use the Plan Body editor of PRACTIONIST *Studio*. In Fig. 6 it is shown the Plan Diagram where the plan library of the *HoldingStocksManager* was modelled.

The *HoldingStockManager* agent has to manage the profit and the risk of a holding stock every time a new price is available or the investor has placed a new buy order for a stock already held, or a new stock is bought. Therefore it was equipped with the *ManageProfitForNewOrder*, *ManageProfitForNewPrice* plans, regarding the profit management, and *ManageRiskForNewOrder* and *ManageRiskForNewPrice* plans, regarding the risk management; as shown in Fig. 6. Success and cancel conditions of these plans refer to the goal success and cancel conditions, whereas they have a proper applicable condition.

Finally, other plans were also modelled, for example to handle the stimuli received from the environment (i.e. a stock price updating or a stock placed order) and to compute the investor's trading rules (that is, to manage the *ComputeStopLoss* goal, etc.).

VI. Coding with PRACTIONIST *Studio*

As stated, PRACTIONIST *Studio* supports the actual implementation of BDI agent systems by providing an automatic code generation facility, which produces template or partially filled parts of source code according to the developed models and relying on the PRACTIONIST Framework.

In this section the source code related to the *ManageHoldingStock* goal generated by PRACTIONIST *Studio* is shown. A snippet of the goal follows:

```
/**
 * @generated
 */
public class ManageHoldingStock implements Goal
{
   ....

   /**
    * @generated
    * @see org.practionist.core.GoalProfile#applicable())
    */
   public boolean applicable()
   {
        // TODO: Insert the right variables value
       return
       beliefBase.bel(AbsPredicateFactory.create
           ("newStockPriceReceived(arrived: X)"))
        || beliefBase.bel(AbsPredicateFactory.create
```

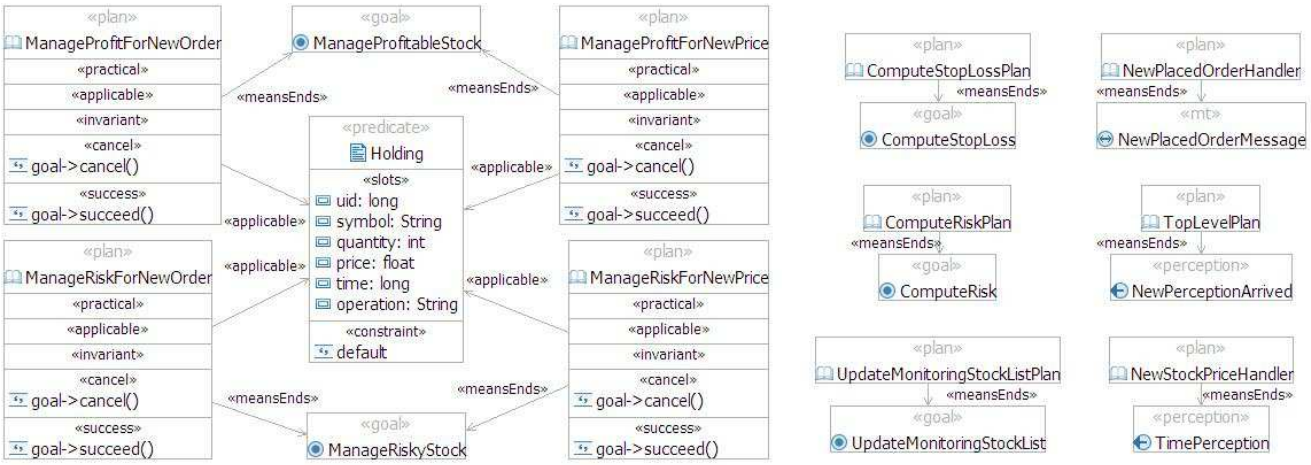


Fig. 6. Plan Diagram related to the *HoldingStocksManager* agent.

```
        ("newPlacedOrderReceived(uid: X, stockSymbol: X,
            operation: X, quantity: X, price: X)"));
   }

   /**
    * @generated
    *
    * @see org.practionist.core.GoalProfile#succeed())
    */
   public boolean succeed()
   {
      // TODO: Insert the right variables value
      return beliefBase.bel(AbsPredicateFactory
         .create("managed(investor: X, stockName: X)"));
   }
   ....
}
```

This general implementation produced by the code generator of PRACTIONIST *Studio* should be customized according to the specific requirements for this goal. An example follows:

```
/**
 * @generated
 */
public class ManageHoldingStock implements Goal
{
   .....

   /**
    * @generated
    * @see org.practionist.core.GoalProfile#applicable())
    */
    public boolean applicable()
    {
       return beliefBase.bel(AbsPredicateFactory
          .create("newStockPriceReceived(arrived: true)"))
          || beliefBase.bel(AbsPredicateFactory
             .create("newPlacedOrderReceived
                (uid: %, stockSymbol: %,
                operation: %, quantity: %, price: %)",
                uid, stockSymbol,
                operation, quantity, price));
   }

   /**
    * @generated
    *
    * @see org.practionist.core.GoalProfile#succeed())
    */
    public boolean succeed()
    {
       return beliefBase.bel(AbsPredicateFactory
          .create("managed(investor: %, stockName: %)",
            investorID, symbol));
    }
    ....
}
```

In this snippet, the designer just needs to detail the right variables of the predicates (in this example a parametrized form of the predicates has been adopted, using the symbol % and then adding values). That is, in order to express the applicability and success conditions of the goal, the corresponding beliefs were customized by replacing the aforementioned variables with the values that characterise the goal.

A code snippet of the dependency relation between the `ManageHoldingStock` and the `ManageProfitableStock` goals follows:

```
/**
 * @generated
 */
public class GR_ManageHoldingStock_ManageProfitableStock
          implements DependencyRel
{
   public Goal verifiesRel(SerializableGoal goal1,
        SerializableGoal goal2)
   {
      if (goal1 instanceof ManageHoldingStock
         && goal2 instanceof ManageProfitableStock)
            return new ManageProfitableStock();
      return null;
   }
}
```

In this example, every `ManageHoldingStock` goal depends on the `ManageProfitableStock` goal, without specifying any information about stocks. Thus, this code should be now customized according to the designer's needs:

```
/**
 * @generated NOT
 */
public class GR_ManageHoldingStock_ManageProfitableStock
      implements DependencyRel
{
   public Goal verifiesRel(SerializableGoal goal1,
      SerializableGoal goal2)
      {
         if (goal1 instanceof ManageHoldingStock
            && goal2 instanceof ManageProfitableStock)
            {
               ManageHoldingStock g =
                   (ManageHoldingStock) goal1;
               return new
                 ManageProfitableStock(g.getUID(),
                             g.getSymbol());
            }
            return null;
      }
}
```

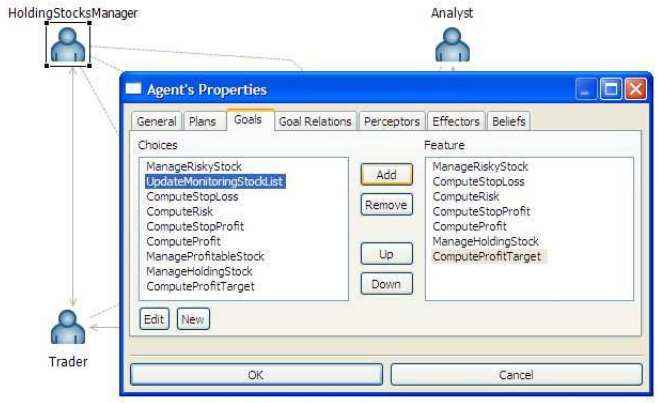


Fig. 7. Agent Diagram related to the *HoldingStocksManager* agent.

In this snippet, the designer has just to detail some properties of the dependee goal; here the dependee goal refers to the same user and symbol of stock of the dependent goal.

Finally, a code snippet of the HoldingStockManager agent class is shown:

```
/**
 * @generated
 */
protected void initialize()
{
   addBeliefSet("/home/pl/holdingstockmanager.pl");

   /****************Goals****************/
   // TODO:Remember to put the goal's parameters here
   registerGoal(new ManageHoldingStock(), "");
   registerGoal(new ManageProfitableStock(), "");
   registerGoal(new ComputeProfitTarget(), "");

   /***********Goals**Relations***********/
   // TODO:Remember to put the relation's parameters here
   registerRelation(new
       GR_ManageHoldingStock_ManageProfitableStock(), "");
   registerRelation(new
       GR_ManageRiskyStocks_ComputeRisk(), "");

   /****************Plans****************/
   // TODO:Remember to put the plan's parameters here
   addPlan(ManageRiskyStock.class, "ManageRiskyStock");
   addPlan(NewStockPriceHandler.class,
       "NewStockPriceHandler");
}
```

The Fig. 7 shows a snapshot of the Agent Diagram. More in detail, by means of such a diagram it is possible to look at the list of all the intentional elements were modelled and to choose that ones that the designer wants associate to an agent.

It is worth noting that the code generator of PRACTIONIST *Studio* is able to produce the agent source code by putting the entities which were designed in the previous modelling phases together; obviously, this code should be customized according to the designer's needs.

VII. Conclusions and Future Work

In this paper, through a running example, i.e. the PSTS, we presented PRACTIONIST *Studio*, the visual modelling, design and development environment of the PRACTIONIST suite, which is the set of tools we have been developing to implement agent systems according to the BDI model.

PRACTIONIST *Studio* supports the development of agents endowed with a lot of useful built-in capabilities and with a computational model which is more flexible and adaptive than the agent models underlying several commercial and non-commercial frameworks.

Our tool allows the design and development of BDI agent systems from several perspectives, including the representation of intentional attitudes and relationships among them, the way the agents interact with their environment, the activities within a plan. and so forth.

As part of our future work, we aim at further developing PRACTIONIST *Studio* by adding editors for other diagrams (i.e. dynamic views, such as interactions). We also intend to improve service features of the tools, such as reverse engineering and documentation management and automatic generation.

Finally, we have been developing some other real-world applications by using the PRACTIONIST framework, methodology and *Studio*.

References

[1] J. McCarthy, "Ascribing mental qualities to machines," Stanford University, Tech. Rep. STAN-CS-79-725, 1979.

[2] D. Dennett, *The Intetional Stance*. MIT Press, 1989.

[3] M. P. Georgeff, B. Pell, M. E. Pollack, M. Tambe, and M. Wooldridge, "The belief-desire-intention model of agency," in *ATAL '98: Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*. London, UK: Springer-Verlag, 1999, pp. 1–10.

[4] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-architecture," in *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, pp. 473–484. [Online]. Available: http://citeseer.nj.nec.com/rao91modeling.html

[5] V. Morreale, S. Bonura, G. Francaviglia, F. Centineo, M. Puccio, and M. Cossentino, "Developing intentional systems with the practionist framework," in *Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN07)*, July 2007.

[6] O. J., "Objects and agents: how do they differ?" *Journal of Object-Oriented Programming*, pp. 50–53, 2000.

[7] J. Pavon, C. Sansores, and J. J. Gomez-Sanz, "Modelling and simulation of social systems with ingenias," *Int. J. Agent-Oriented Softw. Eng.*, vol. 2, no. 2, pp. 196–221, 2008.

[8] D. Bertolini, A. Perini, A. Susi, , and H. Mouratidis, "The tropos visual modeling language. a mof 1.4 compliant meta-model." *Contribution for the AOSE TFG meeting.*, 2005.

[9] H. Wang, J. Mylopoulos, and S. Liao, "Intelligent agents and financial risk monitoring systems," *Commun. ACM*, vol. 45, no. 3, pp. 83–88, 2002.

[10] X. Feng and C.-H. Jo, "Agent-based stock trader," in *Computers and Their Applications*, 2003, pp. 275–278.

[11] D. N. Davis, Y. Luo, and K. Liu, "Combining kads with zeus to develop a multi-agent e-commerce application," *Electronic Commerce Research*, vol. 3, no. 3-4, pp. 315–335, 2003.

[12] A. Ricci, M. Viroli, and A. Omicini, "Programming MAS with artifacts." in *PROMAS*, 2005, pp. 206–221.

[13] I. Trencansky and R. Cervenka, "Agent modelling language (aml): A comprehensive approach to modelling mas," *Informatica*, vol. 29, pp. 391–400, 2005.

[14] O. M. Group, "Unified Modeling Language, Superstructure."

[15] E. S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," pp. 226–235. [Online]. Available: citeseer.ist.psu.edu/article/yu97towards.html

[16] V. Morreale, S. Bonura, G. Francaviglia, M. Cossentino, and S. Gaglio, "PRACTIONIST: a new framework for BDI agents," in *Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS'05)*, Brussels, Belgium, 2005, p. 236.

[17] V. Morreale, S. Bonura, G. Francaviglia, F. Centineo, M. Cossentino, and S. Gaglio, "Goal-oriented development of BDI agents: the PRACTIONIST approach," in *Proceedings of Intelligent Agent Technology*. Hong Kong, China: IEEE Computer Society Press, 2006.

Arguments and Artifacts for Dispute Resolution

Enrico Oliva
Mirko Viroli
Andrea Omicini
DEIS, ALMA MATER STUDIORUM—Università di Bologna
via Venezia 52, 47023 Cesena, Italy
E-mail:{enrico.oliva,mirko.viroli,andrea.omicini}@unibo.it

***Abstract*—In a social context cultural differences, individual interests, and partial awareness are often the causes of disputes. Alternative Dispute Resolution (ADR) is usually considered to be alternative to litigation, and can also be used to allow disputing parts to find an agreement. A dispute resolution is not an easy task and usually involves more entities including mediator or arbitrator with multiple dialogue sessions.**

In the paper we focus the attention on dispute resolution system in artificial society proposing a model and a technology to support the persuasive processes. The persuasion is the principal form dialogue used in an ADR system where agents exchange arguments to support their positions.

The general architecture proposed to build an ADR system exploits two artifacts abstractions – Co-Argumentation Artifact and Dialogue Artifact – that provide the right abstractions to coordinate the agents during the argumentative process. The technological support for the artifacts is provided by the TuCSoN infrastructure, also exploiting a meta-programming technique in Prolog. Finally, in the paper we present a simplified example of the execution of a persuasion dialogue ground on the commitments.

I. ALTERNATIVE DISPUTE RESOLUTION

People develop systems and methods in order to settle conflicts in a fair way. Human societies define norm systems, infrastructure (such as court) and methods (such as trial) to achieve the dispute resolution.

In a global business process scenario there is a increasing need of speed-up the processes, and to make faster the conflict resolution. The new systems have to support legal process when for instance a negotiation is broken, they have to combine mediation and legal service to avoid litigation.

Alternative Dispute Resolution (ADR) is usually considered to be alternative to litigation. It also can be used as a colloquialism for allowing a dispute to drop or as an alternative to violence. ADR is generally classified into at least four subtypes: negotiation, mediation, collaborative law, and arbitration. Walker and Daniels [1] underline that legal negotiation is a part of traditional dispute resolution system rather than a component of the ADR movement. The legal negotiation directly occurs among agents that represent the disputants in a context similar to a courtroom.

Arguments have a central role in the process of formalising legal system, and in the trial, too. The paper [2] contains a survey of logic in computational model on legal argument. The authors present the main architecture of legal arguments with a four layer architecture: 1) logical layer, 2) dialectical layer, 3) procedural layer, and 4) strategic layer. Disputants use arguments in order to persuade the other parts of the dispute and also the decision makers—juries, judges, clients and attorneys. In [3] the use of arguments in an ADR systems is considered, and an analysis of arguments in different contexts such as arbitration, mediation and multi-party facilitation is presented. Argumentation plays an important role in conflict resolution systems, where it drives the ADR to obtain a successful solution of the dispute. The argumentation process promotes the values of justice, equality and community that are desirable in a dispute resolution system.

In an open agent society, the same issue as in human society holds: it is undesirable to resolve dispute by litigation. The development of a system for internal resolution of disputes in virtual organisations is presented by Jeremy Pitt et al in [3], which proposes a norm-government MAS and an ADR protocol specification for virtual organization exploited by intelligent agents.

ADR supplies a theoretical bases for Online Dispute Resolution (ODR) as defined in [4]. ODR has the purpose to extend the ADR process, moving it towards virtual environments while providing computation and communication support. In ODR, the role of technology used to facilitate the resolution of disputes between parties is crucial. It provides a structured communication, as well as an informed environment that helps to the successful conclusion of the conflict.

ODR could be seen as an instance of an ADR system, with a communication infrastructure and Artificial Intelligence (AI) techniques aiming at supporting the parties toward agreements. The reasoning and argumentation capabilities of the parties are achieved by exploiting AI methods.

Walton and Godden [5] show that argument-based dialogue, in particular persuasion dialogue, contributes to the construction of effective dispute resolution system. The main type of dialogue usually considered by ADR is negotiation, which could be interpreted as a particular sort of communication for the purpose of persuasion. In argumentation theory both types of dialogues are present: persuasion dialogue and negotiation dialogue. These two types of dialogue have a different structure and different goals, and in the context of ODR systems should be managed by different procedural rules.

A fundamental problem in ODR and ADR systems is that it is difficult to structure and process the information exchanged between negotiating parties. In order to resolve this problem

in this work we propose to build a ADR system based on the A&A meta-model [6] with Co-Argumentation Artifact (CAA) [7] and Dialogue Artifact (DA) [8] abstractions. We aim at providing a framework for conflict resolution in an agent-based society supplying a supporting infrastructure in order to manage arguments, to retrieve information and to bargain.

Our framework provides structured information based on logic tuple along with the control of dialogue processes through a mediated form of communication over a programmable infrastructure. These two features are useful in order to build MAS in a scalable and flexible architecture, and also to build ADR that supports multi-party dialogue sessions.

The aim of the work is to provide a more formal (functional) connection among the two types of argumentation artifacts CAA and DA in order to support a dialogue for dispute resolution. In particular we make explicit a set of functionalities useful during the dialogue to control the relation with the argumentative commitment store. For the CAA we collect a list of operations to manage a commitment store based on the argumentation system. On the other hand for the DA we describe by operational semantics the use of the CAA operations during the dialogue.

The result is a powerful architecture where it is possible to specify a dialogue grounded on the state of the commitment store enabling a partially automate dialogue execution through DA and CAA infrastructure. Using that dialogue specification the DA can automatically drive the sequence of actions based on the state of the CAA.

In Section II we introduce the architecture of the framework with the definition of the specific CAA and DA; in Section III we explain the argumentation and dialogue system by introducing the new operators to describe the interaction with the commitment store; finally in Section IV we present the case study, implementing a persuasion dialogue protocol.

II. Architecture

We propose our architecture for MAS based on A&A meta model to design a ADR/ODR application. An ADR system, especially on-line, exploits the forms of negotiation, arbitration or mediation required to achieve a solution. There, typically, the entities involved are more than two: at least, two participants and a third entity to help the dispute resolution such as in mediator and arbitrator procedure. The parties involved choose the procedure, terms and conditions of their dispute. For instance, in [3] an arbitration protocol is presented, along with concepts for decision making through formation and voting protocol.

In order to find a solution, the parties have the possibility to share any pertinent argument, make demands and evaluate the acceptability of an argument with respect to normative context. To do that, a multi-party dialogue protocol is required, and also an impartial computation over the shared knowledge. When the dispute involves an increasing number of participants it is necessary to introduce a mediated form of communication in order to have a scalable system. The essential point, here, is that in the act of mediating there are a number of evaluations that could be done automatically.

In that scenario our architecture provides the required abstractions: *(i)* Dialogue Artifact, *(ii)* Co-Argumentation Artifact to made a flexible system. In the DA we store the arbitration, mediation or negotiation protocol. The parties exploit the DA to take part of the discussion, which drives the dialogue ground on the commitments. The advantages are: the management of dialogue between multiple entities, and the automatic interaction with commitment/argument store. The CAA provides the right abstraction to made a commitment/argument store where it is possible evaluate automatically the argument validity respect of normative context. Also, it provides default function to exchange information, data and arguments, and to record their public commitments in private or public form. For instance, in a bargain among three or more entities handled by a CAA, the final set of arguments stored in the CAA during the bargain represent a form of contract among the parties.

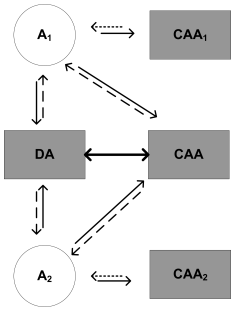


Fig. 1. General architecture of multi-agent argumentation system

A possible architecture for a Multi-agent argumentation system is shown in Figure 1 where A_1 and A_2 represent two rational agents. The suggested architecture exploits both local and global DA and CAA. The global CAA and DA provide services and functionalities for the entire agent society. Ideally, in the model, DA and CAA are separate entities with different and orthogonal functionalities. However, in an actual implementation, both shared artifacts could collapse in one unique global entity without loss of generality. The local CAA_1 and CAA_2 are used by agents in order to coordinate their mental state. Classically, those functions are provided by an internal argumentation component hidden inside the agent.

In the following, we focus our attention on the persuasion dialogue that is among the most common and useful dialogue in ADR. An interesting observation in [5] put in evidence the fact that a negotiation dialogue could naturally include or shift to persuasion dialogue in two points: 1) to follow an offer, and 2) to follow a rejection of an offer. In both cases reasons (by argument) are provided to prove the (un)acceptability of an offer. A dialogue model for persuasion could be composed of: 1) a commitment store for each participant, 2) an inference rule to draw conclusion from commitments in the commitment store made by the participant, and 3) practical rules that govern the sequence of locutions and their consequence. We

foresee that our architecture provides the desired abstraction and properties to implement the persuasion dialogue with agreement purpose in an agent society.

A. Co-Argumentation Artifact

The CAA provides co-ordination services to agents, allowing them to share, store and exchange arguments with one another working as a commitment store. In particular, for persuasion dialogue we exploit the ability of the CAA to automatically calculate argument and belief acceptability according to the agent attitudes and the argumentation semantics. In [9] are introduced agent attitudes in order to provide some acceptability criteria. An agent may have one of three acceptance attitudes about proposition: *(i)* a *credulous* agent can accept any formula for which there is an argument S; *(ii)* a *cautious* agent can accept any proposition for which there is an argument if no stronger rebutting argument exists; *(iii)* a *skeptical* agent can accept any proposition for which there is an acceptable argument S.

Exploiting our argument definition 1 and referring to our argumentation system in section III-A, we resolve the argument acceptance problem following the preferred semantics on argumentation: an argument is credulous acceptable if it belongs to some preferred extension; an argument is skeptical acceptable if it belongs to all preferred extensions.

The CAA validates the argument committed verifying their correctness and also it evaluates their acceptability verifying which of preferred sets belong to. The state of a CAA is represented by a collection of arguments with also the related list of conflict free sets, admissible sets and preferred extension. These sets are update for each argument insertion or removal.

In tuple notation these sets are expressed by a tuple name like `conflictfree`, `admissible` and `preferred` and a parameter composed by a list of lists of argument names indicated by `arg1,...,argN` i.e. `conflictfree([[arg1,...,argN],...])`. In particular for persuasion dialogue the CAA supports the agent `credulous` and `skeptical` attitudes, calculating in which argument set the reference argument belong to. Here, we list a set of operation provided by the CAA where Arg parameter means a generic input argument:

- $acceptable(Arg, Attitude)$: the CAA verifies the acceptance of the argument Arg respect of state of the commitment store with the type of acceptability specified by $Attitude$
- $read(ArgTemplate)$: the CAA returns an argument that logically unifies with $ArgTemplate$
- $conflict(Arg)$: the CAA verifies the existence of an argument $\in$ CAA in rebuttal relation with Arg (see section III-A)
- $attack(Arg)$:the CAA verifies that Arg is in undercut relation with an argument $\in$ CAA (see section III-A)
- $defeat(Arg)$ the CAA verifies the existence of an argument $\in$ CAA in undercut relation with Arg
- $remove(Arg)$: the CAA removes the argument Arg
- $commit(Arg)$: the CAA stores the argument Arg and it recompute the conflict free sets, the admissible sets and the preferred extensions

For further details, including an implementation and examples of this argumentation framework, we refer to the paper [10].

B. Dialogue Artifact

The DA is the abstraction to encapsulate the rules of dialogue and it coordinates the entities during persuasion process. We follow the definition provided of this artifact in [8] where the DA is composed of three components: a collection of specifications of dialogue protocols; a collection of commitments stores; and a collection of specifications of interaction control. Basically, the function of the DA is to drive the agents general type of dialogue keeping trace in the commitments stores of the partial results of the communication. Moreover the DA keep in charge to suggest of the agents the possible right moves constrained by the state of the commitment store. Here a list of operation provided by DA:

- *nextlocutions([L])*: the DA provides the list of possible locutions
- *lastlocution(L)*: the DA provides the last locutions
- *state(S)*: the DA provides the protocol state
- *act(L)*: the DA store the locution L and updates the state of protocol
- *cs(A)*: the DA executes an action A over the commitment store

From a general architecture point of view the commitment store of DA is provided by CAA correctly implemented and the global state of the system is represented by the state of the CAA and the state of the protocol in the DA.

III. Argumentation & Dialogue System

In order to achieve agreement among agents a common dialogue system and a shared argumentation system are required in the agent society. Following, we propose a formalization of both systems.

A. Argumentation System

Our reference argumentation system is introduced in detail in [7] as an extension of the Dung's framework [11] with the definition of the structure inside the arguments. Here we report briefly the argument definition and the object language.

The object language of our system is a first-order language, where Σ contains all well-formed formulae. The symbol $\vdash$ denotes classical inference (different styles will be used like deduction, induction and abduction) $\equiv$ denotes logical equivalence, and $\neg$ or non is used for logical negation.

Definition 1 (argument): An *argument* is a triple $A = \langle B, I, C\rangle$ where $B = \{p_1, \ldots, p_n\} \subseteq \Sigma$ is a set of *beliefs*, $I \in \{\vdash_d, \vdash_i, \vdash_a\}$ is the inference style (respectively, *deduction*, *induction* or *abduction*), and $C = \{c_1, \ldots, c_n\} \subseteq \Sigma$ is a set of *conclusions*, such that:

1) B is consistent
2) $B \vdash_I C$

Deductive Inference		Inductive Inference	
MP	$\frac{A \quad A \to B}{B}$	θ-su	$\frac{B}{R} \quad where \quad R\theta \subseteq B$
MT	$\frac{\neg A \quad B \to A}{\neg B}$	Abductive Inference	
MMP	$\frac{B_1,\ldots B_n \quad (B_1,\ldots B_n) \to C}{C}$	Ab	$\frac{B \quad A \to B}{A}$

TABLE I
DEDUCTIVE INFERENCE: (MP) MODUS PONENS, (MMP) MULTI-MODUS PONENS AND (MT) MODUS TOLLENS. INDUCTIVE AND ABDUCTIVE INFERENCE: (θ-SU) θ-SUBSUMPTION, (AB) ABDUCTIVE

3) B is minimal, so no subset of B satisfying both 1 and 2 exists

The types of inference I we consider for deduction, induction and abduction are shown in Table I. Modus Ponens (MP) is a particular case of Multi-Modus Ponens (MMP) with only one premise. The inference process θ-subsumption derives a general rule R from specific beliefs B, but is not a legal inference in the strict sense.

Definition 2 (contrary): The contrary (or attack) relation R is a binary relation over Σ that $\forall p_1, p_2 \in \Sigma$, $p_1 R p_2$ iff $p_1 \equiv \neg p_2$.

For defeat of arguments there are two possible types of attack based on the contrary relation: 'conclusions against conclusions', called *rebuttals*, and 'conclusions against beliefs', called *undercuts*.

Definition 3 (undercut): Let $A_1 = \langle B_1, I_1, C_1 \rangle$ and $A_2 = \langle B_2, I_2, C_2 \rangle$ be two distinct arguments, A_1 is an *undercut* for A_2 iff $\exists h \in C_1$ such that hRb_i where $b_i \in B_2$.

Definition 4 (rebuttal): Let $A_1 = \langle B_1, I_1, C_1 \rangle$ and $A_2 = \langle B_2, I_2, C_2 \rangle$ be two distinct arguments, A_1 is a *rebuttal* for A_2 iff $\exists h \in C_1$ such that hRc_i where $c_i \in C_2$.

The definitions of conflict-free set, admissible set, preferred extension are the basic ones in our argumentation system. These sets are composed of arguments that together feature different kinds of properties like absence of conflicts or common defence, formally introduced in [11].

We consider also important argument extensions such as acceptability in order to determine whether a new argument is acceptable or not. An argument is acceptable in the context of preferred semantics if an argument is in some/all preferred extensions (credulous/skeptical acceptance).

B. Dialogue System

Our intention here is to capture the rules that govern legal utterance and the effect of the utterances on the commitment store of the dialogue. We use a process algebra approach to represent the possible paths that a dialogue may take, and to represent explicitly the operations to and from the commitment store. Our *communication language* is a set of locutions L_c where a locution is a expression of the form $perf_{name}(Arg_1, \ldots, Arg_n)$ in particular $perf_{name}$ is a performative and Arg_x is either a fact or an argument. An agent performing a dialogue using the communication language can utter a locution composed of facts and arguments. An argument is represented with the tuple `argument(B,I,C)`; also a fact is considered an argument but with an (true) implicit premise and it is represented by syntax `argument(true,I,C)`.

Definition 5 (action): An *action* A is defined by the syntax $A ::= s : L_c | s[t_1, \ldots, t_n] : L_c$ where s indicates the source, and $[t1, \ldots, t_n]$ indicates the (optional) targets of the message. On the other, beyond this, we include additional atomic operations K over commitment stores—many of them can actually occur into one argumentation artifact.

Definition 6 (term action): A *term action* K has the syntax $K ::= commit(C, X) | read(C, X) | conflict(C, X) | attack(C, X) | defeat(C, X) | acceptableS(C, X) | acceptable(C, X)$, where C is a term representing the commitment store identifier, and X is a term representing the commitment.

A protocol P specifies by standard process algebra operator $(., +, \|)$ respectively sequence, parallel and choice, the set of actions and term actions that the agents and DA might execute. For example, an abstract dialogue protocol definition is given by $D := s : a_1 \parallel s : a_1 \parallel s : a_1 \parallel t : a_2 \parallel t : a_3$ where agent s invokes a_1 three times, agent t can invoke a_2 and a_3 only once, but in whichever order. For more detailed protocol definition with the process algebra approach and the related operational semantic we refer to the work [8].

Enriching the previous work we augment the set of K term actions in order to clarify the relations with a commitment store based on arguments. The behaviour of term actions is defined by operational semantics. This semantics describes the evolution over time of the dialogue state and the states of commitment store (seen as the composition of all commitment stores). In essence, the commitment store is the knowledge repository of the dialogue as a whole, and it is expressed in our framework as a multiset of arguments.

Definition 7 (commitment store): A commitment store C is a multiset of arguments and it is defined by the syntax $C ::= 0 | (C|C) | X$ where X is a argument, and 0 is the empty set.

We use also notation $t\{x/y\}$, to mean term t after applying the most general substitution between terms x and y—x should be an instance of y, otherwise the substitution notation would not make sense. Finally, we define the semantics of K operation that describe the interaction and evolution over time of the commitment store C in function of protocol P:

$$(C)commit(x).P \xrightarrow{\tau} (C'|x)P \quad (1)$$

$$(C|x)read(y).P \xrightarrow{\tau} (C|x)P\{x/y\} \quad (2)$$

$$(C|x)remove(y).P \xrightarrow{\tau} (C)P\{x/y\} \quad (3)$$

$$(C|x)conflict(y)).P \xrightarrow{\tau} (C|x)P \text{ if } \{x \text{ rebuttal } y\} \quad (4)$$

$$(C|x)attack(y)).P \xrightarrow{\tau} (C|x)P \text{ if } \{y \text{ undercut } x\} \quad (5)$$

$$(C|x)defeat(y)).P \xrightarrow{\tau} (C|x)P \text{ if } \{x \text{ undercut } y\} \quad (6)$$

$$(C|\mathcal{E})acceptS(y).P \xrightarrow{\tau} (C|\mathcal{E})P \text{ if } \{\forall E \in \mathcal{E}, y \in E\} \quad (7)$$

$$(C|E)acceptable(y).P \xrightarrow{\tau} (C|E)P \text{ if } \{y \in E\} \quad (8)$$

As usual, we write $s \to is'$ in place of $\langle s, i, s' \rangle \in \to$, meaning the dialogue system moves from state s to s' due to interaction i—either an action a, or an internal step τ (an operation over the commitment store).

Rule (1) provides the semantic of `commit` operation, expressing that x term is added to the commitment store C,

and the state of commitment store is updated recalculating conflict free set and preferred extensions after that the process continuation can carry on. Rules (2) and (3) to read and remove terms from commitment store C: the use of substitution operator guarantees that the term x in the commitment store is an instance of the term x to be retrieved. Rules (4), (5) and (6) provide the semantics for attack, conflict and defeat relations using the standard definition of undercut and rebuttal. Finally, rules (7) and (8) express the semantics for acceptable operators for skeptical(7) and credulons(8) acceptance, where $\mathcal{E}$ is the set of all preferred extension E in the commitment store C.

IV. Persuasion Dialogue Application

In persuasion dialogue the goal of a participant is to prove his/her thesis and to rationally persuade the other parties. With the word "persuasion" we mean not a psychological persuasion but rather a rational persuasion supported by arguments. Walton & Krabbe [12] observe that disputes is a subtype of persuasion dialog—where the parties disagree about a single proposition φ. So, for instance, at the beginning of the dialogue a party beliefs in φ while the other belief in $\neg\varphi$, so they have a contrary opinion about a proposition. Generally the following moves are allowed in the dialogue: asking question, answering question, and putting forward arguments. Following Walton [5], a proponent in a persuasion dialogue has successful when: 1) the responded has committed all the premises of the argument 2) each argument is corrected 3) the chain of argument has the proponent thesis as its conclusion

In [13] is presented a survey of formal system of persuasion dialogue that point out the crucial role of the regulating interaction among agents rather than design of behaviour in individual agent within a dialogue. Among the main approaches to design persuasion dialogue and communication between agents based on arguments we draw inspiration from the Parson and McBurney's [9] and Prakken's [14] approaches; and also from [15], where the authors show how each move of a dialogue could be specified by rationality rules, dialogue rules and update rules explicating the relation with the commitment store.

The more common locutions of persuasion dialogue that can be found in literature are well collected in [13], and briefly listed here:

- *claim* φ (assert): The agent asserts a formula φ to start the persuasion.
- *why* φ (challenge): The agent asks for reasons about the φ formula.
- *concede* φ (accept): The agent accepts the validity of φ.
- *reject* φ(retract): The agent no commits the φ. In some cases it retracts the formula from the commitment store previously stored.
- S *since* φ (argue): The agent provides reasons for φ formula by an argument.

```
dialog_persuasion(X,Y,P):=
    X:assert(argument(true,I,P)).
    dialog_response(X,Y,argument(true,I,P))

dialog_response(X,Y,argument(true,I,P)):=
    Y:accept(argument(true,I,P)) +
    Y:reject(argument(true,I,P)) +
    Y:why(argument(true,I,P)).
       X:argue(argument(B,I1,P)).
       dialog_argue(X,Y,argue(argument(B,I1,P))).

% Evaluation of chain argument support of P assertion
dialog_argue(X,Y,argument(B,I,P)):=
    Y:accept(argument(B,I,P)) +
    Y:reject(argument(B,I,P)) +
    Y:argue(argument(B1,I1,P1).(
       X:retract(P) +
       X:argue(argument(B2,I2,P2)).
       dialog_argue(X,Y,argument(B,I,P)))).
```

Fig. 2. Persuasion dialogue without interaction with the CS

A. Protocol Specification

In order to make a persuasion dialogue concrete, a persuasion protocol is typically to be defined among two parties—proponent and respondent. We formalise through our process algebra a generic persuasion dialogue with and without automatic action to the commitment store. The protocol draws inspiration from [16, 15] and adds repetition rule proposed by [13]. The dialogue could be partially driven through the state of commitment store by the actions listed in II-A that are specifiable in the protocol.

Figure 2 shows a dialogue protocol for persuasion where an agent can accept or reject an assertion P based on its attitudes by an internal evaluation of facts and argument acceptability. Then an argumentation phase starts that concludes with either an acceptance or rejection of the assertion P expressed by an argument with "true beliefs". The relation among dialogue and commitments is not explicitly expressed. In a dialogue, each move could be specified by rationality rules, dialogue rules and update rules [15]: the rationality rules specify the pre and post conditions for playing a move; The update rules specify the modification of commitment store; And the dialogue rules specify the next moves. With our process algebra we have the expressive power to cover the three types of dialogue rules. For instance, we propose modified version of the persuasion protocol in the figure 3 where we provide the specification of the automatic evaluation of some preconditions (rationality) and the consequent modification of the commitment store (update). In that version of the dialogue specification the DA automatically drives the sequence of action through the state of the commitment store using the term actions: *commit* and *acceptable*. In the choice points some locutions are automatically chosen by preconditions based on the state of acceptability of arguments. In particular the proponent agent (X) is constrained to retract the proposal if its supporting argument is not acceptable during the arguing phases. Also, the opposer (Y) is constrained to accept the proposal if its opposing argument is not acceptable respect to the state of the commitment store. We exploit the ability of the CAA in order to find argument acceptability following the credulous

argumentation semantic.

This protocol formalisation is very flexible, and opens a number of different courses of actions. The problems could be the termination of dialogue and the determination of the dialogue result. The dialogue is partially automated through DA and CAA infrastructure. Agents have the time the control over own actions, and can decide in every moment to suspend the dialogue.

```
dialog_persuasion(X,Y,P):=
  X:assert(argument(true,I,P)).
  dialog_response(X,Y,argument(true,I,P))

dialog_response(X,Y,argument(true,I,P)):=
  Y:accept(argument(true,I,P)).commit(argument(true,I,P)) +
  Y:reject(argument(true,I,P)) +
  Y:why(argument(true,I,P)).
     X:argue(argument(B,I1,P)).commit(argument(B,I1,P)).
     dialog_argue(X,Y,argue(argument(B,I1,P)))

% Evaluation of chain argument support of P assertion
dialog_argue(X,Y,argument(B,I,P)):=
  Y:accept(argument(B,I,P)).commit(argument(B,I,P)) +
  Y:reject(argument(B,I,P)) +
  Y:argue(argument(B1,I1,P1)).commit(argument(B1,I1,P1)).(
     acceptable(argument(B1,I1,P1)).(
       X:retract(argument(B,I,P)) +
       X:argue(argument(B2,I2,P2)).commit(argument(B2,I2,P2)).(
         acceptable(argument(B2,I2,P2)).
         dialog_argue(X,Y,argument(B,I,P)) +
         not(acceptable(argument(B2,I2,P2)).
         X:retract(argument(B,I,P))
       )
     ) +
     not(acceptable(argument(B1,I1,P1))).
       Y:accept(argument(B,I,P)).commit(argument(B,I,P))
  )
```

Fig. 3. Persuasion dialogue with CS interaction: Automatic evaluation of acceptability

B. Technology Support

Logic programming and meta logic programming are two useful techniques to prototype quickly complicated software systems with rational behavior. The technological support to build artifacts is provided here by TuCSoN, a coordination infrastructure for MAS introduced in [17]. TuCSoN infrastructure following a Linda like coordination model provides a programmable environment based on logic tuples.

To realize CAA and DA implementing the necessary operators listed in section II, an obvious choice is to exploit a TuCSoN logic tuple centre. In fact, on the one hand a typical argumentation process is composed of two parts: (1) knowledge representation; and (2) computation over the set of arguments. On the other hand, the tuple centre architecture is also composed of two parts: an ordinary tuple space where the information are stored in form of tuples, and a behaviour specification that defines the computation over the tuple set. Thus, a TuCSoN tuple centre could support the argumentation process by representing knowledge declaratively in terms of logic-tuple arguments, and by specifying the computation over argument set in term of ReSpecT specification tuples.

From a practical point of view, we exploit the Prolog language and ReSpecT to implement the meta programs for managing the argument set with the ability to calculate: (1) the argument validity; (2) the relations of undercut and attack between argument; (3) the conflict-free sets; and (4) the preferred extensions. Each argument has its own context, where the argument is true. The context is provided in the argument and is composed only by the set of beliefs – facts and rules – directly declared in the tuple. The connection between the premises and the conclusion is expressed in terms of the corresponding inference process, which is specified in the argument too.

The meta programs are useful also to realize the control of dialogue interaction. The engine of process algebra management is implemented exploiting a transition system defined in Prolog by the predicates `transition(Currentstate, Action, Newstate)`. The program has to have the ability to change dialogue state after an agent action, to search of next admissible move after an agent request, and also to make the automatic interaction with the commitment store by argumentative actions. For a more detailed explanation of the use of meta-program technique to manage argument and dialogue process we forward the interested reader to [18].

C. Example of a Run

In this section we provide an example of run of a simplified version of persuasion dialogue exploiting the TuCSoN infrastructure and showing its use. In order to perform the dialogue simulation TuCSoN provides useful tools: *CLIAgent* to simulate agents interaction and *Inspector* to inspect current state of tuple space . The Inspector tool shown in figure 4 allows users to observe and debug the communication state and the behaviour of a tuple centre. In particular, it makes possible to inspect the tuple set, the pending query set, the triggered reaction set, and the behaviour specification set.

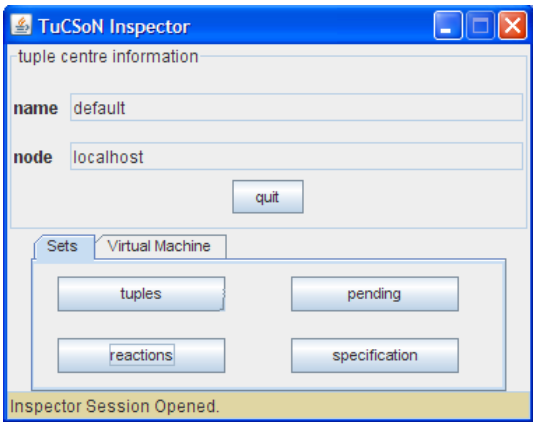


Fig. 4. Inspector tool

The *CLIAgent* tool allows users to invoke the commands of the TuCSoN coordination language. For our purpose we exploit the *CLIAgent* to utter agent locution in the form `out(move(Dialog,Id,Locution))`.

The rules to manage the dialogue in the DA are programmed with the ReSpecT code in [18]; for the commitment store

the same tuple space of dialogue is considered, and the initial dialogue state is expressed by the tuple

```
dialogstate(persuasion,[act(X,assert1(P)),
  (act(Y,accept(P))+act(Y,reject(P)))+act(Y,assert1(non(P)))+
  act(Y,why(P),act(X,argue(argument(N,bel(B),inf(I),conc(C)))),
  (act(Y,accept(N))+ act(Y,reject(N)))]).
```

The locutions that could be uttered in that dialogue are: *assert*, *accept*, *reject*, *why*, and *argue*. We start the simulation sending a *assert* locution in tuple centre from agent *Paul* by the CLIAgent shown in the figure 5. After that move, the infrastructure reacts and calculates next dialogue state.

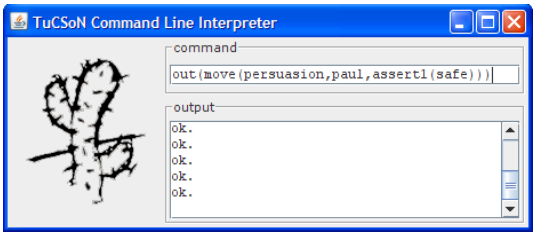


Fig. 5. CLIAgent

```
move(persuasion,paul,assert1(safe))
dialogstate(persuasion,
  ['+'('+'('+'(act(_4,accept(safe)),act(_4,reject(safe))),
  act(_4,assert1(non(safe)))),act(_4,why(safe))),
  act(paul,argue(argument(_3,bel(_2),inf(_1),conc(_0)))),
  '+'(act(_4,accept(_3)),act(_4,reject(_3)))])
```

The responder agent *Olga* can ask the possible admissible next locutions by the tuple `rd(nextlocutions(persuasion,L))`, and the tuple centre responds by new tuple `nextlocution`.

```
nextlocution(persuasion,
  [act(_2,accept(safe)),act(_2,reject(safe)),
  act(_1,assert1(non(safe))),act(_0,why(safe))])
```

At this point, the responder chooses a move either from the state of commitment store or independently from our knowledge base—for instance in this case the choice could be *why(safe)*. Figure 6 shows the state of the tuple centre after *Olga* locution by the inspector tool. The new `dialogstate` expresses the remaining locution constrained by previous logical unification of *paul* and *olga* identifiers.

```
dialogstate(persuasion,[act(paul,
  argue(argument(_3,bel(_2),inf(_1),conc(_0)))),
  '+'(act(olga,accept(_3)),act(olga,reject(_3)))])
```

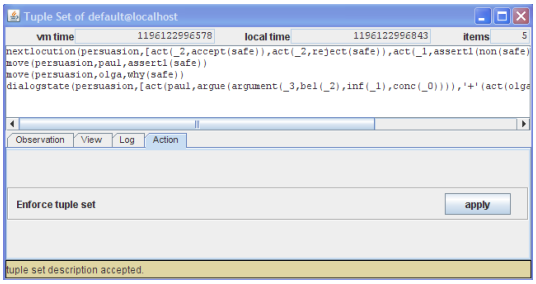


Fig. 6. Tuple Set

References

[1] G. B. Walker and S. E. Daniels, "Argument and alternative dispute resolution systems," *Argumentation*, vol. 9, no. 5, pp. 693 – 704, 1995. [Online]. Available: http://www.springerlink.com/content/m1263hp73g344127

[2] H. Prakken and G. Sartor, *Computational Logic: Logic Programming and Beyond. Essays In Honour of Robert A. Kowalski, Part II.*, 2048th ed., ser. Lecture Notes in Computer Science 2048. Berlin: Springer, 2002, ch. The Role of Logic in Computational Models of Legal Argument: A Critical Survey, pp. 342–380. [Online]. Available: http://www.springerlink.com/content/e0j2bhdq8gm8cg98

[3] J. Pitt, D. Ramirez-Cano, L. Kamara, and B. Neville, "Alternative Dispute Resolution in Virtual Organizations," in *Proceedings of The Eighth Annual International Workshop "Engineering Societies in the Agents World" (ESAW 07), Athens, Greece*, 2007.

[4] T. Schultz, G. Kaufmann-Koheler, D. Langer, V. Bonnet, and J. Harms, "Online dispute resolution: State of the art, issues, and perspectives," Faculty of Law and Centre Universitaire Informatique, University of Geneva, Tech. Rep., October 2001, draft Report.

[5] D. Walton and D. M. Godden, "Persuasion dialogue in online dispute resolution," *Artificial Intelligence and Law*, vol. 13, pp. 273–295, 2005. [Online]. Available: http://www.springerlink.com/content/k813173822154532

[6] A. Omicini, A. Ricci, and M. Viroli, "Artifacts in the A&A meta-model for multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 3, Dec. 2008, special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.

[7] E. Oliva, P. McBurney, and A. Omicini, "Co-argumentation artifact for agent societies," in *Argumentation in Multi-Agent Systems*, ser. LNAI, S. Parsons, I. Rahwan, and C. Reed, Eds. Springer, Apr. 2008, vol. 4946, ch. 3, pp. 31–46, 4th International Workshop (ArgMAS 2007), Honolulu, HI, USA, 15 May 2007. Revised Selected and Invited Papers. [Online]. Available: http://www.springerlink.com/content/5817w1n882861170/

[8] E. Oliva, M. Viroli, A. Omicini, and P. McBurney, "Argumentation and artifact for dialogue support," in *5th International Workshop "Argumentation in Multi-Agent Systems" (ArgMAS 2008)*, I. Rahwan and P. Moraitis, Eds., AAMAS 2008, Estoril, Portugal, 12 May 2008, pp. 24–39.

[9] S. Parsons and P. McBurney, "Argumentation-based communication between agents," in *Communication in Multiagent Systems*, ser. LNCS, M.-P. Huget, Ed., vol. 2650. Springer, Berlin, September 2003, pp. 164–178.

[10] E. Oliva, P. McBurney, and A. Omicini, "Co-

argumentation artifact for agent societies," in *4th International Workshop "Argumentation in Multi-Agent Systems" (ArgMAS 2007)*, S. Parsons, I. Rahwan, and C. Reed, Eds., AAMAS 2007, Honolulu, Hawai'i, USA, 15 May 2007, pp. 115–130.

[11] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artificial Intelligence*, vol. 77, no. 2, pp. 321–358, 1995. [Online]. Available: citeseer.ist.psu.edu/dung95acceptability.html

[12] D. N. Walton and E. C. W. Krabbe, *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY Press, 1996.

[13] H. Prakken, "Formal systems for persuasion dialogue," *Knowledge Engineering Review*, vol. 21, no. 2, pp. 163–188, 2006.

[14] ——, "Coherence and flexibility in dialogue games for argumentation," *Journal of Logic and Computation*, vol. 15, no. 6, pp. 1009–1040, 2005.

[15] L. Amgoud, N. Maudet, and S. Parsons, "An argumentation-based semantics for agent communication languages," in *ECAI*, F. van Harmelen, Ed. IOS Press, 2002, pp. 38–42.

[16] S. Parsons, M. Wooldridge, and L. Amgoud, "Properties and Complexity of Some Formal Inter-agent Dialogues," *Journal of Logic and Computation*, vol. 13, no. 3, pp. 347 – 376, 2003.

[17] A. Omicini and F. Zambonelli, "Coordination for Internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sep. 1999.

