Matteo Baldoni, Antonio Boccalatte, Flavio De Paoli,
Maurizio Martelli, Viviana Mascardi (eds.)

Dagli Oggetti agli Agenti

Agenti e Industria:
Applicazioni tecnologiche degli agenti software

Ottava Edizione, WOA 2007
Genova, 24 e 25 Settembre 2007
Atti del Convegno

Seneca Edizioni, ISBN 978-88-6122-061-4

WOA 2007 Home Page:
`http://woa07.disi.unige.it/`

Prefazione

Il workshop "WOA dagli Oggetti agli Agenti" rappresenta da ormai otto anni una consueta occasione di scambio di idee per tutti i ricercatori che operano nell'ambito dei sistemi ad agenti.

La tecnologia ad agenti sta assumendo un ruolo centrale non solo nel settore dell'intelligenza artificiale, ma anche in settori più tradizionali dell'informatica quali l'ingegneria del software e i linguaggi di programmazione, dove il concetto di agente viene considerato una naturale estensione di quello di oggetto. L'importanza di tecniche orientate agli agenti è dimostrata anche in ambito industriale dall'interesse per il loro utilizzo nella realizzazione di strumenti e applicazioni in molteplici aree. L'edizione di WOA di quest'anno, patrocinata dall'Associazione Italiana per l'Intelligenza Artificiale (AI*IA) e dall'Associazione Italiana Tecnologie Avanzate Basate su concetti Orientati ad Oggetti (TABOO), è infatti dedicata alle applicazioni delle tecnologie ad agenti in ambito industriale ed alle possibilità future che l'industria vede dell'utilizzo degli agenti.

Questo volume contiene sedici articoli scelti dal Comitato di Programma per essere presentati al workshop e quattro articoli che descrivono altrettanti prototipi mostrati nella sessione demo. Il volume contiene anche l'articolo invitato di Giovanni Rimassa, Whitestein Technologies AG, Svizzera, e Birgit Burmeister, DaimlerChrysler AG, Germania, su un approccio alla gestione dei processi di business che sfrutta la tecnologia ad agenti per ottenere un comportamento agile. Come di consueto il workshop sarà preceduto da una miniscuola per studenti di dottorato e laureandi.

I contributi degli articoli coprono aree di ricerca estremamente attuali che includono modelli logici per sistemi ad agenti, tecnologie ed approcci per il recupero di risorse guidato dalla semantica, applicazioni di agenti e sistemi multiagente, agenti situati nell'ambiente, ruoli e fiducia tra agenti, agenti ed approcci orientati ai servizi.

Il Comitato Scientifico Organizzatore ringrazia sentitamente tutti coloro che, con il loro lavoro ed il loro entusiasmo, hanno contribuito al successo di questa ottava edizione di WOA: i componenti del Comitato di Programma, il Dipartimento di Informatica, Sistemistica e Telematica (DIST) ed il Dipartimento di Informatica e Scienze dell'Informazione (DISI) dell'Università degli Studi di Genova, gli organizzatori locali, gli organizzatori della sessione demo e della miniscuola, gli sponsor e tutti collaboratori che hanno partecipato all'organizzazione.

10 Settembre 2007

Matteo Baldoni
Antonio Boccalatte
Flavio De Paoli
Maurizio Martelli
Viviana Mascardi

Comitato Scientifico Organizzatore

Matteo Baldoni, Università di Torino
Antonio Boccalatte, Università di Genova
Flavio De Paoli, Università di Milano - Bicocca
Maurizio Martelli, Università di Genova
Viviana Mascardi, Università di Genova

Organizzazione Sessione Demo

Alfredo Garro, Università della Calabria

Coordinatore Miniscuola

Matteo Baldoni, Università di Torino

Comitato Organizzatore Locale

Antonio Boccalatte, Università di Genova
Giovanni Casella, Università di Genova
Alberto Grosso, Università di Genova
Maurizio Martelli, Università di Genova
Viviana Mascardi, Università di Genova
Andrea Passadore, Università di Genova
Christian Vecchiola, Università di Genova