[18] E. Oliva, "Argumentation and artifacts for intelligent multi-agent systems," Ph.D. dissertation, Dottorato in Ingegneria Elettronica, Informatica e delle Telecomunicazioni, Cesena, Italy, Mar. 2008.

Towards a New Inheritance Definition in Multi-Agent Systems

Antonino Ciuro, Massimo Cossentino, Giuseppe Fontana, Salvatore Gaglio, Riccardo Rizzo and Monica Vitali

***Abstract*—Growing complexity of software systems leads some researchers to explore new paradigms like self-organization and genetic programming. We regard this problem as a new occurrence of a need that has been partially solved in the past with the introduction of object-orientation whose most innovative feature can probably be agreed to be inheritance. In this work, the authors propose a different approach for solving the initially discussed process. The definition of a new "nature-inspired" inheritance is discussed. Agents according to this approach can reproduce by mixing their genome and generate new agents that can better fit a specific problem.**

***Index Terms*—Agent-oriented software engineering, Agents and objects, Relationships between agents and other development technologies.**

I. Introduction

SOFTWARE complexity continuously grows up and pushes researchers towards the definition of new techniques for facing this complexity.

In 1970s, the well-known software crisis encouraged the adoption of new programming paradigms (object-orientation) and new design philosophies (waterfall design approaches were fully exploited and then overcome by more modern evolutionary and iterative/incremental approaches). Despite the relevant advantages offered by modern technologies (service-oriented architectures, model driven engineering, and so on) new challenges have to be faced. Several researchers think that a solution to growing software complexity is in the definition of evolutionary paradigms for software behaviour. Some of them adopt self-organization [7] as the key strategy for solving complex problems. The idea of designing a rather simple system (usually composed by autonomous entities called agents) that can dynamically evolve its behaviour towards the achievement of a goal is for sure very fascinating. The designer needs to define the goal (for instance by using a formal logics or a kind of fitness function) and the entities composing the system will reorganize their individual behaviours and collaborations in order to collectively achieve the goal. The research presented in this paper starts from a similar motivation, it is largely inspired by evolutionary concepts and it aims at exploring the following issue: how can system entities evolve themselves in order to achieve the best fitness for solving a problem? This question has usually been faced by adopting evolutionary criteria of the information manipulated by an algorithm (this is the typical application context of genetic algorithms [6] [5]). If the concept of agent as a highly encapsulated, autonomous, proactive and social entity is introduced in this scenario, a different strategy should be identified because object-oriented inheritance cannot be applied because it violates the high encapsulation of agents. Since agents are frequently regarded as *live entities* the obvious path for achieving such an evolution is to look at natural evolutionary processes and the problem becomes defining an agent-oriented inheritance as the propagation of parents knowledge and abilities to their child with the necessary changes that may ensure evolution. Modern agent-based architectures are mostly based on diffused object-oriented languages and therefore do not offer a specific (agent-oriented) inheritance feature. Sometimes, delegation is adopted as a surrogate for that. In order to define a natural evolution paradigm for agents, it is possible to look at the biological inheritance among individuals. The proposed approach consists in defining a genetic representation of the agent (a DNA-like representation of agent physical and behavioural features), and in allowing the reproduction of couples of individuals by mixing portions of their DNA. Casual changes introduced in the DNA mixing phase also ensure the possibility of pursuing an evolution of species just like it happens in nature. The paper is organized as follows: section 2 introduces the proposed approach, section 3 discusses the Genoma framework we developed to implement the agents' reproduction process, section 4 introduces a case study where the proposed approach is used to approximate some geometrical shapes. Finally some conclusions are drawn in section 5.

Antonino Ciuro is with EMC Corporation, aciuro@gmail.com.

Massimo Cossentino and Riccardo Rizzo are with CNR-ICAR, cossentino@pa.icar.cnr.it, ricrizzo@pa.icar.cnr.it

Giuseppe Fontana is with Accenture S.P.A., gfontana13@gmail.com

Salvatore Gaglio is with Universita' di Palermo, CNR-ICAR, gaglio@unipa.it

Monica Vitali is with Universita' di Palermo, monicavit164@gmail.com

II. The Proposed Approach

In object oriented approaches using inheritance makes portions of useful and general code available to a large set of other classes [9]. Such a code can be easily specialized by programmers in order to fulfil a new goal.

In the agent oriented world such an inheritance process doesn't exist. Some attempts have been proposed in the past (for instance in [4][8]).

In our approach the focus of inheritance is not the code but knowledge and ability of the agents, and the "user" is not a programmer but another agent (the CrosserAgent) that manages the deployment of new agents with new knowledge and abilities. We realize this inheritance feature by means of a reproduction process that starting from two parent agents (that are not able to solve a specific problem), generates a new

agent that (hopefully) better fits the problem. The solution is ensured by the generation of several new individuals that in turn reproduce themselves until the proper result is achieved.

Knowledge and abilities are stored in chromosomes and modified using techniques coming from genetic programming (crossover, mutation, and so on). Using the genetic algorithm point of view, we are looking to a solution in the space obtained by joining the ability and knowledge spaces. In so doing, we suggest an agent reproduction process characterized by two different phenomena:

- from an individualistic point of view: it is highly probable that a part (probably the most useful) of parents' knowledge and ability is forwarded to their children;
- from a social point of view: the behaviour of the new agents "moves" towards the achievement of the assigned goal. This means that plans and knowledge evolve in order to fulfil the desired service.

The first phenomenon concerns the conservation of the useful knowledge and abilities of the agent in the reproduction process. The second phenomenon is about the whole point of genetic programming: the desired behaviour emerges from the agent society during evolution and it can be considered as a "specialization" of the agent.

In order to realize our ideas, we developed the Genoma Framework; this framework allows us to obtain a society of agents capable to increment their efficiency and/or to learn new capabilities. A Genoma agent (an agent that belongs to the framework) is composed of chromosomes defining its knowledge about the world and the plans (addressed as Abilities) that define its behaviours. All the knowledge and the abilities of the agent are coded in a genome structure. Using the information in the genome structure it is possible to implement a reproduction process for the agent. The agent is developed using the JADE platform so it is able to send and receive messages to other JADE agents. The knowledge of the agent is implemented by using Java and it is an instance of ontological elements. In the Genoma framework, the agent society is composed by one or more Genoma agents that can make available many services, and by one agent that manages the reproduction process: the CrosserAgent. Once activated, each Genoma agent sends a chromosome containing all the information that defines the agent to the CrosserAgent.

Inside the agent society all the agents that need a service can ask for that service to all the others inside the society. If the required service is not available, or not satisfying, the agent can ask to the CrosserAgent to activate an evolution procedure. The CrosserAgent will form a society composed by descendants of all agents that can deliver a similar service and it will start the crossover procedure among the selected individuals. The generation zero will contain a lot of duplicate agents, if there are not enough individuals. The following generation will contain many agents of the preceding generation and new agents obtained by crossing genome of the original agents. This evolution process will continue until a suitable agent is obtained (according to some fitness criterion). This resulting agent will now be introduced in the agent society and the CrosserAgent will send to the agent that requested the unsatisfied service its name in order to properly fulfil the service request (Figure 1). The evolution process operates at two different levels: knowledge and plan. The knowledge level is related to instances of (knowledge) objects that belong to the agent, plans are related to how the agents behave and execute the required service. The evolution procedure is different for knowledge and plan and some details will be reported later.

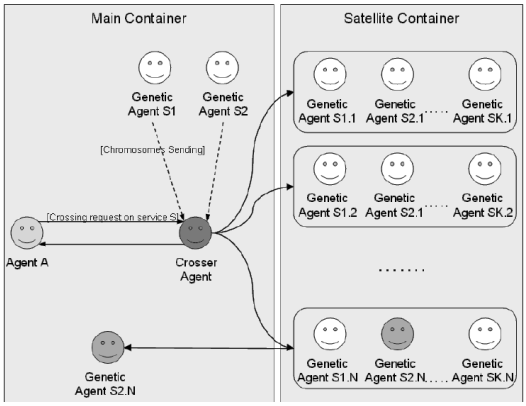


Fig. 1. A sketch of the agent crossover procedure, Genoma agents are white, selected agents are light gray.

An example will help to explain the full mechanism. Imagine we want to develop an agent capable to escape from a labyrinth. A simple (but not efficient) way to escape from a labyrinth is to go ahead and turn on the same side each time a wall or a turn is met. This plan can be expressed as a graph where each node corresponds to a choice or an action. In the upper part of Figure 2 we have two plans for agents that are trying to solve the labyrinth: the former (agent A on the left) changes its direction whether it finds a wall or not. The latter (agent B on the right) tries to move forward in every circumstance. The nodes of the plans refer to the knowledge of the agent. This is a consequence of the ontology structure adopted in PASSI [2] that is composed of: concepts describing categories of the world, predicates asserting the status of the previous cited concepts and finally actions that can affect concepts status. For instance, the action "turn" is referred to a concept "direction" of the agent. This knowledge piece, once instantiated in the agent, can have four values: right, left, forward or backward. We can suppose it has value left. Both of these agents are unable to solve the problem. The crossover of these agents generates a new individual who has acquired the ability to solve the proposed problem as shown in the lower part of Figure 2. The new agent in Figure 2 inherits from agent B the ability to go ahead if there are not walls (part of the plan on the *no* branch of the second decision node) and from agent A the ability to turn according to the actual value of "direction". This value actually depends from the crossover process applied to the knowledge of the two agents. In a second scenario, we can suppose agent A having the value "back" for its direction and agent B having the value "left" (even if it doesn't use it in its plan). The new agent can inherit one or both parents' knowledges and it will be able to solve the problem only if it inherits the knowledge direction with value left. Figure 3 shows a possible crossover of the

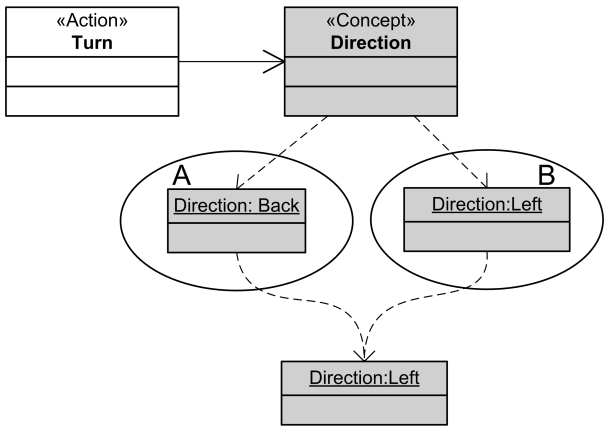


Fig. 3. Diagram which shows the relations between the action "Turn" and the concept "Direction". The instance on the left belongs to the upper left agent in Figure 3, the one on the right belongs to the upper right agent. The instance in the lower part is the knowledge of the new agent generated by crossover.

knowledge "direction". The instance of "direction" on the left belongs to agent A in Figure 2, the one on the right belongs to agent B. Different tecniques for crossing knowledge will be explained further on.

III. The Genoma Framework

The Genoma Framework blends genetic programming to the JADE agent platform[1][3]. All the information required to define an agent and its behaviour are coded in a data structure that is comparable to a genome, as it will be described later. Genomes belonging to two different exemplars of agents are mixed in order to obtain a new agent with some new behaviours.

The basic structure of the framework, as reported in this paper, is applied to the realization of services delivered by single agents, but because of the generality of the approach it can be used for other broader scope tasks. The Genoma Framework is based on the GenomaAgent agent that is a specialization of the JADE Agent. The reproduction process, governed by the CrosserAgent, is applied to couples of these agents and produces new individuals. In the following a description of the GenomaAgent agent will be provided in terms of its structure (subsection III-A) and chromosome (subsection III-B); after that, subsection III-C provides a description of the CrosserAgent.

A. The Genoma Agent Structure

The GenomaAgent agent structure is based on a class that is an extension of the JADE Agent class. The GenomaAgent class is composed of:

- a global plan representing the ability of the agent;
- a set of knowledge items (instances of ontology elements);
- a set of tasks that are used in the global plan;

These elements represent the chromosomes of the agent; more specifically we can talk of an Ability chromosome (representing the agent global plan and composed by activities and control nodes), a knowledge chromosome and a set of task chromosomes (representing the agent activities). Task chromosomes can, in turn, be represented as the composition of an Ability and Knowledge chromosome. The Ability chromosome is composed of activities and control nodes while the knowledge chromosome is composed of referred knowledge (that is the portion of the agent knowledge that is referred in the plan). The global plan (stored in the agent's Ability chromosome) manages the agent's life, defines the roles the agent can assume during its life-cycle, and manages the interactions with the environment and the service delivery. The activities used in the Ability chromosome are elementary pieces of behaviour that constitute the whole set of the agent capacities (usually implemented by using tasks in the JADE platform). Each activity is the representation in the Ability chromosome of an ontology action (stored in the Knowledge chromosome as an Action gene). Tasks realizing activities can have an internal plan that leads the agent towards the achievement of the related sub-goal. Tasks refer to knowledge items for manipulating entities of the environment. As a consequence, Tasks, even though they constitute one of the agent's chromosomes, contain an Ability and a Knowledge chromosome themselves.

All of these elements (ability, knowledge and tasks) define the agent chromosome structure. In order to simplify the operations related to the reproduction process, a plan is represented by a tree structure that is composed by nodes. A node is the basic element of the tree and can contain an action/activity or a predicate. It can have one or more predecessors and one or more successors. If the action or the predicate related to the node are satisfied, all the successors nodes are activated.

B. Chromosomes

The agent structure can be defined using three chromosome categories: a Knowledge chromosome, an Ability chromosome and a Task chromosome. Each chromosome is made by genes; in the first chromosome, each gene describes an instance of the ontology (predicates, concepts and actions). The ability chromosome contains a plan that represents what the agent is able to do.

Figure 4 reports the genome structure in form of a UML class diagram.

1) Knowledge Chromosome Crossing: The generated agent will have a set of knowledge elements derived by the two parents. If parents own a similar knowledge they will generate a new knowledge that will be in relation to both the knowledge of the parents. If one of the parents has a knowledge that does not have any correspondent to the knowledge of the other parent, the new knowledge will be in relation only will the knowledge of one parent (the right parent). In this way, for each knowledge piece of the parents there will be only one knowledge piece in the child. On the other side, the knowledge of the child will be in relationship with at least one knowledge of the two parents.

The evolution process is realized by applying several different crossover techniques to the parents' genome. This process

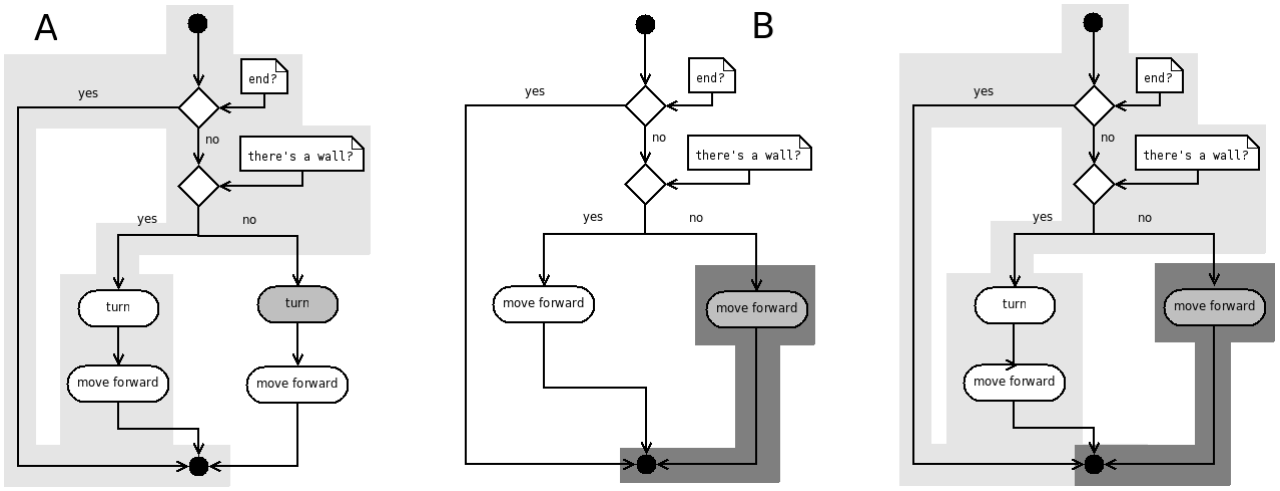


Fig. 2. On the left part of the figure the two plans of the parent agents and on the right the resulting plan. The parts of the plan selected for the crossover procedure are highlighted.

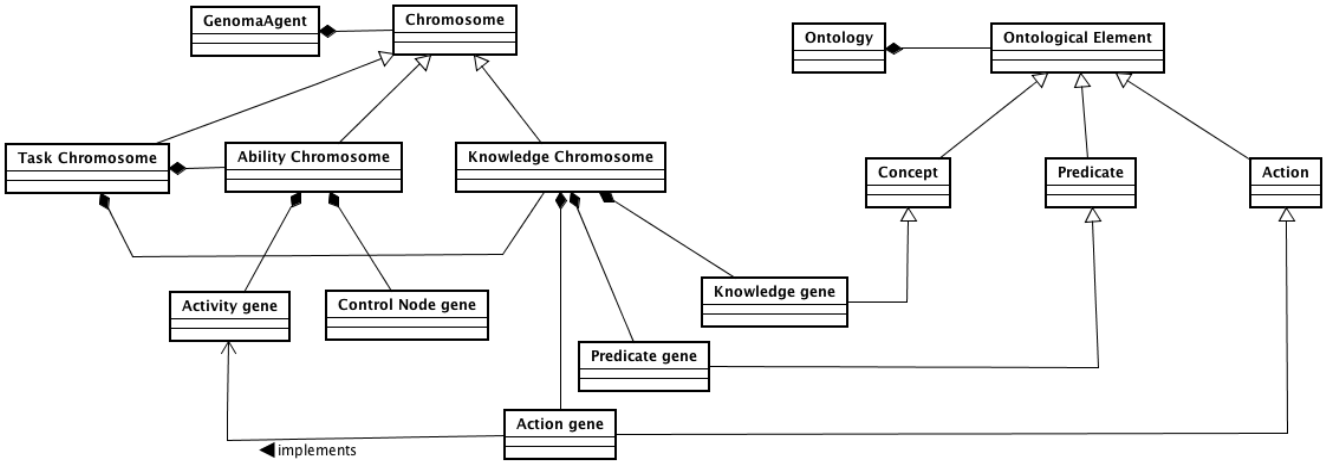


Fig. 4. A UML class diagram representing the agent genome

is strongly conditioned by the need of obtaining a working agent. In order to achieve this objective, it is necessary that knowledge crossover will happen only between similar knowledge elements. There are four techniques to obtain a new knowledge:

- **fusion:** the two parents' knowledge is unified into a single body of knowledge that will contain a weight or algebraic average of the parent knowledge
- **selection:** as in fusion, the two parents' knowledge originate a unique knowledge in the child but, in this case, one of the two is copied and the other one is discarded
- **union:** the new individual's knowledge will be composed by the union of the two parents' bodies of knowledge. A frequent use of this technique may produce a very redundant agent
- **copy:** it is used if one of the two parents has a knowledge the other parent has not. In this case the knowledge is simply copied from the parent to the child.

As it is obvious, we assume that all the knowledge pieces of an agent may have a reference to other knowledge pieces or may be referred by portions of the agent plan. Of course, it is necessary to maintain these references in the child agent during the reproduction process; otherwise it will contain some knowledge portions without any reference (and therefore useless).

An example is in Fig. 5 where the concept "brush" is linked to the concept "grid" (an agent can draw in the grid by using a brush). If we remove this link, the crossover may generate

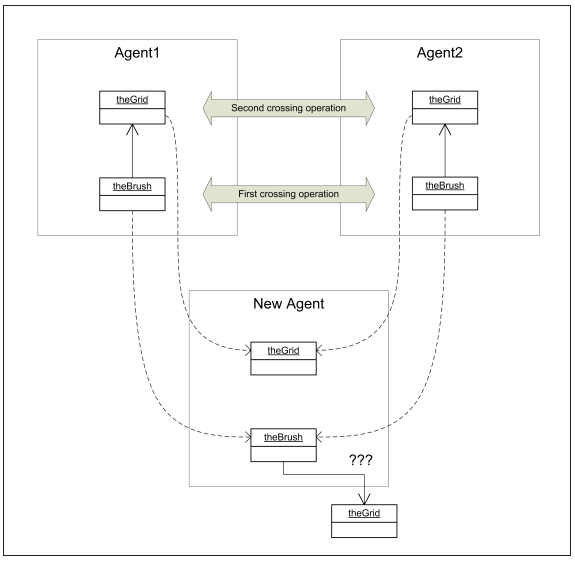


Fig. 5. Crossover where there are referred knowledge pieces

a concept "grid" without any reference to the object "grid" referred by the two parents.

The algorithm applied for the knowledge chromosomes crossing is the following:

1) A crossover operation (fusion, selection, union, copy) is assigned to each knowledge gene element contained in the agent chromosome;
2) An item, containing the selected operation and the parents' knowledge, is added to the list of crossover operations to be done.
3) The first item of the list is selected and the corresponding operation is performed. Then the resulting knowledge is verified:
   a) if both the objects referred by the parents' knowledge were not marked then a new mark is created with the same crossover operation and it is added to the bottom of the crossover list
   b) if at least one of the knowledge was marked for the crossover operation, a reference is created.
4) if the list is empty then end, else go to step 3.

2) Plan Chromosome Crossing: Plan crossing is the operation that allows obtaining the child plan from the two plans of the parents. Plan crossing is some way less complex than the previous discussed knowledge crossing, because all the referred knowledge genes have been already crossed at this point. Generic tasks composing the plan (and referred in the knowledge as actions) are crossed by using the same techniques explained in the previous subsection. In this case it is necessary to distinguish the two parents' agents in, a randomly labelled, "mother" and "father" agent. The plan fragment obtained from the "mother" agent will always contain the starting node and will be obtained as follows:

1) randomly select a cut node
2) cancel all the links that start from the cut node
3) cancel all the nodes that are not reachable

An example of "mother" agent plan is represented in the top-left part of Figure 2. The plan fragment obtained from the "father" agent will always contain the end node and will be obtained as follows:

1) randomly select a cut node
2) cancel all the nodes that are not reachable from the cut node

An example of "father" agent plan is reported in the top-right part of Figure 2. The "child" agent plan is obtained substituting the cut node of the "mother" with the cut node of the "father" (see bottom plan in Figure 2).

3) Mutation: Mutation allows obtaining a new individual from a single parent by modifying in a random way one of its characteristics. If mutation is applied to a knowledge item, an attribute is selected and a random value is given to the node. If mutation is applied to a plan a node is replaced with an equivalent one or a link is added in a random way. The probability that an efficient individual is obtained after mutation is low but mutation adds an unpredictable variation that sometimes allows obtaining unexpected improvements.

C. The Crosser Agent

The CrosserAgent is responsible for managing the reproduction process that includes the creation of agent generations, the execution of new agents, and the evaluation of the results achieved by each agent. The behaviour of this agent is composed by a set of tasks executed according to a specific plan (Figure 6 represents it as a PASSI [2] Task Specification diagram). More in details, the most relevant components of this behaviour are:

- **Listener:** this task is responsible for message receiving and management. Messages containing agent genome and crossing requests are usually received by this agent.
- **ManageRequest:** this task is responsible for creating the necessary file system structure (namely directories) for storing next agent generations.
- **GenerateChildren:** it is responsible for child agent creation and chromosome crossover operations.
- **CompileChildren:** it compiles the classes (agents) built by the GenerateChildren task.
- **PollChildren:** it instantiates agents of the current generation and introduces them in the agent platform. Then it requests them the service that is to be evaluated to esteem the agent fitness to solve the assigned problem. All the agents of this generation have a fixed time for performing the assigned duty. If they complete the work in time, their fitness is evaluated, otherwise they are discarded.
- **EvaluateChildren:** it calculates the fitness function value for each agent and stores it.
- **ManageGeneration:** it selects the agents that will compose the next generation. This new generation will be composed of elite agents (the best agents of the current generation) and the children of this generation that better realised the required service. If the crossover process is

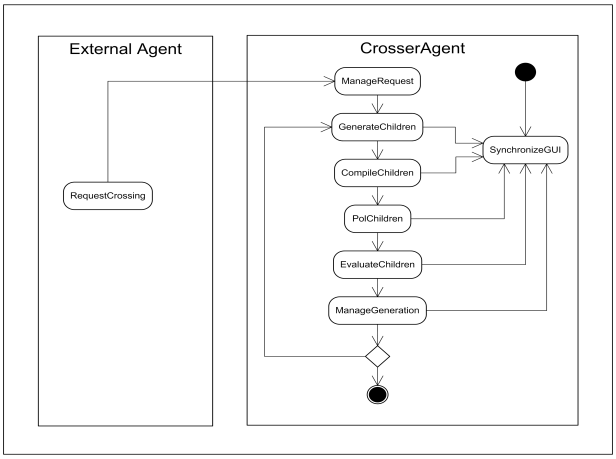


Fig. 6. The CrosserAgent plan represented as a PASSI Task Specification diagram

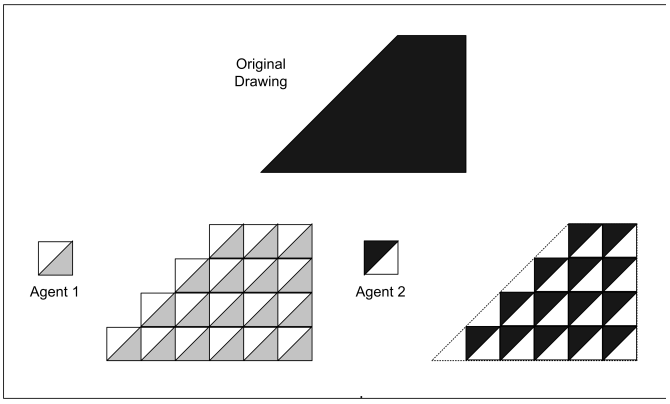


Fig. 7. The objective of the case study is reproducing the shape at the top of this figure. Initial agents are able to achieve a limited approximation of the required goal.

completed (maximum number of generations or maximum value of fitness function reached) then it selects the best agent.

In the current implementation of the framework, crossover operations can be done by only considering agents providing the same service.

IV. Experimental Results

The discussed approach has been applied to a simple case study where the goal is to be achieved by a single agent; the goal consists in reproducing a design in shape and colour (see the shape at the top of Figure 7).

A. Reproduction of a Graphical Structure

This case study uses an agent-oriented system for solving a problem characterized by several different variables. The problem consists in reproducing a shape in both colour and geometry; the fitness function considers both shape similarity and colour proximity.

Figure 7 reports the original shape to be reproduced (experiments have been done by adopting different shapes, sizes and filling colours). Each agent of the first generation (composed by the two individuals depicted in Figure 7) was able to draw by using elementary cells (brushes) of square shape. Inside the brush, it was able to draw a triangle with a vertex on the lower-right corner (the other agent was able to draw a triangle with a vertex on the higher-left corner). Each agent designed its brush in a different colour. An optimal solution to this problem requires that agents cross their genome in order to learn how to draw brushes that can produce a good reproduction of the goal picture in both shape and colour. During the evolution process, brushes of different size can emerge and the triangle can change its orientation and even its shape. The fitness function calculates the percentage of the shape that is filled (and not left blank) and the similarity in the filling colour to the goal picture. Figure 8 reports results obtained by individuals of the fourth generation; they are interesting because they show the different evolution paths that can be undertaken during

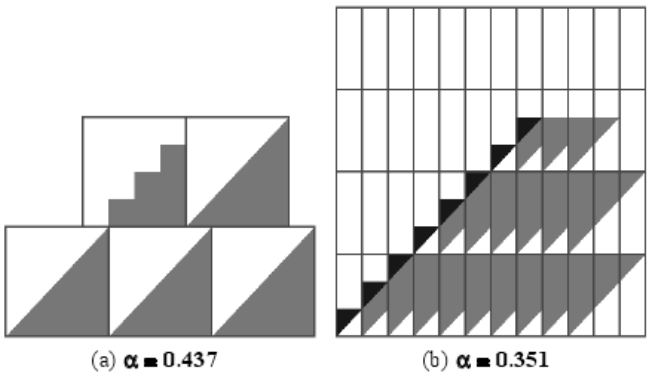


Fig. 8. Results obtained by individuals of the fourth generation and the corresponding fitness function

the process. We found that the number of generations that is necessary to build in order to find the optimal solution to the problem depends (as it was expected) on the complexity of the goal picture. In the reported example, nine generations have been necessary in order to obtain some individuals (more than one obtained the same score) achieving the optimal solution. A few examples of good and optimal solutions obtained from individuals of the ninth generation are reported in Figure 9.

V. Conclusions and Future Works

We proposed a new inheritance approach that focuses on agent knowledge and abilities as a subject of reuse.

In our approach the agents do not inherit code or classes but behaviours and pieces of knowledge; genetic programming techniques modify and specialize them in order to fulfil a defined goal or delivery a requested service. In principle, the management of this procedure is delegated to an agent so that this process is automatic and transparent to the user: agents are evolved by reusing and modifying their knowledge and abilities without the programmer help.

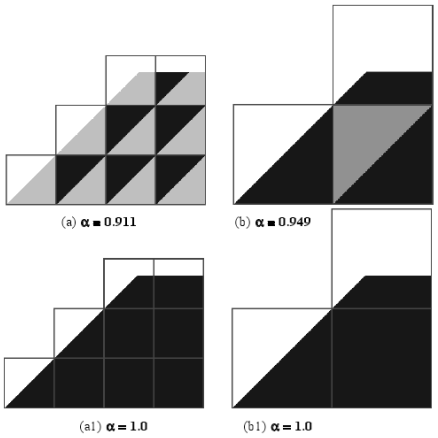


Fig. 9. Results obtained by individuals of the eight (subfigures (a) and (b)) and ninth generations (subfigures (a1) and (b1))

Many aspects of the proposed method need further investigations: for example we noticed that there are also part of the plan of the agent that are not used but merely transmitted from parent to children and they remain present in the agents that reach the final goal. These parts of plan and knowledge can be considered a sort of "memory" of the agent society and they can be useful if another service is requested. In fact from an optimized set of knowledge pieces and set of abilities it can be difficult to obtain something new. It is possible to think that if we optimize the agent by deleting unused knowledge and abilities future generations will cover a smaller area of the "solution" space. The impact on the agent functionality, and on the agent society, of these parts of plan and knowledge that are not used are worth of more investigations.

REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. Jade - a fipa2000 compliant agent development environment. *Agents Fifth International Conference on Autonomous Agents (Agents 2001)*, 2001.

[2] M. Cossentino. *From requirements to code with the PASSI methodology. In Agent Oriented Methodologies.* Idea Group Publishing, june 2005.

[3] M. Cossentino and L. Sabatucci. *Agent System Implementation*. CRC Press, April 2004.

[4] L. Crnogorac, A.S. Rao, and K. Ramamohanarao. *Analysis of inheritance mechanisms in agent-oriented programming*. Morgan Kaufmann Publishers Inc., 1997.

[5] R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection*. MIT Press, 1992.

[6] Nils J. Nilsson. *Artificial Intelligence, a New Synthesis*. Morgan Kaufmann Publishers, Inc., 1998.

[7] G. Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self organisation and emergence in mas: An overview. *Informatica Journal*, 2005.

[8] O. Shehory, K. Sycara, P. Chalasani, and S. Jha. Agent cloning: An approach to agent mobility and resource allocation. *IEEE Communication Magazine*, 1998.

[9] A. Snyder. Encapsulation and inheritance in object-oriented programming languages. In *In proc. of Conference on Object Oriented Programming Systems Languages and Applications*, pages 38–45, 1986.

Nature-inspired Spatial Metaphors for Pervasive Service Ecosystems

Cynthia Villalba[1], Alberto Rosi[1], Mirko Viroli[2], Franco Zambonelli[1]
1) University of Modena and Reggio Emilia – 42100 Reggio Emilia, Italy
2) University of Bologna – 40723 Cesena (FC), Italy
cynthia.villalba@unimore.it, alberto.rosi@unimore.it, mirko.viroli@unibo.it, franco.zambonelli@unimore.it

Abstract

Innovative paradigms and frameworks have to be identified to enable the effective deployment and execution of pervasive computing services. Such frameworks must be conceived so as to match the spatially-situated nature of pervasive services, and must be able to exhibit properties of self-organization and self-adaptability, self-management, and of long-lasting evolvability. This paper discusses how such frameworks should get inspiration from natural systems, by enabling modeling and deployment of services as autonomous individuals, spatially-situated in an ecosystem of other services, data sources, and pervasive devices, all of which acting, interacting, and evolving according to a limited set of spatial "eco-laws". In this context, this paper presents a reference architecture to uniformly frame ecosystem concepts, surveys and critically analyzes different nature-inspired spatial metaphors to realize the idea, and details our current research agenda concerning the development of service frameworks inspired to the ecological metaphor.

1 Introduction

Pervasive and mobile computing devices increasingly populate our everyday environments [7]. These, together with the increasing amount of Web tools that makes it possible to produce and access spatially-situated information about the physical world [5], will eventually define a comprehensive, integrated, and very dense, decentralized shared infrastructure for general-purpose usage. At the user level, the infrastructure can be used to access innovative services for better perceiving/interacting with the physical world and for acting on it. It is also expected that users themselves will be able to personalize the infrastructure by deploying customized services over it (in other words, the overall pervasive infrastructure will be as open the same as the Web currently is). In addition, the infrastructure will be used as a way to enrich more traditional classes of digital services with the capability of dynamically and autonomously adapting their behavior to the context in which they are invoked and exploited.

The inherent spatial nature of the above infrastructure and of all the services that will be deployed over it is very sharp. On the one hand, the infrastructure will be embedded into physical space, will have to deal with spatial concepts and spatial data, and its devices will typically interact based on spatial proximity (as induced by wireless communications). On the other hand, services will have to deal with spatially-situated activities of users, and with their interacting with the physical world.

The effective development and execution of services in the above infrastructure calls for a deep rethinking of current service models and for service frameworks, in order to:

- Naturally match the inherent spatial nature of the environment and of the services within.
- Inherently exhibit those properties of self-organization, self-adaptation and self-management that are necessarily required in highly-decentralized and highly-dynamic scenarios (as the envisioned infrastructure is, due to its distributed nature, the unreliability of its components, and its openness to user contributions).
- Flexibly tolerate evolutions of structure and usage over time. This is necessary to account for increasingly diverse and demanding needs of users as well as for technological evolution, without forcing significant (and economically unbearable) re-engineering to incorporate innovations and changes.

To reach this goal, we should no longer conceive services and their interactions as it is usual made in standard SOA architectures [9]. There: services are simply considered as "loci" of functionalities, whose activities are orchestrated according to specific pre-defined patterns with the support of middleware services (such as discovery, routing, and context services) that either miss in accounting spatial

concepts or do not elect them as primary abstractions; self-organization, self-adaptability and self-management are not intrinsic properties of the system, but are typically enforced via ad-hoc one-of solutions, e.g., via the introduction of specific control tools [10]; long-term evolvability is simply not ensured, and most likely it can be achieved only at very high re-engineering costs.

Thus, the most promising direction is that of taking inspiration from natural ecosystems [16, 8], where spatial concepts, self-organization, self-management, and long-lasting evolvability are inherently there because of the basic "rules of the game". We are aware that nature-inspired solution have already been extensively exploited in the area of distributed computing (see e.g. [2, 11] for two recent extensive surveys). However, most of these proposals exploit the natural inspiration only for the effective implementation of specific algorithmic solutions or for the realization of specific distributed services. Here we go further, and argue that natural ecosystem can act as the key metaphor around which to conceive, model, and develop, fully-fledged pervasive service framework and all the components within.

You can think at physical systems, at chemical systems, at biological systems, as well as at the most properly called ecological systems. In all of them, you can always recognise the following characteristics: above a spatial environmental substrate, individuals of different kinds (or species) interact, compete, and combine with each other in respect of the basic laws of nature. Accordingly, in our scenario, the shared pervasive infrastructure substrate will have to be conceived as the space in which bringing to life an ecosystem of services, intended as individuals whose computational activities are subject to some basic laws of the ecosystem, and for which the dynamics of the ecosystem (as determined by the enactment of its laws) will provide for naturally enforcing features of self-organization, self-management, and evolvability.

In this context, the contributions of this paper are as follows:

- We introduce a unifying reference architecture for nature-inspired pervasive service ecosystems, to show how ecosystem concepts can be framed into a unifying conceptual scheme (Section 2).
- We survey the different metaphors that can be adopted for such ecosystems (Section 3), and discussed their advantages and limitations w.r.t. the capability of supporting self-organization and self-adaptation, self-management and decentralized control, and evolution over time (Section 4).
- We go into more details about the so called ecological metaphor, and sketch our current research work and our research agenda in that area (Section 5), and conclude (Section 6).

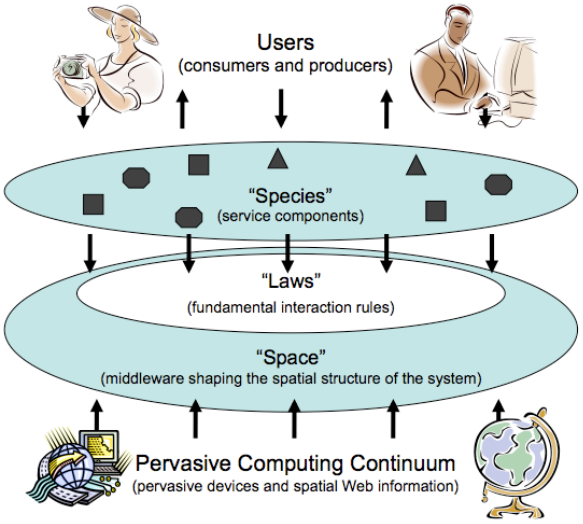


Figure 1. A Reference Architecture for Pervasive Service Ecosystems

2 A Reference Architecture for Pervasive Service Ecosystems

A unifying reference architecture can be identified around which to frame the key abstractions and the conceptual structure for spatial pervasive service ecosystems, independently of the specific metaphor adopted (see Figure 1).

At the lowest level is the physical ground on which the ecosystem will be deployed, i.e., a very dense infrastructure (ideally, a pervasive continuum) of networked computing devices and information sources. These includes all the devices that are going to increasingly pervade all our everyday environments (e.g., PDAs, smart phones, sensors, tags), all interconnected with each other, and most of which generating a large amount of information about the surrounding environment. In addition, such ground can also include the increasing amount of Web tools and data sources that already collect spatially-situated knowledge about nearly every aspect of the world.

At the highest level, service developers, producers and consumers of services and data, access the open service framework for using/consuming data or services, as well as for producing and deploying in the framework new services and new data components.

At both the bottom and the top levels, the architecture exhibits a high-degree of openness: new devices can join/leave the system at any time, and new users can interact with the framework and can deploy new services and data items on it. In between these two levels, there are the components of the pervasive ecosystem architecture.

The level of "Species" is the one in which physical and virtual devices of the pervasive system, digital and network resources of any kind, persistent and temporary knowledge/data, contextual information, events and information requests, and of course software service components, are all abstracted as "living entities" of the system (i.e., the *ecosystem individuals*) that populate the pervasive ecosystem space. Although such individuals are expected to be modelled (and computationally rendered) in a uniform way, they will have specific characteristics very different from each other, i.e., they will be of different "species".

In general terms, an ecosystem is expected to be populated with a set of individuals physically deployed in the environment (physical and network resources, contextual information, initialization data and services, and so on). Yet, the population of individuals is far from being static. First, the set of individuals is subject to changes (to tackle the physical system's mobility, faults, and evolution). Second, service developers and producers inject in the system new individuals at any time (they can insert new services and virtual devices, as well as data and knowledge). Third, producers and consumers can keep control and influence the behavior of (a limited set of) the individuals.

The "Space" level provides the spatial fabric supporting individuals, their spatial activities and interactions, as well as their life-cycle. From a conceptual viewpoint, the "Space" level gives shape to and defines the structure of the virtual world in which individual lives. Given the inherent spatial nature of pervasive services, it is clear that this level should consider that individuals exist in a specific portion of some metric space, and that their activities and interactions are directly dependent on their positions in space and on the shape of the surrounding space. What the actual structure and shape could be, might depend on the specific abstractions adopted for the modeling of the ecosystem.

From a more practical viewpoint, the spatial structure of the ecosystem will be implemented by means of some minimal middleware substrate, i.e., a software infrastructure deployed on top of the physical deployment context. Such middleware substrate will provide for supporting the execution and life cycle of individuals, and will enforce concepts of locality, local interactions, and mobility, coherently to a specific structure of the space.

The way in which individuals live and interact (which may include how they produce and diffuse information, how they move in the environment, how they self-compose and/or self-aggregate with each others, aggregate, how they can spawn new individuals, and how they decay or die) is determined by the set of fundamental "Laws" regulating the eternal service ecosystems model. Such laws, or "eco-laws", are expected to act on the basis of spatial locality principles, as in real laws of nature: the enactment of the laws on individuals will typically affect and be affected by the local space around them and by the other individuals on. The enactment of the eco-laws requires the presence of some meaningful description (within the uniform modeling on individuals) of the information/service/structure/goals of each species, and of proper "matching" criteria to define, based on such description, how the eco-laws apply to specific species in specific conditions of the space.

The dynamics of the ecosystem will be overall determined by having individuals in the ecosystem act based on their own internal goals, yet being subject to the eco-laws for their actions and interactions. The fact that the way eco-laws apply may be affected by the presence and state of other individuals, provides for closing the feedback look which is a necessary characteristic to enable self-* features. Indeed, the typical evolution patterns that can be driven by such laws may include forms of self-organization (e.g., service aggregation or service orchestration, where the eco-laws can play an active role in facilitating individuals to interact with each other and orchestrate their actions), self-adaptation (changing conditions will reflect in changes in the way individuals in a locality are affected by the eco-laws) and of decentralized self-management (the injection of new individuals can be used to modify the way eco-laws affect other individuals and, thus, to somehow control the evolution of the ecosystem dynamics form within the system). As far as adaptation over time and long-term evolution are concerned, the very existence of the eco-laws can make the overall ecosystem sort of eternal, and capable of tolerating dramatic changes in the structure and behavior of the species living in the ecosystem (i.e., the presence of brand new classes of services). Simply said in ecological terms: while the basic laws of life (i.e., the basic infrastructure and its laws) are eternal and do not change (i.e., do not require re-engineering), the forms under which it manifests continuously evolve (i.e., the actual service and data species), naturally inducing new dynamics for the interactions between individuals and for the ecosystem as a whole.

3 Metaphors for Pervasive Service Ecosystems

The key difference in the possible approaches that can be undertaken towards the realisation of eco-inspired service frameworks (as from the described reference architecture) stands in the metaphor adopted to model the ecosystem, its individuals, the space in which they live, and its laws. Without excluding the existence of other useful natural metaphors or the possibility of conceiving interesting non-natural metaphors, the main metaphors that can be adopted and have been suggested so far are: physical metaphors [6, 12], chemical metaphors [3, 14], biological metaphors [2, 4, 15], together with the most properly called ecological metaphors [1, 13].

	Species	*Space*	*Laws*
Physical	Particles (computational components) and messages (computational fields)	The Universe (a network), as shaped by waves and particles.	Navigation and activities driven by fields (gradient ascent by components)
Chemical	Atoms (semantically described) and Molecules (composed semantic descriptions)	Space (localities/bags of components)	Chemical Reactions (matching of semantic descriptions and bonding of components)
Biological	Cells (amorphous computing cells, modules of self-assembly components)	Space (Abstract computational landscapes, or physical landscapes)	Diffusion of chemical gradients and morphogens, differentiation of behaviour and activity
Ecological	Organisms (Agents) and Species (Classes) and Resources (Data)	Niches (Pervasive computing environments)	Survive (goal-orientation), eat, produce, and reproduce

Figure 2. Metaphors for Service Ecosystems

As far as we know, none of these metaphors has been so far adopted to extensively studying and prototyping an actual, open and general-purpose service framework: either the metaphor has been applied to specific application scenarios [12, 4, 15] or its potential general adoption has been only envisioned [6, 1].

Let us now come to the distinguishing characteristics of each metaphor, a summary of which is in Figure 2.

Physical metaphors consider that the species of the ecosystem are sort of computational particles, living in a world of other particles and virtual computational fields, which act as the basic interaction means. In fact, all activities of particles are driven by laws that determine how particles should be influenced by the local gradients and shape of some computational field (those whose description "matches" some criterion). In particular, they can change their status based on specific perceived fields, and they can move or exchange data by navigating over such fields (i.e., by having particles that move following the gradient descent of a field, or by making them spread sort of data particles to be routed according to the shape of fields). The space in which such particles live and in which fields spread and diffuse can be either a simple (euclidean) metric world, or it could be a sort of relativistic world, in which shapes and distances in the environment are not "inherent" but are rather shaped by fields themselves (as in gravitational space-time).

Chemical metaphors consider that the species of the ecosystem are sorts of computational atoms/molecules, with properties described by some sort of semantic descriptions which are the computational counterpart of the description of the bonding properties of physical atoms and molecules. Indeed, the laws that drive the overall behaviour of the ecosystem are sort of chemical laws. They dictate how chemical reactions and bonding between components take place (i.e., relying on some forms of pattern-matching between the semantic description of components), and can lead to both the production of aggregates (e.g., of aggregated distributed components) or of new components (e.g., of composite components). In this case, the space in which components live is typically formed by a set of localities, intended as the "solution" in which chemical reactions can occur, altough of course it is intended that components can flow/diffuse across localities to ensure globality of interactions.

Biological metaphors typically focus on biological systems at the small scale, i.e., at the scale of individual organisms (e.g., cells and their interactions) or of colonies of simple organisms (e.g. ant colonies). The species are therefore either simple cells or very simple (unintelligent) animals, that act on the basis of very simple goal-oriented behaviours (e.g., move and eat) and that are influenced in their activities by the strength of specific chemical signals in their surroundings to which they are sensitive to (i.e., with which there is a match). Similarly to physical systems, in fact, components are expected (depending on their status) to be able to spread and diffuse (chemical) signals around, that can then influence the behaviour of other components. The laws of the ecosystem together with the shape of the spatial computational landscape in which individuals live determine how such signals should diffuse, and how they could influence the behaviour and characteristics of components.

Ecological metaphors focus on biological systems at the level of animal species and of their interactions. The components of the ecosystem are sort of goal-oriented animals (i.e., agents) belonging to a specific species (i.e., agent classes), that are in search of "food" resources to survive and prosper (e.g., specific resources or other components matching specific criteria). The laws of the ecosystem determine how the resulting "web of food" should be realised, that is, they determine how and in which conditions animals are allowed to search food, eat, and possibly produce and reproduce, thus influencing and ruling the overall dynamics of the ecosystem and the interaction among individuals of different species. Similarly to chemical systems, the shape of the space is typically organized around a set of localities, i.e., of ecological niches (think at a set of local pervasive computing environments), yet enabling interactions and diffusion of species across niches.

4 Critical Analysis

As already stated in the introduction, a pervasive service ecosystem should be able to exhibit features of self-organization and self-adaptation (i.e., the capability of autonomously and adaptively self-organize and self-adapt the distributed spatial activities of the components) and self-management (here mostly intended as the possibility of ex-

erting control and directing the behavior of the system form within the system itself, a fundamental feature not to lose control over the system and not to be forced to introduce complex management solutions), and should tolerate evolution and adaptation over time (i.e., should adaptively accommodate new species, should survive the extinction of species, and should be capable of accommodating very diverse and composite behaviour with the same limited set of eco-laws). All of these features, of course, should be enforced without paying the price of dramatically increasing the complexity of the ecosystem, i.e., the number and complexity of eco-laws, and the structure of its components and of the space in which they live.

The analysis of the extent to which the presented metaphors can be able to accommodate (and how easily and naturally) the above features is very complex, and would require much more room than the few pages of this paper. Nevertheless, we can try at least to draw some considerations about this.

Physical metaphors have been extensively studied for their spatial self-organization features, and in particular for their capability of facilitating the achievement of coherent behaviours even in large scale system (e.g., for load balancing and data distribution), and the conceptual tools available for controlling the spatial behaviour and the dynamics of such systems are well-developed. However, the physical metaphor seems to fall short in evolution and time adaptation, in that it hardly tolerates the presence of very diverse components with very diverse behaviours (at least if we want to preserve the simplicity of the eco-laws).

Chemical metaphors, on the other hand, can effectively lead to local self-organizing structures (e.g., local composite services) and, to a more limited extent, to some sorts of global structures (e.g., networks of distributed homogeneous components, as in crystals). Real chemistry, and so chemical computational metaphors, can accommodate an incredible amount of different components and composites, yet with the same set of simple basic laws. This is an important pre-condition for facilitating evolution over time. As far as self-management is concerned, one can think at using sort of catalyst or reagent components to control the dynamics and the behaviour of a chemical ecosystem.

Biological metaphors appear very flexible in enabling the spatial formation of localised morphological and activity patterns, and this has been shown to have notable applications in a variety of applications to distributed systems. However, the number of patterns that can be enforced by the spread of chemical gradients and by the reactions of simple individuals seems (as it is in physical metaphors) quite limited, and this does not match with the need for time evolution and adaption. Moreover, it is quite difficult to understand how to properly control the overall behavior of such systems (just think at the fact that, so far, the mechanisms

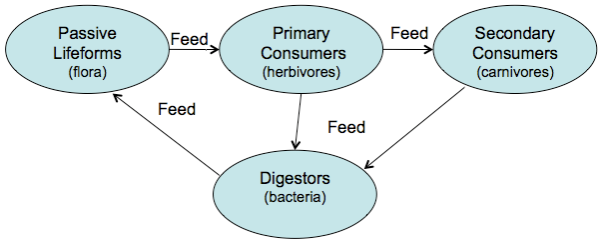


Figure 3. Key Elements for an Ecological System

of morphogenesis are not fully understood by scientists).

Ecological metaphors, the same as chemical ones, promise to be very suitable for local forms of spatial self-organization (think at equilibria in ecological niches), and are particularly suited for modeling and tolerating evolution over time (think at how biodiversity has increased over the course of evolution, without ever mining the health existence of life in each and every place on earth). However, unlike chemical systems, understanding how to properly control the local and global equilibria of real ecological system is a difficult task, and it would probably be very difficult also in their computational counterparts.

In summary, it is very difficult to assess once and for all which of the metaphors is the best for next generation of adaptive service ecosystems. Some exhibit suitable features for certain aspects, but fall short for others.

Personally, and having already extensively studied in the physical metaphor in the past [12], we are now very interested in studying both the chemical metaphor (see for a preliminary study about [14]) and the ecological one (which is the specific subject of the PhD studies of the first author, and which is detailed in the next section), and in possibly ending up with a sound new "hybrid" metaphor, getting the best of the above.

5 Our Current Approach and Research Agenda

As from Section 3, the development of a pervasive service ecosystem inspired by the ecological metaphor should conceive the individuals within as the life forms of an ecosphere, each of which having the trivial ego-centric goal of surviving by finding the appropriate food and resources. The eco-laws thus reduce to simply ruling the dynamics of the food web (who eats who and when), and the spatial structure of the system (typically structured around spatially confined ecological niches) determines how life forms can find and look for food.

In general, an ecological system can consider the presence of different classes of living forms (see Figure 3). Pas-

sive life forms (i.e., the flora system) do not actively look for food, although their existence and survival must be supported by nutrients that have to be spread in space. Primary consumers (i.e., herbivors) need to eat vegetables to survive and prosper. Secondary consumers (i.e. carnivors) typically need to eat other animals to survive, though this does not exclude that can also act as primary consumers (eating vegatables too). The result of the metabolization of food by both primary and secondary consumers typically ends up in feeding lower-level "digestors" life forms (e.g., bacteria), densely spread in space, and that in their turn produce and diffuse necessary resources and nutrients for the flora.

Let us now translate the above concepts in computational terms. Passive life forms represent the data sources of the ecosystem, which are not to be considered proactive computational entities, i.e., they do not need to "do something" to exist and be used. Primary consumers represent those services that require to digest information to be of any use, and yet are computationally autonomous (they do not require external computational functionalities). Secondary consumers, instead, are those services that, to be of any use, need the support of other services (whether primary or secondary in their turn), other than possibly of information sources. Digestors can be generally assimilated to all those background computational services that are devote to monitor the overall activities of the system, and either produce new information about or influence the existing information.

To better clarify, let us present a simple case study we are currently in the process of developing.

Consider a scenario like a thematic park or an exhibition center, densely pervaded with digital screens where to display information, movies, advertisements, or whatever. We can consider each of these screens (i.e., the computational resources associated with each of them) as a spatially confined ecological niche. Different classes of visitors will watch these screens to look for different types of information (intended as passive life forms). Thus, we can think at sort of "user agents" executing on the users' PDAs that, once in the proximity of a screen (i.e., while finding themselves into that specific ecological niche) start looking for specific information to eat (i.e., to have it displayed). User agents would thus act as primary consumers. Concurrently, we can think at "advertising agents" that, acting on behalf of some advertising company, roam from screen to screen in search of specific classes of user agents (i.e. those interested in specific types of information), with the ultimate goal of displaying advertisements there where they could be more effective. Advertising agents would this act as secondary consumers. Background monitoring agents, executing on each ecological niche and possibly interacting with each other, can contribute replicating and spreading information there where it appears to be more appreciated, and can also contribute in supporting the spatial roaming of advertiser agents by directing them there where they could find more satisfaction. Thus, they would act as digestors.

The feedback loop that derives from the above activities can contribute the properly rule the overall dynamics of the screen ecosystem, by continuously self-organizing and self-adapting the way information flows in the system, as well as the way advertising agents move, act, and coordinate with each other. The possibility of exerting control over the dynamics of the system is ensured by the possibility of injecting in the system new classes of "digestor agents" that can radically influence the dynamics of information diffusion and the activities of advertising agents. The adaptation of the system over time is ensured by the fact that it is mostly irrelevant, for the overall functioning of the system, what specific classes of information user agents want, or what the specific goal of advertising agent is. In fact, independently of the specific species of life forms that will populate the system, the basic eco-laws will ensure that such life forms will either find their way of living and their role in the system (e.g., as it can be the case of useful information and of advertising agents that find appropriate users to which to display their ads), or will simply disappear (as it can be the case of useless information or of advertisements no users is interested in).

We personally believe that, within the above simple conceptual framework (mostly in line with that envisioned in [1]), we will be able to identify a simple yet usable general-purpose model for the design and development of specific data/service/control components, and will be able to develop a practical software framework for the execution of a wide class of distributed spatial services for pervasive environment. To this end, we are currently in the process of:

- Trying to identify a proper semantic representation of the needs and characteristics of each life forms (what food one agent class needs, and what kinds of nutrient it can represent to others);
- Define a simple agent-inspired computational model to have the action of agents, as well as their propagation and diffusion over space, driven by a simple set of "eat to survive" eco-laws;
- Implement a simple "middleware" infrastructure to test and put at work our ideas. Such middleware will typically rely on a spatially-structured network of nodes, each of which acting as the basic ecological niche for the local execution of components and their interactions, and interacting with close niches to enforce spatial diffusion and propagation of life forms;
- Put our ideas at work in a variety of application scenarios, to verify their generality and their extent of applicability;

- Evaluate how and to which extent to integrate and extend the sketched ecological approach with features and characteristics inspired by other metaphors, if at all needed.

6 Conclusions

In this paper, we have elaborated on the idea of getting inspiration from natural ecosystem to model and develop next generation pervasive service framework. That is, of conceiving future pervasive service frameworks as a spatial ecosystem in which services, data items, and resources are all modeled as autonomous individuals that spatially act and interact in accord to a simple set of well-defined "laws of nature". In this way, it is possible to deliver self-organization, self-adaptation, self-management, and long-lasting evolvability as inherent properties of the framework, rather than as complicated ad-hoc solutions.

The road towards the actual deployment of usable and effective pervasive service ecosystems, however, still requires answering to several challenging questions. Among the others: what actual metaphor is the best to be adopted among the possible ones? What should be the actual modeling of individual and of eco-laws? What should be the actual shape and properties of the space in which individual will live and interact? And how can we practically implement this? Finding at least some of these answers is the current goal of our research work.

References

[1] G. Agha. Computing in pervasive cyberspace. *Commun. ACM*, 51(1):68–70, 2008.

[2] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes. Design patterns from biology for distributed computing. *ACM Trans. Auton. Adapt. Syst.*, 1(1):26–66, 2006.

[3] A. P. Barros and M. Dumas. The rise of web service ecosystems. *IT Professional*, 8(5):31–37, 2006.

[4] J. Beal and J. Bachrach. Infrastructure for engineered emergence on sensor/actuator networks. *IEEE Intelligent Systems*, 21(2):10–19, 2006.

[5] G. Castelli, A. Rosi, M. Mamei, and F. Zambonelli. A simple model and infrastructure for context-aware browsing of the world. *Pervasive Computing and Communications, 2007.*, pages 229–238, 19-23 March 2007.

[6] J. Crowcroft. Toward a network architecture that does everything. *Commun. ACM*, 51(1):74–77, 2008.

[7] D. Estrin, D. Culler, K. Pister, and G. Sukjatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59 – 69, 2002.

[8] S. Herold, H. Klus, D. Niebuhr, and A. Rausch. Engineering of it ecosystems: design of ultra-large-scale software-intensive systems. In *ULSSIS '08: Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*, pages 49–52, New York, NY, USA, 2008. ACM.

[9] M. N. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[10] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[11] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. Case studies for self-organization in computer science. *Journal of Systems Architecture*, 52(8-9):443–460, 2006.

[12] M. Mamei and F. Zambonelli. *Field-based Coordination for Pervasive Multiagent Systems*. Springer Verlag, 2006.

[13] M. D. Peysakhov, R. N. Lass, and W. C. Regli. Stability and control of agent ecosystems. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1143–1144. ACM, 2005.

[14] R. Quitadamo, F. Zambonelli, and G. Cabri. The service ecosystem: Dynamic self-aggregation of pervasive communication services. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, page 1, Washington, DC, USA, 2007. IEEE Computer Society.

[15] W.-M. Shen, P. Will, A. Galstyan, and C.-M. Chuong. Hormone-inspired self-organization and distributed control of robotic swarms. *Autonomous Robots*, 17(1):93–105, 2004.

[16] M. Ulieru and S. Grobbelaar. Engineering industrial ecosystems in a networked world. In *5th IEEE International Conference on Industrial Informatics*, pages 1–7. IEEE Press, 23-27 June 2007.

Ontology Agents in FIPA-compliant Platforms: a Survey and a New Proposal

Daniela Briola
DISI, Univ. di Genova,
Via Dodecaneso 35, 16146, Ge, IT
E-mail: daniela.briola@unige.it

Angela Locoro
DIBE, Univ. di Genova,
Via Opera Pia 11/A, 16146, Ge, IT
E-mail: angela.locoro@unige.it

Viviana Mascardi
DISI, Univ. di Genova,
Via Dodecaneso 35, 16146, Ge, IT
E-mail: viviana.mascardi@unige.it

***Abstract*—In 2001, FIPA delivered a specification suggesting that each MAS should integrate an "Ontology Agent" (OA) offering services for ontology management. These services should include ontology discovery, maintenance, matching, as well as translation of expressions between different ontologies or content languages. Currently, no FIPA-compliant OA exists that implements all of them. One of the reasons is that providing a service for ontology matching is not an easy task, and coping with translation between ontologies and/or content languages may be even harder. In this paper we survey the state of the art in the area, and we describe our prototypical implementation of an OA for Jade able to match ontologies. Besides "standard" ontology matching algorithms, our OA offers a "matching via upper ontologies" method that, as we showed in a recent technical report, improves the precision of the matching w.r.t. the use of traditional techniques.**

I. Introduction

In 2001, FIPA delivered a specification for organizing and managing ontologies in a MAS [7]. This specification suggests that each MAS should integrate an "Ontology Agent" (OA) providing services to deal with ontologies. The OA should be at the same conceptual level as the Directory Facilitator Agent, and should be able to deal with ontologies explicitly represented in some ontology language, and stored somewhere (perhaps in some server), where agents can access, query, and in case update them.

In particular, an OA should offer the following services to the agents in the MAS:

1) discover public ontologies in order to access them,
2) maintain (for example, register with the Directory Facilitator, upload, download, and modify) a set of public ontologies,
3) translate expressions between different ontologies and/or different content languages,
4) answer queries about relationships between terms or between ontologies,
5) facilitate the identification of a shared ontology for communication between two agents.

It's not mandatory for an OA to be able to realize all these services, provided that it is able to answer that it cannot process the required service.

The FIPA specification assumes that each ontology accessible through the OA's services adheres to the OKBC model [19]. OKBC supports an object-oriented representation of knowledge and provides a set of representational constructs commonly found in object-oriented knowledge representation systems. This standard is widely used there, but differs from the standards commonly accepted in the semantic web area, where OWL [27] is the most widespread model. The RDF, RDFS, and OWL languages are represented as a graph of triples $<$*object, property, subject*$>$. The semantics of OWL is based on Description Logic [2]. Even when the modelling primitives look similar to OKBC, the semantics is different, thus converting the operations associated to an OWL ontology into OKBC operations is not feasible. The lack of support to OWL ontologies is one of the main limitations of the FIPA OA specification, and motivates the lack of fully implemented FIPA-compliant OAs.

Another reason why no OAs exist that implement *all* the services suggested by the FIPA specification is that "*answering queries about relationships between terms or between ontologies*", as the specification suggests, is definitely an hard task if we consider *semantic* relationships, and not just *structural* ones. Answering a query on the structure of ontologies, such as "Is concept $c_1 \in o$ a subconcept of $c_2 \in o$ (or even a subconcept of $c' \in o'$)?" is almost trivial; answering a query on the semantics of terms, such as "What is the confidence in $c \in o$ and $c' \in o'$ having the same meaning?" is much more difficult. This activity requires that the OA is able to "match" ontologies o and o', namely, it is able to compute an "ontology alignment" between them.

Many algorithms for ontology matching exist, and some of them have been implemented and are available to the research community[1]. Surprisingly, none of them has been integrated into an OA. Thus, to the best of our knowledge, no existing OAs offer services for ontology matching.

In this paper we describe our implementation of an OA for Jade, able to provide both "standard" and new ontology matching services, as well as services for comparing one or more alignments to a reference one. In particular, our OA offers services for ontology matching via upper ontologies, a new approach that has proven to give good results in precision

[1]For example, the Alignment API developed by J. Euzenat and his team at INRIA-Rhône Alpes is available at http://alignapi.gforge.inria.fr/ under GNU Lesser General Public License. It provides string-based and simple language-based ontology matching methods, as well as methods for computing precision and recall of a given alignment w.r.t. a reference one.

and recall [16].

Our OA is far from implementing all the services suggested by the FIPA specification as, at the time of writing, it only offers the matching one. Since most of the existing FIPA-compliant OAs provide services for ontology discovery and interrogation, whereas none of them provides a service for ontology matching, we started our research by implementing the latter, in order to complement existing proposals. As other researchers, we deviate from the FIPA specification since we assume that ontologies are represented in OWL instead than OKBC.

The paper is organized as follows: Section II discusses the state of the art of FIPA-compliant OAs, after providing a short background on ontology matching and upper ontologies. Section III describes the functionalities of our OA, briefly presents the algorithms for matching OWL ontologies, discusses the implementation of a simple MAS consisting of a Request Agent and one OA, and shows experimental results. Finally, Section IV concludes and highlights future improvements of our work.

II. Background

A. Ontology Matching

A formalization of the ontology matching process can be found in [6]. Quoting the authors, we define a matching process as *"a function f which takes two ontologies o and o', an input alignment a, a set of parameters p and a set of oracles and resources r, and returns an alignment a' between o and o'"*.

A correspondence (also named "mapping") between an entity e belonging to ontology o and an entity e' belonging to ontology o' is a 5-tuple $< id, e, e', R, conf >$ where:

- id is a unique identifier of the correspondence;
- e and e' are the entities (e.g. properties, classes, individuals) of o and o' respectively;
- R is a relation such as "equivalence", "subsumption", "disjointness", "overlapping", holding between the entities e and e';
- $conf$ is a confidence measure (typically in the [0;1] range) holding for the correspondence between the entities e and e'.

An alignment of ontologies o and o' is a set of correspondences between entities of o and o'.

Among the matching techniques, we just discuss those that fall under the "Granularity / Input Interpretation" classification described in [6], based on the granularity of the matcher and on the interpretation of the input information.

- *String-based methods.* These methods measure the similarity of two entities just looking at the strings (seen as mere sequences of characters) that label them. Among them we may cite substring distance, where two strings are compared to find the longest common substring, n-gram distance [4], where two strings are the more similar the more n-grams (sequence of n characters) they have in common, and SMOA measure [24], which is a function of the commonalities (in terms of substrings) as well as of differences between two strings.
- *Language-based methods.* These methods exploit natural language processing techniques to find the similarity between two strings seen as meaningful pieces of text rather than sequences of characters. Some of them exploit external resources like WordNet, and exploit the semantic relations that it offers to compute the correspondences.

B. Upper Ontologies and their Application to Ontology Matching

An upper ontology (also named top-level ontology, or foundation ontology) is "*an attempt to create an ontology which describes very general concepts that are the same across all domains*" [28]. Few upper ontologies exist: BFO [10], Cyc [13], DOLCE [9], GFO [11], PROTON [5], Sowa's ontology [23], and SUMO [17]. They vary in dimension, ranging from the 30 classes of Sowa's ontology to the 300,000 of Cyc, in representation language (OWL, KIF [1], First Order Logic), in structure (monolithic vs. decomposed into modules), and in developed applications. Nevertheless, all of them describe general concepts (also named "classes") and share the aim to have a large number of ontologies accessible under them.

In our previous work, we implemented different algorithms that used upper ontologies for boosting the ontology matching process [16]. We run experiments with SUMO-OWL (a restricted version of SUMO translated into OWL), OpenCyc (the open version of Cyc, which is a commercial ontology), and DOLCE. The experiments demonstrate that when the "structural matching method via upper ontology" uses an upper ontology large enough (OpenCyc, SUMO-OWL), the recall is significantly improved and the precision is kept w.r.t. not using upper ontologies. Instead, the "non structural matching method" via OpenCyc and SUMO-OWL improves the precision and keeps the recall. The "mixed method", that combines the results of structural alignment without using upper ontologies and structural alignment via upper ontologies, improves the recall and keeps (improves, with OpenCyc) the F-measure, whatever the upper ontology used.

C. FIPA-compliant OAs: the State-of-the-Art

The FIPA reference model for the services provided by the OA is shown in Figure 1.

In the literature there have been few attempts to realize the FIPA OA, and each one adopts a particular point of view of the problem.

Some solutions implement only a subset of the OA's services, others change the FIPA specification in order to use the OWL specification language, others realize a Web Service playing the OA role. Besides being different from the design point of view, these solutions also differ in the choice of the middleware where the OA is integrated, and hence of the language used to implement it.

In the following sections we discuss the most relevant solutions for the integration of the OA in a FIPA compliant framework.

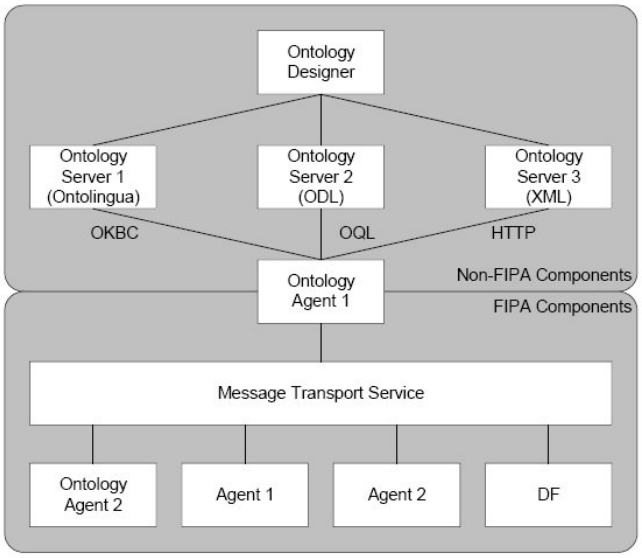


Figure 1. FIPA Ontology Service Reference Model

Implementation over the COMTEC platform

The first attempt to realize an OA was made in 2001 by Suguri, Kodama, Miyazaki [25]. They realized an OA for the COMTEC platform.

The OA is divided into two parts. The first part is an interface to the OKBC front-end. From the OKBC point of view, the OA is one of the front-end user applications. The second part is the FIPA interface where the agent wrapper is implemented. The FIPA interface is an agent wrapper that takes care of generating and interpreting SL [8] actions, predicates and ACL communicative acts based on appropriate interaction protocols. It also processes the registration with the DF, the management of ontology names and the relationship between the ontologies.

The COMTEC OA implements a subset of the services of a generic FIPA-compliant OA, in particular

1) register an ontology in the framework;
2) operate over an ontology (create, delete frames, slot, modify the hierarchies);
3) answer queries about ontologies's structure, and their level of similarity.

No ontology matching service is provided by this OA.

From a design point of view, this solution is one of the best because it tries to faithfully adhere to the FIPA specification. Its main problem is that the COMTEC platform on which it was implemented is no longer available.

Implementation over the AgentService platform

Vecchiola, Grosso, and Boccalatte implemented a FIPA compliant framework called AgentService [26], based on the .NET platform. Together with Passadore, they integrated an OA into AgentService [20].

Ontologies in AgentService are represented in OKBC: the implementation of the OA is thus fully compliant with the FIPA specification.

The services that the OA offers are a subset of the possible ones: the OA implements the discovery and publication of the ontology and its maintenance, allows two agents to check whether they use the same ontologies and if not, it helps them to download the "missing" ones. However, neither ontology matching nor translation are supported.

AgentService uses Protégé [22] to support the designer from the creation of the ontology to the development of an agent that can communicate using that ontology, and MS Visio (http://office.microsoft.com/en-us/visio/default.aspx) to design agent interaction protocols. Visio allows the designer to import an ontology and supports him/her in the creation of message contents compliant with concepts expressed in the ontology itself.

Thus, the main usefulness of having ontologies integrated into AgentService is to support developers in designing message exchange in a clear and well-founded way. This makes the design easily sharable among developers. AgentService OA has been conceived for managing ontologies within closed MASs, rather than for implementing open systems where heterogeneous/unknown agents interact and choose an ontology on the fly.

Implementation over the Jade platform

One of the most used FIPA-compliant platform is Jade [3]. This framework is fully compliant to the FIPA standards, and is based on Java. Jade offers some utilities to model ontologies, but in the spirit of integrating the ontology into the agent's code. Thus, Jade supports the hard coding of the ontology both at design and compile time: all the entities of the ontology have to be transformed in classes and objects, in order to allow a Jade agent to use them. This approach gives little help to a MAS developer who wants to create agents that can communicate with others automatically, after having agreed on an ontology known and available to all of them. Then, Jade lacks a real support to the use of ontologies in open MASs.

We have implemented an OA in Jade offering services for matching OWL ontologies using different algorithms, and for evaluating the result of the matching. Its functionalities are described in Section III.

The only other attempt of integrating a FIPA-compliant OA into Jade we are aware of, is that by Obitko and Snáěl [18]. Their implementation follows the FIPA specification but adapts it to ontologies represented in OWL, as we do.

Since Obitko and Snáěl intended to store OWL ontologies only, they had to adapt the language for describing actions performed by the ontology agent. Their OA agent exploits Jena [12] and implements the basic functionalities of the ontology services as specified in the FIPA proposal, i.e. the possibility of modifying ontologies (assert and retract) and of querying ontologies using RDQL.

Obitko and Snáěl's OA is well organized and closely follows the FIPA specification except for the usage of OWL instead to OKBC.

Non FIPA-compliant solutions

Ontology Services as the result of Distributed Cooperation: A good effort to design and implement a MAS with a support to ontology matching comes from Li, Wu and Yang [14], [15]. Their work concentrates on the process of mapping and integrating ontologies: these functionalities are integrated in the MAS thanks to a set of agents which collaborate to offer them to the other agents. The agents which are involved in the delivery of ontology services are:

1) User Agent (UA): assists the user in formulating his/her requests, posts queries (e.g. tasks) to the proposed system via the IA and visualises the required results according to the user's requirement. The UA only knows the IA.
2) Interface Agent (IA): acts as an interface among agents in the MAS and the UA. Every agent knows the IA.
3) Ontology Agent (OA): acts on behalf of the corresponding ontology, is in charge of ontology related tasks. It provides as much information of the ontology it acts on as possible. The OA operates over the ontology structure and the mapping result file. When a new ontology is loaded in the system, a corresponding OA is created to manage it.
4) Mapping Agent (MA): maps, if possible, concepts from an ontology to the concepts of a second ontology, using also the SA.
5) Similarity Agent (SA): maintains a thesaurus for the purpose of similarity. It holds a list of common words and synonyms of words.
6) Query Agent (QA): operates over the mapping results to investigate ontology-understandable of heterogeneous ontologies after executing ontology mapping.
7) Integration Agent (InA): merges two ontology in a new one (in RDF format). Is based on the result of ontology mapping.
8) Checking Agent (CA): checks the consistency of the integrated ontology (assuming all given ontology are consistent at the beginning).

The purposes of this system, namely providing a large set of ontology services that include ontology matching ones, make it very close to our proposal. However, the way these services are implemented make the system really far from the FIPA OA specification, since services are distributed among different agents, and are not integrated within a FIPA-compliant framework.

An OA implemented as a Web Service: In [21] Peña, Sossa and Gutierrez implement the OA as a web service, in order to offer its services also over the Internet. The OA carries out the management of the ontologies through an interface between the application agents and the ontologies. It is responsible for catching the requests arriving from the agents, interpreting them, forwarding them to the Ontology Manager in charge of the Ontology referenced in the request, and forward back the response from the Ontology manager. Ontologies are in OWL format, and each Ontology Manager answers only to requests about the structure of the ontology or for changing its structure, but no support is offered to the mapping, translating or more complex queries about two ontologies.

III. Our OA in Jade

The OA we have implemented in Jade provides the following services:

1) matching two OWL ontologies through a direct matching;
2) matching two OWL ontologies via an upper ontology (represented in OWL too);
3) evaluating an alignment against a reference alignment.

The implementation of a fourth service, namely the repair of an alignment based on word sense disambiguation techniques, is under way. We do not discuss it here for space constraints.

In this section, we describe the algorithms that realize all the implemented services of the OA. It is worth specifying that in our matching algorithms we only consider concepts as entities to match, and equivalence as relation.

In order to be compliant with the Align API mentioned in Section I, and to be able to exploit some of the methods it provides, our alignments are represented in RDF and their format is like the one shown below.

```
<rdf:RDF .. namespaces here ...>
<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>**</type>
  <time>15</time>
  <method>StringDistAlignment</method>
  <onto1>file:///ka.owl</onto1>
  <onto2>file:///edumit.owl</onto2>
  <uri1>file:///ka.owl</uri1>
  <uri2>file:///edumit.owl</uri2>
  ...
  <map> <Cell>
    <entity1 rdf:res=ka.owl#Book/>
    <entity2 rdf:res=edumit.owl#Book/>
    <relation>=</relation>
    measure>1.0</measure>
  </Cell> </map>
  <map> <Cell>
    <entity1 rdf:res=
       ka.owl#TechnicalReport/>
    <entity2 rdf:res=
         edumit.owl#Techreport/>
    <relation>=</relation>
    <measure>1.0</measure>
  </Cell> </map>
  ...
</Alignment>
</rdf:RDF>
```

With respect to the definition provided in Section II-A, our correspondences are simpler due to the lack of the correspondence identifier, which is strictly necessary only in case we

need to uniquely identify them (e.g. when storing them in a repository).

Direct Matching

The direct matching service is implemented by a function that we named $parallel\_match(o, o', \{WordNet\}, th)$, because it runs in parallel different "standard" matching methods, and combines their outputs. The matching methods that we used are the *substring*, *n-gram*, *SMOA* and *WordNet* ones that we introduced in Section II-A. We used the implementation of these methods offered by Euzenat's Align API[2]. The th parameter is used as a threshold for cutting all correspondences below it. In our experiments, we set it at 0.5. To obtain a final alignment from the ones obtained by the individual matching methods, we use an $aggregate$ function that composes the four alignments by making the union of all their correspondences. In case different alignments produced correspondences between the same concepts c and c', $aggregate$ keeps the correspondence with the highest confidence measure.

Matching via Upper Ontology

For matching two ontologies o and o' via an upper ontology uo we compute the $parallel\_match(o, uo, \{WordNet\}, th)$ and $parallel\_match(o', uo, \{WordNet\}, th)$, obtaining two alignments between o and uo, and o' and uo respectively. These two alignments are given in input to a $merge(a, a')$ function. $Merge$ produces the final alignment between o and o' by combining the correspondences of o-uo and o'-uo in such a way that: if $\exists$ a correspondence $< c, c_u, r, conf1 >$ in o-uo and $\exists$ a correspondence $< c', c_u, r, conf2 >$ in o'-uo, then the $merge$ function creates a new correspondence $< c, c', r, conf1 * conf2 >$ and adds it to the final alignment.

Alignment Evaluation

The Alignment API provided by Euzenat and colleagues offers a *PrecEval* method for measuring the goodness of an alignment based on precision, recall and F-measure. Precision is the number of correctly found correspondences with respect to the reference alignment (true positives), divided by the total number of found correspondences (true positives and false positives), and recall is the number of correctly found correspondences (true positives) divided by the total number of expected correspondences (true positives and false negatives). F-measure is the harmonic mean of precision and recall.

We have integrated the *PrecEval* method into our OA; an example of evaluation produced by the OA is given by the following RDF fragment:

```
<rdf:RDF ..namespaces here ..>
  <map:output rdf:about=''>
    <map:in1 rdf:resource="ka.owl"/>
    <map:in2 rdf:resource="edumit.owl"/>
    <map:precision>0.097</map:precision>
    <map:recall>0.084</map:recall>
    <map:fMeasure>0.09</map:fMeasure>
    <map:nbcorrect>7</map:nbcorrect>
   <map:nbfound>72</map:nbfound>
  </map:output>
</rdf:RDF>
```

Implementation and Experiments

In order to test the behaviour of our OA we implemented a Request Agent (RA) that acts as an interface between the user and the OA, and allows the user to request services to the OA. The RA receives a set of parameters from the user, and saves them for later use. These parameters are:

- the URI of the OWL file containing the first ontology to match, o;
- the URI of the OWL file containing the second ontology to match, o';
- the URI of the file where the computed alignment will be stored;
- the URI of the file containing the reference alignment for performing an evaluation of the computed alignment;
- the matching method ("direct matching" or "matching via upper ontology");
- a further optional parameter providing the URI of the file containing the upper ontology in OWL. This parameter is needed only in case of matching via upper ontology.

The system architecture is depicted in Figure 2.

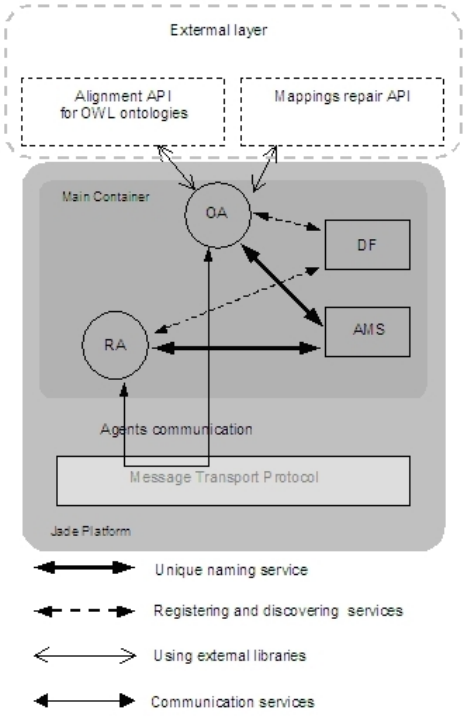


Figure 2. Architecture of the Ontology Agent integrated into Jade.

[2]In particular, we used the Alignment API version 3.1. The methods we used are *StringDistAlignment* that provides `subStringDistance`, `ngramDistance` and `smoaDistance` string metrics, and *JWNLAlignment* that computes a substring distance between two concepts exploiting their WordNet 3.0 synsets.

During its start up phase, RA searches and identifies an OA inside the Jade Platform.

After having received the parameters from the human user, and having found OA, RA:

- sends as many INFORM messages to the OA, as the parameters it wants to send to OA; the parameters are sent as the message content (one parameter for each message); the conversation id of the message is used to keep track of the conversation just started;
- sends a REQUEST message asking for the matching and evaluation services (which, in this first prototype, are always coupled); the ontologies to match, and the method to use, are those passed in the previous INFORM messages.

OA starts its life with the registration of its ontology services to the DF. When RA begins the interaction phase, OA reacts as follows:

- it receives all the parameters from the RA and sends an ACL Message in reply to each message, setting the same conversation id for correctly keeping track of the conversation, after having successfully received and saved each parameter;
- on reception of a REQUEST message, it runs the required matching method on the ontologies specified in the previous interaction steps with the RA;
- sends an INFORM message to notify the RA of the accomplishment of the matching service and to communicate the alignment file URI and the evaluation file URI; in case something goes wrong, it sends a FAILURE message to the RA.

Both agents terminate after the completion of these activities.

OA and RA have been structured according to Jade recommendations for the development of agents. RA's `setup()` method includes the steps to save parameters from the command line, the instantiation of a `wakerBehaviour` object in order to find the OA via DF query and the call to the `RequestPerformer` behaviour. This one includes the `action()` method where the agent sends the parameters, sends the alignment and evaluation requests, and waits for the results. It has been implemented as a generic `Behaviour` class.

OA's `setup()` method starts with the registration of OA's services to the DF and the instantiation of its two behaviours which are of type `ReceiveParameters` and `Align` respectively. The `action()` method of `ReceiveParameters` is dedicated to receive the parameters and has been implemented as a `OneShotBehaviour` class. The `action()` method of `Align` first waits for an ontology matching request. After the request message has been received, the matching and evaluation activities are performed, and the result is notified to the sender. This class has been modelled extending the generic `Behaviour` class.

To better synchronize the exchange of messages each of them is received by the two agents in blocking mode and the `MessageTemplate` class has also been used to deal with conversation id patterns.

For the development of our MAS consisting of the OA and the RA we used Jade 3.6.

In order to verify the feasibility of our approach, we have run a large set of experiments. We discuss only two of them in details; the other ones have been carried out in a similar way.

In the first experiment, the RA asks the OA to execute a direct alignment between *Ka* (protege.cim3.net/file/pub/ontologies/ka/ka.owl) and *Bibtex* (oaei.ontologymatching.org/2004/Contest/304/onto.rdf). *Ka* has 96 concepts dealing with the academic domain, including Event (Activity, Meeting, Conference, Workshop), Publication (concepts similar to those of the *Bibtex* ontology), Organisation (Department, Enterprise, Institute, University), ResearchTopic, whereas *Bibtex* has only 15 concepts including Article, Book, Conference, Manual, Person, Publisher, etc.

Before using *Ka* and *Bibtex* in our experiments, we preprocessed them by hand in order to remove properties and individuals that we do not use in our matching algorithms. We also built a reference alignment between *Ka* and *Bibtex* by hand because we wanted to evaluate how good the alignments resulting from the automatic matching methods were, w.r.t. the reference one.

We passed the URIs of the two ontologies to match as input parameters to RA together with the URI of the file that had to contain the alignment, the URI of the file that contains the reference alignment, and the chosen method (direct alignment).

The second experiment also aimed at aligning *Ka* and *Bibtex*, but using the "matching via upper ontology" method. The upper ontology we used is SUMO-OWL, a lossy translation into OWL of the SUMO ontology, whose full version is encoded in KIF.

The results of these two experiments are shown below; the table summarizes the total correspondences found w.r.t. the correct ones and the precision, recall and f-measure computed by the *PrecEval* method integrated into our OA.

Method	**Found**	**Correct**	**Prec.**	**Recall**	**F-meas.**
Direct	27	7	0.26	**0.08**	**0.13**
SUMO-OWL	9	6	**0.67**	0.07	**0.13**

While running this experiment, we activated the Jade sniffer agent to follow the interactions between RA and OA. A screenshot of the "sniffed" messages is given in Figure 3. The messages exchanged in the first and second experiment are very similar. In both experiments OA and RA exchange messages to pass and save parameters. When the parameter negotiation phase ends, the RA sends a request to OA for performing the matching and the evaluation. The task has been successfully executed in both experiments. As a consequence OA sends an INFORM message to RA to inform it of the successful outcome of the matching, and both agents terminate.

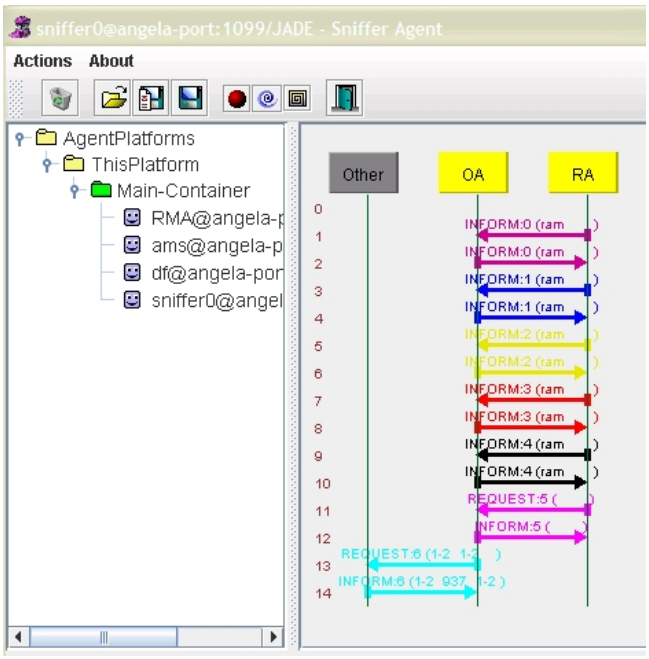


Figure 3. A sniffer screenshot with OA and RA agents.

The first experiment uses a direct matching method that took less than a minute to complete[3], while the second one required about 10 minutes due to the usage of a more sophisticated matching algorithm involving a large upper ontology (SUMO-OWL has 4,393 concepts). From a comparison of the alignments computed by the OA in the two experiments, which are reported in Table III, it turns out that matching *Ka* and *Bibtex* via SUMO-OWL gives the best precision (67% against 26% of the direct alignment), while recall and F-measure are the same (recall: 7% against 8%; F-measure: 13% for both).

We have run 9 more tests. Besides *Ka* and *Bibtex*, the ontologies that we have used in our experiments and the URLs from where they may be downloaded are:

- *Agent*, 212.119.9.180/Ontologies/0.2/agent.owl
- *Biosphere*, sweet.jpl.nasa.gov/ontology/biosphere.owl
- *Ecology*, wow.sfsu.edu/ontology/rich/EcologicalConcepts.owl
- *Food*, silla.dongguk.ac.kr/jena-owl1/food
- *Geofile*, www.daml.org/2001/02/geofile/geofile-ont.daml
- *HL7_RBAC*, lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/ontologies/HL7_RBAC.owl
- *MPEG7*, dmag.upf.es/ontologies/2003/03/MPEG7Genres.rdfs
- *Restaurant*, guru-games.org/ontologies/restaurant.owl
- *Resume*, statistic.gunadarma.ac.id/research/WorkGroupInformationSystem/DownLoad/onto_colection/resume.owl
- *Space*, 212.119.9.180/Ontologies/0.3/space.owl
- *Subject*, www.library.yale.edu/ontologies/subject.owl
- *Top-bio*, www.co-ode.org/ontologies/basic-bio/top-bio.owl

[3]We run our experiments on a HP Pavillon Notebook with Intel Core Duo T2250 processor, 1.73 GHz of clock, 2 GB of RAM, and Windows XP.

- *Travel*, lsdis.cs.uga.edu/projects/meteor-s/downloads/Lumina/ontology/travelontology.owl
- *Vacation*, www.guru-games.org/ontologies/vacation.owl
- *Vertebrate*, www.co-ode.org/ontologies/basic-bio/basic-vertebrate-gross-anatomy.owl (Vertebrate has been reduced by hand when running our experiments)

The experiments run consisted of matching the following couples of ontologies:

- **1:** Ka - Bibtex;
- **2:** Biosphere - Top-bio;
- **3:** Space - Geofile;
- **4:** Restaurant - Food;
- **5:** MPEG7 - Subject;
- **6:** Travel - Vacation;
- **7:** Resume - Agent;
- **8:** Resume - HL7_RBAC;
- **9:** Ecology - Top-bio;
- **10:** Vertebrate - Top-bio

The obtained results are summarized in Table I. In 7 experiments out of 10, matching via SUMO-OWL allowed us to obtain the best precision. In many cases, the recall of the two methods can be compared even if, in some experiments, the direct matching performs definitely better. The suitability of the "matching via upper ontologies" method strictly depends on the ontologies to match. From our experiments we have learnt that for example, the type of English words used by both upper and matched ontologies counts, as well as the terminology used by both upper and matched ontologies.

IV. Conclusions

This paper describes a FIPA-compliant Ontology Agent integrated in Jade, able to deal with OWL ontologies.

Our OA is able to produce alignments between two OWL ontologies via direct matching or via an OWL version of an upper ontology. It evaluates the obtained alignment with respect to a given reference one and is going to be extended with more "standard" services such as registration, discovery, and maintenance of ontologies. The exploitation of an upper ontology for performing an alignment between two ontologies is a new approach that we only described in a recent technical report. This approach represents an original contribution to the ontology matching process.

For what concerns our decision to match concepts only, and to limit ourselves to considering the equivalence relation, our intention is to extend the matching methods towards the exploitation of properties and individuals, and to take into consideration at least subsumption and disjunction relations.

Our OA can be improved through a deeper analysis of the FIPA-Agent-Management ontology in order to see if it is feasible to join the description of the services provided by OA with concepts of the Jade `BasicOntology`, thus allowing the matching process, the evaluation and the correspondences repair services to be a part of the Jade Agents semantic. Future extensions we would like to carry on are: the modelling of more complex behaviours which would better reflect the

Test	Method	Found	Correct	Prec.	Rec.	F-meas.
1	Manual	83	83	1.00	1.00	1.00
1	Direct	27	7	0.26	**0.08**	**0.13**
1	SUMO-OWL	9	6	**0.67**	0.07	**0.13**
2	Manual	604	604	1.00	1.00	1.00
2	Direct	26	6	**0.23**	**0.01**	**0.02**
2	SUMO-OWL	2	0	0.00	0.00	0.00
3	Manual	513	513	1.00	1.00	1.00
3	Direct	164	38	0.23	**0.07**	**0.11**
3	SUMO-OWL	49	22	**0.45**	0.04	0.08
4	Manual	1041	1041	1.00	1.00	1.00
4	Direct	107	12	0.11	0.01	0.02
4	SUMO-OWL	82	28	**0.34**	**0.03**	**0.05**
5	Manual	637	637	1.00	1.00	1.00
5	Direct	323	94	0.29	**0.15**	0.20
5	SUMO-OWL	224	93	**0.42**	**0.15**	**0.22**
6	Manual	262	262	1.00	1.00	1.00
6	Direct	50	19	**0.38**	**0.07**	**0.12**
6	SUMO-OWL	26	8	0.31	0.03	0.06
7	Manual	1122	1122	1.00	1.00	1.00
7	Direct	157	58	0.37	**0.05**	**0.09**
7	SUMO-OWL	71	31	**0.44**	0.03	0.05
8	Manual	295	295	1.00	1.00	1.00
8	Direct	127	36	0.28	**0.12**	0.17
8	SUMO-OWL	60	34	**0.57**	**0.12**	**0.19**
9	Manual	308	308	1.00	1.00	1.00
9	Direct	88	28	**0.32**	**0.09**	**0.14**
9	SUMO-OWL	140	17	0.12	0.06	0.08
10	Manual	19	19	1.00	1.00	1.00
10	Direct	12	2	0.17	**0.11**	0.13
10	SUMO-OWL	7	2	**0.29**	**0.11**	**0.15**

Table I
COMPLETE RESULTS OF OUR EXPERIMENTS

alignment, evaluation and correspondences repair tasks (i.e. parallel behaviours for string based and language based partial alignments which could be nested inside a sequential behaviour able to produce the final alignment and the merged alignment via an upper ontology), the extension of OA life to prolong the availability of these services for the whole time the platform is running and the exploitation of FIPA standard interaction protocols to manage the conversation between our agents. Separating the alignment process from the evaluation one by providing different requests for alignment, evaluation and correspondences repair, hence exploiting a modular architecture, is a further goal we will pursue.

ACKNOWLEDGMENTS

The authors acknowledge the "Iniziativa Software" CINI-FINMECCANICA project that partially funded this work.

REFERENCES

[1] American National Standard. KIF Knowledge Interchange Format – dpANS NCITS.T2/98-004, 1998.
[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
[3] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. WILEY, 2007.
[4] E. Brill, S. Dumais, and M. Banko. An analysis of the askmsr question-answering system. In *Conference on Empirical Methods in Natural Language Processing, EMNLP 2002, Proceedings*, 2002.
[5] N. Casellas, M. Blázquez, A. Kiryakov, P. Casanovas, M. Poblet, and R. Benjamins. OPJK into PROTON: Legal domain ontology integration into an upper-level ontology. In R. Meersman and et al., editors, *WORM 2005, 3rd International Workshop on Regulatory Ontologies, Proceedings*, volume 3762 of *LNCS*, pages 846–855. Springer, 2005.
[6] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer, 2007.
[7] *FIPA Ontology Service Specification*, 2001. http://www.fipa.org/specs/fipa00086/XC00086D.pdf.
[8] *FIPA SL Content Language Specification*, 2002. http://www.fipa.org/specs/fipa00008/SC00008I.html.
[9] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with DOLCE. In A. Gómez-Pérez and V. R. Benjamins, editors, *EKAW, 13th International Conference, Proceedings*, volume 2473 of *LNCS*, pages 166–181. Springer, 2002.
[10] P. Grenon, B. Smith, and L. Goldberg. Biodynamic ontology: Applying BFO in the biomedical domain. In D. M. Pisanelli, editor, *Ontologies in Medicine*, volume 102 of *Studies in Health Technology and Informatics*, pages 20–38. IOS Press, 2004.
[11] H. Herre, B. Heller, P. Burek, R. Hoehndorf, F. Loebe, and H. Michalek. General formal ontology (GFO): A foundational ontology integrating objects and processes. part i: Basic principles. Technical report, Research Group Ontologies in Medicine (Onto-Med), University of Leipzig, 2006. Version 1.0, Onto-Med Report Nr. 8, 01.07.2006.
[12] *JENA*. http://jena.sourceforge.net/.
[13] D. Lenat and R. Guha. *Building large knowledge-based systems*. Addison Wesley, 1990.
[14] L. Li. Agent-based ontology management towards interoperability. Master's thesis, Swinburne University of Technology, 2005.
[15] L. Li, B. Wu, and Y. Yang. Agent-based ontology integration for ontology-based application. In *AOW 2005, associated with the 18th CRPIT Conference series by Australian Computer Society, Vol 58*, pages 53–59, 2005.
[16] V. Mascardi, A. Locoro, and P. Rosso. Exploiting DOLCE, SUMO-OWL, and OpenCyc to boost the ontology matching process. Technical Report DISI-TR-08-08, University of Genoa, 2008. http://www.disi.unige.it/person/MascardiV/Software/OntologyMatchingViaUpperOntology.html.
[17] I. Niles and A. Pease. Towards a standard upper ontology. In C. Welty and B. Smith, editors, *FOIS 2001, 2nd International Conference on Formal Ontology in Information Systems, Proceedings*, pages 2–9. ACM Press, 2001.
[18] M. Obitko and V. Snáěl. Ontology repository in multi-agent system. In M. H. Hamza, editor, *Artificial Intelligence and Applications, AIA 2004, Proceedings*, 2004.
[19] *OKBC*. http://www.ai.sri.com/~okbc/.
[20] A. Passadore, C. Vecchiola, A. Grosso, and A. Boccalatte. Designing agent interactions with Pericles. In *ONTOSE 2007, 2nd International Workshop*, 2007.
[21] A. Peña, H. Sossa, and F. Gutierrez. Web-services based ontology agent. In *Distributed Frameworks for Multimedia Applications, 2006. The 2nd International Conference on*, pages 1–8, 2006.
[22] *Protégé*. http://protege.stanford.edu/.
[23] J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing, 1999.
[24] G. Stoilos, G. B. Stamou, and S. D. Kollias. A string metric for ontology alignment. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *4th International Semantic Web Conference, ISWC 2005, Proceedings*, volume 3729 of *Lecture Notes in Computer Science*, pages 624–637. Springer, 2005.
[25] H. Suguri, E. Kodama, M. Miyazaki, H. Nunokawa, and S. Noguchi. Implementation of FIPA Ontology Service. In *Workshop on Ontologies in Agent Systems, Proceedings*, 2001.
[26] C. Vecchiola, A. Grosso, and A. Boccalatte. AgentService: a framework to develop distributed multi-agent systems. *Int. J. Agent-Oriented Software Engineering*, 2(3):290 – 323, 2008.
[27] W3C. OWL Web Ontology Language Overview – W3C Recommendation 10 February 2004, 2004.
[28] Wikipedia. Upper ontology – Wikipedia, the Free Encyclopedia, 2008. [Online; accessed 30-March-2008].

From Agents to Artifacts Back and Forth: Operational and Doxastic use of Artifacts in MAS

Michele Piunti
Università degli studi di Bologna
DEIS - Bologna, Italy
Email: michele.piunti@unibo.it

Alessandro Ricci
Università degli studi di Bologna
DEIS - Bologna, Italy
Email: a.ricci@unibo.it

Abstract— **Recent approaches in Multi-Agent Systems are focusing on providing models and methodologies for the design of environments and special purpose tools supposed to ease programming in the large and scale up growing complexities. Among others, the Agents and Artifacts (A&A) approach introduced the notion af artifact as first class abstraction providing agents with external facilities, services and coordination medium explicitely conceived for promoting their activities. In this paper we analyse A&A systems by focusing on the *functional roles* played by artifacts. In particular, we here investigate the function of artifacts once they are employed in the context of societies of cognitve agents, i.e. agents capable to reason about their epistemic and motivational states. In this context, a twofold kind of interaction is envisaged. On the one side, artifact rapresentational function allows agent to improve epistemic states, i.e., by representing and sharing strategic knowledge in the overall system (*doxastic use*). On the other side, artifacts operational function allows agents to improve the repertoire of actions, i.e., by providing additional means which can be purposively triggered by agents to achieve goals (*operational use*). Some of the outcomes of this approach are discussed along with test cases showing agents engaged in goal-oriented activities relying on the transmission of relevant knowledge and the operations provided by artifacts.**

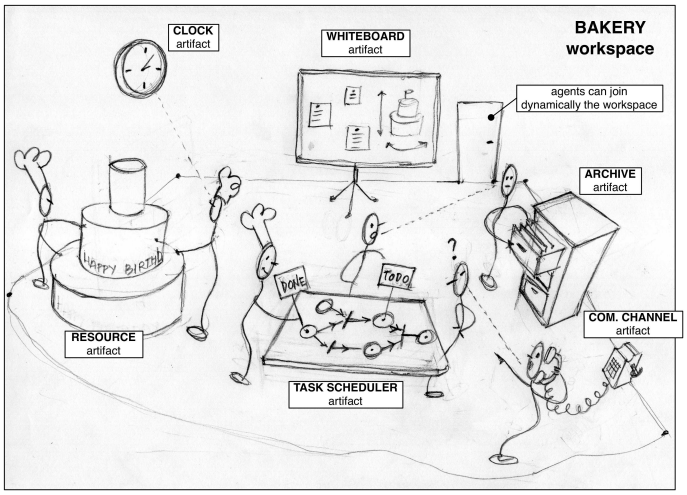


Fig. 1. A metaphorical representation of a MAS according to A&A.