Comitato di Programma

Stefania Bandini, Università di Milano - Bicocca
Cristina Baroglio, Università di Torino
Pietro Baroni, Università di Brescia
Federico Bergenti, Università di Parma
Lorenzo Bettini, Università di Firenze
Enrico Blanzieri, Università di Trento
Paolo Bouquet, Università di Trento e IRST
Nadia Busi, Università di Bologna
Giacomo Cabri, Università di Modena e Reggio Emilia
Nicola Cannata, Università di Camerino
Massimo Cossentino, ICAR-CNR, Palermo

Francesco Donini, Università di Tuscia - Viterbo
Rino Falcone, ISTC-CNR, Roma
Giancarlo Fortino, Università della Calabria
Alfredo Garro, Università della Calabria
Laura Giordano, Università del Piemonte Orientale
Paolo Giorgini, Università di Trento
Letizia Leonardi, Università di Modena e Reggio Emilia
Marco Mamei, Università di Modena e Reggio Emilia
Sara Manzoni, Università di Milano - Bicocca
Rebecca Montanari, Università di Bologna
Massimo Paolucci, Università di Genova
Paola Quaglia, Università di Trento
Alessandro Ricci, Università di Bologna - Cesena
Giovanni Rimassa, Whitestein Tech, Zurich, Svizzera
Corrado Santoro, Università di Catania
Carla Simone, Università di Milano - Bicocca
Emilio Tuosto, University of Leicester, Regno Unito
Eloisa Vargiu, Università di Cagliari
Mario Verdicchio, Universit di Bergamo
Mirko Viroli, Università di Bologna - Cesena
Giuseppe Vizzari, Università di Milano - Bicocca

Revisori addizionali

Aliaksandr Birukou	Leonardo Vito	Hugo Andres Lopez
Francesca Piersigilli	Stefano Schivo	Michele Loreti

Direttivo WOA

Giuliano Armano Università di Cagliari
Matteo Baldoni Università di Torino
Antonio Corradi Università di Bologna
Flavio De Paoli Università di Milano - Bicocca
Emanuela Merelli Università di Camerino
Andrea Omicini Università di Bologna - Cesena
Agostino Poggi Università di Parma
Franco Zambonelli Università di Modena e Reggio Emilia

Sponsor

Indice dei Contenuti

Articolo invitato

Prototipi di Sistemi Multiagente (Sessione Demo)

Modelli Logici per Sistemi ad Agenti

Tecnologie ed Approcci per il Recupero di Risorse Guidato dalla Semantica

Applicazioni di agenti e Sistemi Multiagente

Agenti Situati nell'Ambiente

Ruoli e Fiducia tra Agenti

Agenti ed Approcci Orientati ai Servizi

Achieving Business Process Agility in Engineering Change Management with Agent Technology

Giovanni Rimassa, Birgit Burmeister.

***Abstract*— The importance of business processes for a successful enterprise cannot be overestimated. They are core assets through which a business turns its potential into actual competitiveness on the market. To face the challenges posed by today's changing and uncertain business environment, traditional BPM approaches are not sufficient anymore. This paper presents an approach to business process management, which leverages Agent Technology features to obtain agile business process behavior. Beyond the problem and solution description, this work presents a concrete case study in the domain of Engineering Change Management..**

***Index Terms*— agents, business process management, engineering change management.**

I. Introduction

Business processes are a fundamental component of any enterprise across all kinds of industries. Their effective setup, execution and evolution are of paramount importance to successful business operations. By definition business processes consist of a set of activities, connected in a structured whole. They describe the modes of operation of a business organization in given situations and their importance is manifold:

They constitute the *organizational knowledge* of the enterprise. The ways of operating that are captured by business processes belong to the organization, and they are made public and explicit in the face of personnel turnover and growth.

They gather and structure the *identity of the enterprise*, express its specific way to conduct business and are often the key to realizing an organization's competitive advantage. Moreover, they explicitly represent current organizational setup and are amenable to assessment and *continuous improvement*.

The whole set of activities that an organization performs in order to create, maintain, control and evolve its business processes is named *Business Process Management* (*BPM* for short). BPM is an approach to administering business processes that involves people, organizations and technologies. In addition, BPM can be carried out with varying levels of automation.

The trend toward more flexible ways of working, shorter organizational reaction times and fully embracing market and business unpredictability, along with the increase in distribution and the need to preserve understandability despite more and more complexity, characterizes the past years and shows no signs of abating.

We believe that in the face of the challenges present in today's dynamic business environments, BPM falls short of what it is commonly intended to achieve. This paper presents *agile business process management* as an effective approach to the challenges mentioned above. Moreover, the role and contribution of Agent Technology is analyzed, and the application to a concrete case is presented.

This paper is organized as follows. Section II presents the problem of achieving agility in BPM, with particular reference to the domain of *Engineering Change Management* (*ECM* for short). Section III presents the role played by Agent Technology in conceiving and realizing a solution for the problem of agile BPM. Sections IV and V illustrate the major concept of the solution, namely *goal-oriented and autonomic BPM*. Lastly, Section VI introduces the concrete application of the approach.

II. Problem Definition

Today any development project in the automotive industry has to be supported by a powerful engineering change management. The increasing complexity of the product, the shortening of time-to-market and the growing dependencies on the suppliers increase the number and the complexity of change requests in all phases of the product development. Therefore an efficient management of product changes is an important success factor.

Compared to typical business processes, e.g. in call centers or financial services, managing engineering processes is even more challenging: Engineering processes are long running tasks. Constructing a car lasts for many years. During this time period many things change – what has been an up-to-date approach in the beginning may be outdated at the end. Also, engineering processes have to cope with uncertainty because of their mixture of creative tasks, collaborative work and repeating activities. This results in very complex processes

Giovanni Rimassa is with Whitestein Technologies AG, Advanced Technologies, Zurich, 8032 Switzerland (e-mail: gri@whitestein.com).

Birgit Burmeister is with DaimlerChrysler AG, Group Research, 71034 Böblingen, Germany (e-mail: birgit.burmeister@daimlerchrysler.com).

with many alternative paths and sections that cannot be planned in advance.

Traditionally, BPM systems have been developed based on a mind model of business processes as process chains or task chains. Changes, uncertainty, and hidden processes are seen (and sometimes handled) as exceptions instead as regular events. Hence, support for the special demand of engineering processes is limited [2]. Adequate support for engineering processes in terms of modeling and execution obviously requires a completely new approach for process management that is able to deal with the requirements for flexibility, transparency, and efficiency, both in design and execution of the process.

A. Achieving Business Agility

A new modelling approach to enable agile processes has to

- Support the design of huge, complex processes, by using a modular process model but also allowing for an overall picture of the process.
- Decrease the effort for changing and maintaining the process model.
- Allow flexibility and agility not only in process modelling but also in process execution through software systems.

We think that agent technology can offer approaches and methods to meet these requirements. Agent-oriented software technology was first introduced to deal with large-scale, distributed software systems, which are embedded in dynamic environments, and allow for the interaction of different partners. The term "agent" is used as a name for an autonomous software component, which is able to deal with the dynamic environment and may interact with other agents [3].

Inspired by agent technology and especially by the concepts of goal orientation and decomposition the research department of Daimler has developed the idea of a goal- and context-oriented business process modelling. The main ideas of this approach are (i) to have a modular process model that describes the single steps of a process (sub-processes, activities) separate from the goals of the process and the different contexts in which the process can be executed; (ii) to have different modelling levels, for the different parts of the process model; and (iii) to have a seamless "translation" of the process model into process execution. This modular, goal- and context-based process model can then be directly executed as an agile process, by considering current goal and context when determining the next step in the process, just as realized in the BDI agent architecture (see section III). For details see [4].