I. Introduction

The *artifact* abstraction has been recently introduced in Multi-Agent System (MAS) [13] and MAS programming [20] as a basic building block to model and design agent environments and, more exactly, agent *work environments*. The notion of work environment used here refers to that part of the MAS – so developed by MAS designers and developers – which is perceived and used by agents as a first-class entity of their world, and which provides suitable functionalities and services that agents can exploit to ease their individual and social activities [18]. Artifacts – as introduced by the A&A conceptual model – can be conceived as basic module to structure such work environments, representing *non-autonomous* computational objects[1] that agents can dynamically instantiate, share and use as *resources* and *tools* to support and promote their activities. Mutuating the notion of ecosystems[2] or human societies, where individuals are supposed to behave and interact by means of shared knowledge, cultural transmission, memes [6], A&A states a clear separation of concern between the entities dwelling in a MAS: whereas agents can be considered as the proactive actors of the systems, exhibiting a purposive and autonomous behavior, artifacts are the non-autonomous entities, providing agent with facilities and special purpose tools to be exploited as external resources in order to serve a functional role [14]. According to this view, a MAS is designed and programmed in terms of an ensemble of agents that play together in a common (work) environment not only by communicating through some high-level Agent Communication Languages, but also co-constructing and co-using different kinds of artifacts, organised in *workspaces* (Figure 1 shows a metaphorical picture of a MAS in this perspective). The main source of inspiration underlying this view comes from human societies and research works in Activity Theory [12], remarking the fundamental role that play artifacts in our society in mediating and supporting human work, in particular cooperative work. Besides A&A, CARTAGO [20] has been introduced as a platform and infrastructure providing a concrete computational model to program artifacts and a distributed runtime for executing artifact-based work environments, making it possible for agents developed on different agent platforms to dynamically join and work inside such environments [18].

[1]The notion of object is used here in its general term, meaning a dynamic entity with a proper identity

[2]Introduced by Cristopherson in [5], ecosystem has been defined as "a natural unit consisting of all plants, animals and micro-organisms (biotic factors) in an area functioning together with all of the non-living physical (abiotic) factors of the environment".

At first, for MAS designers the usefulness of the artifact abstraction concerns the availability of an explicit level of abstraction and technologies – based on A&A concepts – for modeling, design and programming work environments for different kinds of purposes. This is the main line followed by most of the existing work exploiting the notion of artifact, where work environments are mainly devoted at ruling and promoting complex social interactions. Recent examples are [9], [8], where the notion of artifacts is used respectively to conceive and design organisation infrastructures for open MAS and to support the design of reputation mechanisms.

Then, a further – more challenging – level can be devised, in which MAS designers conceive and design agents – typically *cognitive* agents – which are capable to reason about their work environment and dynamically decide how to exploit it depending on their goals and tasks. This level is fundamental when *open* MAS are of concerns, and introduces many interesting and challenging issues, both on the artifact side and the agent side.

On this line, in this paper we report on preliminary investigation concerning the *functional roles* played by artifacts in the context of cognitive agent societies, and we relate such roles to the *epistemic* and *motivational* states of agents working with artifacts. As a result, we identified two fundamental general functions (described in Section III): (i) *doxastic*, which allows agent to improve their epistemic states, by representing and sharing strategic knowledge in the overall system; (ii) *purposive*, which allows agents to improve the repertoire of actions, by providing additional means which can be purposively triggered by agents in order to achieve their goals. Besides the conceptual aspects, in Section III we show some practical outcomes of the work reporting simple examples involving agents programmed with the *Jason* agent platform [1] (based on the AgentSpeak BDI-based agent language) engaged in goal-oriented activities involving artifacts programmed with CARTAGO. In order to ground the discussion and the examples provided in Section III, Section II provides an overview about the computational model of artifacts provided in CARTAGO and the repertoire of actions provided to agents for playing within artifact-based environments. Concluding remarks on the approach are reported in Section IV.

II. The CARTAGO Platform

CARTAGO (Common ARtifact infrastructure for AGent Open environment) is an infrastructure and a platform for programming and executing artifact-based work environments for MAS [20], implementing the A&A conceptual model. In detail, the platform includes a Java-based API for programming artifacts, defining new types of artifacts following the A&A programming model, an agent API to be used in agent-oriented programming platforms to play within CARTAGO environments – including a basic set of actions for creating and interacting with artifacts, and managing and joining workspace – and a runtime and related tools, to execute and manage the artifact-based environments. CARTAGO technology is open-source[3] and implemented on top of the Java platform.

A. Environment Model

A work environment in CARTAGO is conceived as collection of *workspaces* possibly distributed on different nodes where the infrastructure has been installed (referred in the following as CARTAGO nodes). Agents (possibly in execution on multiple and heterogeneous agent platforms) can dynamically join and quit the workspaces, possibly working in multiple (and distributed) workspaces at the same time. A Role-Based Access Control (RBAC) model is adopted for specifying and managing security aspects at workspace level, ruling agent entrance and exit, and agent access and interaction with artifacts.

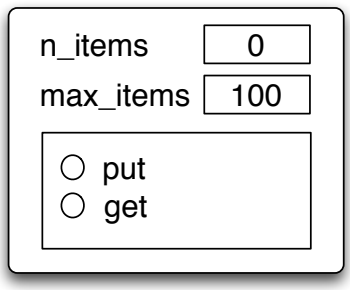
By default, each workspace contains a set of pre-defined artifacts, created at the workspace creation time, which provides some fundamental functionalities and facilities for agents working inside the workspace and for workspace(s) management. Among the others, a *factory artifact* is used to instantiate artifacts, specifying the artifact template and a name and a *registry* artifact is used to keep track of the set of artifacts actually available in the workspace. The general idea is to *reify* in terms of a suitably designed artifact every infrastructure part and functionality so as to make it observable, controllable, adaptable by agents themselves (meaning agents that have the permission to do that according to their role), besides human administrators.

B. Programming Artifacts: An Example

To give a taste of artifact programming model, here we briefly describe a simple example of artifact, a *bounded-inventory*, which contains main aspects of the artifact model, namely observable properties and a usage interface, besides basic synchronization functionalities. The bounded-inventory is a kind of coordination artifact designed to function as a shared inventory mediating the exchange of some kind of *items* between a possibly dynamic number of *producer* agents and *consumer* agents [11]. The producers-consumers problem is typical in concurrent systems, where agents are supposed to adopt effective strategies with respect of the shared resorce and taking into account further bounded resources like time and space (memory). This requires some kind of coordination strategy between agents, i.e., in order to coordinate the cyclic production of items by producer and the activities performed by consumer agents. To this end, the bounded-inventory provides a coordination mechanism to uncouple and – at the same time – synchronize the activities of producers and consumers, thus providing a locus of design (the size of the inventory) for tuning the performance of the system.

Looking at the CARTAGO implementation in Figure 2, the bounded-inventory artifact provides a usage interface with two operation controls to respectively insert (`put`) e consume (`get`) items, and two observable properties,

[3] Available at `http://cartago.sourceforge.org`.



OBSERVABLE PROPERTIES:

n_items: int+
max_items: int

Invariants:
n_items <= max_items

USAGE INTERFACE:

put(*item*:Item) / (n_items < max_items):
[obs_prop_updated, op_exec_completed]

get / (n_items >= 0) :
[obs_prop_updated, *new_item*(item:Item),
op_exec_completed]

```
import alice.cartago.*;
import java.util.*;

public class BoundedInventory extends Artifact {
  private LinkedList<Item> items;

  @OPERATION void init(int nmax){
    items = new LinkedList<Item>();
    defineObsProperty("max_items",nmax);
    defineObsProperty("n_items",0);
  }

  @OPERATION(guard="inventoryNotFull") void put(Item obj){
    items.add(obj);
    updateObsProperty("n_items",items.size()+1);
  }

  @OPERATION(guard="itemAvailable") void get(){
    Item item = items.removeFirst();
    updateObsProperty("n_items",items.size()-1);
    signal("new_item",item);
  }

  @GUARD boolean itemAvailable(){ return items.size() > 0; }

  @GUARD boolean inventoryNotFull(Item obj){
    int maxItems = getObsProperty("max_items").intValue();
    return items.size() < maxItems;
  }
}
```

Fig. 2. A simple bounded-inventory artifact, exploiting operation control guards to synchronize agent use of the inventory.

`max_nitems`, showing the maximum capacity of the inventory, and `n_items`, showing the current number of items stored in the inventory. Internally, a simple linked list is used to store items. The synchronization functionality provided by the artifact is realised here by exploiting a basic feature of the artifact programming model, which accounts for the possibility of defining *guards* that specify when an operation controls is either enabled or disabled. In the example the `put` control is allowed only when the inventory is not full (`inventoryNotFull` guard), and `get` is allowed when the inventory is not empty (`itemAvailable` guard). Hence, if an agent selects the `put` operation control and the inventory is full, the action is suspended. Analogously for the `get` control, if the inventory is empty. A most detailed description of the Java-based API and of this use case, along with other examples, can be found in [19].

C. Integration with Agent Platforms

CARTAGO is envisaged for enabling full interoperability across heterogeneous agent platforms, hence it has been designed to be orthogonal to the specific models and technologies adopted for the agents working in it. It allows integration and exploitation in principle of *any* agent programming platform, enabling agents of heterogeneous platforms (and finally models and architectures) to interact and interoperate as part of the same MAS, sharing common artifact-based environments [18].

To realise such integration, both from a conceptual and engineering point of view, the notion of *agent body* is exploited, as that part of an agent which is belonging to a CARTAGOworkspace. Whereas the *agent mind* remains in execution externally – whithin the agent platform – an agent body is physically running in a CARTAGOsystem. Hence, the agent body logically and physically *situates* an agent in a CARTAGOworkspace: in particular, it contains proper *effectors* that make it possible essentially to act upon (*use*) artifacts, and *sensors* to detect and perceive observable events generated by artifacts, possibly applying filters and specific kinds of "buffering" policies. From an architectural point of view, to connect agent mind and agent body, platform-specific *bridges* are introduced, functioning as wrappers on the agent mind side to control the body and perceive stimuli collected by body's sensors. Currently, bridges exists for *Jason* [1], an interpreter for an extended version of AgentSpeak, Jadex [17], a BDI agent platform based on Java and XML as mainstream language / technologies to develop intelligent software agent systems, and simpA [21], a Java-based agent-oriented framework based on the A&A conceptual model.

D. The Tenet of Agent-Artifact Interaction Model: Use and Observation

To enable interactions between agents and artifacts the repertoire of actions natively provided by the agent platforms

is extended with a new set of special-purpose actions envisaged for playing inside an artifact-based environment. The overall set of new actions can be grouped in four groups, as depicted in Table I: *(i)* join and leave workspaces; *(ii)* create, lookup, dispose of an artifact; *(iii)* use an artifact; *(iv)* observe an artifact without directly using it.

The core part of this set is given by actions in the last two groups, concerning artifact *use* and *observation*. The agent activities bleonging to artifact use and observation are the tenet of the agent-artifact interaction model and their understanding is the pivotal underpinning of this work.

The `use` action is provided to agents so as to act upon the artifact by selecting an operation control. To use an operation its description needs to be part of the artifact usage interface, and eventually specifying parameters required by the control (see Figure 3). If the use action succeeds, then a new instance of the operation linked to the operation control starts its execution inside the artifact. The execution of the operation eventually generates a stream of observable events that may be perceived both by the agent which is responsible of the operation execution and by all the agents that are *observing* the artifact. Some basic types of events are meant to be generated by default by artifacts, in spite of their specific type, in correspondence to particular situations (i.e., the completion or the failure of an operation, the update of an observable property, or rather the disposal of an artifact). Two aspects are important here. First, the execution of a use action upon an operation control involves a *synchronous* interaction between the agent and the artifact: action success means that the operation linked to the control has started its execution. Second, the execution of the operation is completely *asynchronous with respect to agent activity*. Hence, use does not involve any transfer of control as it happens in the case of remote procedure call or method invocation in procedure-based or object-oriented systems.

Besides use, *observation* is the second main aspect concerning agent-artifact interaction. To perceive the observable events generated by the artifact two basic modalities are possible, called here *active* and *passive*. In the active modality, the agent doing a `use` explicitly indicates a *sensor* as a parameter. The sensor is thus meant to collect all the observable events (which can be reffered to the triggered operation) as soon as they are generated; then, a further `sense` action is provided to the agent to actively fetch those events as percepts, possibly specifying filtering rules. In so doing the agent actively retrieve the percepts from the sensor on demand, *as soon as it needs it*. Sensors in this case play the role of *perceptual memory* explicitly manageable by the agent, who can use them to organise in a flexible way the processing of the events, possibly generated by multiple different artifacts that an agent can be using for different, even concurrent, activities.

In the passive modality events generated by an artifact are authomatically made observable to the agent directly as native/internal events, without the explicit mediation of sensors. In other terms, the bridge mechanism translate the events coming from the scrutinized artifact into events holding the agent architecture. Besides, the agent passively receives those events as native signals to be handled within the reasoning cycles. Those events are supposed to contain relevant information about the occurrence of the originating artifact event (i.e. , the source of the event, associated labels, contents etc.).

As an additional interaction modality, besides perceiving the events related to a previous use, a support for *pure observation* – that is, observation without use – is provided, concerning both observable properties and observable events. For *continuous* observation of properties and events a specific action called `focus` is provided (see Figure 4): by executing a focus on a specific artifact, an agent can continuously perceive the state of artifact observable properties and thus is notified of all the observable events that the artifact will generate from that moment on, even if it is not actually using it. Observable properties are directly mapped onto agent percepts and then, for cognitive agent architecture in particular, can be related to percepts or beliefs indicating the situated state of the artifact. For observable events, the two perceptive (active and passive) modalities are available also for the `focus` action, either specifying or not a sensor. The semantics is the same as in the use case: by specifying a sensor all the observable events generated by the artifact are detected by the sensor and eventually fetched by the agent through a `sense` internal action. A further action concerning observation is `observeProperty`, which makes it possible read the current value of a specific observable property, which is returned directly as feedback of the action.

It's worth noting that continuous observation of properties and perception of events have different characteristics (and then purposes, from the designer point of view). In particular, observable properties represent the *state* of the environment (structured in terms of artifacts) and, as such, it could change with a frequency that could be beyond agent perceiving (and related) capabilities. Instead, observable events represent *changes* in the world and typically are buffered and processed in some kind of order that could depend on event priorities or agent actual promptness. This is true in particular for cognitive agents which can indeed follow adaptable strategies in allocating their attentive resources.

Agents using mental states are the ideal candidate to manage complex interactions from agents to artifacts involving interleaved operation calls performed on heterogeneous artifacts distributed across nodes and workspaces. Based on the various execution models employed by the various integrated agent platforms, more complex form of loosely coupled interaction between agents and artifacts can be suitably conceived. In what follows we'll provide a systematization on the functional terms at which a complex interaction can be conceived in cognitive terms. Whereas agents perceptive capabilities allow to dynamically store and situate information which is relevant for the ongoing purposes, reasoning capabilities may promote the use of external services provided by artifacts, orchestrating a tight composition of chained sequences of operation calls.

(1) `joinWorkspace(+Workspace[,Node])`
(2) `quitWorkspace`
(3) `makeArtifact(+Artifact,+ArtifactType[,ArtifactConfig])`
(4) `lookupArtifact(+ArtifactDesc,?Artifact)`
(5) `disposeArtifact(+Artifact)`
(6) `use(+Artifact,+UIControl([Params])[,Sensor][,Timeout][,Filter])`
(7) `sense(+Sensor,?PerceivedEvent[,Filter][,Timeout])`
(8) `focus(+Artifact[,Sensor] [,Filter])`
(9) `stopFocussing(+Artifact)`
(10) `observeProperty(+Artifact,+Property,?PropertyValue)`

TABLE I

ACTIONS INTRODUCED IN AGENT'S REPERTOIRE ALLOWS THE INTERACTION IN CARTAGOWORKING ENVIRONMENT. THEY ARE FUNCTIONALLY DIVIDED IN FOUR MAIN GROUPS: FOR MANAGING WORKSPACES (1–2), FOR CREATING, DISPOSING AND LOOKING UP ARTIFACTS (3–5), FOR USING ARTIFACTS (6–7), AND FOR OBSERVING ARTIFACTS (8–10). SYNTAX IS EXPRESSED IN A LOGIC-LIKE NOTATION, WHERE ITALICISED ITEMS IN SQUARE BRACKETS ARE OPTIONAL. CONCRETE REALISATION OF THE ACTIONS DEPENDS ON THE SPECIFIC AGENT PROGRAMMING PLATFORM WHICH HAS BEEN INTEGRATED [18], [15]

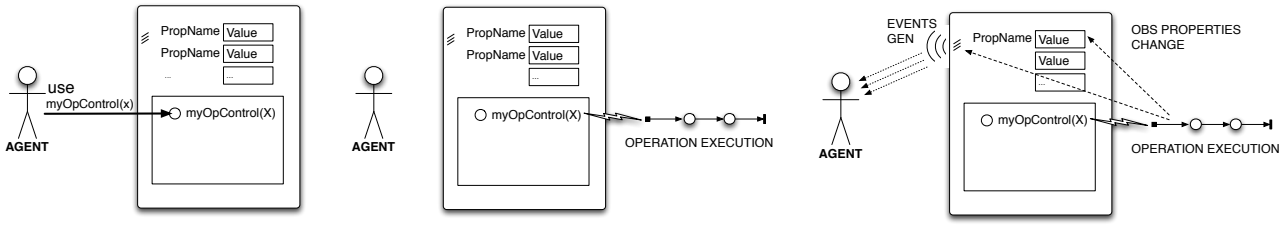


Fig. 3. Using an artifact: by selecting the `myOpControl` control belonging to the usage interface, a new operation instance starts its execution inside the artifact. The execution of the operation will eventually generate events observable to the user agents – and to all the agents observing the artifact – and possibly update artifact observable properties.

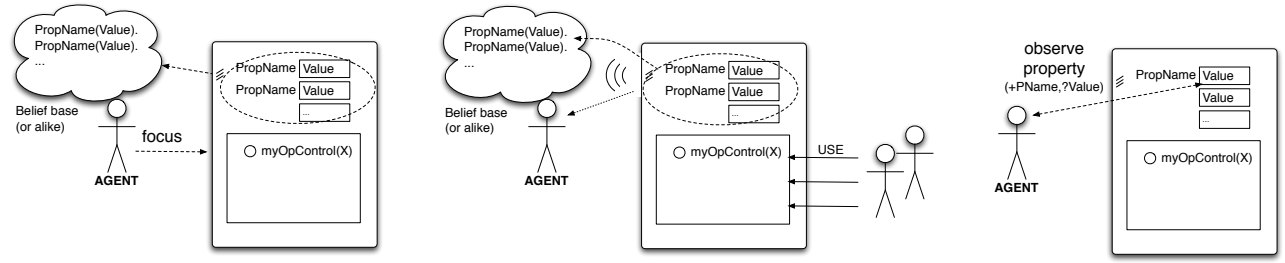


Fig. 4. Observing an artifact: by focussing an artifact, an agent is (1) continuously made aware of observable properties value as percepts typically mapped into agent belief base, and (2) receives all the observable events generated by the artifact in executing operations possibly triggered by other user agents.

III. COGNITIVE USE OF ARTIFACTS

A main aspect of cognitive system study concerns the investigation of how information is represented and how those representations are transformed, combined and propagated so as to form a behavior [24]. In particular, we here refer to a general explanation of cognitive agents built upon the two pronged notion of epistemic and motivational states. In this view, cognitive agents can be described as *intentional systems* able to autonomously reason about their resources – mappable upon internal representations – in order to pro-actively reach a desired state of affairs. On the epistemic dimension, cognitive agents are assumed to support their reasoning processes on the basis of their internal knowledge, namely "beliefs". Beliefs can be viewed as those *doxastic representation* related on the information agents are able to find, integrate and take into account. Besides epistemic states, motivational states allow agents to "pursue" a given course of actions, i.e. by committing an intention (among the achievable ones) through the execution of some action or plan they have in repertoire. On these bases, in this section we refer to artifacts which can be cognitively used, read and created by cognitive agents.

In this view, artifact are supposed to be (not only) computational components structuring the environment (but also) resources which can be interactively and cognitively exploited by agents to attain their goals. Given the model abstractly described in Section II the following sections provide a deeper analysis on the cognitive terms underlying interactions from agents to artifacts. Before detailing the operational and dox-

astic use of artifacts, the next section describes the cognitive use of active and passive perception styles.

A. Active and Passive Interaction Styles

As seen in Subsection II-D, two basic approaches have been envisaged for agents in order to manage their perceptive activities upon scrutinized artifacts, namely *active* and one *passive* modalities. Sensors can be seen as part of an *agent body*, logically situating an agent into a workspace and containing both sensors and effectors to act upon artifacts of that workspace. Hence, in the active modality sensors play the role of *perceptual memory* or external working memory, whose functionality accounts for keeping track of stimuli arrived from the CARTAGO environment. Accordingly, sensors can be programmed by defining rules, filters and specific kinds of "buffering" policies. This allows agents to retrieve relevant events, even interleaved and generated by multiple artifacts that the agent may use for different, even concurrent, purposes. This approach provides to agent developers the possibility to customize the perceptive activities at an intentional level. Percepts generated by artifacts can be situated in the context of the adopted goals, and thus managed through internal actions to be executed within the plan workflow. In so doing, active perception makes it possible for agents developper to organize perceptive activities – at the programming level – as flexibly as they wish. For instance, in active modality, a given sensor can be devoted to filter relevant events coming from a scrutinized artifact so as to suddenly become aware on artifact changes. Accordingly, filterd events can be proactively and intentionally processed, i.e. in order to update beliefs or check goal achievement.

The passive modality allows the automatic propagation of native internal events at runtime, generated by translating on the fly the events coming from the scrutinized artifacts. This makes it possible for agents to react to observable event asynchronously, as soon as they are perceived. This functioning is supposed to ease agent's reasoning allowing pivotal processes as goal adoption and plan selection to be governed by internal events, which in turns can be targeted on the basis of the events coming from artifacts. This might be the case when agents perform activities in a reactive fashion, for instance when they have to check the execution of operations wich has been externalised in artifacts, as well as a control activity is being automated in the human case. As showed in [15], [16], using events coming from artifacts the agent can handle events so to trigger plans and thus decide the next course of actions. This has a special importance once the agent needs to manage low level and routinized interaction activities. Besides reactive behavior, events coming from artifacts can signal to the agent situation requiring particular servicing: once abnormal values are encountered or exceptional situations arrive, agents can arouse and suitably exploit such signals for reentering the deliberation process or for reconsider their intention. Besides, becoming aware about those relevant facts, agents can elicit reallocation of resources, recovery policies, exception handling etc.

B. Operational function

Artifact operations, controlled by the usage interface, encapsulate artifact's intended purposes[4]. From the agent viewpoint, operations can be suitably used to improve agent repertoire of actions, providing additional means to achieve agents' goals. They can be targeted dynamically by agents so as to *externalise* and distribute (part of) their goal-oriented activities. For doing this, operation outcomes have to be taken into account by agents in their practical reasoning. In fact, by changing the actions required for achieving a given goal, artifact operations change agent means-end reasoning[5] stages.

This aspect can be tackled at different conceptual levels. The first, most obvious, solution is to integrate artifacts functionalities in the agent's developing phases. In so doing artifacts use can be defined at the language level, by defining the operation control in an off-line fashion, at design time. Referring to the bounded-inventory example introduced in Subsection II-B, an agent having the goal to produce a new item and put it in the buffer may use the following intention (agent's specification is provided with *Jason* language):

```
+!produceItems : nextItemToProduce(Item)
  <- cartago.lookupArtifact("my-inventory", InvID)
     cartago.use(InvID,put(Item),5000).

-!produceItems: true
  <- cartago.use(console,
              println("Insertion failed due to timeout.")).
```

The agent here selects the intention to store an item on the inventory once an `Item` has been prepared and is available in the belief base. Then the adopted intention first lookups the `my-inventory` artifact to retrieve its system identifier `InvID` and then stores the item by selecting the `put` control provided by the artifact usage interface. Notice here the presence of a 5000 milliseconds timeout, after which the action (and the plan) is considered failed and a message is printed on the console by the goal deletion plan `-!produceItems`.

Besides, a consumer agent can cyclically adopt the following plan to attain an item on the inventory:

```
+!consume
  : myInventory(InvID) & mySensor(S)
  <- cartago.use(InvID, get, S, 1000);
     cartago.sense(S, new_item(Item));
     !consumeItem(Item);
     !consume.

+!consumeItem(Item) : true <- ...
```

Notice that a sensor identified by `S` is explicitly used by the consumer to detect and manage the final event of type `new_item(Item)` generated by the artifact at the end of the `get` operation. This event represents for the agent the signal indicating a goal achievement.

[4]Notice that before beeing in the intention of an agent who wants to use the artifact, the intended purpose is in the mind of artifact designer, who conceive it in order to serve an operation or a function.

[5]We here refer to the notion of cognitive agents able to find a successful sequence of actions, between the ones he has in repertoire, in order to attain an adopted goal. Several agent architectures founded on this reasoning principle have been presented in the last decades, many of which can be related to the conceptual model provided by [2]

C. Doxastic function

A secondary function, dual to operational one, is about informational, observable and retrievable knowledge provided by artifacts and represented by their observable properties. In this case, from an agent point of view, artifacts are informational units functioning to maintain, make it observable and pre-process information which is exploitable in a situated way. In other terms, by embedding machine-readable representations, an artifact can be a target for agents epistemic actions [10]. This entails for agents the opportunity to use, read and observe artifacts to attain new information and possibly update beliefs, solely with the aim to improve the knowledge base with information which is strategic for their tasks. In this view, artifacts are supposed to provide observable cues in order to highlight relevant information (thus improving agent's situated cognition). This turns to be important for shaping *goal-supporting beliefs*, i.e. those beliefs required to agents for ruling over deliberation and practical reasoning [4]. Accordingly, information available with artifacts can ease agent reasoning, for instance simplifying and improving agent's decision making and remarkably easing belief update processes.

As a simple example of doxastic use, we consider here an extension of the producer-consumer scenario in which two bounded inventories are used instead of one (to avoid centralisations, for instance). By continuously observing the number of items of both the inventories, consumer agents must dynamically decide which artifact to use to consume a new item, choosing the one with more items so as to minimise the probability to get stuck because of the inventory is empty. To this end the continuous observation of artifact observable properties is exploited:

```
+!consumeActivity  : true
  <- +min_items(-1);
     cartago.lookupArtifact("my-inventory-1", InvID1);
     cartago.focus(InvID1);
     cartago.lookupArtifact("my-inventory-2", InvID2);
     cartago.focus(InvID2);
     +selectedInv(InvID1,0);
     !consumeAction.

+n_items(N) [source(percept), artifact(InventoryID)] :
  selectedInv(_,N1) & N > N1
  <- -+selectedInv(InventoryID,N).

+!consumeAction : selectedInv(InvID,_)
  <- cartago.use(InvID, get, mySensor);
     cartago.sense(mySensor, new_item(Item));
     cartago.use(console,println( " Consumed Item: ", Item));
     !consumeAction.
```

The agent here uses the goal-supporting belief `selectedInv(InventoryID,NItems)` to store the identifier of the inventory with the greatest number of items, among the observed inventories. Such a belief is initially set for `my-inventory-1` artifact in the `consumeActivity` plan, then it is updated by the second plan of the agent each time a new percept about the actual value of the observable property `n_items` of any observed inventory is perceived. In the plan, the annotations `[source(percept), artifact(InventoryID)]` make it possible to retrieve the identifier of the artifact source of the percept.

So, in this case agents are aware of the current state of the artifacts and can rule their means-end reasoning based on goal-supporting beliefs which are *read* on the artifacts. A similar strategy can be implemented for the producer agents (the code is here omitted for brevity) that can use a twofold strategy for choosing the inventory where to put a new item.

D. Discussion

Some final remarks are worth to be taken into account on these cognitive aspects. A first pivotal aspect in threating operational functionalities of artifacts relates on the motivational attitudes of agents. Actually abilities to handle goals are variously characterized by mainstream agent platforms [22]. The procedural goal approaches can be related to agents functioning according to transitions within the action selection policy, where the goal is not explicit in agents specification and where a behavioral policy is rather constructed by the programmer through procedures taking into account the intended goal states. We refer, in the case of agents adopting procedural goals, to a *goal-oriented* use of artifacts. On the contrary, declarative goal approaches refer to agents able to process goals explicitly represented as internal states, where declarativeness stands for explicit representation of goals described either in terms of end-states either in terms of execution states within the reasoning process. As discussed in [15] describing integration between the Jadex agent platform and CARTAGO, we refer, in the case of agents dealing declarative goals, to a stronger notion of usabilty, namely *goal-directed* use of artifacts. A more advanced approach in exploiting operational function envisages an on-line integration, by which agents are enabled to dynamically discover and afford artifact which are not known at design time. This approach requires the additional capability for agents to afford artifacts and map operations, learned form artifact's machine readable descriptions, in their planning and means-end processes. As in the human case, once once an artifact has been acknowledged in terms of its descriptions (i.e. through *manuals*), agents can learn to use operations. In addition, by introducing planning capabilities, an agent can switch actions of his repertoire with operations provided by artifacts to achieve goals.

As for the doxastic function, the contribute of artifacts in easing epistemic activities is remarkable also in the context of Multi Agent scenario. Here the pivotal aspect is the distribution of information in the overall society of agents. In particular, information can be spread over several orthogonal dimensions: *(i)* across agents: by organising and making available relevant information as permanent side-effect of artifact use (modification of artifact state); *(ii)* across platforms: once interactions between agents are mediated by artifacts, heterogeneous platforms can be integrated at the same domain level. Moreover agents acquire an additional option to communicate, being artifacts a suitable alternative to protocols based on message exchange; *(iii)* across time: artifacts are designed to hold strategic information whic can persist also over interleaved presence of individual agents; *(iv)* across space: the topological

notion of work environments makes it possible for agents to distribute their activities between many nodes and workspaces. This entails no need for agents mutual presence within a given location/place.

IV. Conclusion and Related Works

In this work we provided a common grounding for theories and programming approaches based on A&A interaction. In particular, we investigated cognitive aspects of interactions between agents and artifacts, describing the terms of the interaction since the definition of the perceptive activities needed for agents to cognitively operate with artifacts. By adopting a functional approach, we described the twofold role played by artifact once they are used by a cognitive agent. On the one side artifacts are supposed to provide operations, wich agents can exploit to perform activities and attain goals (operational function). On the other side artifacts embeds information which is readable by agents to improve their epistemic states and can be considered as repositories of relevant information in working environments (doxastic function).

Nevertheless the role of the environment as first-class abstraction in designing complex MAS has been largely acknowledged in literature (see [25] for a survey), few works consider the issue of cognitive agents interacting in properly designed environments. Among others, Brahms [23] is a programming language and platform to develop and simulate multi-agent models of human and machine behavior, based on a theory of work practice and situated cognition. Another approach has been developed by Holvoet and Valckenaers [7], who introduce Delegate MAS as a mean to design environment in BDI-based agent architectures. A further work is GOLEM [3], that introduces a platform for modeling situated cognitive agents in distributed environments by declaratively describing the representation of the environment in a logic-based form.

References

[1] Rafael Bordini, Jomi Hübner, and Mike Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, Ltd, 2007.

[2] M. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

[3] Stefano Bromuri and Kostas Stathis. Situating Cognitive Agents in GOLEM. In *Engineering Environment-Mediated Multiagent Systems (EEMMAS'07)*, 2007.

[4] C. Castelfranchi and F. Paglieri. The role of beliefs in goal dynamics: Prolegomena to a constructive theory of intentions. *Synthese*, 155:237–263, 2007.

[5] R.W. Christopherson. *Geosystems: An Introduction to Physical Geography*. 1996.

[6] Richard Dawkins. *The Selfish Gene*. Oxford University Press, 1976.

[7] Tom Holvoet and Paul Valckenaers. Beliefs, desires and intentions through the environment. In *AAMAS'06, Proceedings*, pages 1052–1054, New York, NY, USA, 2006. ACM.

[8] Jomi F. Hübner, Olivier Boissier, and Laurent Vercouter. Instrumenting multi-agent organisations with reputation artifacts. In Virginia Dignum and Eric Matson, editors, *Coordination, Organizations, Institutions and Norms (COIN@AAAI), held with AAAI 2008*, 2008.

[9] Rosine Kitio, Olivier Boissier, Jomi Fred Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations: "giving the power back to the agents". In J. Sichman, P. Noriega, J. Padget, and S. Ossowski, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems III*, LNCS. Springer, 2008.

[10] Paul P. Maglio and David Kirsh. Epistemic action increases with skill. In *18th Annual Conference of the Cognitive Science Society*, pages 391–396. Erlbaum, 1996.

[11] Thomas Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[12] B. A. Nardi. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996.

[13] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17 (3), December 2008.

[14] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, and Luca Tummolini. Coordination Artifacts: Environment-based Coordination for Intelligent Agents. In *Proceedings of AAMAS'04*, volume 1, pages 286–293, New York, USA, 2004.

[15] M. Piunti, A. Ricci, L. Braubach, and A. Pokahr. Goal-Directed Interactions in Artifact-Based MAS: Jadex Agents playing in CARTAGO Environments. In *2008 IEEE/WIC/ACM Conferences on Web Intelligence and Intelligent Agent Technology (IAT-2008)*. IEEE, 2008.

[16] Michele Piunti and Alessandro Ricci. Cognitive Artifacts for Intelligent Agents in MAS: Exploiting Relevant Information residing in Environments. In *Workshop on Knowledge Representation for Agents and Multi-Agent Systems (KRAMAS 2008)*. Sydney, 2008.

[17] A. Pokahr, L. Braubach, and W. Lamersdorf. *Jadex: A BDI Reasoning Engine*, chapter Chapter of Multi-Agent Programming. Kluwer Book, 2005.

[18] A. Ricci, M. Piunti, L. D. Acay, R. Bordini, J. Hubner, and M. Dastani. Integrating Artifact-Based Environments with Heterogeneous Agent-Programming Platforms. In *AAMAS'08, Proceedings*, 2008.

[19] Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. Environment programming in CARTAGO. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*. To appear. The draft of the chapter is available at: http://137.204.107.188/ aricci/Drafts/chapter-mas-programming.pdf.

[20] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. The A&A programming model & technology for developing agent environments in MAS. In *ProMAS'07,Post-proceedings*, volume 4908 of *LNAI*, pages 91–109. Springer, 2007.

[21] Alessandro Ricci, Mirko Virolil, and Giulio Piancastelli. simpA: A simple agent-oriented Java extension for developing concurrent applications. In Mehdi Dastani, Amal El Fallah Seghrouchni, Joao Leite, and Paolo Torroni, editors, *Languages, Methodologies and Development Tools for Multi-Agent Systems (LADS 2007*, volume 5118 of *LNAI*, pages 176–191. Springer-Verlag, Durham, UK, 2007.

[22] M. Birna Van Riemsdijk, Mehdi Dastani, and Michael Winikoff. Goals in agent systems: A unifying framework. In *Intern. Conf. on Autononous agents and Multi-Agent Systems (AAMAS08)*, 2008.

[23] Marteen Sierhuis and William J. Clancey. Modeling and simulating work practice: A human-centered method for work systems design. *IEEE Intelligent Systems*, 17(5), 2002.

[24] Herbert Alexander Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, Mass., 1981.

[25] Danny Weyns and H. Van Dyke Parunak. Special issue on environments for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):1–116, February 2007.

powerJADE: Organizations and Roles as Primitives in the JADE Framework

Matteo Baldoni, Guido Boella, Mauro Dorni, Andrea Mugnaini, and Roberto Grenna

Abstract. **This document shortly describes powerJADE, an improved JADE framework [4] which provides the primitives to manage organizations and roles.**

I. THE MODEL OF ORGANIZATIONS AND ROLES

Organizations are the subject of many recent papers in the MAS field, and also among the topics of workshops like COIN, AOSE, CoOrg and NorMAS. They are used for coordinating open multiagent systems, providing control of access rights, enabling the accommodation of heterogeneous agents, and providing suitable abstractions to model real world institutions [12].

Many models have been proposed [16], applications modeling organizations or institutions [22], software engineering methods using organizational concepts like roles [26]. However, up to now, on the one hand, despite the development of several agent programming languages among which 3APL [25], few of them have been endowed with primitives for modeling organizations and roles as first class entities (exceptions are MetateM [15], J-Moise+ [19], and the Normative Multi-Agent Programming Language in [24]). On the other hand, frameworks for modelling organizations like S-Moise+ [20] and MadKit [17] offer limited possibilities to program organizations.

The heterogeneity of solutions show a lack of a common agreement upon a clear conceptual model of what is an organization; the ontological status of organizations has been studied only recently and thus it is difficult to translate the organizational model in primitives for programming languages.

Our proposal, the introduction of primitives for organizations and roles in JADE, is mainly based on the ontological model of organizations and roles of [7]. However, since this model disregards the problem of how agents play roles in organizations, in [6] we integrated our model with the model of role playing of [11]. The model of [7] is focused on the definition of the structure of organizations, given their ontological status, which is only partly different from the one of agents or objects. On the one hand, roles do not exist as independent entities, since they are linked to organizations. Thus they are not components like objects. Moreover, organizations and roles are not autonomous and act via role players. On the other hand, organizations and roles are description of complex behaviours: in the real world, organizations are considered legal entities, so they can even act like agents, albeit via their representative playing roles.

Thus, in our model, roles are entities, which contain both state and behaviour: we must distinguish with the term "role", the role instance associated with a player, while the general specification of a role is defined as a "role type". For each agent, when it asks to an organization to play a role of a given type, an instance is created, representing the possibility for the player of interacting with the organization, and with the other roles, and the state of the interaction between the agent and the organization. As recognized by [10] this feature is quite different from other approaches which use roles only in the design phase of the system, as, e.g., in [26].

The goals, together with the beliefs, attributed to a role (as also in [11]) describe the behaviour expected from the player of the role, since an agent pursues his goals based on his beliefs. The player should be aware of the goals attributed to the roles, since it is expected to follow them (if they do not conflict with other goals of the agent).

Most importantly, roles work as "interfaces" between organizations and agents: they give so called "powers" to agents, extending their abilities, allowing them to operate inside the organization and inside the state of other roles. An example of such powers, called "institutional powers" in [21], is the signature of a director which counts as the commitment of the entire institution. If, on the one hand, roles offer powers to agents, they request from agents who want to play roles a set of requirements, abilities that the agents must have, like also in [9]. Thus, the set of roles an agent can play is not determined a priori, but it depends on which abilities are required by the role of an institution.

Powers are invoked by players on their roles, but they are executed by the roles, since they have both state and behaviour. Boella and van der Torre's [7] model focuses on the dynamics of roles in function of the communication process: role instances evolve according to the speech acts of the interactants, where speech acts are an example of powers which change not only the state of the role making the speech act, but also of other roles (see [5]). E.g. the commitments made by a speaker of a promise or by commands made by other agents playing roles which are empowered to give orders. In this model, sets of beliefs and goals (as [11] does) are attributed to the roles. They are the description of the expected behaviour of the agent. The powers of roles specify how the state of the roles changes according to the moves played in the interactions by the agents enacting other roles. Roles are a way to structure the organization, to distribute responsibilities and a coordination means.

Roles allow to encapsulate all the interaction between an agent and an organization and between agents in their roles. The powers added to the players can be different for each role and thus represent different affordances offered by the organization to other agents to interact with it [2].

Using the distinction of Omicini [22], we use our model [7] as an objective coordination mechanism, like for example artifacts: organizations are first class entities of the MAS rather than a mental constructions which agents use to coordinate themselves.

However, this model leaves unspecified, how, given a role, its player will behave. In Dastani [11], the problem of formally defining the dynamics of roles, is tackled identifying the actions that can be done in an open system such that agents can enter and leave. In [11] four operations to deal with role dynamics are defined: enact and deact, which mean that an agent starts and finishes to occupy (play) a role in a system, and activate and deactivate, which means that an agent starts executing actions (operations) belonging to the role and suspends the execution of the actions. Although is possible to have an agent with multiple roles enacted simultaneously, only one role can be active at the same time: when an agent performs a power, he is playing only one role in that moment.

II. Organizations, roles, and players in JADE

In this section we describe how we use our model in [7] to extend the JADE framework with primitives for programming organizations and roles.

JADE, to program a MAS, offer as basic elements the class *Agent*, different classes of behaviours (like finite state machines), and protocols with the relative speech act definitions, compliant with FIPA [13]. We have to introduce organizations and roles as first class entities, located in another platform with respect to their member agents, with behaviours - albeit not autonomously executed - and communication abilities. Thus, organizations and roles can be implemented using the same primitives as agents. In JADE this amounts to having special extensions of the *Agent* class, *Organization* and *Role* respectively, which can be further extended to program organizations and roles. Analogously, to implement agents who are able to play roles, the *Player* class is defined, which extends the JADE *Agent* class. Differently from *Organization* and *Role*, this class is used to implement autonomous agents. The behaviours and the communication protocols - described in the next section - which allow organizations to work, are inherited from these three classes. E.g. all organizations must offer a suitable protocol to allow an agent to ask playing a role, to verify if the agent fulfills the requirements of the role, and to enact the role. The extensions of the *Role* class, like, e.g., *Buyer*, represent the role types. Their instances, the role instances associated with an instance of the *Agent*. Organizations and roles, however, differ in two ontological aspects: first, roles are associated to players, second, roles are not independent from the organization offering them. Thus, the *Role* class is subject to an invariant, stating that it can be instantiated only when an instance of the organization offering the role is present. Conversely, when an organization is destroyed all its roles must be destroyed too.

A further difference of role classes is that to define "powers" as described in the previous section, they must access the state of the organization they belong too: only in this way it is possible that the signature of a role has the effect of modifying the private state of the organization adding a commitment. To avoid making the state of the organization public, the standard solution offered by Java is to use the so-called "inner classes", which are classes defined inside other classes (called "outer classes").

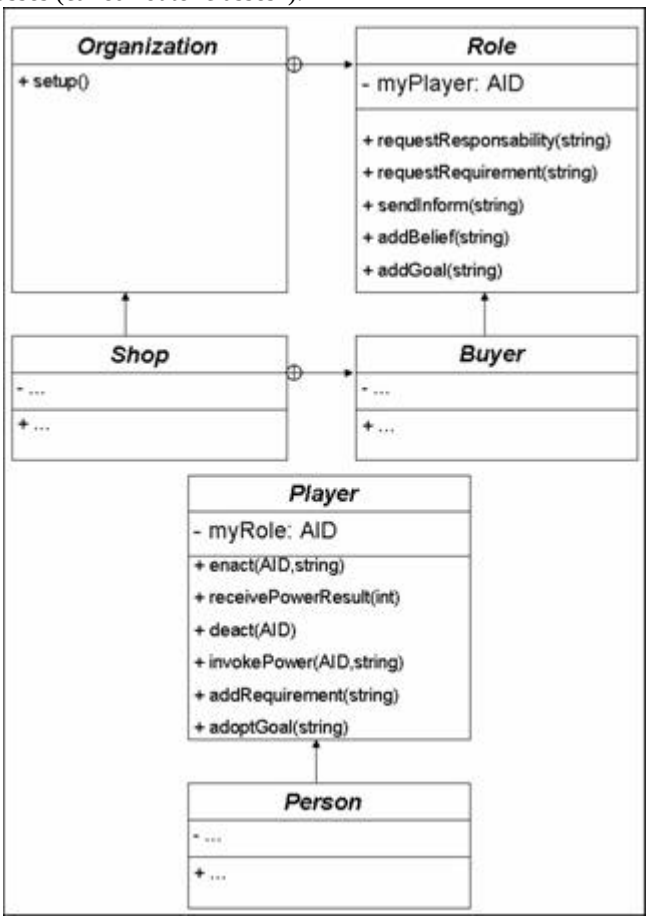


Fig. 1 - UML representation

An inner class shares the namespace of the outer class and of the other inner classes, thus being able to access private variables and methods. Thus the class *Role* is defined as an inner class of the *Organization* class. Class extending the *Role* class must be inner classes of the class extending the Organization class (see Figure 1). E.g., the *Buyer* class

extending the *Role* class must be an inner class of the *Shop* class extending the *Organization* class. In this way the role can access the private state of the organization and of the other roles. Note that this access is not unregulated, since a specific role cannot access the entire private state of the organization and other roles, but only those features which have been decided by the programmer of the role (who is the same as the programmer of the organization, since the role is contained in it). E.g., the methods of the *Buyer* class can access the private state of both the *Shop* class and of other roles of *Shop*, like *Seller*.

If roles are implemented as inner classes, albeit extension of the *Agent* class, this means two facts: first, the role instance must be on the same platform as the organization instance it belongs to; second, the role agent can be seen as an object from the point of view of the organization and of the other roles which can have a reference to it, besides sending messages to it. In contrast, outside an organization the role agent is accessed by its player (which can be on a different platform) only as an agent via messages, and no reference to it is possible. So not even its public methods can be invoked. The inner class solution for roles is inspired to the use of inner classes to model roles in object oriented programming languages like in powerJava [3]. The use of inner classes is coherent with the organization of JADE, where behaviours are often defined as inner classes with the aim to better integrate them with the agent containing them.

Organizations. To implement an organization it's necessary to extend *Organization*, subclass of *Agent*, which offers protocols necessary to communicate with agents who want to play a role, and the behaviours to manage the information about roles and their players.

Moreover, the *Organization* class includes the definition of the *Role* inner class that can be extended to implement new role classes in specific organizations. To support the creation and management of roles the *Organization* class is endowed with the data structures and methods to create new role instances and to keep the list of the AIDs (Agent ID) of role instances which have been created, associated with the AIDs of their players. However, these methods are private, to avoid the misuse by the programmer who could violate the organization invariants (e.g., to instantiate a role first there must be a player, etc.).

The *Enact* protocol allows starting the interaction between player and organization. What happens is that the player sends a message to the organization requesting to play a role of a certain type; if the organization considers the agent authorized to play that role type, it sends to the caller a list of powers (what the role can do) and requirements (what the role can ask to player to do). At this point, the player can compare his requirement list with the one sent from organization and communicate back if he can play the role or if he can't. Here, in fact, we can also consider that a not honest player could lie to play a role for which he doesn't have all the requirements. Only in a second time, and only if the player will show his limits, some player recovering a controlling role could apply sanctions to the dishonest one. It's important to note that only the player can begin the Enact protocol; the organization, in fact, is intended as a collection of roles that players can play. The only way in which an organization could begin an enactment protocol is to be a player that want to play a role inside another organization.

Each organization has to maintain the list of agents playing roles in it, associating with the player agent AID the role type and the role AID. While the association of a player and a role is done automatically during the enactment of a role (see next section), the programmer may query which role instances an agent is playing (specifying the role type) or which is the player of a given role. Finally, the operation of leaving a role (deact), is asked by the player to the role itself, so the class organization does not offer any method or protocols for that.

Since roles are Java inner classes of an organization, the organization code can be written in Java mostly disregarding what is a JADE application. Moreover, the inner class mechanism allows the programmer to access the role state and viceversa, while maintaining the modularity character of classes. For helping players to find quickly one or more organizations offering a specific role, Yellow Pages, a JADE feature, are used. They allow to register a pair (*Organization*, *RoleType*) for each role in each organization; the interested player will only have to query the Yellow Pages to obtain a list of these couples and choose the best for itself.

Then the candidate player can start the enactment protocol with the selected organization.

Roles. As discussed above, a role is implemented by extending the *Role* class, thus, inheriting the facilities offered by that class. In particular, the *Role* class offers the protocols to communicate with the player agent and the methods for the role programmer to use these protocols.

To program a role, it is necessary to extend, inside a class extending the *Organization* class, the inner class *Role* of the *Organization* class.

In particular, the communication protocol with the player essentially allows (see Figure 2): (1) To receive the request of invoking powers. (2) To receive the request to deact the role. (3) To send to the player the request to execute a requirement. (4) To receive from the player the result of the execution of a requirement. (5) To notify the player the failure of executing the invoked power or the failure to receive all the results of the requested requirements. The role programmer, thus, has to define the methods which are the powers which can be invoked by the player, and to specify them in a suitable data structure used by the Role class to select the requests of powers which can be executed.

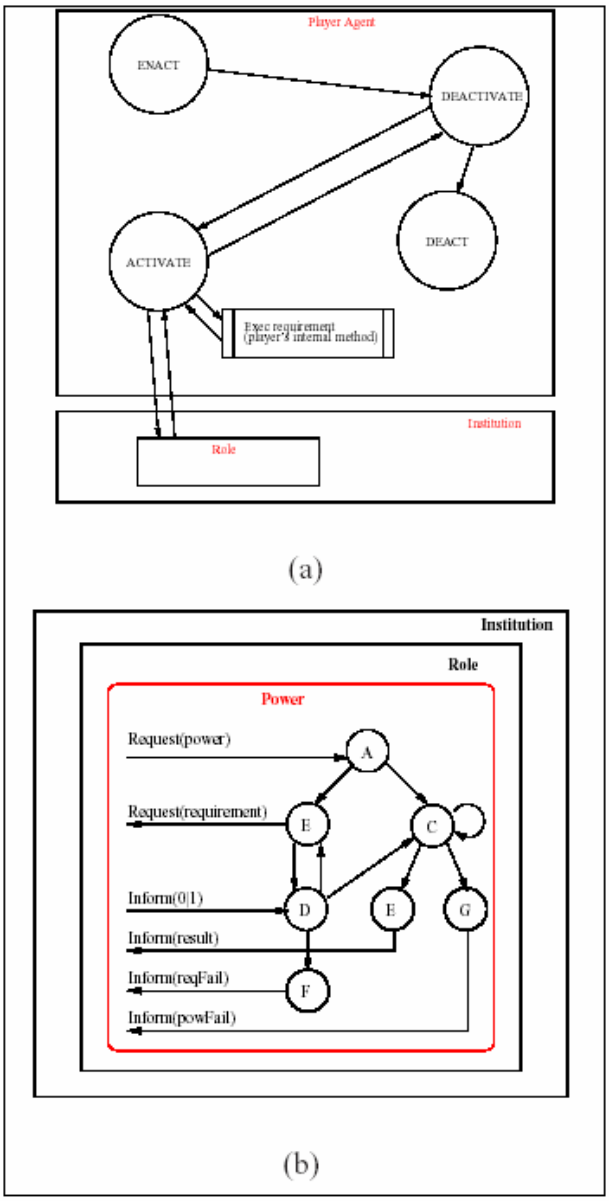


Fig. 2 - (a) The states of role playing. - (b) The behaviour of roles - A: ManagePwRequest (manages the request from player); B: ManageReqRequest (if a requirement is needed); C: Execute (executes the called power); D: MatchReq (checks if all requirements are ok); E: InformResult (sends results to player); F: InformFail (sends fail caused by requirement missing); G: Inform-PowerFail (sends fail caused by power failure)

Allowing a player to invoke a power (which results in the execution of a method by the role) could seem a violation of the principle of autonomy of agents, since it seems that the player delegates to the role the execution. However, two things must be noted: (1) The powers are the only way the players have to act on an organization. (2) The execution of an invoked power may request, in turn, the execution by the player of requirements needed to carry on the power, so the result of the power execution still depends on the player.

The second point is important, since the player may refuse to execute a requested requirement, and the requirements determine the outcome of the power, which thus varies from invocation to invocation and from player to player.

Since roles are not autonomous, the invocation of a power is not subordinated to the decision of the role to perform it or not. In contrast with powers, a requirement cannot be invoked. Rather it is requested by the role, and the player autonomously decides to execute it or not. In the latter case the player is not complying anymore with its role and it is deacted.

To remark this difference we will use the expression "invoking a power" versus "requesting a requirement". Requests for the execution of requirements are not necessarily associated with the execution of a power. They can be requested to represent the fact that a new goal has been added to the role. For example, this can be the result of a task assignment when the overall organization is following a plan articulated in subtasks to be distributed among the players at the right moments, as in [20].

In case the new goal is a requirement of the player, the method *requestRequirement* is executed, otherwise, if it is a power of the role, a *requestResponsibility* is executed.

- *requestResponsibility(String)*: this method asks to the player to invoke a power of the role.
- *requestRequirement(String)*: this method invokes a requirement of the player. It returns the result sent by the player if he complies with the requirement. The failure of executing of a requirement results in the deactment of the role.

Analogously to requirements when the role notifies its player about the responsibility, it cannot be taken for granted that it will invoke the execution of the power. Note that both the *requestRequirement* method and the *requestResponsibility* can be invoked also by other roles or by the organization itself, due to the role's limited autonomy. Other methods are available only when agents endowed with beliefs represented, e.g., as Jess knowledge bases:

- *sendInform(String)*: this method is used to inform the player that the beliefs of the roles are changed. This does not assume that the player adopts the conveyed beliefs as well.
- *addBelief(Belief)* and *addGoal(Goal)*: they are invoked by the role's behaviours or by other roles' to update the state of the role.

Besides the connection with its player, which is regulated by the protocols described in the next section, the role is an

agent like any other, and it can be endowed with further behaviours and further protocols to communicate with other roles of the organization or even with other agents. At the same time it is a Java object as any other and can be programmed, accessing both other roles and the organization internal state to have a better coordination.

Players. Players of roles in organizations are JADE agents, which can reside on different platforms with respect to the organization. However, since to play a role special behaviour is needed, the *Player* class is offered.

An agent which can become a player of roles extends the *Player* class, which, in turn, extends the *Agent* class. This class allows to model the states of the role playing (enact, active, deactivated, deacted), the transitions from one state to the others, and offers the protocols for communicating with the organization and with the role. A player agent can play more than one role. Thus, an instance of the protocols and behaviours is needed for each role played. The list of roles played by the agent, and the state of each role (activated, deactivated), is kept in an hashtable.

The enactment procedure, described in the next section, takes the AID of an organization and of a role type and, if successful, it returns the AID of the role instance associated to this player in the organization. From that moment the agent can activate the role and play it. The activate state allows the player to receive from the role requests for requirement execution and responsibilities (power invocation). Analogously, the *Player* class allows an agent to deact and deactivate a role.

The behaviour of playing a role is modelled in the player agent class by means of a finite state machine behaviour. The behaviour is instantiated for each instance of the role the agent wants to play, by invoking the method enact and specifying the organization AID and the role type. The states are inspired to the model of [11]:

- *Enact*. The communication protocol (which contains another finite state machine itself) for enacting roles is entered. If it ends successfully with the reception of the new role instance AID the deactivated state is entered. The hashtable containing the list of played roles is updated. Otherwise, the deacted state is reached.
- *Activate*. This state is modelled as a finite state machine behaviour which listens for events coming from outside or inside the agent:
    - If another behaviour of the agent decides to invoke a power of the role by means of the *invokePower* method (see below), the behaviour of the activated state checks if the power exists in the role specification, and sends an appropriate message to the role agent. Otherwise an exception is raised. If another behaviour of the agent decides to deactivate the role, the deactivated state is entered.
    - If a message requesting requirements or to invoke powers arrives from the role agent it plays, the agent will decide whether to comply with the new request sent by the role. First of all it checks that the required behaviour exists, or there has been a mismatch at the moment of enacting a role. If the role communicates to its player that the execution of a power is concluded, and sends the result of the power, this information is stored waiting to be passed back to the behaviour which invoked the power upon its request (see *receivePowerResult*).

  The cyclic behaviour associated with this state blocks itself if no event is present and waits for an event.
- *Deactivated*. The behaviour stops checking for the invocation of requirements or powers from respectively the role and the player itself, and blocks until another behaviour activates the role again. The messages from the role and the power invocations from other behaviours pile up in the queue waiting to be complied with, until an activation method is called and the active state is entered.
- *Deact*. The associated behaviour informs the role that the agent is leaving the role and cleans up all the data concerning the played role in the agent.

One instance of this finite state machine, that can be seen in Figure 2, is created for each role played by the agent. This means that, for a role, only one power at time is processed, while the others wait in the message queue. Note that the information whether a role is activated or not is local to the player: from the role's point of view there is no difference. However, the player processes the communication of the role only as long as it is activated, otherwise the messages remains in the buffer. More sophisticated solutions can be implemented as needed, but they must be aware of the synchronization problems (e.g., what happens if in the same moment a role sends a request to its player and the player puts the role in a deactivated state?).

The *Player* class offers some methods. They can be used in programming the other behaviours of the agent when it is necessary to make changes in the state of role playing or to invoke powers.We assume that invocations of powers to be asynchronous. The call returns a call id which is used to receive the correct return value in the same behaviour if necessary. It is left to the programmer how to manage the necessity of blocking of the behaviour till an answer is returned with the usual block instruction of JADE. This solution is coherent with the standard message exchange of JADE and allows to avoid using more sophisticated

behaviours based on threads. The methods offered by this class are the following.

- *enact(organizationAID, roleClassName)*: to request to enact a role an agent has to specify the AID of the organization and the name of the class of the role. It returns the AID of the role instance or an exception is raised.
- *invokePower(roleAID, power)*: to invoke a power it is sufficient to specify the role AID and the name of the behaviour of the role which must be executed. It returns an integer which represents the id of the invocation.
- *receivePowerResult(int)*: to receive, if needed, the result of the invocation of a power (which is identified by means of the id).
- *deact(roleAID), activate(roleAID), deactivate(roleAID)* respectively definitively deacts the role, killing it and managing the data structure to remove all references to it, activates a role agent that is in the deactivate state, and temporarily deactivates the role agent (e.g., immediately after a successfully enact, the role agent goes in the deactivate state).
- *addRequirement(String)*: when extending the *Player* class it is necessary to specify which of the behaviours defined in it are requirements. I.e., the list of behaviours which can be requested by a role which is played by the agent. This information is used in the *canPlay* private method which is invoked by the enact method to check if the agent can play a role. This list may contain non truthful information, but the failure to comply with the request of a commitment may result in the deactment to the role as soon as the agent is not able to satisfy the request to execute a certain requirement.

Moreover, defining a player requires to implement an abstract method to decide whether to execute the requirements upon request from the roles. The decision about the implementation of the method is

- *adoptGoal(String)*: it is used to make the player autonomous with reference to requests of role he plays: when the execution of a requirement is requested by the role, this method returns true if the agent decides to execute it.

III. Interaction between our actors

In this section we describe the different protocols used in the interaction between agents who want to play roles and organizations, and between players and their roles. All protocols use standard FIPA messages, to enable also non JADE agents to interact with organizations without further changes. Note that the protocols are always split in two part: the side of the initiator and the one of the responder. While the organization is only the responder of a protocol, roles and players can be both initiators and responder.

We refer to Figure 3, that describes the sequence diagram for interaction.

Agents and the organization. Behind the enacting state of the player described in the previous section, there is an enactment protocol inherited, respectively, as concerns the initiator and the receiver, from the classes *Player* and *Organization*. It forwards from the player to the organization the request of enacting a specified role, and manages the exchange of information: sending the specification of requirements and powers of the roles and checking whether the player complies with the requirements.

The organization listens from messages from any agent (even if some restrictions can be posed at the moment of accepting to create the role), while the subsequent communication between player and role is private. After a request from an agent, the behaviour representing the protocol forks creating another instance of itself to be ready to receive requests of other agents in parallel. The first message is sent by the player as initiator and is a request to enact a role. The organization, if it considers the agent authorized to play the role, returns to the candidate player a list of specifications about the powers and requirements of the requested role which are contained in its knowledge base, sending an inform message containing the list; otherwise, it denies to the player to play the role, answering with an inform message, indicating the failure of the procedure.

In case of positive answer, the player, invoking the method *canPlay* using the information contained in the player about the requirements, decides whether to respond to the organization that it can play the role (agree) or not (failure). The first answer results in the creation of a new role instance of the requested type (e.g., *Buyer*) and in the update of the knowledge base of the organization with the information that the player is playing the role. To the role instance the organization passes the AID of the role player, i.e., the initiator of the enactment, so that it can eventually filter out the messages not coming from its player. An inform is sent back to the player agent, telling him the played role instance's AID (since it is implemented as an agent it has an AID), The player, in this way, can address messages to the role and it can identify the messages it receives from the role it plays. Then the agent updates its knowledge base with this information, labeling the role as still deactivated. The protocol terminates in both the player and the organization. This completes the interaction with the organization: the rest of the interaction, including deacting the role, passes through the role instance only. The final part of the protocol shows how the player and the role communicate, for example, for a power request. The role can require the execution of one or more requirements, from which can depend the power execution.

Players and their roles. The interaction between a player and its role is regulated by three protocols: the request by the role of executing a requirement, the invocation of a power by the player, and the request of the role to invoke a power. In all cases, the interaction protocol works only between a player and the role instances it plays. Messages following the protocol, but which do not respect this constraint are discharged on both sides.

We start from the first case since it is used also in the second protocol during the execution of a power. According to Dastani et al. [11], if a role is activated, the player should (consider whether to) adopt its goals and beliefs. Since our model is distributed, the role is separated from its player: the goals (i.e., the requirements) and beliefs of the role have to be communicated from the role to its player by means of a suitable communication protocol. Each time the state of the role changes, since some new goal is added to it, the agent is informed by the role about it: either a requirement must be executed or a power must be invoked. In this protocol, the initiator is the role, which starts the behaviour when its method *requestRequirement* is invoked.

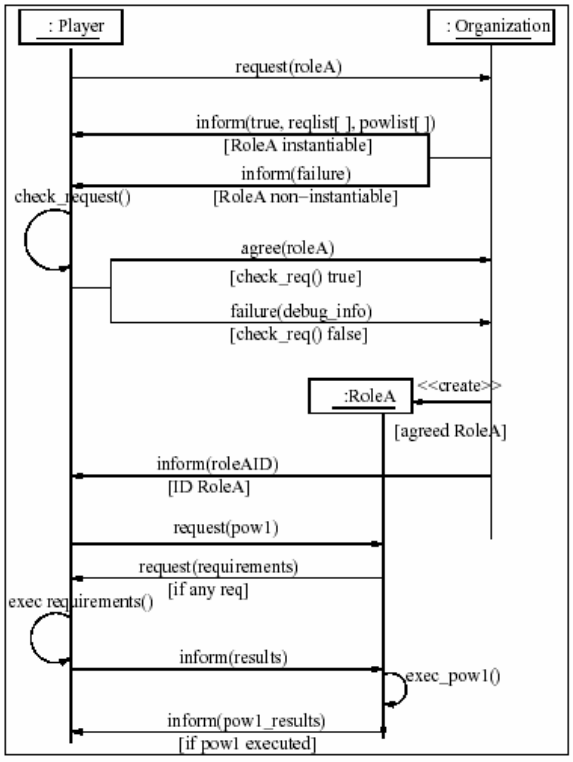


Fig. 3 - The interaction protocol

First of all, the agent checks if the requested requirement is in the list of the player's requirements, but this does not mean that it will be executed. Since the player agent is autonomous, before executing the requirement, it takes a decision by invoking the method *adoptGoal* which is implemented by the programmer of the player.

The protocol ends by informing the role about the outcome of the execution of the requirement or the refusal of executing it, using an "inform" (see bottom of Figure 2 b). This protocol is used inside the protocol initiated by the player for invoking a power of the role. After a request from the player, the role can reply with the request of executing some requirements which are necessary for the performance of the power.

In fact, in the behaviour corresponding to the power, some invocation of the method *requestRequirement* can be present. The protocol ends with the role informing the agent about the outcome of the execution of the power. A third protocol is used by the role to remind the agent about its responsibilities, i.e., the role asks its player to invoke a power executing the method *requestResponsibility.* In this case, the object of the request is not a requirement executable by the player, but a power, i.e., a behaviour of the role. So the player has to decide whether and when to invoke the power.

In principle, the programmer could have invoked a power directly from the role, instead of requesting it by means of *requestResponsibility*. However, with this mechanism we want to model the case where the player is obliged to invoke the power, but the decision of invoking the power is left to the player agent, who can have more information about when and how invoke the power. It is left to the programmer of the organization to handle the violation of such obligations.

The final kind of interaction between a player and its role is the request of a player to deact the role. While deactivation is an internal state of the player, which is not necessarily communicated to the role, deacting requires that the role agent is destroyed and that the organization clears up the information concerning the role and its player, removing the couple (*Player*, *Role*) from its data structures.

IV. Future developments

Our extension of JADE is inspired to the model of [7] which has been implemented in Java creating the language powerJava [3]. With respect to powerJava, there are similarities and differences, which are due to the fact that in powerJava agents have been reduced to objects, losing some features.

Few agent languages are endowed with primitives for modeling organization. MetateM [14] is one of these, and introduces the notion of group by enlarging the notion of agent with a context and a content. The context is composed by the agents (also groups are considered as agents, like in our model organizations are agents) which the agent is part of, and the content is a set of agents which are included. The authors propose to use these primitives to model organizations, defining roles as agents included in other agents and players as agents included in roles. This view risks

to forget the difference between the play relation and the role-of relation which have different properties (see, e.g., [23]).

Moreover it does not distinguish between powers. Finally MetateM is a language for modeling BDI agents, while JADE has a wider applicability and is built upon on the Java general purpose language. About SMoise+ features [20], we will improve our system with agent sets and subsets as particular inner classes in the *Organization* class. Very interesting is the matter of cardinality, constraint that we will implement considering both minimum than maximum cardinality allowed for each group. Concerning J-Moise+ [19], that is a combination of Jason [8] and Moise+ [18], we will enrich our platform integrating it with a rule engine (like Jess), becoming able to write beliefs and goals as rules. Very interesting is the matter of groups and schemes, that we will consider to implement.

The principles of permission will be implemented through a specific new protocol, called *Permissions*, which will allow to a role a call to another role's power, if and only if the first role's player can show (at the time of execution) his credentials (additional requirements); if no additional requirement is given, the other role's power invocation cannot be done. Another future work is related to *Obligations* [20]; we are going to implement them by particular requirements that have to produce some result in a fixed time. If no result is produced, then a violation occours and this behaviour is sanctioned in some way.

Planning goals too will be realized by requirements, that can be tested one after another to play single missions.

V. CONCLUSIONS

In this paper we use the ontological model of organizations proposed in [7] to program organizations. We use as agent framework JADE since it provides the primitives to program the MAS in Java. We define a set of Java classes which extends the agent classes of JADE to have further primitives for building organizations structured into roles.

Organizations and roles are implemented as extension of the Agent JADE class: this allows to communicate remotely with them using protocols based on FIPA speech acts, to identify organizations and roles with an AID rather than with a memory reference, to put on yellow-pages services the information about them, and to program them using JADE *Behaviours*.

Roles, modelled as inner classes in Java, can be viewed as agents and as objects at the same time, depending on the perspective: for the external world they are agents with an AID and communication capabilities, from the perspective of the organizations they are objects which can be programmed in the traditional way. Being inner classes, all roles of an organization can be programmed as a single program, since they all belong to the same namespace.

Organizations and roles are agents with a limited autonomy, since they can act only via the actions of the players of the roles. The interaction between them is made via a set of protocols: a protocol between the organization and an agent to start enacting a role, a protocol between the player and its role to invoke powers of a role, a protocol between the role and its player to invoke the requirements, etc. The interaction among roles can happen via normal agent protocol or directly by method invocation, since they belong to the same namespace.

To play a role, an agent has first to contact the organization and then to execute the requirements requested by the role. These requests for requirements represent the expected behaviour of an agent in its role. The agent who plays a role can be in different states with respect to the roles: enacting it, activated, deactivated and deacted.

It is possible to verify that a player fulfills the requirements during the execution and not only before the enactment of a role. In the JADE framework, it is up to the programmers to decide which requirements to provide to an agent implementation at design time by extending the class that contains the role specification. The advantage of our proposal is that we only verify the presence of those requirements that are actually used by the agent during the specific execution, without considering all those requirements that are not necessary to achieve the current goal. In perspective, if the protocol specification were available for inspection before the decision of playing a role, an agent could verify a priori if it owns all the requirements needed for achieving its goal, disregarding all the others, in the line of what presented in [1]. This approach is also important whenever an agent is allowed to decide whether satisfying a requirement depending on the context of the execution. In some cases, it might either decide not to use a requirement that it actually has, or select which requirement to use among a set of available requirements.

To define the organizational primitives, JADE offered advantages, but also posed some difficulties. First of all, being based on Java, it allowed to reapply the methodology used to implement roles in powerJava [3] to implement roles as inner classes. Moreover, being based on Java it provides a general purpose language to create new organizations and roles. Finally, being based on FIPA speech acts, it allows agents programmed in other languages to play roles in organizations, and viceversa, JADE agents to play roles in organizations not implemented in JADE. However, the decision of using JADE has some drawbacks. For example, the messages used in the newly defined protocols can be intercepted by other behaviours of the agents. This shows that a more careful implementation should use a more complex communication infrastructure to avoid this problem. Moreover, since JADE behaviours, differently from methods,

do not have a proper return value, they make it difficult to define requirements and powers.

Finally, due to the possible parallelism of behaviours inside an agent, possible synchronization problems can occur. Under the programming and debugging point of view, an objection could be that the (possible) execution of player and role on different platform (and, then, in a separate way), can create difficult during the debugging phase. To solve this problem it is necessary to develop a new debugging protocol, based on message passing, that will be used to communicate, for example, when an exception is raised by a power. This solution is perfectly lined up to the JADE philosophy.

In this paper we do not consider the problem of structuring organizations in suborganizations nor to model federated organizations residing on different platforms. A prototype implementation has been constructed, as described in the paper. The requirements mechanism will be used to implements many other important features for MAS: obligations and permissions, groups (sets) and subgroups (subsets), plans.

REFERENCES

[1] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Reasoning on choreographies and capability requirements. International Journal of Business Process Integration and Management IJBPIM, 2(4), 2007.

[2] M. Baldoni, G. Boella, and L. van der Torre. Modelling the interaction between objects: Roles as affordances. In Knowledge Science, Engineering and Management, First International Conference, KSEM 2006, volume 4092 of LNCS, pages 42–54. Springer, 2006.

[3] M. Baldoni, G. Boella, and L. van der Torre. Interaction between Objects in powerJava. Journal of Object Technology, 6(2):7 –12, 2007.

[4] F. L. Bellifemine, G. Caire, and D. Greenwood. Developing Multi-Agent Systems with JADE. Wiley, 2007.

[5] G. Boella, R. Damiano, J. Hulstijn, and L. van der Torre. ACL semantics between social commitments and mental attitudes. In International Workshops on Agent Communication, AC 2005 and AC 2006, volume 3859 of LNAI, pages 30–44. Springer, Berlin, 2006.

[6] G. Boella, V. Genovese, R. Grenna, and L. der Torre. Roles in coordination and in agent deliberation: A merger of concepts. In Proceedings of Multi-Agent Logics. PRIMA 2007. Lecture Notes in Computer Science, Springer, 2007.

[7] G. Boella and L. van der Torre. Organizations as socially constructed agents in the agent oriented paradigm. In Engineering Societies in the Agents World V, 5th International Workshop (ESAW'04), volume 3451 of LNAI, pages 1 –13, Berlin , 2005. Springer.

[8] R. H. Bordini, J. F. Hubner, and M. Wooldrige. Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley, 2007.

[9] G. Cabri, L. Ferrari, and L. Leonardi. Agent roles in the brain framework: Rethinking agent roles. In The 2004 IEEE Systems, Man and Cybernetics Conference, session on "Role-based Collaboration", 2004.

[10] A. Colman and J. Han. Roles, players and adaptable organizations. Applied Ontology, 2007.

[11] M. Dastani, B. van Riemsdijk, J. Hulstijn, F. Dignum, and J.-J. Meyer. Enacting and deacting roles in agent programming. In Procs. of AOSE'04, pages 189–204, New York, 2004.

[12] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: an organizational view of multiagent systems. In Agent-Oriented Software Engineering IV, 4th International Workshop (AOSE'03), volume 2935 of LNCS, pages 214–230, Berlin, 2003. Springer.

[13] FIPA. FIPA ACL message structure specification. Technical Report XC00061, Foundation for Intelligent Physical Agents, 2001.

[14] M. Fisher. A survey of concurrent metatem - the language and its applications. In ICTL, pages 480–505, 1994.

[15] M. Fisher, C. Ghidini, and B. Hirsch. Organising computation through dynamic grouping. In Objects, Agents, and Features, pages 117–136, 2003.

[16] D. Grossi, F. Dignum, M. Dastani, and L. Royakkers. Foundations of organizational structures in multiagent systems. In Procs. of AAMAS'05, pages 690–697, 2005.

[17] O. Gutknecht and J. Ferber. The madkit agent platform architecture. In Agents Workshop on Infrastructure for Multi-Agent Systems, pages 48–55, 2000.

[18] J. F. Hubner, J. S. Sichman, and O. Boissier. Developing organised multi-agent systems using the moise+ model: Programming issues at the system and agent levels. International Journal of Agent-Oriented Software Engineering., 2007.

[19] J. F. Huebner. J-Moise+ programming organizational agents with Moise+ and Jason. In http://moise.sourceforge.net/doc/tfg-eumas07-slides.pdf, 2007.

[20] J. F. Huebner, J. S. Sichman, and O. Boissier. S-moise+: A middleware for developing organised multi-agent systems. In O. Boissier, J. A. Padget, V. Dignum, G. Lindemann, E. T. Matson, S. Ossowski, J. S. Sichman, and J. Vzquez-Salceda, editors, AAMAS Workshops, volume 3913 of Lecture Notes in Computer Science, pages 64–78. Springer, 2005.

[21] A. Jones and M. Sergot. A formal characterisation of institutionalised power. Journal of IGPL, 3:427–443, 1996.

[22] A. Omicini, A. Ricci, and M. Viroli. An algebraic approach for modelling organisation, roles and contexts in MAS. Applicable Algebra in Engineering, Communication and Computing, 16(2-3):151–178, 2005.

[23] F. Steimann. On the representation of roles in object-oriented and conceptual modelling. Data and Knowledge Engineering, 35:83–848, 2000.

[24] N. Tinnemeier, M. Dastani, and J.-J. C. Meyer. Orwell's nightmare for agents? programming multi-agent organisations. In Sixth international Workshop on Programming Multi-Agent Systems PROMAS'08, 2008.

[25] W. van der Hoek, K. Hindriks, F. de Boer, and J.-J. C. Meyer. Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems, 2(4):357–401, 1999.

[26] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. IEEE Transactions of Software Engineering and Methodology, 12(3):317– 370, 2003.

Matteo Baldoni is Associated Professor at the Dipartimento of Informatica, Università di Torino, Corso Svizzera 185, Torino, 10149, Italy. E-mail: baldoni@di.unito.it .

Guido Boella is Associated Professor at the Dipartimento of Informatica, Università di Torino, Corso Svizzera 185, 10149, Torino, Italy. E-mail: guido@di.unito.it.

Mauro Dorni is student at the Dipartimento of Informatica, Università di Torino, Corso Svizzera 185, 10149, Torino, Italy. E-mail: sp064535@educ.di.unito.it.

Andrea Mugnaini is student at the Dipartimento of Informatica, Università di Torino, Corso Svizzera 185, 10149, Torino, Italy. E-mail: sp064278@educ.di.unito.it.

Roberto Grenna is PhD student at the Dipartimento of Informatica, Università di Torino, Corso Svizzera 185, 10149, Torino, Italy E-mail: grenna@di.unito.it.

Supporting the Design of Self-Organizing Ambient Intelligent Systems Through Agent–Based Simulation

Stefania Bandini, Andrea Bonomi, Giuseppe Vizzari
Complex Systems and Artificial Intelligence research center, University of Milano–Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{stefania.bandini,andrea.bonomi,giuseppe.vizzari}@csai.disco.unimib.it

Abstract— **The ambient intelligence scenario depicts electronic environments that are sensitive and responsive to the presence of people. The aims of this kind of system is not necessarily to provide some form of electronic service to its users, but also to enhance the everyday experience of people moving inside the related physical environment. For this second type of application, computer simulation represents a useful way to envision the behaviour of responsive environments without actually bringing them into existence in the real world. This paper will describe the simulation of an adaptive illumination facility, a physical environment endowed with a set of sensors that perceive the presence of humans (or other entities such as dogs, bicycles, cars) and interact with a set of actuators (lights) that coordinate their state to adapt the ambient illumination to the presence and behaviours of its users. This system is made up of a model managing the self-organization of the adaptive illumination facility and an agent–based model to simulate pedestrian dynamics in the physical environment in which the system is deployed.**

I. Introduction

The ambient intelligence scenario [14] depicts future human environments endowed with a large number of electronic devices, interconnected by means of wireless communication facilities, able to perceive and react to the presence of people.

The goals of these facilities can be very different, from providing electronic services to humans accessing these environment through computational devices (such as personal computers or PDAs), to simply providing some form of ambient adaptation to the users' presence (or voice, or gestures), without thus requiring him/her to employ a computational device. Ambient intelligence comprises thus those systems that are designed to autonomously adapt the environment to the people living or simply passing by in it in order to improve their everyday experience.

Besides the specific aims of the ambient intelligent system, there is an increasing interest and number of research efforts on approaches, models and mechanisms supporting forms of self-organization and management of the components (both hardware and software) of such systems. The latter are growingly viewed in terms of autonomous entities, managing internal resources and interacting with surrounding ones so as to obtain the desired overall system behaviour as a result of local actions and interactions among system components. Examples of this kind of approach can be found in both in relatively traditional pervasive computing applications (see, e.g., [8]), but also in a new wave of systems developed in the vein of amorphous computing [2] such as the one on paintable computers described in [7]. In this rather extreme application a whole display architecture is composed of autonomous and interacting graphic systems, each devoted to a single pixel, that must thus interact and coordinate their behaviours even to display a simple character.

Computer simulation plays an important role in supporting the design and realization of adaptive, self-organizing ambient intelligence systems. In fact, traditional design and modeling instruments can provide a suitable support for evaluating static properties of this kind of environment (e.g. through the construction of 3D models representing a mock-up, proof of concept of the desired appearance or also adaptation effect but in a single specific situation), but they are not designed to provide abstractions and mechanisms for the definition and simulation of reactive environments and their behaviours. Through specific models and simulators it is possible to obtain an envisioning of the static features of the ambient intelligence system as well as its dynamic response to the behaviour of humans and other relevant entities situated in it. This allows performing a *face validation* [13] of the adaptation mechanisms and also to perform a tuning of the relevant parameters.

This paper describes the application of a modeling and simulation approach to support the design of an adaptive illumination facility that is being designed and realized by the Acconci Studio[1] in Indianapolis. In particular, the designed system should be able to locally enhance the overall illumination of a tunnel in order to highlight the position and close surrounding area of pedestrians (as well as other entities such as dogs, bicycles, cars). In this case, the simulation offers both a support to the decisions about the number and positioning of lights and, more important, it encapsulates the self-organization mechanisms guiding the adaptive behaviour of lights reacting the the presence of pedestrians and other relevant entities in the environment. By providing the current state of the environment, in terms of simulated outputs of sensors detecting the presence of pedestrians, as an input to the self-organization model it is possible to obtain its simulated response, and the current state of lights. A schema

[1] http://www.acconci.com

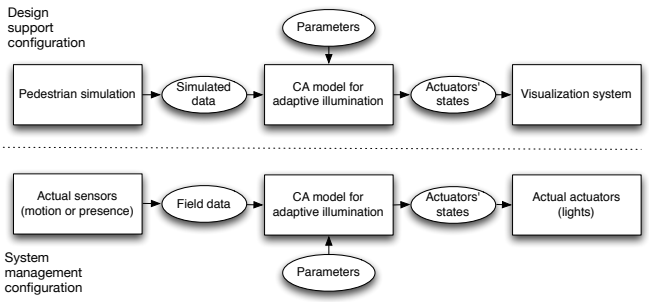


Fig. 1. A schema describing the modules of the design support system prototype.

of the overall simulation system is shown in Figure 1: it must be noted that the self-organization model adopted for the simulator could be effectively used to manage the actual system, simply providing actual inputs from field sensors and employing its outputs to manage actual lights rather that a virtual visualization of the actual environment.

The following section will introduce more in details the specific scenario in which this research effort is set, describing the requirements for the adaptive illumination system and the environment adaptation model. Section IV introduces the pedestrian modeling approach, while the self-orga-nization model guiding the adaptive illumination facility is described in Section III. A description of the developed environment supporting designers will follow, then conclusions and future works will end the paper.

II. The Scenario

The Acconci Studio, partner of the described research effort, has recently been involved in a project for the renovation of a tunnel in the Viginia Avenue Garage in Indianapolis. The tunnel is currently mostly devoted to cars, with relatively limited space on the sidewalks and its illumination is strictly functional. The planned renovation for the tunnel includes a set of interventions, and in particular two main effects of illumination, also depicted in a graphical elaboration of the desired visual effect shown in Figure 2: an overall effect of *uniformly coloring* the environment through a background, ambient light that can change through time, but slowly with respect to the movements and immediate perceptions of people passing in the tunnel; a *local effect of illumination* reacting to the presence of pedestrians, bicycles, cars and other physical entities.

The rationale of this local and dynamic adaptive illumination effect is better explained by the following narrative description of the desired effect:

> The color is there to make a heaviness, a thickness, only so that the thickness can be broken. The thickness is pierced through with something, theres a sparkle, its you that sparkles, walking or cycling though the passage, this tunnel of color. Well no, not really, its not you: but its you that sets off the sparkle a sparkle here, sparkle there, then another sparkle in-between one sparkle affects the other, pulls the other, like a magnet a point of sparkle is stretched out into a line of sparkles is stretched out into a network of sparkles.
> These sparkles are above you, below you, they spread out in front of you, they light your way through the tunnel. The sparkles multiply: its you who sets them off, only you, but – when another person comes toward you in the opposite direction, when another person passes you, when a car passes by some of these sparkles, some of these fire-flies, have found a new attractor, they go off in a different direction.

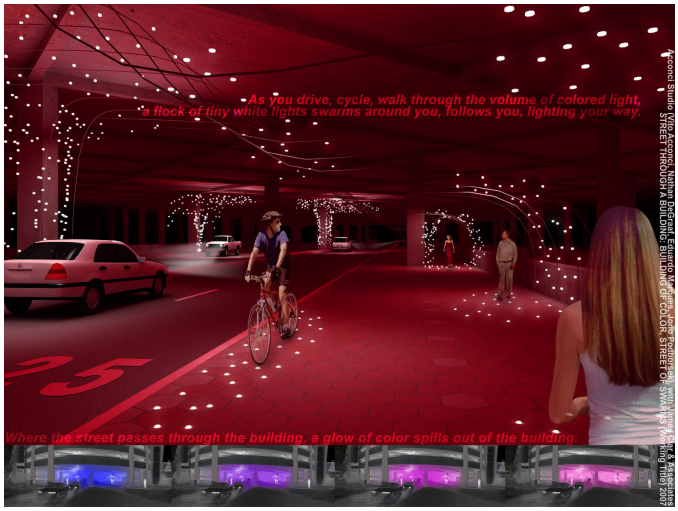


Fig. 2. A visual elaboration of the desired adaptive illumination facility (the image appears courtesy of the Acconci Studio).

The first type of effect can be achieved in a relatively simple and centralized way, requiring in fact a uniform type of illumination that has a slow dynamic. The second point requires a different view on the illumination facility. In particular, it must be able to perceive the presence of pedestrians and other physical entities passing in it, in other words it must be endowed with sensors (detecting either the presence or the movement of relatively big objects). Moreover, it must be able to exhibit local changes as a reaction to the outputs of the aforemetioned sensors, providing thus for a non uniform component to the overall illumination. The overall environment must be thus split into parts, cells that represent proper subsystems: Figure 3 shows a schema of the approach we adopted to subdivide the physical environment into autonomous units, provided with motion/presence sensors (able to detect the arrival/presence of relevant entities) and lights (to adapt the ambient illumination, highlighting the presence of pedestrians).

However, the effect of the presence of a pedestrian in a portion of space should extend beyond the borders of the occupied cell. In fact, the illumination effect should "light the way" of a pedestrian through the tunnel. Cells must thus be able to interact, in order to influence neighboring ones whenever a pedestrian is detected, to trigger a (maybe less intense) illumination. The model we adopted to manage this form of self-organization of the illumination facility is a Cellular Automata (CA) [15], whose transition rule defines and

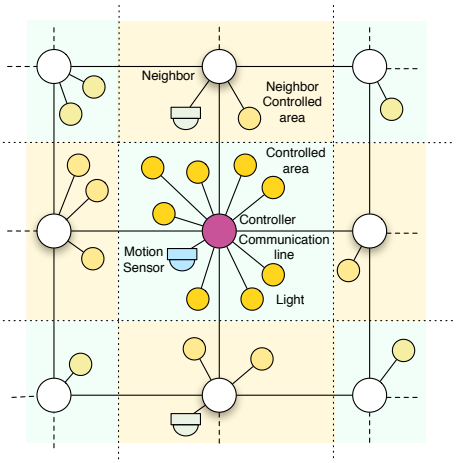


Fig. 3. A schema of the CA model for the adaptive illumination facility.

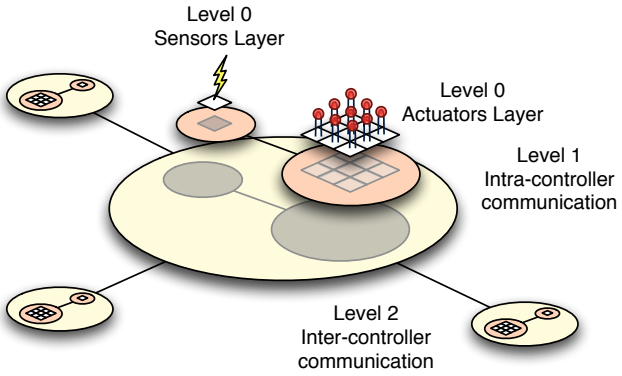


Fig. 4. The proposed automata network for the D-MAN.

manages the interaction among cells and thus the influence of the presence of a pedestrian on neighboring ones.

III. Adaptive Illumination Model

We employed a Cellular Automata model to realize the local effect of illumination as a self-organized reaction to the presence of pedestrians. CA cells, related to a portion of the physical environment, comprise sensors and actuators, as schematized in Figure 3. The former can trigger the behaviours of the latter, both through the interaction of elements enclosed in the same cell and by means of the local interaction among adjacent cells. The transition rule models mechanisms of reaction and diffusion, and it was derived by previous applications to reproduce natural phenomena such as percolation processes of pesticides in the soil, in percolation beds for the coffee industry and for the experimentation of elasticity properties of batches for tires [3]. In this specific application the rule manages the interactions of cells arranged through a multilayered architecture based on the Multilayered Automata Network model [6], schematized in Figure 3.

Multilayered Automata Network have been defined as a generalization of Automata Networks [10]. The main feature of the Multilayered Automata Network is the explicit introduction of a hierarchical structure based on nested graphs, that are graphs whose vertexes can be in turn be a nested graph of lower level. A Multilayered Automata Network is directly obtained from the nested graph structure by introducing states and a transition function.

The irregular nature of the cellular space is not the only difference between the adopted approach and the traditional CA models. In fact, CAs are in general closed and synchronous systems, in which cells update their state in parallel triggered by a global clock. Dissipative Cellular Automata (DCA) [16] differ from the basic CAs mainly for two characteristics: while CA are synchronous and closed systems, DCA are open and asynchronous. DCA cells are characterized by a thread of control of their own, autonomously managing the elaboration of the local cell state transition rule. DCA can thus be considered as an open agent system [12], in witch the cells update their state independently of each other and they are directly influenced by the environment.

The model we defined and adopted, Dissipative Multilayered Automata Network (D-MAN), takes thus the advantages of both the Multilayered Automata Network and the Dissipative Cellular Automata. An informal definition this model describes D-MAN as Multilayered Automata Network in which the cells update their state in an asynchronous way and they are open to influences by the external environment.

The multilayered cellular structure of the D-MAN is composed of three layers: the first level is related to the basic *discretization* of the physical environment into cells, corresponding to a local controller. Each of these cells comprises two additional levels, respectively devoted to the *perception* and *actuation* responsibilities of the higher level cell. This structure is schematized in Figure 4. Specific transition rules must thus be defined to manage different interactions and influences that take place in this structure, and mainly (i) the direct influence of a sensor that detected a pedestrian to the actuators in the same cell, and (ii) the influence of a high level cell to the neighboring ones (given the internal structure of each cell, due to the presence of a specific level of actuators inside it, this interaction effectively affects *a part* of a neighboring cell). Moreover, the effect of external stimuli must gradually vanish, and lights must fade in absence of pedestrians.

The adaptive illumination model is thus characterized by several features that make it difficult to predict how it will react to particular stimuli (i.e. patterns of pedestrian movement in the related environment), from the number and positioning of sensors and actuators, to the parameters of the transition rule. To couple this model with a pedestrian simulation model sharing the discrete representation of the spatial aspect of the environment allows to simulate the behaviour of the adaptive illumination facility as a response to specific patterns of usage of the environment by pedestrians.

A. Model Architecture

The designed system is an homogeneous peer system. As shown in Figure 3, every controller has the responsibility of managing the sensors and actuators belonging to a fixed area of

the space. Controllers are homogeneous in terms of hardware and software capabilities. Every controller is connected to a motion sensor, which roughly covers the controlled area, some lights (about 40 LED lights) and neighbouring controllers.

As shown in Figure 4, the external layer (level 2) is the communication layer between the controllers of the system. Every controller is an automata network of two nodes, one node is a sensor communication layer and represents a space in which every sensor connected to the microcontroller has a correspondent cell. The other node represents the actuators' layer in which the cells pilot the actuators (lights, in our case). Since the external layer is a physical one and every cell is an independent microcontroller, it cannot be assumed that the entrire network is synchronized. In same cases, a synchronous network can be constructed (for example, a single clock devices can be connected to each microcontrollers or the microcontrollers can be synchronized by a process without a master node), but the most general case is an asynchronous network.

B. Sensors Layer

The Sensor Layer is a Level 0 Dissipative Automata. As previously introduced, it is composed of a single cell, since only one sensor is connected to each microcontroller. It is a Dissipative Automata because the internal state of the cell is influenced by the external environment. The state of the cell is represented by a single numerical value $v_s \in \mathbb{N}_{8bit}$, where

$$\mathbb{N}_{8bit} \subset \mathbb{N}_0, \forall x : x \in \mathbb{N}_{8bit} \Rightarrow x < 2^8$$

The limit value was chosen for performance reasons because 8-bit microcontrollers are widely diffused and they can be sufficiently powerful to manage this kind of situation. The value of v_s is computed as

$$v_s(t+1) = v_s(t) \cdot m + s(t+1) \cdot (1-m)$$

where $m \in \mathbb{R}, 0 \leq m \leq 1$ is the *memory coefficient* that indicates the degree of correlation between the previous value of v_s and the new value, $s(t) \in N_{8bit}$ is the reading of the sensor at the time *s(t)*. If the sensor is capable of distance measuring, *s(t)* is inverse proportional to the measured distance (so, if the distance is 0, the value is 255, if the distance is ∞ the value is 0). If the sensor is a motion detector sensor (it able to signal 1 if an object is present or 0 otherwise) *s(t)*, s(t) is equal to 0 if there is not detected motion, *c* in case of motion, where $c \in \mathbb{N}_{8bit}$ is a constant (in our tests, 128 and 192 are good values for *c*).

C. Diffusion Rule

In this section we describe the diffusion rule, that is used to propagate the sensors signals through the system. At a given time, every level 2 cell is characterized by an intensity of the signal, $v \in \mathbb{N}_{8bit}$. Informally, the value of v at time $t+1$ depends of the value of v at time t and on the value of $v_s(t+1)$, to capture both the aspects of interaction with neighbouring cells and the memory of the previous external stimulus caused by the presence of a physical entity in the area associated to the cell.

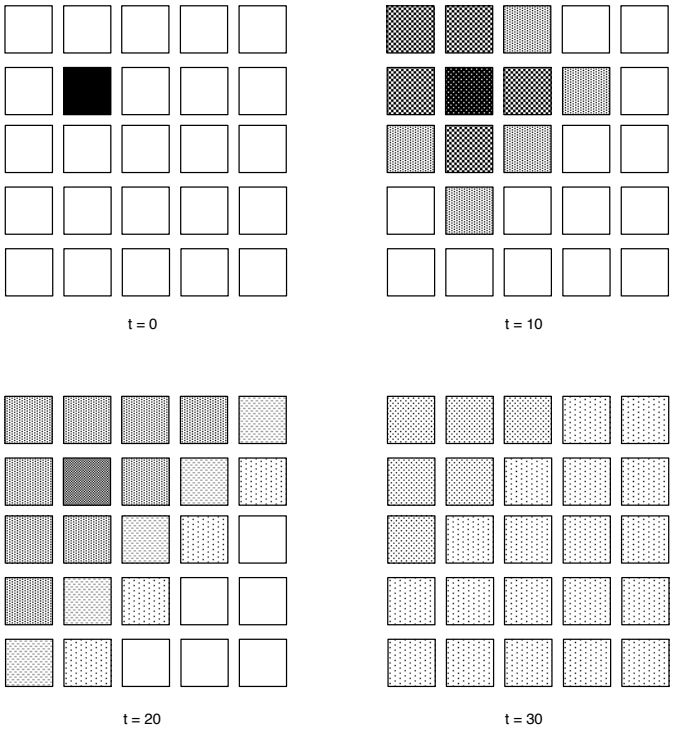


Fig. 5. An example of the dynamic behaviour of a diffusion operation. The signal intensity is spread throughout the lattice, leading to a uniform value; the total signal intensity remains stable through time, since evaporation was not considered.

The intensity of the signal decreases over time, in a process we call evaporation. In particular, let us define $\epsilon_{evp}(v)$ as the function that computes the quantity of signal to decrement from the signal and is defined as

$$\epsilon_{evp}(v) = v \cdot e_1 + e_0$$

where $e_0 \in \mathbb{R}^+$ is a constant evaporation quantity and $e_1 \in \mathbb{R}, 0 \leq e_1 \leq 1$ is the evaporation rate (e.g. a value of 0.1 means a 10% evaporation rate).

The evaporation function $evp(v)$, computing the intensity of signal v from time t to $t+1$, is thus defined as

$$evp(v) = \begin{cases} 0 & \text{if } \epsilon_{evp}(v) > v \\ v - \epsilon_{evp}(v) & \text{otherwise} \end{cases}$$

The evaporation function is used in combination with the neighbours' signal intensities to compute the new intensity of a given cell. We first present the formula for a regular neighbourhood and than we generalize to the irregular structure.

1) Regular neighbourhood: The automaton is contained in the finite two-dimensional square grid $\mathbb{N}^2$. We suppose that the cell $C_{i,j}$ is located on the grid at the position *i, j*, where $i \in \mathbb{N}$ and $j \in \mathbb{N}$. According to the von Neumann neighbourood [11], a cell $C_{i,j}$ (unless it is placed on the border of the lattice) has 4 neighbours (as shown in figure 6), denoted by $C_{i-1,j}$, $C_{i,j+1}$, $C_{i+1,j}$, $C_{i,j-1}$.

For simplicity, we numbered the neighbours of a cell from 1 to 4, so for the cell $C_{i,j}$, N_1 is $C_{i-1,j}$, N_2 is $C_{i,j+1}$, N_3 is $C_{i+1,j}$, and N_4 is $C_{i,j-1}$

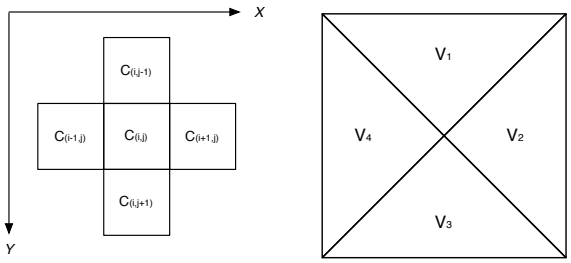


Fig. 6. On the left, the von Neumann neighbourhood of the cell $C_{i,j}$, on the right, the internal structure of a cell of the regular automaton.

At a given time, every cell is characterize by an intensity of the sensor signal. Each cell is divided into four parts (as shown in Figure 6), each part can have a different signal intensity, and the overall intensity of the signal of the cell is the sum of the parts intensity values. The state of each cell $C_{i,j}$ of the automaton is defined by $C_{i,j} = \langle v_1, v_2, v_3, v_4 \rangle$ where $v_1, v_2, v_3, v_4 \in \mathbb{N}_{8bit}$ represent the intensity of the signal of the 4 subparts. $V_{i,j}(t)$ represents the total intensity of the signals (i.e. the sum of the subparts signal intensity) of the cell i, j at time t. The total intensity of the neighbours are denoted by V_{N1}, V_{N2}, V_{N3}, and V_{N4}. The signal intensity of the subparts and the total intensity are computed with the following formulas:

$$v_j(t+1) = \begin{cases} \frac{evp(V(t))\cdot q + evp(V_{Nj}(t))\cdot(1-q)}{4} & \text{if } \exists N_j \\ \frac{evp(V(t))}{4} & \text{otherwise} \end{cases}$$

$$V(t+1) = \sum_{i=1}^{4} v_i(t+1)$$

where $q \in \mathbb{R}, 0 \leq q \leq 1$ is the conservation coefficient (i.e. if q is equals to 0, the new state of a cell is not influenced by the neighbours values, if it is equals to 0.5 the new values is a mean among the previous value of the cell an the neighbours value, if it is equals to 1, the new value does not depend on the previous value of the cell but only from the neighbours). The effect of this modeling choice is that the parts of cells along the border of the lattice are only influenced through time by the contributions of other parts (that are adjacent to inner cells of the lattice) to the overall cell intensity.