B. The ACM Project

The feasibility of the sketched goal- and context-oriented modelling approach was first shown in a software demonstrator implemented by Daimler Group Research and applied to the area of engineering change management. This demonstrator used the JadeX agent tool as the process execution engine. JadeX is implemented by the University of Hamburg enhancing the Jade platform with a BDI agent architecture [5], see also next section.

III. The Role of Agent Technology

As stated earlier agent technology is a specific approach to software engineering. A system is composed of a number of agents being autonomous in their behavior and interacting with each other to achieve the desired overall functionality. A specific architecture of an agent is the so-called BDI-agent. A BDI- agent is described by its Beliefs, i.e. the information an agent has about itself, its environment and possibly other agents; its Desires, i.e. motivations of the agents that drive its course of action; and finally its Intentions; i.e. the short-term goals that the agent wants to achieve, derived from its desires and external events, to which the agents wants to react. Additionally an agent has certain plans how the intentions/goals can be achieved. A plan consists of certain actions/steps that have to be executed to achieve the corresponding goal.

The BDI architecture was first implemented by [6]. The execution of the formal framework sketched above is as follows: The activities of an agent can be described as a permanent jump between two different types of actions: on the one hand the execution of basic tasks, which the agent uses to fulfill currently active goals ("execution activity"), and on the other hand the reasoning about the next basic action, which he will execute ("control activity"). Execution activities can be interacting with the environment, e.g. with the user of the system, performing some kind of computation, manipulating the agent's own data base (belief base), and sending and receiving messages to and from other agents.

A control activity results in the choice of an execution activity, which will be performed next. To find out which activity to execute next, the agent introspects its goal base, the set of possible execution activities and the belief base. From the goal base it extracts the goals, which are not yet fulfilled. Then it collects all plans, which could be used to fulfill these goals. Next, it checks which of the plans could be performed, by checking the current context (i.e. the current belief base) whether it fits to the context the plan was designed for. Different plans are designed for different contexts, which is described in the so-called context condition of the plan. Thus the agent has to drop all plans that would fulfill a goal, but only in another context. Among the remaining plans he chooses now the one he will execute next (see Figure III-1). The single steps of the plan are then executed as defined in the plan.

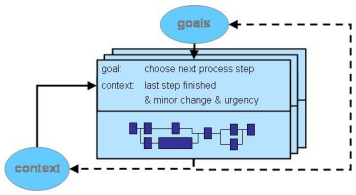


Figure III-1. Choosing and Executing of Plans

The BDI architecture is well-established agent architecture with several agent tools and applications supporting the architecture. Georgeff also used the ideas of the BDI architecture for business process modeling and management in the Agentis platform [7].

Based on the ideas of goal-oriented and context-aware execution of agent plans, and of using it for business process modeling and execution, we have enhanced the ideas for a new form of business process modeling.

IV. GOAL-ORIENTED BPM

In day-to-day management operations, it is natural to set goals, decompose a goal into sub-goals, define or reuse plans, and routinely track and check the execution of chosen plans in order to detect problems as they occur (or even better before they do), and to take appropriate actions.

On the other hand, today's dominant IT approaches focus almost exclusively on procedures. The concept of what the procedure is meant to achieve, and why, typically remains implicit in the mind of the humans who designed it. Because of this, the increase in process management automation that occurred with BPM systems has also shifted the focus away from goals and plans and toward procedures.

The limiting consequence is that processes have become more efficient in execution but less flexible in adaptation. To maintain effectiveness without sacrificing agility, the concept of plan and goal must be brought back to center stage in BPM solutions.

A. Plans and Goals to Express Processes

Using a goal-oriented approach separates the statement of what the desired system behavior is, from the possible ways to perform such behavior. More precisely:

The desired result is described by *achievement conditions* to make true and as *maintenance invariants* whose violation must be avoided.