2) Irregular neighbourhood: The irregular structure automata is a generalization of the regular one. The automaton is composed of cell numbered from 1 to N, so we use C_i for $0 \leq i \leq N$ to indicate the i-th cell. Every cell C_i can have an arbitrary number of neighbours $L_i, 0 \leq L_i \leq L \leq N-1$ where L_i is the numbers of neighbours of the cell C_i and $L = \max(L_i)$ is the maximum numbers of neighbours of every cell the system. Neighbouring cells of cell i can be denoted as $N_{i,l}$.

As for the regular neighbourhood case, each cell is divided into L parts, each part can have a different signal intensity, and the overall intensity of the signal of the cell is the sum of the parts intensity values. The state of each cell C_i of the automaton is defined as $V_i = \sum_{l=0}^{L_i} v_{i,l}$ where $v_{i,l} \in \mathbb{N}_{8bit}$ represent the intensity of the signal of the L subparts. Finally, the intensity of each neighbouring cell of C_i is denoted by $V_{i,l}$.

The signal intensity of the subparts and the total intensity can thus be computed according to the following formulas:

$$v_{i,l}(t+1) = \begin{cases} \frac{evp(V_i(t))\cdot q + evp(V_{i,l}(t))\cdot(1-q)}{L} & \text{if } \exists N_{i,l} \\ \frac{evp(V_i(t))}{L_i} & \text{otherwise} \end{cases}$$

$$V_i(t+1) = \sum_{l=1}^{L_i} v_{i,l}(t+1)$$

In the real system, the maximum number of neighbours (L) is constrained by the number of available inputs on the microcontrollers.

D. Actuators Layer

The cells of the actuator layer determinate the actuators actions. In this project the actuators are LED lamps that are turned on and of according the the state of the cell. Instead of controlling a single LED from a cell, every cell is related to a group of LEDs disposed in the same (small) area.

In the case of regular neighbourhood, each controlled area in divided into 9 sub-areas and each sub-area contains a group of LEDs controlled by the same actuators layer cell. The state of each cell is influenced only by the state of the signal intensity of the upper layer cell. The correlation between the upper layer cell subparts and the actuators layer cells is shown in Figure 7.

The state of the actuators cells $A_1..A_9, A_j \in N_{8bit}$ is computed with the following formula:

$$A_i(t+1) = \begin{cases} v_i(t+1) & 1 \leq i \leq 4 \\ \frac{v_4(t+1) + v_1(t+1)}{2} & i = 5 \\ \frac{v_1(t+1) + v_2(t+1)}{2} & i = 6 \\ \frac{v_2(t+1) + v_3(t+1)}{2} & i = 7 \\ \frac{v_3(t+1) + v_4(t+1)}{2} & i = 8 \\ \frac{1}{4}\sum_{j=1}^{4} v_j(t+1) & i = 9 \end{cases}$$

There are differents approache to associtates the LEDs to the cells. A first approach consists to directly connected the lights intensity to the signal levels of the correspondent cell. Another approach consists to turn on a numbers of LEDs proportionals to the signal intensity of the controller cell.

IV. Pedestrian Simulation Model

The adopted pedestrian model is based on the Situated Cellular Agent model, a specific class of Multilayered Multi-Agent Situated System (MMASS) [4] providing a single layered spatial structure for agents environment. A thorough description of the model is out of the scope of this paper, but we briefly introduce it to give some basic notion of the elements that are necessary to describe the SCA crowd modeling approach.

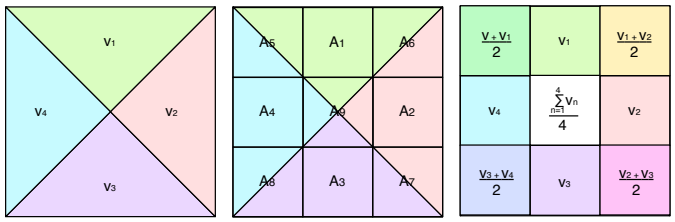


Fig. 7. Correlation between the upper layer cell subparts and the actuators layer cells.

A. Situated Cellular Agents

A *Situated Cellular Agent* system is defined by the triple $\langle Space, F, A \rangle$ where *Space* models the environment where the set A of agents is situated, acts autonomously and interacts through the propagation of the set F of fields and through reaction operations. *Space* consists of a set P of sites arranged in a network (i.e. an undirected graph of sites). The structure of the space can be represented as a neighborhood function, $N : P \rightarrow 2^P$ so that $N(p) \subseteq P$ is the set of sites adjacent to $p \in P$; the previously introduced *Space* element is thus the pair $\langle P, N \rangle$. Focusing instead on the single basic environmental elements, a site $p \in P$ can contain at most one agent and is defined by the 3–tuple $\langle a_p, F_p, P_p \rangle$ where:

- $a_p \in A \cup \{\perp\}$ is the agent situated in p ($a_p = \perp$ when no agent is situated in p that is, p is empty);
- $F_p \subset F$ is the set of fields active in p ($F_p = \emptyset$ when no field is active in p);
- $P_p \subset P$ is the set of sites adjacent to p (i.e. $N(p)$).

A SCA agent is defined by the 3–tuple $< s, p, \tau >$ where τ is the *agent type*, $s \in \Sigma_\tau$ denotes the *agent state* and can assume one of the values specified by its type (see below for Σ_τ definition), and $p \in P$ is the site of the *Space* where the agent is situated. As previously stated, agent *type* is a specification of agent state, perceptive capabilities and behaviour. In fact an agent type τ is defined by the 3–tuple $\langle \Sigma_\tau, Perception_\tau, Action_\tau \rangle$. Σ_τ defines the set of states that agents of type τ can assume. $Perception_\tau : \Sigma_\tau \rightarrow [\mathbf{N} \times W_{f_1}] \ldots [\mathbf{N} \times W_{f_{|F|}}]$ is a function associating to each agent state a vector of pairs representing the *receptiveness coefficient* and *sensitivity thresholds* for that kind of field. $Action_\tau$ represents instead the behavioural specification for agents of type τ. Agent behaviour can be specified using a language that defines the following primitives:

- $emit(s, f, p)$: the $emit$ primitive allows an agent to *start the diffusion of field* f on p, that is the site it is placed on;
- $react(s, a_{p_1}, a_{p_2}, \ldots, a_{p_n}, s')$: this kind of primitive allows the specification of a *coordinated change of state* among adjacent agents. In order to preserve agents' autonomy, a compatible primitive must be included in the behavioural specification of all the involved agents; moreover when this coordination process takes place, every involved agents may dynamically decide to effectively agree to perform this operation;
- $transport(p, q)$: the $transport$ primitive allows the definition of *define agent movement* from site p to site q (that must be adjacent and vacant);
- $trigger(s, s')$: this primitive specifies that an agent must *change its state* when it senses a particular condition in its local context (i.e. its own site and the adjacent ones); this operation has the same effect of a reaction, but does not require a coordination with other agents.

For every primitive included in the behavioural specification of an agent type specific preconditions must be specified; moreover specific parameters must also be given (e.g. the specific field to be emitted in an emit primitive, or the conditions to identify the destination site in a transport) to precisely define the effect of the action, which was previously briefly described in general terms.

Each SCA agent is thus provided with a set of sensors that allows its interaction with the environment and other agents. At the same time, agents can constitute the source of given fields acting within a SCA space (e.g. noise emitted by a talking agent). Formally, a field type t is defined by $\langle W_t, Diffusion_t, Compare_t, Compose_t \rangle$ where W_t denotes the set of values that fields of type t can assume; $Diffusion_t : P \times W_f \times P \rightarrow (W_t)^+$ is the diffusion function of the field computing the value of a field on a given space site taking into account in which site (P is the set of sites that constitutes the SCA space) and with which value it has been generated. It must be noted that fields diffuse along the spatial structure of the environment, and more precisely a field diffuses from a source site to the ones that can be reached through arcs as long as its intensity is not voided by the diffusion function. $Compose_t : (W_t)^+ \rightarrow W_t$ expresses how fields of the same type have to be combined (for instance, in order to obtain the unique value of field type t at a site), and $Compare_t : W_t \times W_t \rightarrow \{True, False\}$ is the function that compares values of the same field type. This function is used in order to verify whether an agent can perceive a field value by comparing it with the sensitivity threshold after it has been modulated by the receptiveness coefficient.

B. SCA Based Pedestrian Model

The above introduced SCA model has beed applied to represent a very simple tunnel with two ends and some columns in it; pedestrians enter the tunnel from one end and they move towards the other end, avoiding obstacles either immobile (i.e. columns), and mobile (i.e. other pedestrians moving in the opposite direction).

The SCA *Space* is the same cellular space defined for the D-MAN described in Section III. To support agent navigation in this space, in each end of the tunnel we positioned an additional site in which a "beacon" agent (a static agent emitting a simple presence field) is situated. In the environment, thus, only two types of field are present.

To exploit this environmental specification in order to obtain the above overall system behaviour, we defined two types of agent, respectively interpreting the one type of field as attractive and ignoring the other one. This can be achieve through a simple $transport$ primitive, specifying that the agent should move towards the free adjacent site in which the intensity of the field considered attractive is maximum.

The behavioural specification of these agents is completed by an obstacle avoidance rule (another $transport$ that moves the agent towards a random different lane whenever the best possible destination is occupied by an obstacle). Finally, agents reaching their destination, that is, one of the tunnel ends, are removed from the environment and they are positioned at the other end, so they start over their crossing of the tunnel.

V. The Design Support Environment

The design of human environments (e.g. buildings, stores, squares, roads) is a complex task, composed of several subtask evolving the initial idea into a detailed project, through the production of intermediate and increasingly detailed models.

After the initial phases, in which the designer usually expresses his/her creativity with sketches on the paper or on the computer, a Computer-Aided Design (CAD) software is used to develop the project in details. CAD softwares (e.g AutoCAD), and also 3D modelling applications (e.g. Autodesk 3DStudio Max, Blender) are used to create the digital models for the projects and to generate photo realistic renderings and animations. For a compact overview of the typical design process see [9].

Together with these software applications supporting designers in the definition of general architectural spaces, other tools supporting designers in very specific tasks can also be adopted: these tools vary from presenting the elaboration of the building shadows, to elaborating their impact on wind conditions, up to the simulation of vehicles and pedestrian movements in the designed scenario.

The proposed design environment is one of these tools; in particular, it helps the designers in the definition and specification of an adaptive illumination facility through the simulation of its dynamic behaviour. The output of the system is not only a graphical simulation but also a static configuration of the illumination facility (number of lights and their positioning) and an unambiguous specification of their dynamic behaviour (general lights self-organization model plus specific parameters).

The design environment is composed of two main modules: a simulation environment (that is in turn decomposed into a pedestrian simulation module and an adaptive illumination module) and a visualization facility. In the following paragraphs these modules will be described.

A. The Simulation Environment

The simulation environment actually comprises two models, one managing the network of controllers (with sensors and actuators), the other simulating the environment in which the adaptive illumination facility is situated and the pedestrians situated in it. The two simulations are connected: in particular, the state of sensors of is influenced by the state of the environment simulation.

The environment simulator, that is based on the MMASS [5], can be used to perform pedestrian simulation. This module actually feeds the self-organization model with simulated field data. The previously described CA model managing the self-organization of the illumination facility will react according to the current occupation of the space in the environment and according to its own parameters.

In this way, the designer can effectively envision the interaction between the people an the specified adaptive environment. The simulation environment allows the designer in configuring the network, defining the type, number, position of the sensors and actuators, and in specifying the behavior of the controllers, by means of defining the parameters of the CA model.

B. The Visualization Facility

The system supports both a 2D and 3D visualization of the simulated environment and the state of the two different enclosed models. The 2D visualization can be interactive, so it is possible to define an action event to be fired on a click (e.g. simulate the perception of a pedestrian when the designer clicks on a cell). This is useful because allows the designer to test the system behavior before specifying in an extensive way a pedestrian simulation scenario.

The 3D visualization is useful to understand the behavior of the system. It is not a photo realistic rendering but a real-time representation of simulated system. During the simulation, the user can navigate the 3D space, changing his/her point of view, for instance, taking the perspective of one of the pedestrians walking in the environment. It is possible to load 3D models both for the space and for pedestrian agents. The 3D visualization is based on the jMonkey engine[2]; the API of this open source project allows loading several 3D model formats. A screenshot of the visualization system is shown in Figure 8.

VI. Future development

The paper introduced a simulation approach to supporting the design of an ambient intelligence infrastructure aimed at improving the everyday experience of pedestrians and people passing through the related environment. A specific scenario related to the definition and development of an adaptive illumination facility was introduced, and a CA-based model specifying its dynamic behaviour was defined. An agent-based pedestrian model simulating inputs and stimuli to the adaptation module was also introduced. A prototype of a system supporting designers in the definition of the relevant parameters for this model and for the overall illumination facility was finally described.

The renovation project is currently under development on the architectural and engineering side, whereas the CA-based model has shown its adequacy to the problem specification, both in order to provide a formal specification of the behaviour for the system components and possibly as a centralized control mechanism. The realized prototype explored the possibility of realizing an ad hoc tool that can integrate the traditional CAD systems for supporting designers in simulating and envisioning the dynamic behaviour of complex, self-organizating installations. It has been used to understand the adequacy of the modeling approach in reproducing the desired self-organized adaptive behaviour of the environment

[2]http://www.jmonkeyengine.com/

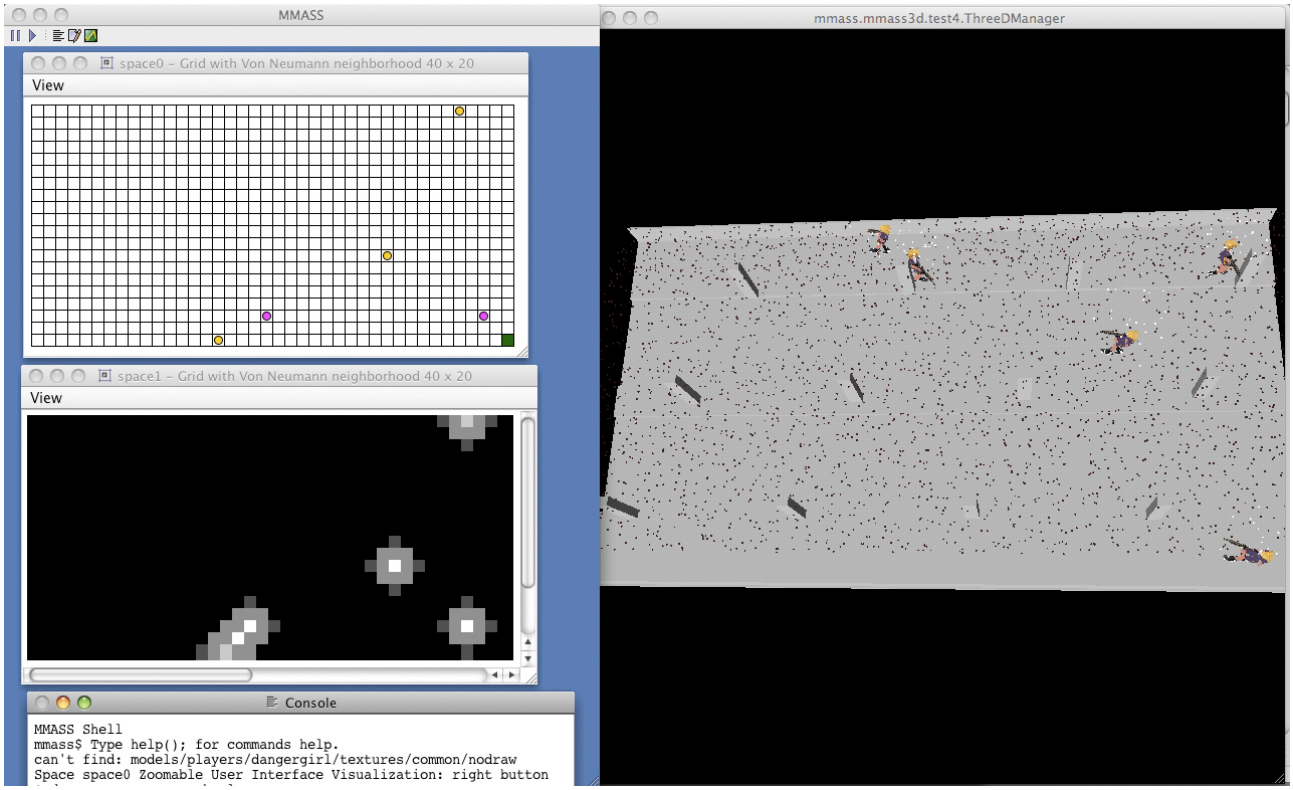


Fig. 8. Screenshot of the simulation environment: on the left, the top panel shows the position of pedestrians in the environment, while the bottom one shows the intensity of cells. The right panel shows a 3D visualization of the environment, including columns, lights and pedestrians.

to the presence of pedestrians. We are currently improving the prototype, on one hand, to provide a better support for the Indianapolis project and, on the other, to realize a more general framework for supporting designers of dynamic self-organizing environments.

Acknowledgements

The work presented in this paper has been partially funded by the University of Milano-Bicocca within the project "Fondo d'Ateneo per la Ricerca - anno 2007".

References

[1] *Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007, Boston, MA, USA, July 9-11, 2007*. IEEE Computer Society, 2007.

[2] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. K. Jr., R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, 2000.

[3] S. Bandini, G. Erbacci, and G. Mauri. Implementing Cellular Automata Based Models on Parallel Architectures: The CAPP Project. In V. E. Malyshkin, editor, *PaCT*, volume 1662 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 1999.

[4] S. Bandini, S. Manzoni, and C. Simone. Dealing with space in multi-agent systems: a model for situated MAS. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1183–1190. ACM Press, 2002.

[5] S. Bandini, S. Manzoni, and G. Vizzari. Towards a Platform for Multilayered Multi Agent Situated System Based Simulations: Focusing on Field Diffusion. *Applied Artificial Intelligence*, 20(4–5):327–351, 2006.

[6] S. Bandini and G. Mauri. Multilayered Cellular Automata. *Theor. Comput. Sci.*, 217(1):99–113, 1999.

[7] W. Butera. Text display and graphics control on a paintable computer. In *SASO* [1], pages 45–54.

[8] A. E. S. Filho, E. C. Lupu, N. Dulay, S. L. Keoh, K. P. Twidle, M. Sloman, S. Heeps, S. Strowes, and J. Sventek. Towards supporting interactions between self-managed cells. In *SASO* [1], pages 224–236.

[9] J. Frazer. Computing without computers. *Architectural Design*, 75(2):34–43, 2005.

[10] E. Goles and S. Martinez. *Neural and Automata Networks: Dynamical Behavior and Applications*. Kluwer Academic Publishers, 1990. ISBN 0-792-30632-5.

[11] H. Gutowitz. *Cellular Automata: Theory and Experiment*. MIT Press/Bradford Books, Cambridge Mass., 1991. ISBN 0-262-57086-6.

[12] N. R. Jennings. On agent-based software engineering. *Artif. Intell.*, 117(2):277–296, 2000.

[13] F. Klügl. A Validation Methodology for Agent-Based Simulations. In R. Menezes and M. Viroli, editors, *Symposium on Applied Computing*, pages 39–43. ACM Press, 2008.

[14] N. Shadbolt. Ambient Intelligence. *IEEE Intelligent Systems*, 18(4):2–3, 2003.

[15] J. von Neumann. *Theory of Self-Reproducting Automata*. University of Illinois Press, 1966.

[16] F. Zambonelli, M. Mamei, and A. Roli. What can cellular automata tell us about the behavior of large multi-agent systems? In A. F. Garcia, C. J. P. de Lucena, F. Zambonelli, A. Omicini, and J. Castro, editors, *SELMAS*, volume 2603 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 2002.

Applying Tropos to Socio-Technical System Design and Runtime Configuration

Fabiano Dalpiaz*, Raian Ali*, Yudistira Asnar*, Volha Bryl* and Paolo Giorgini*
*Dipartimento di Ingegneria e Scienza dell'Informazione
Università degli Studi di Trento
Email: {dalpiaz,ali,yudis.asnar,bryl,pgiorgio}@disi.unitn.it

***Abstract*—Recent trends in Software Engineering have introduced the importance of reconsidering the traditional idea of software design as a socio-tecnical problem, where human agents are integral part of the system along with hardware and software components. Design and runtime support for Socio-Technical Systems (STSs) requires appropriate modeling techniques and non-traditional infrastructures. Agent-oriented software methodologies are natural solutions to the development of STSs, both humans and technical components are conceptualized and analyzed as part of the same system. In this paper, we illustrate a number of Tropos features that we believe fundamental to support the development and runtime reconfiguration of STSs. Particularly, we focus on two critical design issues: risk analysis and location variability. We show how they are integrated and used into a planning-based approach to support the designer in evaluating and choosing the best design alternative. Finally, we present a generic framework to develop self-reconfigurable STSs.**

I. Introduction

Socio-technical systems, introduced by Emery and Trist [1], [2], identify a particular class of systems characterized by an interplay between their social and technical components; in other words, a socio-technical system is composed not only of hardware and software, but also of human agents. STSs present specific properties, among which [3]:

- emergent properties arising from the system as a whole, rather than from the individual components;
- non-determinism, since humans do not always react in the same way;
- dynamic organizational objectives, because objectives can have different (subjective) interpretations and may vary over time.

A particularly relevant and promising application area for socio-technical systems is Ambient Intelligence (AmI). Figure 1 presents an AmI scenario concerning *crisis management*, which we will use further in the paper as a motivating case study. A camera detects a possible fire in a building and, in order to avoid false alerts, it asks another camera for confirmation (1). A sound alert cannot be activated, because the (in-place) alarm ring is out of order (2). The system should therefore *self-reconfigure*: the alternative is to call the firemen (3), check the current traffic status to support the rescue teams (4), and alert the fire warden (6). The socio-technical nature of this scenario becomes clear during this reconfiguration step: calling firemen and alerting the fire warden involves human activities, whose outcome is unpredictable, whereas checking the traffic status involves humans only in a further step, when traffic should be re-routed (5). Even in such a simplified scenario, the interplay between humans and technical subsystems is manifest, and shows the complex nature of designing and supporting STS at runtime.

Fig. 1. An Ambient Intelligence crisis management scenario for Socio-Technical Systems.

The use of *agents*, with sociality, autonomy, and proactivity as key characteristics, can be beneficial for socio-technical systems (and, consequently, AmI systems) design and runtime support. Indeed, in an STS the social interaction among the computational units is essential in pursuing the system goals. System components (both technical and social) need autonomy to take decisions locally, to choose when and how they need to achieve their objectives.

Tropos [4] is an agent-oriented software engineering methodology, which bases on the Belief-Desire-Intention (BDI) paradigm [5], [6]. Tropos models the system as a set of interacting agents[1]. Each agent has a set of *goals* to fulfill, and a number of *tasks* that describe how to achieve goals. An agent can provide or require *resources* to execute tasks. Soft-goals represent those goals, such as software qualities, for which fulfillment there is no clear-cut criteria. Goal-to-goal connections can be set through (a) *and-decomposition* to split a goal into a number of concurrent sub-goals; (b) *or-decomposition*

[1]An agent can be a human or an artificial agent.

to represent a number of alternative sub-goals. Goals are connected to tasks through *means-end* decomposition: the task is a means to achieve an end (the goal). *Contribution* relations link goals, tasks, and resources to soft-goals; a contribution from an element x to a soft-goal s represents how well x contributes to the satisfaction of s. Finally, agents depend on each other – via *dependencies* – for goal achievement, task execution, and resource provision.

These aspects of Tropos appear useful for the development of socio-technical systems, where hardware/software agents coexist with human actors. However, the original [4] Tropos modeling framework alone is not sufficient to capture all aspects of STSs, and in this paper, we illustrate a collection of Tropos extensions that can be used to better address both the development and the support of STSs at runtime. The first design-time extension concerns the modeling of location variability; the location where an agent is situated can requires specific strategies to be used at runtime. The second extensions we introduce is a framework to handle uncertainty in the development of STSs: the Goal-Risk (GR) framework is a modeling technique, accompanied by analysis tools, whose objective is the minimization of risk. These extensions to the original Tropos framework can be integrated in a planning-based framework that can support a designer in exploring the design-time space of alternatives. The framework allows the designer to look at all possible designs and on the base of a number of criteria decide on such alternatives. Finally, we introduce two approaches to support self-reconfigurable STSs. Essentially, we propose two approaches to use the Tropos goal diagrams to monitor the execution of a system. The former adopts a centralized reconfiguration engine, while the latter a decentralized reconfiguration enacted by each agent.

The paper is organized as follows: Section II discusses two concerns which are relevant in the design of STSs in an AmI setting: location variability and risk; Section III shows how AI planning techniques can be used to automate design-time analysis; Section IV analyzes autonomic STSs and the property of runtime self-reconfiguration. Finally, Section V presents final remarks and future work.

II. Modeling STS in AmI settings

We present two different techniques to support the design of socio-technical systems. The rationale behind these proposals comes from two important aspects in STSs, and whose importance gets even wider when considering Ambient Intelligence settings; these concerns are location-based variability and risk.

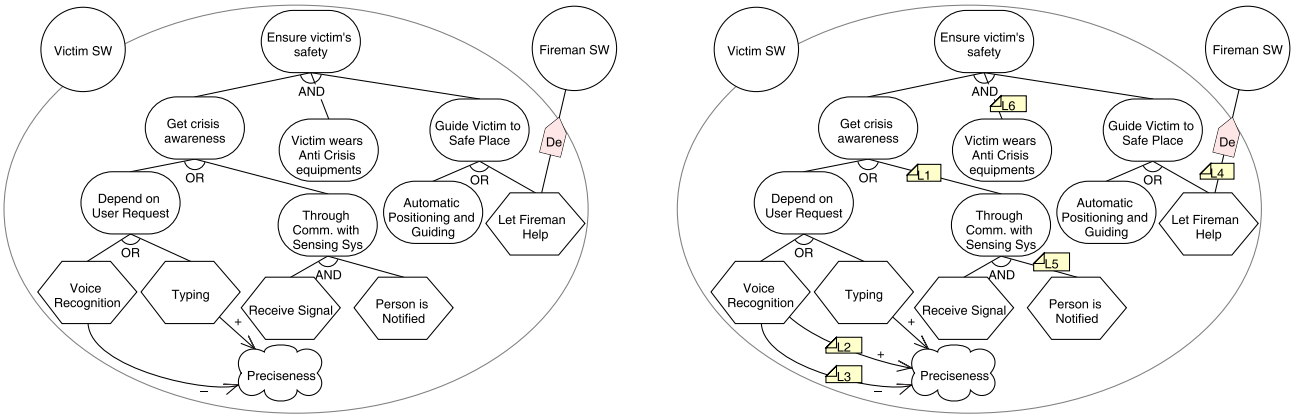
A. Modeling Location-based Variability

The rationale of an agent often contains behavioral variability, where the agent can choose among alternative strategies (*tasks*) to fulfill the same objective (*goal*). A proper variability modeling should include means to represent how the selection between these alternatives is performed, specifying *when* and *where* a certain alternative is applicable. STSs are characterized by a dynamic location, in which both technical and social components vary over time. We claim that location is an important criteria that can constrain and guide the selection of the most suitable alternative.

Figure 2a shows a partial goal model for a software agent working on behalf of a victim of a crisis (e.g., a fire). To ensure victim safety, the software has to be aware of the crisis, which can be done through an explicit request from the victim by a voice or typed command or through the continuous automated analysis of the signal that comes from some sensing system. To ensure safety, the victim might need to wear special dress (e.g. anti-fire coat). Then, the victim has to be guided to a safe place through an automatic tracing and directing, or through the help of a fireman.

The original Tropos goal model supports modeling alternatives for satisfying goals, but lacks tools for specifying the locations where specific alternatives are applicable. (see the model in Figure 2a). In our previous work [7], [8], [9], we extended Tropos goal model to represent the relation between goal satisfaction alternatives and location. Our modeling enables an agent to answer several important questions such as: what are the possible, impossible, or recommended alternatives to satisfy a goal in a specified location. In our approach, (i) a *location property* is a boolean predicate evaluated against the current location; (ii) a *location-based variation point* is an element in the goal model to which a location property (Li on Figure 2b) can be associated. We defined five location-based variation points:

1) Location-based Or-decomposition: Or-decomposition is the basic variability construct; the choice of a specific Or-alternative might be based on a location properties that inhibits, allows, or recommends some alternatives. E.g. having crisis awareness through communicating with a sensing system requires both that a sensing system exists and the user's PDA has the ability to connect to it ($L1$).
2) Location-based contribution to softgoals: the value of the contributions to the softgoals can vary from one location to another. We need to specify the relation between the location and the value of the contribution. E.g. receiving the user request through voice recognition contributes positively to the softgoal *"Preciseness"* when the level of noise is low and the system is trained enough for recognizing that user voice ($L2$), while it contributes negatively in the opposite case ($L3$).
3) Location-based dependency: in some locations, an actor might be unable to satisfy a goal using its own alternatives. In such case, the actor might delegate this goal to another actor that is able to satisfy it. E.g. guiding the person in crisis by a fireman is an alternative that needs a free and skilled fireman that is close to and can reach that person ($L4$).
4) Location-based goal / task activation: an actor, and depending on the location settings, might find necessary or possible triggering (or stopping) the desire of satisfying a goal / the execution of a task. E.g. notifying a person about crisis has to be triggered when the analysis of the signal that comes from the sensing system addresses



(a) Modeling with Tropos

(b) Modeling with location-based Tropos

Fig. 2. Partial goal model for the crisis management scenario.

some potential danger ($L5$).

5) Location-based And-decomposition: a sub-goal might (or might not) be needed in a certain location, that is some sub-goals are not always mandatory to fulfil the top-level goal in And-decomposition. E.g. the need of a person in crisis to wear special equipments depends on the category of the crises, and the skills the person in danger has ($L6$).

The analysis of the location properties will lead to the definition of the location model that can be modeled using class diagram as we did in [8]. In [9], we described a process to derive a location model – describing the location in terms of its entities and the links between them – from the location properties in a goal model. The location model can be instantiated to represent a certain location and enable automated reasoning. The evaluation of the location properties will enable an agent to derive the possible alternatives for satisfying its goals. The proposed extension of Tropos goals modeling constructs are colored in gray in the metamodel of Figure 3.

By formalizing the goal model, the location model, and the location preperties, it becomes possible performing several kinds of analysis. We outline now three types of automated analysis:

1) Location-based goal satisfiability (LGS): it verifies whether a goal is achievable by choosing a certain alternative in a specific location.
2) Location properties satisfiability (LPS): this analysis checks if the current location structure is compliant with a set of goals. This techniques can be used to identify what is missing in a particular location where some top-level goals have been identified as unsatisfiable by LGS.
3) Preferences analysis (PA): this type of analysis requires the specification of preferences over alternatives. Preferences can be modeled using soft-goals as in [10]. We need this analysis in two cases:
   a) when there are several alternatives to satisfy a goal: the selection will be based on the contributions to preferred soft-goals.
   b) when there is no applicable alternative: in this case, LPS might provide several proposals about the needed location modifications.

   The adopted modifications are those leading to better satisfaction of the preferences expressed over soft-goals.

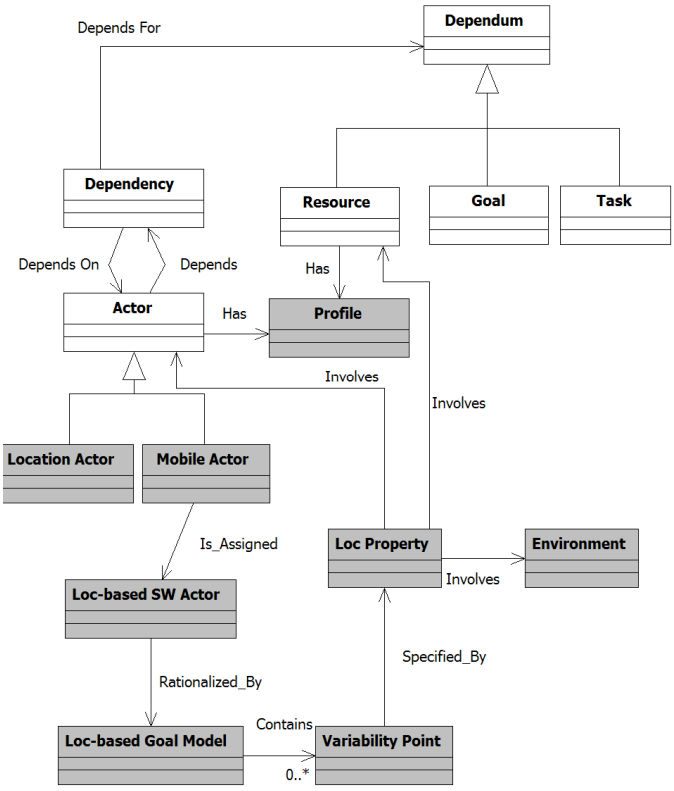


Fig. 3. Metamodel showing the extension of Tropos with location.

B. Modeling Uncertainty through Risk Analysis

STSs are exposed to a wide range of uncertainty during their development, runtime, and maintenance. Some uncertainties can result into system failures, and can even put human lives

in danger. There is no such systems that are free of failure: if something *can go wrong* then it *will go wrong* [11]. Therefore, designers should consider uncertainties that could lead to failures that harm the system and treat them.

In [12], we proposed the Goal-Risk (GR) modeling framework, that extends Tropos by providing modeling constructs to represent uncertain events that may affect the organization negatively (called risks) or positively (called opportunities). Indeed, it is hardly possible to nullify the risks that threaten a system since the system can fail as well in case of normal operations (e.g., operator errors, bad maintenance) or malicious intentions (e.g., attacks, frauds).

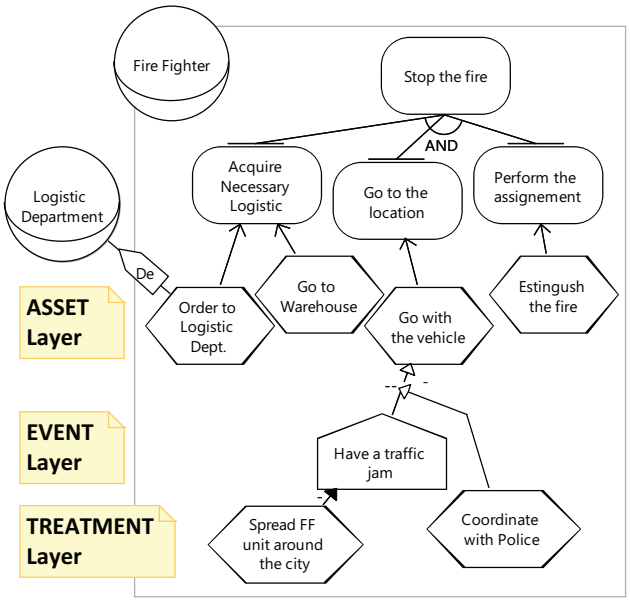


Fig. 4. The Goal-Risk Framework

Conceptually, a Goal-Risk (GR) model (see Figure 4) is composed by three-layers: *asset* to capture the goals of the stakeholders (e.g., firefighter intends to stop a fire), and tasks and resources required to achieve the goals (e.g., "Go to warehouse to obtain the logistic"), *event* (pentagons) to model uncertainty events (e.g., risks, opportunities) that affect the asset layer (e.g., having a traffic jam), and *treatment*, depicted as tasks, to capture additional measures that are required to treat the risks (e.g., spread firefighter units around the city). This framework is equipped with two basic reasoning mechanisms to help designers in making decisions. First, *forward reasoning* aims at calculating the risk level of an organization for a given setting (e.g., value of goals, adopted treatments) and inputs (e.g., likelihood-severity of event). Second, *backward reasoning* aims at eliciting the possible solutions (e.g., strategy to achieve the goals and necessary treatments to mitigate the risks) for a given set of constraints (e.g., tolerable risk level).

A socio-technical system is composed by several agents, each having its own goals, tasks, resources, and, moreover, each exposed to different risks. An agent often cannot fulfill all its goals, and needs to depend on others to satisfy some subgoals, execute some tasks, or provide resources. Thus, a system can be viewed as a network of agent dependencies. In secure and dependable systems, this phenomena emerges as one of the critical points because a vulnerable agent can put the entire system at risk.

In [13], we extend the GR framework to the case of a multi-agents setting and illustrate how risks are propagated from an agent across the organization. In certain cases, agents should depend on other agents that they do not trust due to some reasons (e.g., there are not other choices, the regulation orders to do it). In such a setting, the agents often perceives a higher level of risk as if they depend on the ones they trust. Essentially, we can infer how much risks that an agent perceives is based on its trust relations [14] and evaluate whether adopted treatments are perceived to be effective by agents in mitigating the risks.

Finally, considering risks is critical to ensure the socio-technical systems being dependable and operating securely. Risk analysis is a continuous process, and therefore risks must be monitored during runtime and be reviewed regularly as long as the system is still in use.

III. Automating the Design: a Planning-Based Approach

The planning-based extension of Tropos [15] has been proposed to support a designer in exploring the space of alternative designs of a socio-technical system. Indeed, the fulfillment of each of the system goals is related to a number of choices of how the goal is decomposed and which are the actors the goal (or its subgoals) are delegated to. The idea behind planning-based framework is that the task of constructing a requirements model for a socio-technical system, i.e. a network of delegations among actors for goals, can be framed as a *planning problem* where selecting a suitable social structure corresponds to selecting a plan that satisfies the stakeholders goals.

This work adopts AI (Artificial Intelligence) planning [16] techniques to the domain of requirements engineering. AI planning is about automatically determining a course of actions (i.e., a plan) needed to achieve a certain goal where an action is a transition rule from one state of the world to another. To define a planning problem, one should specify (i) the initial state of the world, (ii) the desired state of the world, and (iii) the actions. In our planning-based framework, *goal decomposition*, *delegation* and *fulfillment* are seen as actions that the designer ascribes to the actors of the system-to-be and of its organizational environment. We use PDDL (Planning Domain Definition Language) 2.2 [17] to formally specify the initial organizational setting and actions of the domain. An off-the-shelf planning tool, LPG-td [18], is adopted for the implementation of the planning domain.

In [15] we have presented the basic set of first-order predicates used to formalize the organizational setting in terms

of actors and goals, their properties (e.g. actor capabilities), and social dependencies among actors.

The flexibility of the PDDL specification language makes it possible to accommodate various criteria into the planning problem definition. In the following, we list the extensions of our framework related to three different aspect of socio-technical system development and deployment.

An application of our planning-based framework to the domain of **secure system design** presented in [19] supports trust and permission concepts of Secure Tropos [20]. The planning domain is defined so that it guarantees that the resulting socio-technical model satisfied the trust and permission related constraints imposed on it (e.g., no goal is delegated along an untrusted link). In the crisis management scenario, presented in Section I, the examples of such security constraints can be related to the permission to activate the alarm, which only a limited set of actors possess, or to trust relations between the firemen and the fire warden of the building.

Yet another extension of our framework [21] uses risk-based evaluation metrics for selecting a suitable design alternative, and aims at **agent-based safety critical** applications. In this work, the risk-based criteria (e.g. related to the criticality of a goal satisfaction or minimum acceptable level of trust between agents) and the respective framework discussed in Section II-B of the present paper, are incorporated into the planning-based procedure which supports a socio-system design (as well as a system redesign at runtime). This work aims at proposing a design that maintain the risk level within the acceptable limits. In the crisis management scenario, the examples of risk constraints are the ones related to the way to alert the workers about the danger (the most reliable one should be chosen among the available alternatives), or to the level of trust between the firemen and the fire warden of the building and, accordingly, the goals that can be delegated between these actors.

Location can be used as a metrics for evaluating alternatives, as well. As we showed in Section II-A, location properties associated to variation points can be used to (a) limit the range of alternatives an agent can choose among; (b) express location-dependent contribution to soft-goals. In such a way, the planning-based approach we suggest here can be customized to discard unavailable options and to exploit soft-goal satisfaction for ranking available alternatives.

IV. Autonomic Socio-Technical Systems: Runtime Self-Reconfiguration

The previous sections focused on various facets of the design of STSs, proposing both modeling languages and analysis/reasoning techniques. The design of an STS is a fundamental activity, which helps preventing the development of a system that violates its requirements (both functional and non-functional) at runtime.

Nevertheless, design-time support is not sufficient to provide a comprehensive support for socio-technical systems. Runtime violation of requirements [22] is recognized as an open problem and has been explored since several years. Feather et al. [23] propose an approach to reconcile requirements with runtime behavior, where both the design- and run-time phases are covered: (a) anticipate as many violations as possible at specification time, and (b) detect and resolve the remaining violations at runtime. Though Feather's work is not targeted for socio-technical systems, it points out problems and proposes solutions which apply also in the context of STSs.

There is hence a clear need for a runtime framework which complements the design-time techniques we have presented. An STS exhibits particular properties that set specific requirements for the execution infrastructure:

- humans play an active role and should interact with the technical sub-systems at runtime;
- the location (both the physical and the social aspects of location) is in continuous evolution;
- the system should self-reconfigure adapting to the changing environment where it operates;
- failures should be compensated and an alternative plan should replace the failed plan.

Multi-agent system infrastructures represent a good candidate to support socio-technical systems at runtime. In particular, those based on the BDI paradigm are particularly suitable, since Tropos is founded on BDI. However, STSs exhibit some features which are not considered in the classical BDI paradigm, such as the interplay between software and human agents. Therefore, existing infrastructures need customization to result an effective solution for STSs.

The approach we have taken in our research is to link Tropos to BDI-based software architectures. This solution enables us to combine different state-of-the-art techniques extending the capabilities of BDI. Agents represent the core concept at runtime; each agent has a goal-based specification, executes plans to achieve its active goals, and depends on other agent for plan execution, goal achievement, and resource provision. Two different but complementary approaches can drive the self-reconfiguration process, with distinct properties and application scenarios:

- centralized self-configuration: some types of STS, such as a scientific institution, can work properly only if a centralized knowledge of the various agents is available, and self-reconfiguration is therefore controlled centrally. We explored this first type of self-reconfiguration in [24].
- decentralized self-configuration: this approach presents self-configuration from the local perspective of an individual agent. Each agent commits to achieve its own goals at best, without having a complete knowledge of the STS. This solution cannot achieve the same level of optimality a centralized approach guarantees, but is the only available solution whenever the internals of the agents cannot be disclosed to a centralized supervisor. An example of this situation is two software systems that interact but belong to different companies: the interfaces are available, but the companies will not disclose the internal reasoning. We proposed an initial approach supporting this vision in [25].

In [24] we have presented an approach to dynamic reconfiguration of a socio-technical system structure in response to internal or external changes. The paper suggests a **centralized reconfiguration mechanism**, which aims at making a socio-technical system self-configuring, and proposes a multi-agent architecture for its implementation.

The proposed reconfiguration mechanism

- collects and manages the information about the system;
- evaluates both the system state (e.g. the overall workload), and the local utilities of each agent to decide whether the system needs to be redesigned in response to external or internal changes;
- and, if the above evaluation shows that the reconfiguration is needed, replans the system structure in order to optimize it with respect to the evaluation criteria of interest.

The notification about the change is obtained either from the inside of the system or from the environment. Each system agent is obliged to communicate to some central point if it committed to, or achieved a goal. Four types of triggering events are supported, namely, the situations when a new agent enters the system, or the existing one leaves, when a new system goal is introduced, or one of the old ones is satisfied. However, due to the flexibility of the PDDL representation, it is possible to extend the formalization to support the changes in the agents' capabilities and commitments, failures when achieving goals, etc.

This framework can be applied to the organization of firemen rescue teams, where different alternatives are available (truck type, fire fighting approach, firemen equipment) and several agents are involved. In this scenario, finding the optimal solution is fundamental, and the centralized planning-base approach is the best choice.

Talos [25] is an architectural approach to self-reconfiguration based on a **decentralized reconfiguration mechanism**, where self-reconfiguration is seen from the perspective of each agent/component. Three different subsystems are the core of the self-reconfiguration process each agent performs:

- *Monitor*: the agent should continuously monitor both its internal state and the location where it is running (similarly to what happens for the centralized approach). The internal state is evaluated verifying the status of the agent's goals, detecting new goals, failures, and fulfillments. A mailbox is exploited to figure out the incoming requests from other agents, who want to interact to achieve their own goals. The external context is monitored receiving events from the set of artifacts which can be seen or are used by the agent.
- *Diagnose*: monitored events are linked to the goal model by traceability links, triggering new top-level goals and notifying failures or achievements. Diagnosis provides different levels of detail depending on the chosen goal monitoring granularity; in Talos we exploit a variant of Wang's goal monitoring switches [26]: the closer the monitoring switches are to the plans, the more detailed diagnosis we can obtain. A particular kind of diagnosis is related to the enactment of dependencies, where other agents provide information concerning dependency requests (e.g., refuse, contract, accept).
- *Compensate*: after detecting and diagnosing a failure, the following step consists of taking a countermeasure to this failure. We propose the execution of two sub-tasks to properly carry out this activity:
  - A compensation plan should be executed to "undo" the effects of the failed plan. An important information from diagnosis is to understand which action of the plan failed, or if the plan failed to achieve the goal though it terminated correctly.
  - A self-reconfiguration process is enacted to choose another strategy to achieve the goal of which a failure event was generated. A variant of goal analysis is used to perform this step.

The decentralized solution is suitable in the fire fighting scenario, as well. In the scene described in Figure 1 the out-of-order bell inhibits the best overall alerting strategy (playing a sound alarm), and this failure requires a prompt local reconfiguration (e.g., alerting the fire warden). Involving a central control unit would produce delays and could result in a failure, especially if the fire damaged the physical network enabling the communication with the central unit.

V. Conclusion and future work

In this paper, we have presented and analyzed a number of extensions to the Tropos methodology to support the development and the runtime operation of a complex class of modern systems, namely socio-technical systems. The design and runtime emergent properties of these systems present a lot of challenges for developers and there is a clear need for innovative engineering tools and techniques.

We have presented two design-time modeling and reasoning techniques, focused on location properties of an STS and risk analysis, respectively. Also, the problem of runtime reconfiguration of a socio-technical structure was addresses in the paper with two approaches, centralized and decentralized, suitable each for different application areas.

As future work, we believe it will be important to further elaborate and better integrate the techniques presented in this paper. Particularly, we would like to work on the implementation of an integrated CASE tool for the development of STSs.

VI. Acknowledgements

This work has been partially funded by EU Commission, through the SENSORIA, SERENITY, and MASTER projects, and by the PRIN program of MIUR under the MEnSA project.

References

[1] F. Emery, "Characteristics of socio-technical systems," *London: Tavistock*, 1959.

[2] F. Emery and E. Trist, "Socio-technical systems," *Management Science, Models and Techniques*, vol. 2, pp. 83–97, 1960.

[3] I. Sommerville, *Software Engineering*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2006, ch. Socio-Technical Systems.

[4] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[5] A. Rao and M. Georgeff, "An abstract architecture for rational agents," *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pp. 439–449, 1992.

[6] ——, "Bdi agents: From theory to practice," *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pp. 312–319, 1995.

[7] R. Ali, F. Dalpiaz, and P. Giorgini, "Location-based variability for mobile information systems," *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08)*, 2008.

[8] ——, "Modeling and analyzing variability for mobile information systems," *Proceedings of the 4th Ubiquitous Web Systems and Intelligence Workshop (UWSI 2008)*, 2008.

[9] ——, "Location-based software modeling and analysis: Tropos-based approach," *Proceedings of the 27th International Conference on Conceptual Modeling (ER 2008)*, 2008.

[10] S. Liaskos, S. McIlraith, and J. Mylopoulos, "Representing and reasoning with preference requirements using goals," Tech. rep. CSRG-542, Computer Science Department, University of Toronto, Tech. Rep., 2006.

[11] B. Schneier, *Beyond Fear: Thinking Sensibly about Security in an Uncertain World*. Springer, 2003.

[12] Y. Asnar and P. Giorgini, "Modelling Risk and Identifying Countermeasures in Organizations," in *Proc. of CRITIS'06*, ser. Lecture Notes in Computer Science, vol. 4347. Springer, 2006, pp. 55–66.

[13] Y. Asnar, R. Moretti, M. Sebastianis, and N. Zannone, "Risk as Dependability Metrics for the Evaluation of Business Solutions: A Model-driven Approach," in *Proc. of ARES'08*. IEEE Press, 2008.

[14] Y. Asnar, P. Giorgini, F. Massacci, and N. Zannone, "From Trust to Dependability through Risk Analysis," in *Proc. of ARES'07*. IEEE Press, 2007.

[15] V. Bryl, P. Giorgini, and J. Mylopoulos, "Designing Cooperative IS: Exploring and Evaluating Alternatives," in *CoopIS'06*, 2006, pp. 533–550.

[16] D. S. Weld, "Recent Advances in AI Planning," *AI Magazine*, vol. 20, no. 2, pp. 93–123, 1999.

[17] S. Edelkamp and J. Hoffmann, "PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition," University of Freiburg, Tech. Rep. 195, 2004.

[18] LPG Homepage, "LPG-td Planner," http://zeus.ing.unibs.it/lpg/.

[19] V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone, "Designing Security Requirements Models Through Planning," in *CAiSE'06*, 2006, pp. 33–47.

[20] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling Security Requirements Through Ownership, Permission and Delegation," in *Proc. of RE'05*. IEEE Press, 2005, pp. 167–176.

[21] Y. Asnar, V. Bryl, and P. Giorgini, "Using risk analysis to evaluate design alternatives." in *AOSE*, ser. Lecture Notes in Computer Science, L. Padgham and F. Zambonelli, Eds., vol. 4405. Springer, 2006, pp. 140–155.

[22] A. van Lamsweerde, "Divergent views in goal-driven requirements engineering," *Foundations of Software Engineering*, pp. 252–256, 1996.

[23] M. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard, "Reconciling system requirements and runtime behavior," *Proceedings of the 9th International Workshop on Software Specification and Design*, pp. 50–59, 1998.

[24] V. Bryl and P. Giorgini, "Self-Configuring Socio-Technical Systems: Redesign at Runtime," in *SOAS'06*, 2006.

[25] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "Talos: an architecture for self-reconfiguration," DISI-08-026, Tech. Rep., 2008.

[26] Y. Wang, S. McIlraith, Y. Yu, and J. Mylopoulos, "An automated approach to monitoring and diagnosing requirements," *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering*, pp. 293–302, 2007.

Advancing Object-Oriented Standards Toward Agent-Oriented Methodologies: SPEM 2.0 on SODA

Ambra Molesini*, Elena Nardini†, Enrico Denti* and Andrea Omicini†
*
Alma Mater Studiorum – Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
Email: {ambra.molesini, enrico.denti}@unibo.it
†
Alma Mater Studiorum – Università di Bologna a Cesena
Via Venezia 52, 47023 Cesena, Italy
Email: {elena.nardini, andrea.omicini}@unibo.it

***Abstract*—Building ad-hoc design processes and methodologies has become a key challenge in Software Engineering, and several efforts are being made for developing appropriate *meta-models* both for methodologies and development processes. The Software Process Engineering Meta-model (SPEM) – an OMG object-oriented standard – is a natural candidate for representing, comparing and reusing design processes in a uniform way.**

In this paper we apply SPEM 2.0 to Agent-Oriented Software Engineering methodologies, so as to assess its strengths and limitations. To this end, we take the SODA methodology as a significant case study, and compare the meta-model of its process obtained from SPEM 2.0 with the former meta-model obtained from SPEM 1.0.

I. Introduction

In the Software Engineering (SE) research field, several efforts are underway for developing appropriate *meta-models* for SE methodologies and processes. According to Cernuzzi et al. [1], a *Development Process* is an ordered set of steps that involve all the activities, constraints and resources required to produce a specific output which satisfies a set of input requirements. Typically, a process is composed of different stages/phases in relation with each other: each stage/phase identifies a portion of the work to be done, the resources to be exploited and the constraints to be obeyed for that purpose.

The relation between methodologies and processes is well studied in the literature: as pointed out in [1], methodologies focus more explicitly on *how* an activity or task should be performed in specific stages of the process, while processes may also cover more general management aspects about *who*, *when*, *how much*, etc.

Software development processes and methodologies have always been described in suitable terms for developers [2]: in fact, they talk about what tasks and techniques should be used, what sort of lifecycle is appropriate, and how these process elements should be organised in time and assigned to people. These aspects are often described in a manual or book that the project manager and his/her team of developers closely follow [2]. However, such manuals are not suitable for the automatic tools that typically support the designer's work, such as CASE tools that need specific rules for supporting methodologies and processes—rules stating, for instance, that it is a nonsense to put in a sequence two activities, three techniques and four roles: these rules are usually captured by a *meta-model*.

Although it is possible to describe a methodology / process without an explicit meta-model, formalising their underpinning ideas is valuable for checking consistency, or when planning extensions or modifications: there, meta-models can be exploited to check both the software development process and the completeness and expressiveness of methodologies. More generally, the relevance of meta-model becomes clear when studying the completeness and the expressiveness of a methodology / process, and when comparing or integrating different methodologies / processes together. For these reasons, research efforts are being made to define unified meta-models, aimed at representing the existing methodologies and processes in a uniform way, so as to promote their mutual comparison, their composition and reuse—this area is sometimes referred to as *Method Engineering* [3], [4].

SPEM (Software Process Engineering Meta-model, [5]) is one of the key references for this purpose: as it could be expected, SPEM is conceived for an object-oriented context, since most current methodologies adopt this paradigm as their reference.

SPEM seems a natural candidate for representing the meta-models of Software Engineering methodologies, both because it is an OMG standard, and because it is based on formal descriptions that can lead to consistent, comparable models: so, an interesting challenge is to test its applicability to other, non object-oriented Software Engineering domains. Despite its origin in the object-oriented context, SPEM can be applied to the agent-oriented process quite naturally, since the process of software development is mostly independent of the computational paradigm adopted, and has essentially the same phases in any methodology. However, AOSE methodologies introduce a richer set of abstractions and mechanisms, which

naturally lead to a more articulated definition of the software development process.

In a previous work [6] we explored the applicability of SPEM to the Agent-Oriented Software Engineering (AOSE) domain, whose abstractions and mechanisms are particularly suited to the design and development of complex software systems. There, we highlighted several limitations (briefly recalled in Section IV), exploiting the SODA methodology as a significant case study for stressing SPEM 1.0's strengths and weaknesses because of its focus on modelling the social issues and the application environment, and its mechanisms for capturing the layered structure of complex systems. Other AOSE methodologies modelled by SPEM [7] apparently do not suffer from such limitations (mainly because they do not include some mechanisms, like the SODA layering which is discussed below), so it seems quite difficult to determine general metrics and criteria for assessing the SPEM meta-modelling power.

So, in this paper we explore SPEM 2.0 by modelling the SODA methodology process and comparing the results with the previous ones—in particular, aiming to discover whether and how the previous limitations have been addressed: in a sense, to check whether the extension of the SPEM object-oriented standard has gone farther in addressing the many issues of agent-oriented methods and techniques.

Accordingly, the paper is structured as follows. Section II briefly presents SPEM 2.0 and some considerations about the adoption of SPEM in the AOSE field (Subsection II-A), while Section III presents the corresponding SODA process. Then, Section IV compares the meta-modelling power of SPEM 1.0 and SPEM 2.0, by taking the SODA process as its running example. Conclusions are reported in Section V.

II. SPEM

SPEM is an OMG standard meta-model for formally defining software and systems development processes [5]. The goal of SPEM 2.0 is not only to support the representation of one specific development process ore the maintenance of several unrelated processes, but to provide process engineers with mechanisms to consistently and effectively manage whole families of related processes promoting process reusability [5]. To this end, its meta-model introduces a clear separation between reusable methods content and its application in processes: the first item provides step-by-step explanations of how the development goals are achieved, independently of the placement of these steps within a development lifecycle; then, processes take these methods content elements and relate them into partially-ordered sequences that are customized to specific types of projects.

More in detail, SPEM 2.0 is structured into seven packages: *Core*, *Process Structure*, *Process Behaviour*, *Manage Content*, *Method Content*, *Process With Method*, and *Method Plugin*.

The *Core* package defines the base classes and abstractions for all other meta-model packages, while *Process Structure* provides the base for creating flexible process models — in particular, defining a process model as a breakdown or decomposition of nested *Activities*, with the related *Roles* and input / output *Work Products*. In addition, this package enables process reuse by providing mechanisms such as dynamic binding of *process patterns* (or *capability patterns*), which are reusable best practices for quickly creating new development processes.

The *Process Behavior* package supports the extension of the static breakdown structure of a process by externally-defined behaviour models. *Manage Content*, then, introduces the concepts to document and manage development processes through natural language description—indeed, practice of processes techniques and methods often cannot be formalised, but can only be expressed in natural language. In its turn, *Method Content* makes it possible to build a reusable development knowledge base which is independent of any specific development process: in particular, this package comprises textual step-by-step explanations, describing how specific fine-grain development goals are achieved by which roles, with which resources and results, independently of the placement of these steps within a specific development lifecycle. *Process With Method* provides what is needed to integrate processes with instances of Method Content concepts. Finally, the *Method Plugin* package introduces concepts for 'designing' and managing maintainable, large scale, reusable, and configurable libraries or repositories of method content and processes. In the next Subsection we outline some of the main issues in the use of SPEM in the AOSE context.

A. SPEM&AOSE

As introduced above, AOSE methodologies introduce a richer set of abstractions and mechanisms then OO systems, so the software development process is more articulated; in turn, the wide range of peculiarities of each methodology makes it difficult to define some general metrics and criteria for SPEM testing and evaluation.

Yet, some points can be put in evidence. First, each process and subprocess resulting from methodology representation should be reasonably clear and easy to understand, since failing to do so would make the SPEM representation itself little useful. SPEM's separation between Method Contents and Processes is a natural candidate to support this aspect, although the limited set of symbols offered by SPEM might lead to difficulties in representing elements or state changes. Second, since most methodologies exploit some iterative / incremental processes, the SPEM representation should be able to support such aspect. Third, since methodologies for complex systems typically include conceptual mechanisms for complexity management (such as some form of in/out zooming, the ability to view the system by levels at different abstraction levels, etc), some support should be provided by SPEM in order to capture such aspects in a satisfactory way.

In the following section we will try to exploit SODA as a testbed for evaluating SPEM 2.0's expressiveness with respect to such issues.

III. The SODA Process

SODA (Societies in Open and Distributed Agent spaces) [8], [9] is an agent-oriented methodology for the analysis and design of agent-based systems, which adopts the Agents & Artifacts meta-model (A&A) [10], and introduces a *layering principle* as an effective tool for scaling with the system complexity, applied throughout the analysis and design process.

SODA abstractions are logically divided into three categories: *i)* the abstractions for modelling/designing the system's active part (task, role, agent, etc.); *ii)* those for the reactive part (function, resource, artifact, etc.); and *iii)* those for interaction and organisational rules (relation, dependency, interaction, rule, etc.). In its turn, the SODA *process* is organised in two phases (Figure 1), each structured in two sub-phases: the *Analysis phase*, which includes the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps. Each sub-phase models (designs) the system exploiting a subset of the SODA abstractions: in particular, each subset always includes at least one abstraction for each of the above categories—that is, at least one abstraction for the system's active part, one for the reactive part, and another for interaction and organisational rules.

In order to represent the whole SODA process in a simple yet effective way, we exploited SPEM's separation between Method Contents and Processes (Section II): first, we modelled each sub-phase as a separate and independent Method Content, then we defined a specific process for each sub-phase – see Figures 3, 4, 5 and 6 for details – and re-used these processes to create the whole SODA process presented in Figure 1. In this way the whole process is reasonably easy to understand, since each sub-phase in the Activity Diagram is depicted as a simple activity, hiding the internal complexity of that process portion.

In addition, since the SODA process (Figure 1) is iterative and incremental, each step can be repeated several times, by suitably exploiting the layering principle: so, for instance, if, during the Analysis step, the System Analyst – one of the roles involved in the SODA process – recognises some omissions or lacks in the requirements' definition, he/she can restart the Requirements Analysis step adding a new layer in the system or selecting a specific layer and then refining it. Analogous considerations could be made for both the Architectural Design step – where the Analysis step can be restarted from the layering – and the Detailed Design step—which leads to restart the Architectural Design step.

The layering in Figure 1 is represented as a simply Activity of the process: actually, it is a *capability pattern* (Section II), i.e., a reusable portion of the process, as shown in Figure 2 where the layering process is detailed. In particular, the layering presents two different functionalities: *(i)* the selection of a specific layer for refining / completing the abstractions models in the methodology process, and *(ii)* the creation of a new layer in the system by in-zooming (i.e., increasing the system detail) or out-zooming (i.e., increasing the system

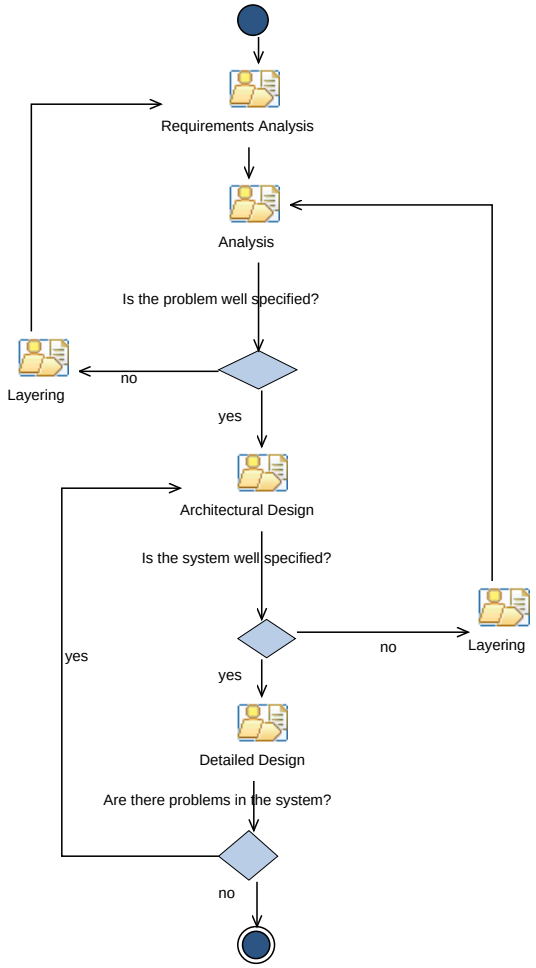


Fig. 1. Activity Diagrams of the whole SODA process.

abstraction) activities. In latter case, the layering process terminates with the projection activity needed to project the abstractions from one layer to another "as they are", so as to maintain the consistency in each layer.

The layering pattern is also used within sub-phases—except in the Detailed Design, where the layering principle is, by definition, not applicable. For instance, Figures 3, 4 and 5 report the sub-process of the Requirements Analysis, of Analysis and of Architectural Design steps, respectively: the layering activity is applied multiple times, both as a refinement or layer selecting technique in the single models (activities) – e.g., task layering, role layering, resource layering, space layering interaction layering, etc… – and as a way for re-starting the stage if some problems arise in the models or just for triggerring a new iteration of the stage. In the following, each sub-phase is presented in short.

a) Requirements Analysis.: Several abstract entities and models are introduced for this purpose. Each model is represented in Figure 3 as an activity, related to the corresponding Task in the Requirements Analysis Method Content. The latter specifies the steps to be completed to achieve the task, as well as the Workproducts to be produced—i.e., the SODA tables

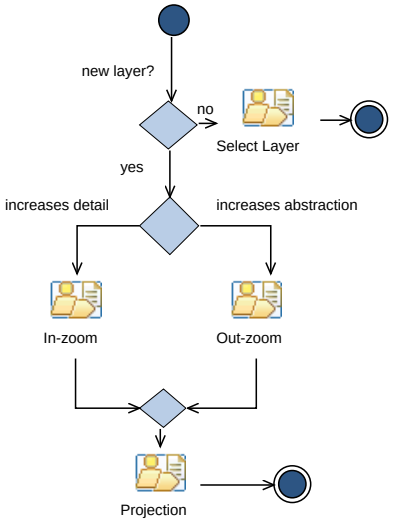


Fig. 2. Activity Diagram of the Layering Pattern.

describing the abstract entities of the Requirements Analysis. During the *Requirements modelling* activity (Figure 3), *requirement* and *actor* are used for modelling the customers' requirements and the requirement sources, respectively, while the *external-environment* notion is used as a container of the *legacy-systems* that represent the legacy resources of the environment in the *Environment modelling* activity. The relationships between requirements and legacy systems are modelled in the *Relation modelling* activity in terms of a suitable *relation*.

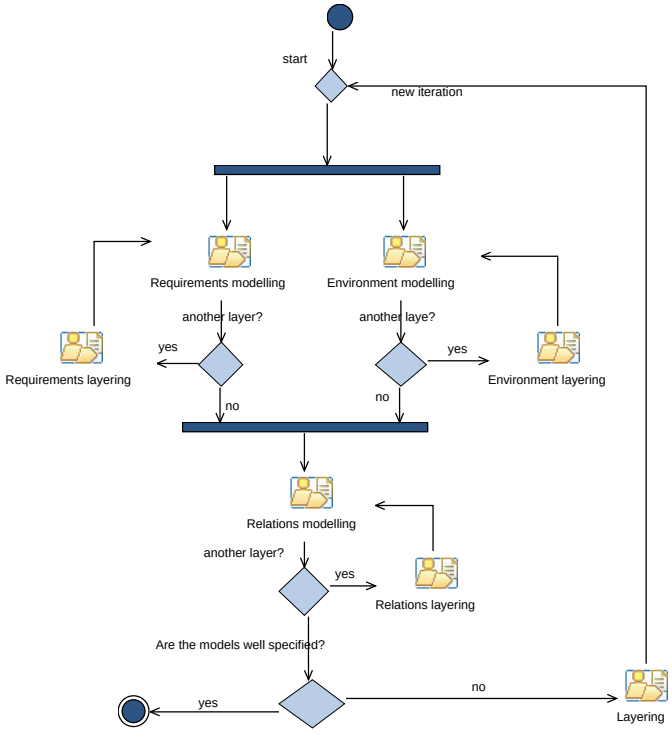


Fig. 3. Activity Diagram of the Requirements Analysis step.

b) Analysis.: The first activity in the Analysis step is *Moving from Requirements* (Figure 4), where the abstractions identified in the previous step are mapped onto the abstractions adopted in this stage to generate the initial version of the Analysis models. In particular, the Analysis step expresses the requirement representation in terms of more concrete entities such as *tasks* and *functions*. Tasks are activities requiring one or more competences and are analysed in the *Task analysis* activity, while functions are reactive activities aimed at supporting tasks analysed in the *Function analysis* activity. The structure of the environment, analysed in the *Topology analysis* activity, is also modelled in terms of *topologies*—i.e., topological constraints over the environment. The relations highlighted in the previous step are here the starting point for the definition of *dependencies* among such abstract entities in the *Dependency analysis* activity.

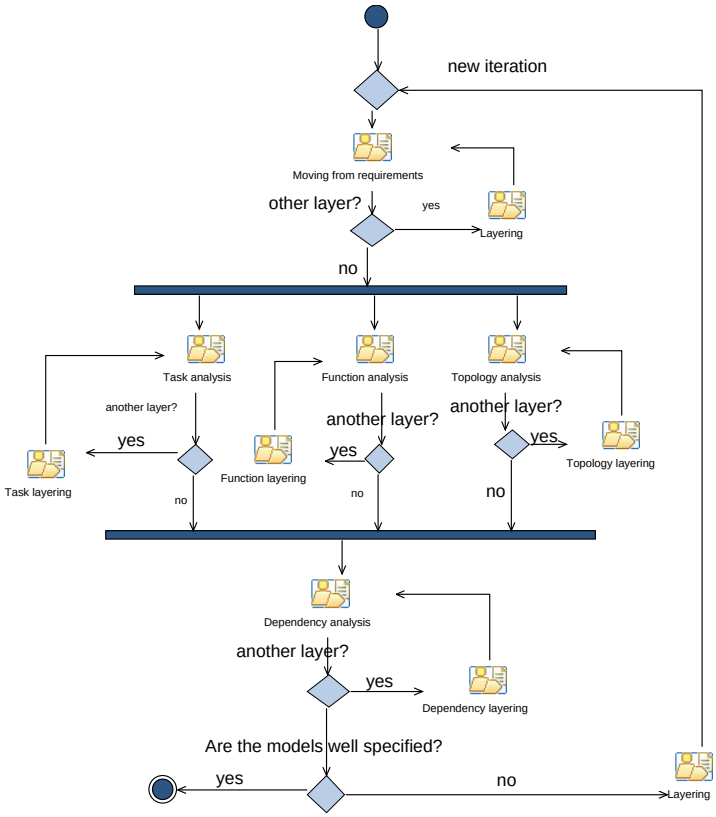


Fig. 4. Activity Diagram of the Analysis step.

c) Architectural Design.: This stage (Figure 5) is one of the more complex sub-phases in SODA. The first activity is *Transition* (Figure 5), where the abstractions identified in the previous step are mapped onto the abstractions adopted in this stage so as to generate the initial version of the Architectural Design models. The main goal is to assign responsibilities for achieving tasks to *roles* – *Role design* activity – and for providing functions to *resources*—*Resource design* activity. In order to attain one or more tasks, a role should be able to perform *actions* – *Role design* activity –; analogously, the

resource should be able to execute *operations* providing one or more functions—*Resource design* activity. The topology constraints lead to the definition of *spaces*, i.e., conceptual places structuring the environment in the *Space design* activity. Finally, the dependencies identified in the previous phase become here *interactions* and *rules*. Interactions represent the acts of the interaction among roles, among resources and between roles and resources, and are designed in the *Interaction design* activity; rules, instead, enable and bound the entities' behaviour and are designed in the *Constraint design* activity.

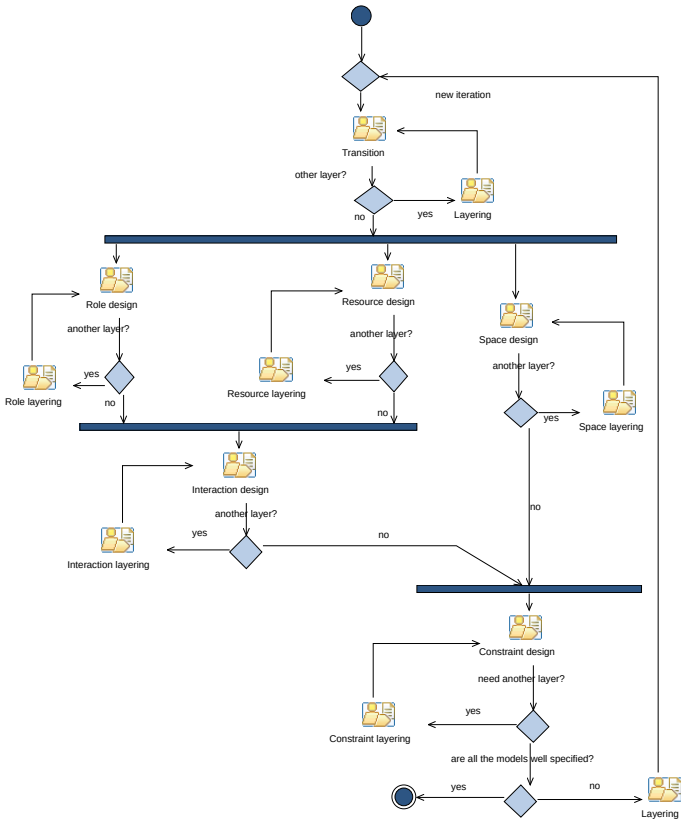


Fig. 5. Activity Diagram of the Architectural Design step.

d) Detailed Design.: The Detailed Design step (Figure 6) is the only stage where the layering principle is not applicable, since its goal is to choose the most adequate representation level for each architectural entity, thus leading to depict one (detailed) design from the several potential alternatives architectures outlined in the previous step. So, as shown in Figure 6, the first activity of this sub-process is *Carving*, which represents a sort of boundary between the Architectural Design and the Detailed Design, where the chosen system architecture is "carved out" from all the possible architectures. We also provide some SPEM's Guidelines for performing the carving activity properly. The next activity is *Mapping* (Figure 6), where the carved abstractions are mapped onto the abstractions adopted in this stage, thus generating the initial version of the Detailed Design models. These models are expressed in terms

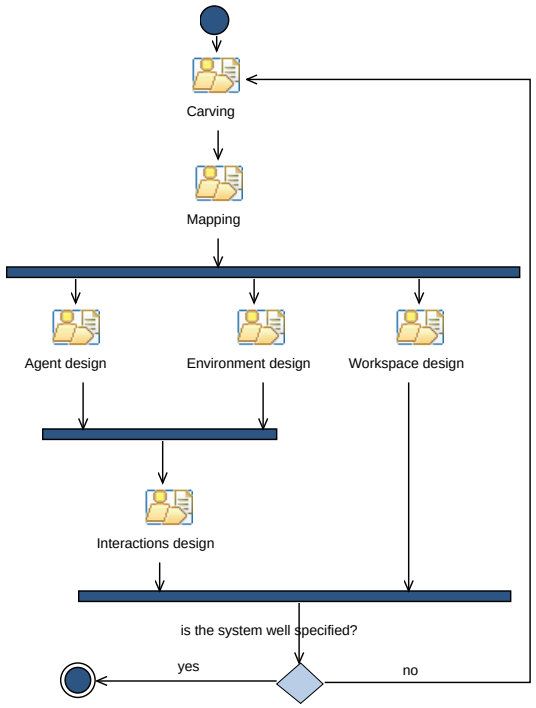


Fig. 6. Activity Diagram of the Detailed Design step.

of *agents*, agent *societies*, *composition*, *artifacts*, *aggregates* and *workspaces* for the abstract entities, while the interactions are expressed by means of *uses*, *manifests*, *speaks to* and *links to* concepts. More precisely, agents are intended here as autonomous entities able to play several roles, while a society can be seen as a group of interacting agents and artifacts whose overall behaviour is essentially autonomous and proactive: they are designed during the *Agent design* activity. The resources identified in the previous step are here mapped onto suitable artifacts, while aggregates are defined as a group of interacting agents and artifacts whose overall behaviour is essentially functional and reactive: they are designed during the *Environment design* activity. *Workspaces* take the form of an open set of artifacts and agents: artifacts can be dynamically added to or removed from workspaces, and agents can dynamically enter (join) or exit workspaces. Workspaces are designed in the *Workspace design* activity. Finally, the *uses*, *manifests*, *speaks to* and *links to* concepts are designed during the *Interactions design* activity.

IV. Discussion

In [6], the SPEM 1.0 meta-modelling power was put to test in the context of AOSE methodologies. There, SODA was taken as a case study to assess the strengths and limitations of SPEM, given its peculiar focus on the modelling and engineering (*i*) social issues, (*ii*) application environments, and (*iii*) complexity management—essential aspects for complex software systems. In order to simplify the comparison among the two versions of SPEM, Figure 7 reports the Activity Diagram of the Architectural Design stage as it was modelled

in SPEM 1.0. Three major problems were put in evidence at that time:

1) Activity Diagrams and abstractions did not easily capture the SODA layering principle: this is quite clear in Figure 7, where layering is represented as a simply activity and there is no way to detail the layering sub-process without reporting in the Activity Diagram all the layering sub-activities;
2) WorkProduct elements are characterised by a unique symbol, which makes it difficult to model the state changes of a WorkProduct during the process evolution (Figure 7);
3) UML Diagrams often become unreadable due to the too many elements required to represent a process: for instance, Figure 7 shows how Activities, Roles, Inputs and Workproducts are depicted in the same diagram.

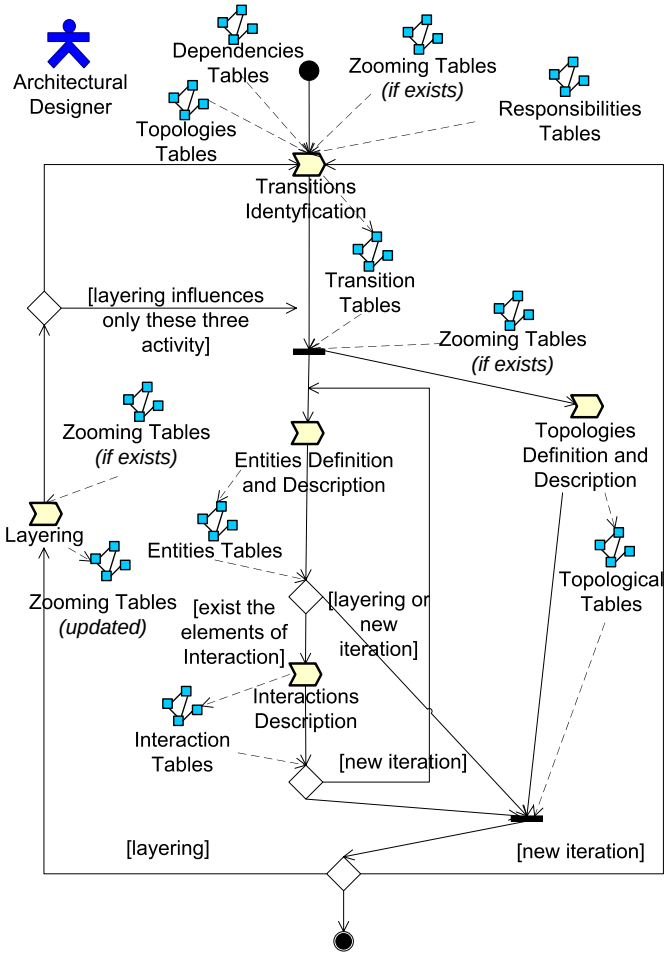


Fig. 7. Activity Diagram of the Architectural Design step (SPEM 1.0).

These limits depend on the fact that SPEM 1.0 does not offer sufficient abstractions for effectively managing the representation complexity of articulated processes like those underpinned by SODA. From this viewpoint, SPEM 2.0 seems to overcome the limits of the previous version. In fact, the first issue is now addressed by providing the *capability pattern* mechanism (Section II) that makes it possible to represent a process pattern as a single activity, hiding its internal structure. As seen in Section III, such a pattern is suitable for modelling the layering principle, and allows engineers to realise more understandable and readable diagrams by hiding the process complexity behind the Activity abstraction. So, the different activities composing the Layering can now be detailed without reporting them in the Activity Diagrams each time, leading to a great simplification (compare Figures 5 and 7).

The second issue is addressed in SPEM 2.0 by extending both the UML Activity Diagrams so as to represent the input and output parameters of an Activity, and the UML State Diagrams so as to annotate the State elements [5]. Such extensions enable UML State Diagrams to model the lifecycle of each WorkProduct, and relate each State element to the corresponding Activity that causes the state change.[1] This makes it unnecessary to represent the Workproducts inside the Activities Diagrams as it was in SPEM 1.0.

The last issue is already partially addressed by the solution adopted for the first issue, since capability patterns simplify the Diagrams structure; in addition, as seen in Section II, SPEM 2.0 introduces the concept of *process reusability* and allows Method Contents to be defined independently of their application in the development lifecycle. So, Method Contents can be re-used by relating their elements into a process that is customised for the specific type of project. As a result, each UML Diagram is now more readable, as it can focus only on a given portion of the Method Content / Process, and does not contain all the "unusable" entities which are not related to the considered portion of the meta-model.

In Section III, for instance, we defined a Method Content for each SODA stage, relating them to the corresponding processes. The Method Content defines the involved Roles, the Tasks to be performed with the corresponding steps, the Inputs and Workproducts, and the relation between the Inputs / Workproducts and Tasks; processes, in their turn, specify the Activities responsible for the tasks achievement and their order inside processes. The resulting Activities Diagrams in SPEM 1.0 and SPEM 2.0 for the Architectural Design stage are shown in Figure 7 and 5, respectively: the latter appears more readable, as it does not contain the Roles and Workproducts that are not necessary in this Diagram.

Summing up, SPEM 2.0 seems to overcome the major limits of its previous version, providing the right abstractions and mechanisms to model articulated process like SODA's, perhaps finding its way in the AOSE context.

V. Conclusions and Future Work

In this paper we took the SODA methodology as a case study for testing the applicability of SPEM 2.0 to AOSE methodologies. Moving from a previous work [6] where the SODA process was modelled in SPEM 1.0, we explored here whether SPEM 2.0 addressed the weaknesses and limits of expressiveness that had clearly emerged—mainly, the readability

[1]Example concerning WorkProduct elements are not reported here for obvious limitations in space.

of UML diagrams, both for the intrinsic complexity of Agent-Oriented methodologies, and for the lack of suitable ad-hoc entities.

Our experience indicates that SPEM 2.0 addresses such limits, by introducing a clear separation between Method Contents and Processes, adding capability patterns, and making it possible to express the ties between the Workproducts'states and the Activities that produce the changes in the Workproducts'states. Our next plans include testing SPEM in other contexts such as modelling the processes underpinned by MAS infrastructures, with the purpose of integrating AOSE methodologies and MAS infrastructures according to the Situational Method Engineering technique [11].

VI. Acknowledgements

This work has been supported by the MEnSA project (M*ethodologies for the* En*gineering of complex software* S*ystems:* A*gent-based approach*) funded by the Italian Ministry of University and Research (MUR) in the context of the National Research 'PRIN 2006' call.

References

[1] L. Cernuzzi, M. Cossentino, and F. Zambonelli, "Process models for agent-based development," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 2, pp. 205–222, March 2005.

[2] B. Henderson-Sellers and C. Gonzalez-Perez, "A comparison of four process metamodels and the creation of a new generic standard," *Information & Software Technology*, vol. 47, no. 1, pp. 49–65, 2005.

[3] S. Brinkkemper, K. Lyytinen, and R. Welke, *Method engineering: Principles of method construction and tool support*. Kluwer Academic Publishers, 1996.

[4] J. Ralyté and C. Rolland, "An approach for method reengineering," in *Conceptual Modeling*. London, UK: Springer-Verlag, 2001, pp. 471–484, 20th International Conference (ER 2001), Yokohama, Japan, 27-30 Nov. 2001. Proceedings. [Online]. Available: http://www.springerlink.com/content/pbtbr52cwya7qyd4/

[5] Object Management Group, "Software & Systems Process Engineering Meta-Model Specification 2.0," `http://www.omg.org/spec/SPEM/2.0/PDF`, Apr. 2008.

[6] E. Nardini, A. Molesini, A. Omicini, and E. Denti, "SPEM on test: the SODA case study," in *23th ACM Symposium on Applied Computing (SAC 2008)*, R. L. Wainwright, H. M. Haddad, R. Menezes, and M. Viroli, Eds., vol. 1. Fortaleza, Ceará, Brazil: ACM, 16–20 Mar. 2008, pp. 700–706, special Track on Software Engineering. [Online]. Available: http://portal.acm.org/citation.cfm?id=1363686.1363853

[7] IEEE-FIPA Methodology Working Group, "Home page," `http://www.fipa.org/activities/methodology.html`. [Online]. Available: http://www.fipa.org/activities/methodology.html

[8] A. Molesini, A. Omicini, E. Denti, and A. Ricci, "SODA: A roadmap to artefacts," in *Engineering Societies in the Agents World VI*, ser. LNAI, O. Dikenelli, M.-P. Gleizes, and A. Ricci, Eds. Springer, Jun. 2006, vol. 3963, pp. 49–62, 6th Inter. Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 Oct. 2005. Revised Paper. [Online]. Available: http://www.springerlink.com/link.asp?id=j68l84713542525p

[9] SODA, "Home page," `http://soda.apice.unibo.it`. [Online]. Available: http://soda.apice.unibo.it

[10] A. Omicini, "Formal ReSpecT in the A&A perspective," *Electronic Notes in Theoretical Computer Sciences*, vol. 175, no. 2, pp. 97–117, Jun. 2007, 5th Inter. Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 Aug. 2006. Post-proceedings.

[11] M. Cossentino, S. Gaglio, N. Gaud, V. Hilaire, A. Koukam, and V. Seidita, "A MAS metamodel-driven approach to process composition," in *9th International Workshop on Agent Oriented Software Engineering (AOSE'08)*, M. Luck and J. Gómez-Sanz, Eds., AAMAS 2009, Estoril, Portugal, 12–13 May 2008.

Towards filling the gap between AOSE methodologies and infrastructures: requirements and meta-model

Fabiano Dalpiaz*, Ambra Molesini†, Mariachiara Puviani‡ and Valeria Seidita§
*Dipartimento di Ingegneria e Scienza dell'Informazione
Università di Trento
Email: dalpiaz@disi.unitn.it
†Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna
ambra.molesini@unibo.it
‡Dipartimento di Ingegneria dell'Informazione
Università di Modena e Reggio Emilia
mariachiara.puviani@unimore.it
§Computer Science and Artificial Intelligence Laboratory
Università degli Studi di Palermo
seidita@dinfo.unipa.it

***Abstract*—Many different methodologies have been proposed in Agent Oriented Software Engineering (AOSE) literature, and the concepts they rely on are different from those adopted when implementing the system. This conceptual gap often creates inconsistencies between specifications and implementation. We propose a metamodel-based approach that aims to bridge this gap, resulting in an integrated meta-model that merges the best aspects of four relevant AOSE methodologies (GAIA, Tropos, SODA and PASSI). The meta-model assembly followed a well-defined process: for each methodology to be integrated in the meta-model, we elicited the requirements, identified a set of process fragments, thoroughly compared the concepts belonging to the various fragments, and finally composed the meta-model.**

I. Introduction

The trend towards agent-oriented software engineering (AOSE) is motivated by the need for a new engineering paradigm to face the increasing complexity and openness of computational systems. Object-oriented software engineering is adequate for the development of a variety of systems, but it falls short when applied to the development of open complex systems. This class of systems introduces the need for a new computing paradigm based on distributed intelligent units – *agents* –, whose characteristics are intrinsically different from objects [1]. This paradigmatic shift involves both the conceptual and the technical levels of the development cycle, ranging from the requirements analysis to the implementation and the deployment over an infrastructure. The work we present here is in the context of the "Methodologies for the Engineering of complex Software systems: Agent-based approach" (MEnSA) project[1], which aims at filling the conceptual gap between AOSE methodologies and multi-agent systems (MAS) infrastructures.

This gap is well known: Molesini et al. [2] examined this problem and proposed a case study concerning the SODA methodology [3]. Integrating an AOSE methodology with a MAS infrastructure requires to compare and relate the concepts, to provide a set of methodological guidelines, and to introduce a set of new concepts acting as a glue to make the integration successful. This task is not trivial, and one of the main reasons that make it complex is the difference in perspectives of methodologies and infrastructures developers. AOSE methodologies follow a top-down approach starting from a real world problem and moving towards a solution (the architecture of a MAS); thus, the concepts and techniques developed are mainly suitable for the use at analysis and design phases. On the other hand, the developers of MAS infrastructures follow a bottom-up approach starting from already existing programming paradigms, often an object-oriented one, and build upon it to form higher level programming constructs that make the development of the agent-based software easier.

MEnSA's "filling the gap" objective requires a complex process, made up of several sub-tasks, whose common element is the usage of a meta-model approach, and will ultimately produce an integrated methodology. This paper is focused on the work we have done concerning the integration of a number of AOSE methodologies; the integration of infrastructures is ongoing, and it will be presented in future publications. The AOSE methodologies taken into consideration by MEnSA are GAIA [4], Tropos [5], SODA [2], and PASSI [6]: they mainly differ in the typical scenarios they are designed to support, and in the phases they better cover. For instance, Tropos exploits a well established technique for requirements analysis (goal

[1]http://www.mensa-project.org

modeling), SODA provides an exhaustive characterization of the environment, PASSI has an extensive coverage of the implementation phase, and GAIA is well suited for the modeling of organizational aspects.

Our approach is founded on the work done by Cossentino et al. [7], [8] and starts from the definition of a set of requirements for the meta-model we want to assembly. Then, we elicit a set of fragments fulfilling the identified requirements, define a semantic conceptual map to precisely relate concepts belonging to various methodologies, and finally compose the fragments into an integrated meta-model.

This paper is structured as follows: Section II discusses the requirements we identified to lead the assembly of the meta-model; Section III describes the selected fragments and presents the conceptual map to compare the methodologies; Section IV focuses on the meta-model, describing the current version of meta-model; Section V terminates the paper by proposing conclusions and future work.

II. Requirements and principles for assembling the meta-model

In order to obtain a good meta-model – and a good methodology – we followed a path similar to that adopted in the engineering of a (software) system and proposed in [7], [9]. After defining the requirements for our product (the meta-model), we identified the fragments that better contribute to the satisfaction of the requirements.

In this section we illustrate how the integrated meta-model was conceived (Section II-A), and describe the requirements that led to a new methodology (Section II-B).

A. Assembling a meta-model

A meta-model describes the structure of all the elements that should be designed when following a specific methodology. Relationships between elements have specific meanings, and they should reflect the phases in the methodology. Different methodologies are built according to specific design philosophies, and comparing their meta-models is not a trivial task: often, the described concepts and relationships share the name but have different semantics.

Previous experiences in meta-models creation (e.g., [8], [9]) made it clear that this activity is much more than the mere selection and composition of concepts from the existing meta-models. Different composition patterns can be encountered:

1) selected elements from existing meta-models present the same name but have different meanings. This is the most common and difficult situation to be faced; a deep analysis of the collected elements has to be done, possibly some new elements have to be introduced, some others have to be modified in order to fill the presented differences;
2) selected elements have the same meaning but different names (the opposite of the previous case): renaming some elements is necessary;
3) all the selected elements present totally disjoint names and definitions, requiring just a simple composition; this is the best situation we could encounter, though the most unusual.

Given the consistency and coherence problems enumerated above, an integrated meta-model normally needs to be completed by concepts and relations acting as glue, introduced to ensure the important features of the original methodologies are not lost. After a sufficient refinement of the meta-model, it is possible to start the new methodology definition by assembling a set of selected process fragments according to the chosen life-cycle. If the selected fragments do not completely cover all the life-cycle phases and the requirements, new fragments will be selected, modified (if needed) and added.

B. The methodology requirements

The definition of a new methodology has to start by specifying the requirements to be satisfied. For the construction of a new integrated methodology, we decided to start from the requirements, choose the more suitable fragments belonging to existing methodologies, and assemble them in a proper way. The evaluation of the resulting integrated methodology is the verification of the extent to which the requirements are satisfied.

Now we list and describe the requirements and sub-requirements for our integrated AOSE methodology:

1) **Fill the gap between design and implementation**:
   a) Transformational approach from requirements elicitation to design and implementation, which refines high-level abstractions into low-level more concrete entities.
   b) Support for traceability: the path from each requirement to the corresponding source code should be clear and easy to identify.
   c) Powerful abstractions during the design phase are needed to provide an appropriate design of the system; they should be close to the infrastructure-level abstractions, but attention should be paid to avoid too fine-grained designs.
2) **Good requirements elicitation and analysis**:
   a) Support for both functional and non-functional requirements.
   b) Support for both goal-oriented and functional-oriented analysis.
3) **Different abstractions in the different phases** should make the comprehension of the design process easier.
4) **Enabling an easy transition towards the new methodology** for designers who are expert with one or more of the input methodologies.
5) **Precise and compact modeling constructs for the concept of agency**:
   i Agent: the definition of what an agent is and what it is supposed to do during its lifetime.
   ii Agent's rationale: the rationale an agent follows to achieve its objectives, that is the general reasoning principles leading the agent's behavior.

iii Situated agents: the environment where agents live requires an explicit representation throughout the whole methodology.
iv Social agents: agent-to-agent and agent-to-environment interactions are essential to engineer a multi-agent system.

Considering these requirements we started the analysis of the four selected methodologies (Tropos, Gaia, SODA, and PASSI), and we discovered more specific and detailed requirements. These requirements are listed below:

1) **Transformational process**: this need comes from requirements 1 and 3. The model-driven engineering paradigm [10] will be therefore adopted. Transformations between the elements of different domains should be clearly defined. For example, the notion of agent exists in different development phases and methodologies with (slightly) different meanings. Following a transformational approach, we can define several types of agent (requirements agent, design agent, ...), and define the way a certain agent transforms (or refines) into another one.
2) **Layering**: the management of different abstraction levels simplifies the design (requirements 1 and 3). SODA supports layering by means of the zooming and projection mechanisms. Zooming makes it possible to pass from an abstract layer to another, while projection projects the entities of a layer into another [3].
3) **Goal-oriented analysis** should be performed **before functional-oriented analysis**. The latter should start from results of the former. The goal-oriented analysis stands as a basis for Tropos, where agents are defined in terms of the functional and non-functional goals they want to achieve. Functional-oriented analysis is then used by eliciting the tasks to be executed to achieve the goals.
4) **Interaction**:
   a) Agent interactions should support *semantic communications* for removing or minimizing the ambiguity of messages contents.
   b) An ontology should be used to model agent knowledge in order to provide a conceptual background to all the agents belonging to a MAS.
   c) Compliance with FIPA ACL (Agent Communication Language) [11] specifications at the communication level is necessary.
   d) Agent interactions with the environment should be explicitly modeled.
   e) Indirect interactions (e.g., blackboard-based) should be supported.
5) **Organizational rules** proved to be a useful approach for modeling some social aspects. Gaia and SODA are examples of methodologies based on (organizational) rules to constrain and direct the agents behaviors.
6) **Environment and topology modeling** can be done by adopting abstractions like SODA's *artifacts and workspaces* in order to explicitly distinguish between active entities (agents) and passive entities (artifacts), and for organizing the conceptual places – workspaces – structuring the environment.
7) **Non-functional requirements should be explicitly modeled** (requirement 2). Tropos is the first AOSE methodology supporting explicit modeling of non-functional requirements, through the concept of soft-goals.
8) **Agent plans** should be modeled but they should not constrain the agent architecture to a specific kind of agent. In other words, the methodology should provide an abstract representation of plans, which can be realized into several implementations.

III. Selected fragments and conceptual map

The starting point for the meta-model creation is the requirements we described in the previous section. Given this, the approach we used to devise a meta-model is the following.

- Firstly, we have derived a set of fragments (Section III-A) satisfying the requirements identified in Section II. These fragments represent the core of the meta-model, which should be analyzed and refined in order to provide a better integration of the fragments.
- Secondly, we built a glossary of terms relevant to the fragments identified in Section III-A. For space reasons the dictionary is not reported here and it can be found in [12].
- Then, based on the glossary, in Section III-B we defined a conceptual map of terms, identifying synonyms and similar terms, and pointing out existing conflicts.
- Finally, on the basis of all the previous work, we defined the first version of the meta-model in Section IV.

A. Selected fragments

Each of the studied methodologies has some strong points, and should give a significant contribution to formulate the final MEnSA methodology. Here we list the coarse-grained fragments we have initially chosen from each methodology, which we extracted from the FIPA TC repository of fragments [13]:

1) **Tropos**
   a) Early requirements phase:
      i) Organization description.
      ii) Analysis.
   b) Late requirements phase:
      i) System identification.
      ii) Environment description.
2) **Gaia**
   a) Analysis phase:
      i) System *roles* identification.
      ii) Role model elaboration.
   b) Design phase:
      i) Service model development.

3) **SODA**
   a) Architectural Design:
      i) MAS Organisational model.
   b) Detailed Design:
      i) MAS Interaction model.
   c) Environment model.
4) **PASSI**
   a) Agent Society:
      i) Domain Ontology design.
      ii) Communication Ontology description.
   b) Agent Implementation:
      i) Multi-agent system design.
   c) System Requirement:
      i) Agent Identification.

B. Linking methodologies: a conceptual map

In order to propose an integrated meta-model, we built a conceptual map for eliciting *synonyms* (or, at least, similar concepts), and *inter-level relations* between concepts used at different abstraction levels. The conceptual map has been built on the basis of the MEnSA glossary [12], which has provided a complete and accurate semantic matching schema connecting the fragments' abstractions.

The conceptual map is shown in Figure 1; we used different colors to depict concepts belonging to fragments coming from distinct meta-models. The concepts are tied by two types of graphical links that represent two different relationships:

- non-directed links represent **horizontal relations**, which relate two concepts that are "synonyms". Identifying concepts sharing the same definition is very unlikely, and the resulting integration would be loose and nearly useless. Therefore we decided to extend the equivalence relation to include those concepts having a similar definition and whose usage in practice is equivalent. Horizontal relations (h) are not transitive: $h(c1, c2) \wedge h(c1, c3) \nrightarrow h(c2, c3)$.
- directed links point out **vertical relations**, which create "inter-level" links (top-down) between concepts belonging to different abstraction levels. We define *abstraction* as the development phase a concept belong to. Since we use h-relations to express similarity (and not only sameness), $h(c1, c2) \wedge v(c1, c3) \nrightarrow v(c2, c3)$.

In order to explain the conceptual map, in Figure 1 we have organized the different relations among concepts in various labeled sets. So, the first diagram chunk (a) concerns non-agentive concepts, which are typically in the system-to-be together with the agents. The Tropos concept *Resource* is horizontally linked to *Function* and *Legacy System* in SODA: more precisely, the former SODA concept is almost equivalent to Tropos resource, whereas the latter is linked because a legacy system defines a set of resources that should be modeled.

Aggregation of agents is examined in (b): Gaia *Organization* is a very high-level view of a set of agents (analysis phase), which is vertically linked to the lower-level concept *Society* of SODA (detailed design). These concepts are useful to

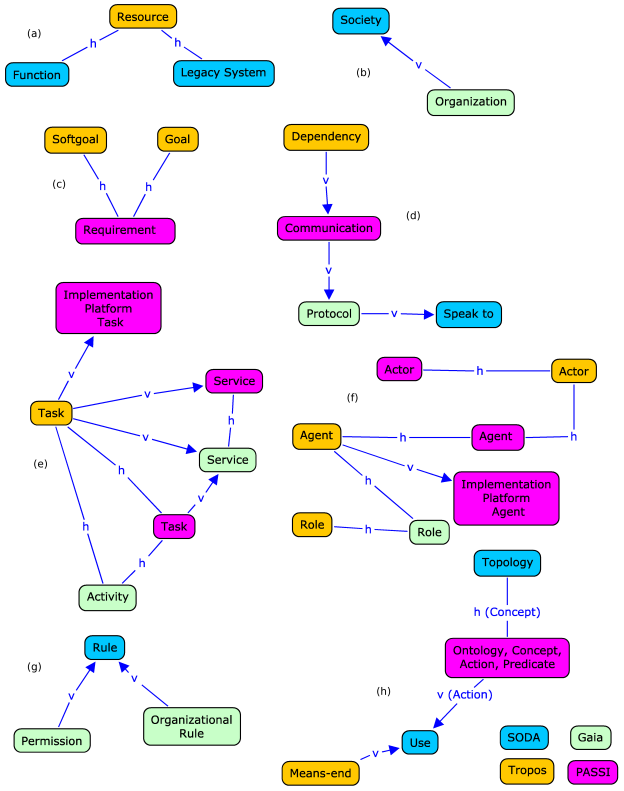


Fig. 1. Conceptual map linking concepts of different methodologies.

support the multi-level definition of the MEnSA meta-model, providing two related abstractions at different levels.