The possible ways to obtain a result are represented by plans: process graphs decorated with the conditions where they are applicable and the results they obtain when successful.

In business organizations there is an upper management level, which coarsely drives the more detailed project planning and tracking. Such a level gives clear direction without unnecessarily limiting the decisional power and the adaptation leeway of the finer-grained management operations.

It is thus natural for upper managers to be more concerned with (and express their views in terms of) what is to be achieved than how to achieve it. Operating at the goal level is a natural approach for such people with the core of the business process captured through goals and sub-goals independently of the actual activities.

When moving to detailed planning in business or project management, there is usually more to the plan than just its tasks and structure. At the very least, the expected objectives of the plan need be stated, and also, in many cases, the initial requirements. Moreover, additional information such as resource and time consumption is also often attached to a plan.

To effectively tackle challenges at the organizational level, management agility has become a strategy of choice; perhaps one of the most decisive weapons in the day-to-day business world.

B. Keeping the Goal Level Alive

The procedural nature of computers and software must not cripple the management processes just described. In particular, a detailed, explicitly directive process specification that identifies precisely what to do in each and every envisaged variation, e.g., a BPEL execution engine, allows agility only up to a certain level.

In fact the more complex and unpredictable the situation, the more convoluted an automated directive process may become. This most often results in brittle behavior specifications that become progressively harder to extend, change and test.

To move forward and enable a BPM system to support the management of complex processes, or execute in a dynamic and unpredictable environment, both an explicit representation and a clear separation of goal and plan levels are essential.

In principle, the steps to perform goal-oriented business process modeling are:

- Expressing the intentions and requirements of the process through goals and sub-goals, connected as necessary ("*Think the end first!*").
- Organizing processes into plans by attaching them the statement of what they require and what they achieve, and grouping them when they achieve the same goals in different ways.
- Decomposing processes into tasks, specifying their aggregation structure.

Once these steps are taken, business processes in an enterprise can be modeled as a set of related goals to be achieved or maintained. One or many plans are attached to these goals, and each plan has its own feasibility requirements. Attributes such as expected completion time or resource cost can also be associated to plans.

Adopting goal-oriented BPM results in several benefits, such as:

- *Business user empowerment*. Users can work at the goal level, expressing what is to be achieved as the defining core of the business process. The details can be left out of this essential picture.
- *Increased process understandability*. The goal level alone already shows what the business

process is supposed to achieve (main goal) and which are the fundamental milestones (intermediate goals). This increased understandability is also leveraged to acquire visibility of the whole process even across organizational boundaries.

- *Improved process tracking and monitoring*. The goal level allows tracking of business process evolution independently of operational details. If needed, the structure at the plan level, together with plan attributes such as cost and time, allows the continuous fine assessment of the current state of the work.
- *Encapsulation of tactics*. The set of plans attached to a given goal represents a collection of different tactics. The details of these tactics do not spark dependency chains across involved systems.
- *Lowered maintenance costs*. The widespread use of declarative specification reduces the dependence on details and makes the business process models, and their implementation, both more stable and easier to change. Moreover, plans can be reused and combined to more efficiently deal with process goals.

C. The GO-BPMN Language

The ideas of goal-oriented BPM are supported at the modeling and execution level by the *Goal-Oriented Business Process Modeling Notation* (GO-BPMN) for modeling processes. GO-BPMN is a visual modeling language for the specification of business processes, enriching BPMN by the explicit modeling of goals, plans and their relationships. Moreover, GO-BPMN precisely specifies the operational semantics of all its elements, including the used standard BPMN ones, so that compliant and unambiguous model execution can be obtained.

A GO-BPMN model explicitly contains elements such as:

- *Achieve goals*. They represent overall or intermediate goals that the system will try to bring about. These goals become active when some context condition is true. Achieve goals are arranged into hierarchies with a decomposition relation.
- *Maintain goals*. These goals are used to describe safety conditions that have to be verified at all times. Whenever one of such conditions is negated, a compensation plan is automatically scheduled.
- *Plans*. They are attached to goals and contain as body a BPMN-compliant activity. Moreover, a plan has a context condition that tells in which situations it can be executed

The Figure IV-1 shows a small sample of a GO-BPMN diagram.