Requirements are considered in (c): Tropos *Goal* and *Softgoal* are horizontally linked to PASSI *Requirement*, the latter representing either a functional or a non-functional requirement.

Sociality of agents is represented in chunk (d). In this particular case, we have been able to point out a well defined hierarchical structure connecting the four examined methodologies. Tropos *Dependency* is used to depict linked actors justifying the reason why they depend on each other (for a goal, a task, or a resource); dependencies are defined during requirements analysis, and in our map they are vertically connected to PASSI *Communication*, which is a design-time concept defining an abstract interaction between two agents. We achieve a lower level of abstraction by linking communication to the Gaia *Protocol*, which defines the way in which roles interact with each other. Protocol is linked to an even lower level to the SODA *Speaks To*, which refers to the act of interaction between agents. It is worth noting that these concepts refer to different types of entity: a dependency involves actors, a communication is between agents, a protocol involves roles, a speaks-to is an atomic interaction act between agents. The meta-model and the derived methodology will handle this heterogeneity by defining exactly where these concepts apply.

Another interesting topic is that related to tasks and services

(e). Tropos *Task* is vertically connected to PASSI *Implementation Platform Task*, the latter being an implementation-level realization of the former (which stands at requirements level). Tropos *Task* is horizontally linked to PASSI *Task*. PASSI *Service* has a definition which is very similar to Gaia *Service*, and hence these two concepts are horizontally linked, providing an abstraction during the design phase. PASSI *Task* is vertically linked to Gaia *Service*. Gaia *Activity* is horizontally linked both to Tropos and PASSI *Task*. From these relations, we can derive a top-down relationship between task (or activity) and service, the former being higher level than the latter.

A crucial part of the conceptual map is the one related to agents, actors, and roles (f), because these are the active entities that glue all the other concepts together. Tropos *Agent* is horizontally linked to PASSI *Agent*, for they are both representations of the same concept. Another horizontal relation is between Tropos and PASSI *Actor*. Moreover, Tropos *Actor* is horizontally linked to PASSI *Agent*. These four concepts are not synonymous, but they are at the same level of abstraction. A third couple of horizontally linked concepts is Gaia and Tropos *Role*, with a further horizontal relation between Tropos *Agent* and Gaia *Role*. There is a vertical relation between Tropos agent and PASSI *Implementation Platform Agent*. This diagram chunk is not very precise: many relations have been identified, but during the meta-model and methodology definition we will have to make some choices and define the selected concepts in a more precise way.

Constraints are an important aspect in multi-agents systems, and our conceptual map contains some related entities and relations in (g). SODA *Rule* is a general design-time concept for regulating agents and their interaction with the environment they live in. Gaia *Permission* and *Organizational Rule* are analysis-level concepts which define the organization in terms of rules and permissions, and are linked vertically to SODA *Rule*.

The last part of the diagram (h) mainly involves the usage of entities by agents. SODA *Uses* is intended to depict a kind of interaction between an agent and an artifact, whereas Tropos *Means-end* is a higher-level abstraction of this behavior, where there is an intentional relation between a goal (or task) and a resource: the latter is the means to achieve the end (the goal or the task). PASSI *Action* is vertically linked to *Use*, whereas PASSI *Concept* is horizontally linked to SODA *Topology*, since a topology is defined in terms of the concepts constituting and regulating it.

IV. MEnSA META-MODEL: A PRELIMINARY VERSION

Starting from the MEnSA glossary and the conceptual map linking concepts of different methodologies, we have created a first version of the MEnSA meta-model. This initial effort is restricted to the phases of requirements and design; the layering is coarse grained, and implementation-level concepts are just sketched. The meta-model we present here slightly refines the initial version described in MEnSA deliverable 1.2 [12].

The key notions around which all the other elements are placed, are *role* and *agent*, which are building blocks for several AOSE methodologies. The meta-model is presented in Figure 2, and we describe it in the next two sub-sections, which refer to (1) requirements phase, and (2) design and implementation phases of the meta-model. The term "phase" is here used in order to represent the logical connection between the three main software process engineering phases and the meta-model elements a designer instantiates while developing each phase.

A. Requirements phase

The requirements phase is mainly based on the fragments of Tropos and Gaia, with some concepts coming from SODA (environment-related), and ontological aspects extracted from PASSI. The meaning of the presented concepts and relations derives from the corresponding methodologies, unless we specified otherwise in the description.

The main notion in the requirements phase in MEnSA meta-model is that of *Requirements Agent*, which is defined in terms of the concepts it connects to (through association links). A requirements agent *plays* one or more *Roles*, and *knows* an *Ontology*. The concept of Role is defined as Tropos' role plus Gaia's rules and permissions, whereas Ontology comes from PASSI. An *Organization* (Gaia) is composed of a set of agents (*made_of* relation between Organization and Requirements Agent), and *has* a set of *Organizational rule*s (Gaia) which define the regulations of the Organization. Every Role *adheres* to the Organizational rules of the Organization where the agent playing that Role lives.

The element Ontology is specified in a slightly different way from PASSI's definition, because we support here a refinement process of the ontology in different development phases. At requirements time, the Ontology is made of a set of *Requirements Concept*s, which can be hierarchically organized (self-transition). This is a coarse-grained representation of an ontology, which will be refined at design-time.

Each Role is *responsible* for one or more *Requirement*s (equivalent to Tropos' abstract goal), and each Requirement can belong to more than one Role. Requirement is specialized (concretized) into *Goal* (hard-goal in Tropos) and *NFR* (Non-Functional Requirement) (soft-goal in Tropos). A Goal can be *and/or-decomposed* into a set of sub-goals, *contribute* to Non-Functional Requirements, and can be *means-end* decomposed. The means to achieve a Goal can be either a *Resource* (in the Tropos sense, which corresponds to SODA Function) or an *Activity* (in Gaia glossary, but we showed this it is horizontally linked to Tropos and PASSI task). Activities, like Goals, contribute to the satisfaction of Non-Functional Requirements, and can be refined through *and/or decomposition*. Resources can be viewed as means to carry out an Activity, in the same manner they are used to achieve Goals.

Another important concept in the requirements phase is that of *Dependency*: this notion is taken from Tropos, and connects a *depender* role to a *dependee* role *for* a certain *Dependum* (the object of the dependency). A Dependum can be either a

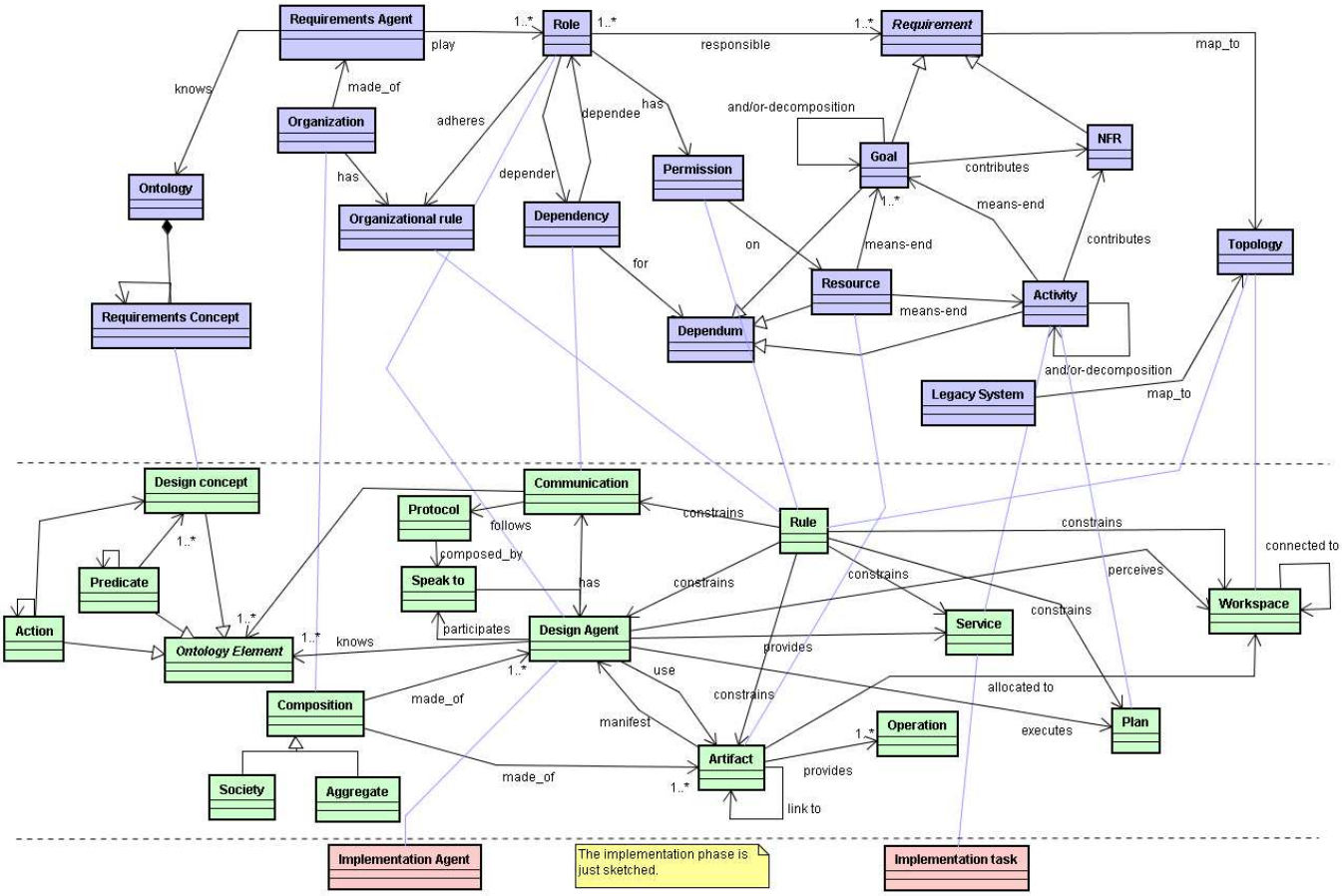


Fig. 2. First version of the MEnSA meta-model.

Goal (the depender wants the dependee to fulfill that Goal), an Activity (the depender wants the dependee to execute an activity), or a Resource (the depender wants the dependee to provide a Resource).

A Role is defined also by expressing which *Permission*s it *has on* certain Resources. This enables the definition of constraints on the usage of/access to resources.

A Requirement has a relation *map_to* with SODA *Topology*, because the achievement of that requirement depends on the topology definition. The concept of *Legacy System* (SODA) *map_to* a topology, as well.

B. Design and implementation phases

In these phases an additional modeling construct is used to define elements, that is the *realization* links between concepts at different abstraction layers, which define the inter-layer relationships that ensure connections between the various phases.

The concept *Design Agent* realizes the Requirements Agent, and is defined in terms of the associations with other design-time entities. In the context of communication, it *has* a set of *Communication*s active at a certain time (zero or more); every Communication follows a *Protocol*, that is the set of rules that govern the interaction between agents. The Protocol is in turn *composed_by* a number of *Speaks To* elements, which are the elementary (atomic) communication actions, involving two different Design Agents through the association *participates*. A Communication is a top-down realization of Dependency, implementing in the meta-model the vertical link of the conceptual map of Figure 1. Moreover, communication is connected to the abstract class *Ontology Element*, which is made concrete by *Action*, *Predicate*, and *Design concept*. Action and Predicate are connected to Design Concept, as prescribed by PASSI. The difference between PASSI's representation of Ontologies and our specification is in the realization of the requirements concept into a design concept, which enables a refinement of the Ontology during the development cycle. Ontology concept has an incoming association from Design Agent labeled *knows*, which represents the ontological knowledge of an agent.

The Design Agent *provides* a number of *Service*s, which are design-time realizations of the requirement-level concept Activity. The second realization of Activity is the concept of *Plan*, which is a common design-time construct to define the behavior of an agent. An Agent also *perceives* a *Workspace*, which realizes Topology, and can be *connected to* other Topology

entities. A Workspace can be *connected to* other workspaces. Design Agents *use* a set of *Artifact*s, which in turn expose their interface (*manifests* relation between Artifact and Design Agent). Artifacts are the realization of the requirements-time concept Resource. The relation between agents and Artifacts comes from SODA, and it is very important to connect active entities to passive entities in the system. Following SODA meta-model, an Artifact *provides* one or more *Operation*s, can *links to* other Artifacts, and is *allocated to* a Workspace.

Composition is another concept derived from SODA: here it is intended as a design-time realization of Organization. Composition is *made_of* Design Agents and Artifacts, and is specialized by *Society* (a collection of agents and artifacts exhibiting proactive behavior) and *Aggregate* (which exhibits a functional behavior).

The concept of *Rule* is quite important at design-time, for it allows constraining a number of other entities. It is a realization of both Organizational rule, Permission, and Topology; this way it enables control over disparate concerns in the multi-agent system. The concept of Rule is linked, via the association *constrains*, to many other concepts: it governs the Communication between agents, puts constraints on the Design Agent behavior, is encoded into Artifacts to define how they can be used and accessed, constrains both the Services provided and the Plans executed by the agents, and regulates the Workspace where Artifacts are located.

We did not put emphasis to the implementation phase here, because we believe that the definition of this meta-model part will be much easier when infrastructures come into place, providing the suitable abstractions to model this phase. From the methodologies we introduce just two realization: Design Agent into the *Implementation Agent*, and Service into *Implementation Task*, both coming from PASSI.

V. Conclusion and future work

In this paper, we have presented an initial version of an integrated agent-oriented meta-model which aims at being the basis for the creation of a new agent-oriented methodology integrated with MAS infrastructures through a well interrelated set of phases from requirements to implementation. The basis of the meta-model are the fragments selected from the four AOSE methodologies: Gaia, PASSI, SODA and Tropos.

The process we followed for defining the meta-model starts from the identification of the requirements for the target methodology. These requirements helped in the selection of a list of fragments from the four considered methodologies. The next two steps were the construction of a glossary, and the definition of a conceptual map of methodologies abstractions. This map was built to reflect the relations of similarity (at the same level of abstraction, that is at the same development phase) and realization (in the form of "requirements concept X is realized by design concept Y") among the abstractions adopted by each considered methodology.

The most immediate work direction is the definition of the meta-model's implementation phase, extracted from a set of MAS infrastructures. This will likely be done by adopting the process presented in Section II.

In addition, the meta-model will certainly be refined as a result of the work on the methodological aspects and the validation phase over a case study.

Another aspect to be considered for refining the presented meta-model concerns the meta-model structure: in the current version we have only two development phases that seem too coarse-grained. We will refine the meta-model splitting the two phases into different and more detailed sub-phases, e.g., the requirements analysis phase could be split into early and late requirements.

All these directions will lead to the creation of the MEnSA methodology, which will be based on the meta-model introduced here. In addition, during the definition of the methodology there will be bidirectional feedbacks between the methodology and the meta-model in order to refine again the meta-model.

Acknowledgements

Part of this work makes use of results produced by the MEnSA project (PRIN 2006) and by the PI2S2 Project managed by the Consorzio COMETA (PON 2000-2006).

References

[1] N. R. Jennings, "On agent-based software engineering," *Artif. Intell.*, vol. 117, no. 2, pp. 277–296, 2000.

[2] A. Molesini, E. Denti, and A. Omicini, "From AO methodologies to MAS infrastructures: The SODA case study," in *Engineering Societies in the Agents World VIII*, ser. LNAI, A. Artikis, G. O'Hare, K. Stathis, and G. Vouros, Eds. Springer, 2008, vol. 4995, pp. 300–317, 8th International Workshop (ESAW'07), 22–24 Oct. 2007, Athens, Greece.

[3] A. Molesini, A. Omicini, E. Denti, and A. Ricci, "SODA: A roadmap to artefacts," in *Proc. of the 6th International Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 Oct. 2005. Revised, Selected & Invited Papers*, O. Dikenelli, M.-P. Gleizes, and A. Ricci, Eds., 2006, pp. 49–62.

[4] F. Zambonelli, N. R. Jennings, and M. Wooldridge, "Developing Multi-agent Systems: the Gaia Methodology," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 3, pp. 417–470, 2003.

[5] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini, "Tropos: An agent-oriented software development methodology," *Autonomous Agent and Multi-Agent Systems (8)*, vol. 3, pp. 203–236, 2004.

[6] M. Cossentino, "From requirements to code with the PASSI methodology," in *Agent Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini, Eds. Hershey, PA, USA: Idea Group Publishing, Jun. 2005, ch. IV, pp. 79–106.

[7] M. Cossentino, S. Gaglio, A. Garro, and V. Seidita, "Method fragments for agent design methodologies: from standardisation to research," *International Journal of Agent-Oriented Software Engineering*, vol. 1, no. 1, pp. 91–121, 2007.

[8] M. Cossentino, S. Gaglio, N. Gaud, V. Hilaire, A. Koukam, and V. Seidita, "A MAS metamodel-driven approach to process composition," in *Proceedings of The 9th International Workshop on Agent Oriented Software Engineering (AOSE'08)*, M. Luck and J. Gómez-Sanz, Eds., Estoril, Portugal, 12–13 May 2008.

[9] V. Seidita, M. Cossentino, and S. Gaglio, "Adapting PASSI to support a goal oriented approach: a situational method engineering experiment," in *Proc. of the Fifth European workshop on Multi-Agent Systems (EUMAS'07)*, 2007.

[10] S. Kent, "Model driven engineering," in *IFM*, ser. Lecture Notes in Computer Science, M. J. Butler, L. Petre, and K. Sere, Eds., vol. 2335. Springer, 2002, pp. 286–298.

[11] FIPA, "Home page," http://www.fipa.org/.

[12] MenSA Group, "Deliverable 1.2," http://www.mensa-project.org/request.php?41.

[13] FIPA Methodologies, "Home page," http://www.pa.icar.cnr.it/~cossentino/FIPAmeth/.

Using multi-coordination for the design of mobile agent interactions

Giancarlo Fortino, Alfredo Garro, Samuele Mascillaro, and Wilma Russo

***Abstract*— This paper proposes a *multi-coordination* approach for the design of mobile agent interactions. The approach is founded on the *multi-coordination* concept, which is a synergic exploitation of multiple coordination models which best fit interaction requirements. In particular, the proposed approach is based on two steps: (i) candidate design solutions are defined through a procedure which allows to identify the most effective coordination models for a given mobile agent interaction scenario; (ii) the defined candidate design solutions are quantitatively evaluated through a discrete-event simulation framework which allows for an easy evaluation of mobile agent interaction scenarios in terms of ad-hoc defined performance indices.**

Index Terms— **Agent Interaction Design, Mobile Agents, Multi-Coordination, Performance Evaluation.**

I. Introduction

Code mobility paradigms have been introduced to support the design and the implementation of flexible, dynamic and reconfigurable distributed applications in terms of software components which are not confined in a single run-time context for their entire lifecycle but can migrate autonomously or on-demand across different contexts [1]. Among them, the most fascinating paradigm is represented by the mobile agents, executing software components capable of autonomous migration by retaining code, data and execution state. Although it is advocated that the exploitation of mobile agents can provide many benefits [2], they have introduced specific and not yet fully addressed issues that actually limit their advertised wide-spread use [3]. An interesting issue concerning with the design of mobile agent interactions regards how to clearly identify which agents will be interacting and how their interactions can be modeled. To deal with mobile agent interactions, communication paradigms and mechanisms as well as coordination models and architectures for non mobile software components have been enhanced to be mobility-aware (message-passing, tuple space, publish/subscribe, etc) and new ones have been purposely defined for logical and physical mobility (meeting, blackboard, shared transiently tuple spaces, reactive tuples) [4, 5, 6, 7, 8, 9]. Although some mobile agent frameworks already offer several mechanisms based on the aforementioned communication/coordination paradigms and architectures, in the current practice mobile agent interactions are designed on the basis of a single paradigm which is mainly based on message passing or, in some application domains, on tuple spaces [4]. As single model based communication/coordination might not be effective for satisfying all needs of mobile agent interactions in all possible application scenarios, the exploitation of multiple communication/coordination paradigms, namely *Multi-Coordination*, can enhance design effectiveness, improve efficiency, and enable adaptability in dynamic and heterogeneous computing environments [10]. In particular, *Multi-Coordination* allows agents to choose among a variety of different communication/coordination paradigms which best fit mobile agent interaction needs. Moreover, although several design patterns have been proposed for driving the design of mobile agent interactions [11, 12] and programmable coordination models and related frameworks (e.g. TuCSoN [13]) are now available, systematic methods for supporting the development of mobile agent interactions which specifically take into account an integrated exploitation of multiple coordination models are surprisingly still lacking. To overcome this lack, this paper proposes a *multi-coordination* approach for the *design* and *evaluation* of mobile agent interactions.

The *design* is based on a procedure which uses suitable agent interaction patterns to fulfill agent coordination requirements. In particular, interaction patterns are first characterized by appositely defined parameters and associated to specific coordination models according to such parameters; then, the most appropriate coordination model is selected for implementing a given interaction pattern so providing a design solution for the related coordination requirement.

The *evaluation* is based on a *discrete-event simulation framework* which allows to evaluate the designed solutions in

G. Fortino is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (corresponding author; phone: +39.0984.494063; fax: +39.0984.494713; e-mail: g.fortino@unical.it).

A. Garro is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: garro @unical.it).

S. Mascillaro is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: samuele.mascillaro@deis.unical.it).

W. Russo is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: w.russo @unical.it).

terms of performance indices with reference to given application scenarios. In particular, the *simulation framework* provides effective abstractions for easily programming mobile agent interaction scenarios and flexibly supporting configuration, execution and evaluation of such scenarios.

The proposed *multi-coordination* approach makes it possible the definition of alternative design solutions and their evaluation and comparison from qualitative (i.e. according to design effectiveness criteria) and quantitative (i.e. according to performance indices) points of view.

To show a concrete application of the proposed approach, a significant case study related to mobile agent-based distributed information retrieval is presented. In particular, some design solutions, which use different coordination models (message-passing, Linda-like tuple space, publish/subscribe), are defined on the basis of specific agent coordination requirements. Among the designed solutions, *multi-coordination-based* and *message-passing-based* solutions have been evaluated against significant performance indices. The evaluation shows that the *multi-coordination-based* solution has the best overall performance.

The remainder of this paper is organized as follows. Section II provides some background concepts about mobile agent coordination and discusses related work. By using a case study Section III exemplifies the design of alternative solutions through a three-step procedure. Section IV briefly proposes a comparison of the results of the performance evaluation of two alternative solutions based on multi-coordination and message-passing. Finally conclusions are drawn and on-going work is briefly elucidated.

II. COORDINATION AMONG MOBILE SOFTWARE AGENTS

Coordination basically implies the definition of a coordination model and related coordination architecture or related coordination language. In particular, in the context of Agents, an agent coordination model [14] is a conceptual framework which should cover the issues of creation and destruction of agents, communications among agents, and spatial distribution of agents, as well as synchronization and distribution of their actions over time. In this framework, the *coordinables* are the coordinated entities (or agents) whose mutual interaction is ruled by the model, the *coordination media* are the abstractions enabling the interaction among the agents, and the *coordination laws* are the rules governing the interaction among agents through the coordination media as well as the behavior of the coordination media itself. To date, agent coordination models have been classified by using several taxonomies [4**,** 15]; for example they can be classified in control-driven and data-driven according to the taxonomy proposed in [15]. However, in this paper the reference taxonomy is that proposed in [4] as the focus is on agents strongly characterized by mobility. It is worth noting that, although mobility can be an enabling feature for improving efficiency and effectiveness in distributed systems, mobility poses further issues on agent coordination as mobile entities demand for more complex coordination frameworks. The reference taxonomy [4] for Internet-based mobile agent coordination takes these issues into consideration and, in particular, classifies coordination models on the basis of the degrees of spatial and temporal coupling induced by the coordination models themselves. *Spatial coupling* requires that the entities to be coordinated share a common name space or, at least, know the identity of their interaction partners; conversely, *spatial decoupling* allows for anonymous interaction, i.e. there is no need for an acquaintance relationship. *Temporal coupling* implies synchronization of the interacting entities whereas *temporal decoupling* allows for asynchronous interactions [4].

On the basis of the reference taxonomy the following coordination models have been classified: Direct, Meeting-oriented, Blackboard-based and Linda-like.

In *Direct* coordination models, agents usually coordinate using RPC-like primitives or asynchronous message passing. The former coordination method implies temporal and spatial coupling whereas the latter implies only spatial coupling as temporal decoupling can be obtained by adopting message reception queues [16]. The majority of the Java-based mobile agent systems [17], particularly the most famous ones, namely Aglets, Voyager, Ajanta and Grasshoppers, rely on this model.

In *Meeting-oriented* models, agents coordinate using implicit or known meeting points which allow for partial spatial decoupling.

In *Blackboard-based* models, agents coordinate via shared data spaces to store and retrieve information under the form of messages so providing only temporal decoupling.

In *Linda-like* models, agents coordinate through tuple spaces which allow for insertion and retrieval of tuples by using associative pattern-matching; this enables both spatial and temporal decoupling.

Recently new coordination models which can be classified as spatially and temporally decoupled have emerged in the context of Internet applications: the *reactive tuple space* models which enable programmable coordination spaces [18, 19], *transiently shared tuple space* models which handle interactions in the presence of active mobile entities [20], and the *publish/subscribe event-based* models [6**,** 21**,** 22].

The *reactive tuple space* model extends the simple tuple space model by introducing computational capability inside the coordination media under the form of programmable reactions, triggered by operations on the tuple space or by other reactions, which can influence the behavior of agents. This model also allows for the separation of concerns between agent computation and coordination issues.

The *transiently shared tuple space* [20] is another Linda-like coordination model. As Linda offers a static, persistent and globally accessible tuple space, which is scarcely usable in presence of (physical or logical) mobility, the transiently shared tuple space model attempts to deal with these issues. In particular, each mobile agent owns a personal tuple space, named ITS (Interface Tuple Space). Whenever a mobile agent migrates, its ITS is carried with it and merged to the other co-

located agent's ITS making a transiently shared tuple space. Shared means that co-located agents can interact through the merged tuple space and transient means that its contents changes according to agent migrations.

In the *Publish/Subscribe event-based model*, agents coordinate through asynchronous publication and notification of events so enabling temporal and spatial decoupling [6]. In particular, to be notified about a published event an agent has to previously subscribe to the topic/type/context of the published event.

Each of the aforementioned coordination models has some features which make them suitable in given interaction patterns but poorly efficient or not usable at all in other patterns [15]. In [23] the authors proposed the use of multi-paradigm to design heterogeneous applications through different programming paradigms. On the basis of the multi-paradigm approach, a multi-coordination model [10] for the design and implementation of coordination among mobile agents executing in heterogeneous and dynamic distributed systems has been proposed.

III. A MOTIVATING EXAMPLE FOR THE MULTI-COORDINATION

This section proposes a simple yet effective case study which motivates the exploitation of multi-coordination for improving design effectiveness and, notably, system performances. The defined case study concerns with a distributed information retrieval task in a distributed computing system which is carried out through a coordinated set (or task force) of mobile agents. In particular, a user can search for specific information over a network of federated information locations by creating and launching a task force of mobile agents (called *searcher agents*) onto different locations. As soon as the task force finds the desired information, the user is notified with the found information. The proposed solution for the coordination of the task force during its information retrieval task implies that the following coordination requirements (CRs) are to be fulfilled:

- CR_1: every time a *searcher agent* visits a location not yet searched by other agents of the same task force, it notifies the other agents that such location has already been searched so avoiding unnecessary and resource-consuming duplicate searches.
- CR_2: as soon as a searcher agent finds the desired information on a given location, it reports the found information to the user.
- CR_3: when a searcher agent finds the desired information on a given location, it signals such event to all the other searcher agents to stop them.

These coordination requirements (CR_1, CR_2, CR_3) can be respectively designed by the following commonly used mobile agent interaction patterns (LBN, R2O, GBN) [4, 11, 12]:

- *Location-based notification* (LBN), which involves agents passing through a given location to be notified about events occurring/occurred in such location.
- *Report to owner* (R2O), which involves a child agent reporting to its owner agent when its task is completed.
- *Group-based notification* (GBN), which involves an agent notifying all its peer agents when a given event occurs.

These interaction patterns must be effectively implemented by exploiting the most appropriate coordination model/s which can be identified through the following subsequent steps:

1. *Characterization* of the interaction patterns according to appositely defined parameters by taking into account some application-level constraints;
2. *Matching* of the characteristics of the interaction patterns with the intrinsic features of the considered coordination models.
3. *Selection* of the most appropriate coordination model according to specific criteria using the results of the *Matching* step.

The defined parameters for the *Characterization* step of mobile agent interactions are:

- *Number of participants (PN)*, which can assume values in the range [2..N].
- *Participant identity (PI)*, which concerns with the mutual knowledge among interacting agents. PI can therefore assume the values *known* or *unknown*.
- *Locus (L)*, which indicates remote or local interactions among agents. L can assume the values *local* or *remote*.
- *Temporality (T)*, which refers to the type of temporal coupling among interacting agents. T can assume two values: *async* for time decoupling and *sync* for time coupling.

The characterization of the considered interaction patterns is reported in Table 1 in which the PI characteristic of the LBN and GBN cannot be fixed as the agents of a task force may or may not know the identity of each other

TABLE 1. CHARACTERIZATION OF THE INTERACTION PATTERNS

INTERACTION PATTERN	DIMENSIONS			
	PN	*PI*	*L*	*T*
LBN	2..N	UNKNOWN / KNOW	LOCAL	ASYNC
R2O	2	KNOWN	REMOTE	ASYNC
GBN	2..N	UNKNOWN / KNOW	REMOTE	ASYNC

To carry out the *Matching* step, it is needed to characterize the considered coordination models with respect to the characteristics of the interaction patterns to identify what characteristics they are able to intrinsically support. In particular the considered coordination models are the following:

- *Queue–based unicast asynchronous message passing* (QUAMP), which supports a variable number of participants, allows for both local and remote interactions and does not require temporal coupling between participants.
- *Local Linda-like tuple space* (LTS), which supports a high number of participants, allows temporal decoupling but only local interaction is supported.
- *Topic-based publish/subscribe* (TPS), which supports a high number of participants, allows for both local and

remote interactions and does not require temporal coupling between participants.

The *Matching* step intersects the characteristics of the defined interaction patterns with the characteristics supported by the considered coordination models to identify which coordination model is more suitable to implement a given interaction pattern. As the PI characteristic of the LBN and GBN depends on mutual knowledge among agents (the considered application-level constraint), the *Matching* step produces two possible matchings, reported in Tables 2 and 3, which are respectively related to the value assumed by the PI characteristic (*unknown* or *known*).

TABLE 2. CHARACTERISTICS OF INTERACTION PATTERNS WHICH CAN BE DIRECTLY SUPPORTED BY A CORDINATION MODEL (PI=UNKNOWN FOR LBN AND GBN)

IP	CM	Characteristics			
		PN	PI	L	T
LBN	**LTS**	**X**	**X**	**X**	**X**
	TPS	X	X		X
	QUAMP			X	X
R2O	LTS	X			X
	TPS	X		X	X
	QUAMP	**X**	**X**	**X**	**X**
GBN	LTS	X	X		X
	TPS	**X**	**X**	**X**	**X**
	QUAMP			X	X

TABLE 3. CHARACTERISTICS OF INTERACTION PATTERNS WHICH CAN BE DIRECTLY SUPPORTED BY A CORDINATION MODEL (PI=KNOWN FOR LBN AND GBN)

IP	CM	Characteristics			
		PN	PI	L	T
LBN	**LTS**	**X**	**X**	**X**	**X**
	TPS	X	X		X
	QUAMP	**X**	**X**	**X**	**X**
R2O	LTS	X			X
	TPS	X		X	X
	QUAMP	**X**	**X**	**X**	**X**
GBN	LTS	X	X		X
	TPS	**X**	**X**	**X**	**X**
	QUAMP	**X**	**X**	**X**	**X**

The *Selection* step, which allows to choose the coordination model which best supports the characteristics of an interaction pattern, is based on the following selection criterion: the coordination models supporting all the characteristics of an interaction pattern will be the candidate models to implement such interaction pattern.

TABLE 4. DESIGN SPACE FOR PI=UNKNOWN

IP	CM	Implementation description
LBN	LTS	When a searcher agent searches in a location which has not been already searched by another agent of its task force, it inserts (by using the *out* primitive) a signaling tuple into the LTS to signal that this location has been searched. As soon as an agent visits a location and reads the signaling tuple (by using the *non-blocking rd* primitive), it avoids searching.
R2O	QUAMP	When a searcher agent finds the desired information, it sends a message containing the found information (by using the *send* primitive) to its owner.
GBN	TPS	When a searcher agent finds the desired information, it publishes an event of a specific topic related to its task force (by using the *publish* primitive) which signals the stop of the retrieval task. All the other agents of the task force will be thus asynchronously notified since they subscribed to the specific topic at creation time.

According to such criterion the only possible solution if PI=*Unknown* (see Table 2) is represented by the following coordination models: LTS for LBN, QUAMP for R2O and TPS for GBN. An implementation of such solution is reported in Table 4 which constitutes the related *design space*.

Conversely, if PI=*Known* (see Table 3), the coordination models which can be selected are LTS and QUAMP for LBN, QUAMP for R2O, and QUAMP and PS for GBN. The related *design space* which contains the implementations of the interaction patterns through the selected coordination models is reported in Table 5.

TABLE 5. DESIGN SPACE FOR PI=KNOWN

IP	CM	Implementation description
LBN	LTS	**see table 4**
	QUAMP	A searcher agent to notify that it has searched a given location sends a message containing the location identifier (by using the *send* primitive) to all the other searcher agents of the task force.
R2O	QUAMP	**see table 4**
GBN	TPS	**see table 4**
	QUAMP	A searcher agent which has found the desired information sends a notification message (by using the *send* primitive) to all the other searcher agents of the task force to stop them.

The choice of a specific solution among alternative solutions (if any) can depend on different criteria bounded to the values which can be assumed by specific characteristics of the interaction patterns. In particular, this choice can be driven by qualitative considerations or by performance evaluation of the alternative design solutions.

With reference to Table 5, the following qualitative considerations based on the values of the PI characteristics can be considered:

- if the number of participants is very large (PI>>2), the GBN interaction pattern could be better supported by TPS as an agent to notify all the others through TPS always needs to send just one notification whereas the same notification based on QUAMP needs the generation of as many messages as the number of the participants. Thus the use of QUAMP leads to a bottleneck at the agent location both for the agent execution and network performances.
- if the number of participants is small, QUAMP could be a more effective choice as TPS requires a distributed middleware-level infrastructure more complex than that required by QUAMP.

The abovementioned considerations also hold for the LBN interaction pattern.

An example of performance evaluation for driving the choice among alternative design solutions is shown in the next section in which the evaluation and comparison of two possible design solutions based on *multi-coordination* and *message-passing* is presented.

IV. A PERFORMANCE EVALUATION OF THE DESIGNED SOLUTIONS: MULTI-COORDINATION VS. MESSAGE-PASSING

The proposed multi-coordination approach uses a *discrete-event simulation framework* for the evaluation of the designed solutions. The simulation framework provides effective *multi-coordination*-based programming abstractions [24] for the implementation of agent-based systems. In particular, the simulation framework is an enhancement of MASSIMO [25, 26] to support multiple coordination spaces through which agents can interact and currently includes an implementation

of the following coordination spaces:

- The asynchronous Message-based coordination space which is based on proxies [16]. In particular, a message is delivered at the agent home location and, from here, forwarded to the actual agent location by following the chain of proxies left during agent migration.
- The Publish/Subscribe coordination space which behaves like a state-full ELVIN event notification system [6]. In particular, before agent migration the system removes all existing subscription of the migrating agent and re-subscribes the agent to the same notifications after the agent arrives at the new location. Moreover the weight of a notification is less than the weight of a message as no source field of the notification is included.
- The Tuple coordination space which is based on TuCSoN [13]. In particular, each location has its own local tuple space, an instance of a TuCSoN tuple space which relies on text-based tuples.

According to the simulation framework two alternative solutions designed in section III (see Table 5), <LTS, QUAMP, TPS> (or *multi-coordination-based* solution) and <QUAMP, QUAMP, QUAMP> (or *message-passing-based* solution), have been implemented and simulated to calculate the ad-hoc defined performance indices reported in Table 6.

TABLE 6. EVALUATION PERFORMANCE INDICES

Name	Definition
T_{TC}	*Task completion time*: the temporal gap between the spawning of the first created Searcher Agent and the first report message received from the User Agent.
T_N	*Notification time*: the temporal gap between the information finding and the notification to the last Searcher Agent.
N_V	*Number of visits after finding the information*: the total number of locations visited by the Searcher Agents after the information finding.
N_S	*Number of searches after finding the information*: the total number of the locations searched by the Searcher Agents after the information finding.
N_M	*Number of coordination messages*: the number of coordination messages transmitted through the network.

The simulation tests rely on the simulation parameters (the number of locations and the number of *searcher agents*) and on the following settings of the simulation topology at network and information level:

- locations are connected through a fully connected logical network composed of FIFO channels. In particular, channels are characterized by the same delay and bandwidth parameters modeled as uniform random variables.
- the information to be found is contained exactly at one location and the locations keep references (randomly generated) to other locations at information level to be all reachable.

Simulations were carried out with the number of locations equals to 100 and the number of *searcher agents* in the range [10..90, step=10]. Moreover, for each simulation run, the *multi-coordination-based* and *message-passing-based* solutions were executed on the same network and information topologies. In Figures 1-5 the simulation results are reported; the obtained values of the performance indices were averaged over 100 simulation runs.

The T_{TC} performance index, which measures the speed with which the information search task is carried out, decreases as the number of searcher agents increases (see Figure 1). In fact, the use of more searcher agents augments the degree of parallelism which, consequently, increases the probability to find the searched information with a smaller number of migrations which are time-consuming. The performances of the *message-passing-based* and *multi-coordination-based* solutions are almost the same.

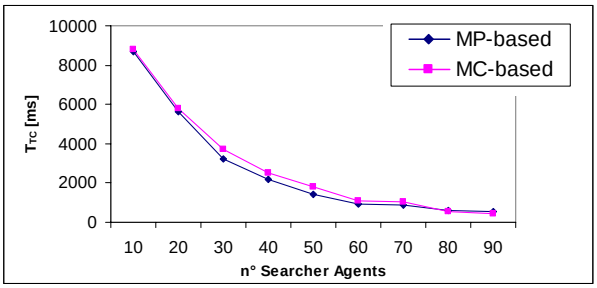


Figure 1: Task completion time

The T_N performance index measures how fast all the *searcher agents* are notified after finding the information. The shorter T_N, the fewer are the resources consumed throughout the networked agent platform. The *multi-coordination-based* solution performs better than the *message-based-solution* when the number of searcher agents is less than 80 (see Figure 2) due to (i) the exploitation of the TPS coordination space which provides faster notifications than the message-based coordination space and (ii) the network load which is lighter than the one obtained in the *message-passing-based* solution (see discussion about the N_M parameter). However, when the number of agents is greater than 80 the *message-passing-based* solution performs better as it avoids the occurrence of many migrations which could slow down the *stop* notification of agents. In fact, when the LBN interaction pattern is carried out through LTS, agents should migrate to a location to understand if such location has been searched. Conversely, when the LBN interaction pattern is carried out through QUAMP, agents send messages to notify a searched location to the others so limiting the number of migrations per agent as the agents are notified without having to migrate to new locations.

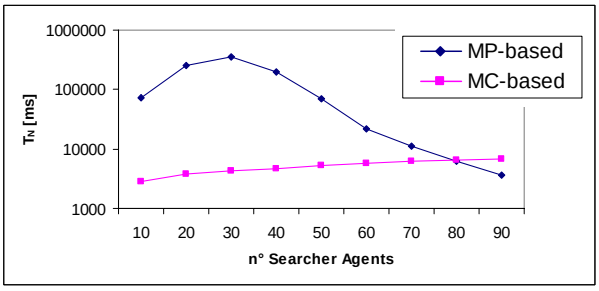


Figure 2: The notification time.

The N_V and N_S parameters are measures of the consumption of resources after the information is found. The values of such parameters should be kept as low as possible. As shown in Figures 3 and 4, the *multi-coordination-based* solution outperforms the *message-passing-based* solution when the number of searcher agents is less than or equal to 40.

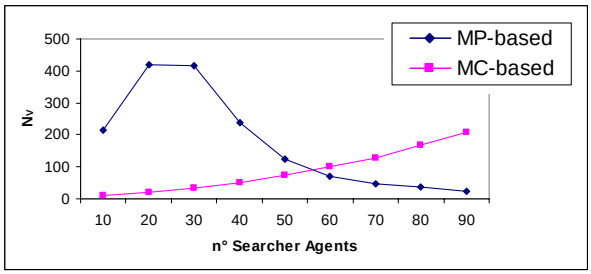


Figure 3: Number of visits after finding information

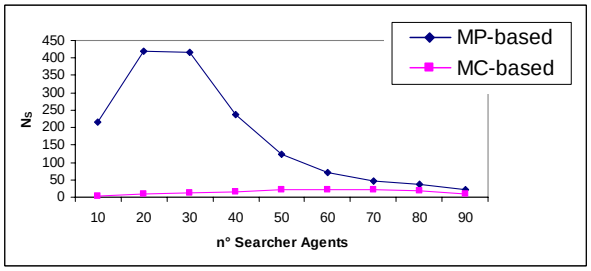


Figure 4: Number of searches after finding information

Finally the N_M parameter (see Figure 5), which measures the network load, is significantly better in the *multi-coordination-based* solution thus saving network resources with respect to the *message-passing-based* solution.

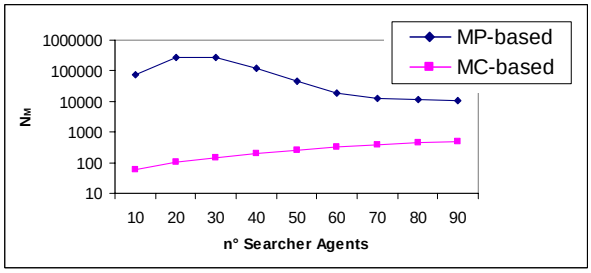


Figure 5: Number of coordination messages

Finally, it is worth noting that a network of locations cannot be flooded by a lot of agents per searching task which would cause an over usage of network resources, even though a numerous task force of agents would significantly decrease the T_{TC} as shown in Figure 1. So a trade-off should be reached in terms of the number of agents constituting the task force which should be appositely set to a percentage of the number of available locations to minimize the resource usage and obtain low task completion times. According to the obtained results (see Figures 1-5) this percentage can be set to 40% for the *multi-coordination-based* solution which is a good trade-off and implies that a task force of 40 agents is created and launched for each information retrieval task.

V. Conclusion

This paper has proposed a multi-coordination approach for the design and evaluation of mobile agent interactions which is based on two subsequent phases: (i) the defined coordination requirements among agents are designed through well-known agent interaction patterns which are then implemented by using specific coordination models according to a three-step procedure which provides alternative design solutions; (ii) these alternative design solutions are evaluated and compared through an agent-oriented discrete-event simulation framework according to ad-hoc defined parameters.

The proposed approach has been applied to a simple yet effective case study which has highlighted its actual applicability and that the exploitation of *multi-coordination* could be both more effective and more efficient than the use of a message-based coordination model.

On the basis of the obtained results work is underway for: (i) testing the proposed three step technique with a wide variety of coordination requirements, agent interaction patterns and coordination models; (ii) relaxing the selection criterion proposed in section III to also consider other solutions which can be implemented by mixing a coordination model with mobility and third-party agent components (e.g. reflectors, mediator, facilitator, etc); (iii) enhancing the simulation framework to include other coordination spaces.

References

[1] A. Fuggetta, G.P. Picco, and G. Vigna, “Understanding Code Mobility”, *IEEE Trans. on Software Engineering*, 24(5), pp. 342-361, 1998.

[2] D.B. Lange and M. Oshima, “Seven good reasons for Mobile Agents”, *Communications of the ACM*, 42, 3, pp 88-89, 1999.

[3] G. Vigna, “Mobile Agents: Ten Reasons For Failure”, *Proceedings of the 2004 IEEE International Conference on Mobile Data Management* (MDM’04), Berkeley, CA, USA, 19-22 January 2004.

[4] G. Cabri, L. Leonardi and F. Zambonelli, “Mobile-agent coordination models for internet applications”, *IEEE Computer*, *33*, *2*, pp 82-89, 2000.

[5] A. Murphy, G. P. Picco, and G. Roman, “LIME: A Middleware for Logical and Physical Mobility”, *Proceeding of 21th International Conference on Distributed Computing Systems*”, IEEE CS, 2001.

[6] A. Padovitz, “Agent communication using Publish-Subscribe genre: Architecture, Mobility, Scalability and Applications”, *Annals of Mathematics, Computing and Teleinformatics*, 1, 3, pp 35-50, 2004.

[7] J. Baumann, F. Hohl, N. Radouniklis, K. Rothermel and M. Strasser, “Communication concepts for Mobile Agent Systems”, *Proceeding of the 1st International Workshop on Mobile Agents (MA’97)*, Berlin, Germany, LNCS 1219, pp. 123-135, April 1997.

[8] S. Choi, H. Kim, E. Byun, C. Hwang, and M. Baik, "Reliable Asynchronous Message Delivery for Mobile Agents", In *IEEE Internet Computing*, vol. 10, no. 6, pp. 16-25, 2006.

[9] J. Cao, X. Feng and S.K. Das, "Mailbox-Based Scheme for Mobile Agent Communications", *Computer 35(9)*, pp. 54–60, 2002.

[10] G. Fortino and W. Russo, “Multi-coordination of Mobile Agents: a Model and a Component-based Architecture”, *Proceedings of 20th Annual ACM Symposium on Applied Computing (SAC’05), Special Track on Coordination Models, Languages and Applications*, Santa Fe, NM, USA, Mar. 13-17, 2005.

[11] Y. Aridor, D.B. Lange, “Agent Design Patterns: Elements of Agent Application Design”, *Proceedings of Autonomous Agent ’98*, Minneapolis, Minnesota, US, 1998.

[12] D. Deugo, M. Weiss, and E. Kendall, “Reusable Patterns for Agent Coordination” published as Chapter 14 in the book: Omicini, A.,

Zambonelli, F., Klusch, M., and Tolksdorf, R. (eds.), Coordination of Internet Agents: Models, Technologies, and Applications, Springer, 2001.

[13] A.Omicini and F. Zambonelli, "Coordination of Mobile Agents for Information Systems: the TuCSoN Model", *Proceeding of 6th AI*IA Convention*, 1998.

[14] P. Ciancarini, "Coordination models and languages as software integrators", *ACM Computing Surveys*, 28, 2, pp 300-302, Jun 1996.

[15] G.A. Papadoupolos, F. Arbab, "Coordination models and languages", In *Advances in Computers 46*, Academic Press, 1998.

[16] X.Y. Zhou, N. Arnason and S.A. Ehikioya, "A proxy-based communication protocol for mobile agents: protocols and performance", *IEEE Conference on Cybernetics and Intelligent Systems*, volume 1, pp 53-58, 1-3, Dec. 2004.

[17] A.R. Silva, A. Romao, D. Deugo and M. Mira da Silva, "Towards a reference model for surveying mobile agent systems", *Autonomous Agent and Multi-Agent Systems*, 4 (3), pp 187-231, 2001.

[18] G. Cabri, L. Leonardi and F. Zambonelli, "Engineering Mobile Agent Applications via Context-dependent Coordination", *IEEE Transactions on Software Engineering, 28, 11*, pp 1040-1056, Nov. 2002.

[19] A. Omicini, and F. Zambonelli, "Tuple centres for the coordination of internet agents", *Proceedings of ACM Symp. on Applied Computing (SAC'99), Special track on Coordination Models, Languages and Applications*, San Antonio, TX, USA, Feb 28-Mar 2, 1999.

[20] G. P. Picco, A. L. Murphy and G. C. Roman, "LIME: Linda meets mobility", 1999

[21] G. Cugola, E. Di Nitto and A. Fuggetta, "The Jedi event-based infrastructure and its application to the development of the OPSS WFMS", *IEEE Transactions on Software Engineering, 27, 9*, pp 827-850, 2001.

[22] A. Carzaniga, D.S. Rosenblum and A. Wolf, "Design and evaluation of a wide-area event notification service", *ACM Transactions on Computer Systems, 19, 3*, pp 332-383, 2001.

[23] P. Zave, "A compositional approach to MultiParadigm Programming", *IEEE Software 6(5)*, pp 15-25, 1989.

[24] G.Fortino, A. Garro, S. Mascillaro and W. Russo, "Modeling Multi-Agent Systems through Event-driven Lightweight DSC-based Agents", *Proceedings of 6th International Workshop From Agent Theory to Agent Implementation (AT2AI'06)*, May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

[25] M. Cossentino, G. Fortino, A. Garro, S. Mascillaro, and W. Russo, "PASSIM: a simulation-based process for the development of multi-agent systems", *International. Journal on Agent-Oriented Software Engineering* 2(2), 132-170, 2008.

[26] G. Fortino, A. Garro, and W. Russo, "A Discrete-Event Simulation Framework for the Validation of Agent-based and Multi-Agent Systems", *Proceedings. of the Workshop on Objects and Agents (WOA'05)*, Camerino, Italy, Nov 14-16, 2005.

Author Index