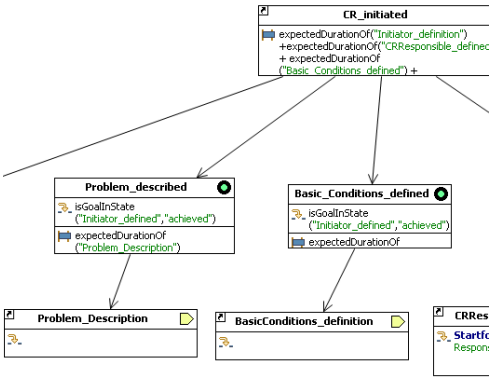


Figure IV-1. Goal-oriented modeling with GO-BPMN

V. Autonomic BPM

Effectively managing complex business processes in the face of a dynamic and unpredictable environment requires striking a careful balance between flexibility and safety.

During operation, the definition and execution of business processes has to be easily adapted to unforeseen changes. It must also be possible to ensure that these changes are correct and do not incur any unfavorable consequences.

While some human inspection tools can and should be provided, it is unreasonable to expect that all safety controls can be handled manually. This would simply void most of the improvements in timeliness and adaptivity gained with an increased level of automation.

The only way out of this is to provide the system with means of *self-management*. This implies that the system itself (i.e., the business process management engine) is able to monitor its own operation and, to a certain extent, recognize and counteract undesirable situations. Following Autonomic Computing terminology, one can mention the major facets of autonomic, self-managing systems:

- *Self-healing*. The system is able to recover from unfavorable conditions that may result in malfunctions, by autonomously attempting to determine compensation actions and then performing them.
- *Self-optimization*. The system continuously assesses its own performance, explores possible courses of actions that would result in performance improvements, and adopts the ones that are sufficiently promising.
- *Self-protection*. The system detects threats and puts in place preventive and corrective measures to ensure correct operation even in the face of these threats.
- *Self-configuration*. The system is able to change its operating parameters to adapt to mutable external conditions, some of which may even be

unpredictable at system design time.

A. From Autonomic IT to Autonomic BPM

The original focus of Autonomic Computing was on IT infrastructure with the targeted problem being the administration and management of complex computing environments. Nevertheless, the basic idea and the primary concepts of Autonomic Computing apply to most systems and even to organizational entities. Introducing self-management properties into applications can yield significant benefits.

Both for infrastructure and applications, a key to the Autonomic Computing vision is the presence of feedback control loops in the system. In principle, a system exhibiting autonomic self-management can be divided into:

- *A base system*, providing concrete functionality that is required to meet the system design goals.
- *An autonomic controller*, monitoring the base system and the external environment, and deciding and enacting self-management policies.

When the base system is not simply a software application, but a whole business process management system, the addition of an autonomic controller results in *Autonomic Business Process Management*.

In autonomic BPM, the "system" is the overall ensemble of software, hardware, human and physical resources, together with the norms and policies defining it. This system is the one that exhibits self-management and in particular the self-management qualities.

The benefits of the autonomic BPM approach result from the effect of self-management at various levels, such as self-healing of process activities through alternative backup tactics, or self-optimization by automatically detecting feasible remedies and proposing reasonable options to a human for selection.

In general, the above benefits can be summed up in two broad cases:

- *Self-management at the process level*. This means that the definition and enactment of the business process itself have some or all the self-management properties. There can be, e.g., special control processes that are added to the base processes. The way people are involved within the business process also has some autonomic traits.
- *Self-management at the engine level*. This means that the BPM runtime environment has self-management built into it. The BPM engine exhibits self-healing, self-optimization and other similar features.

VI. Applying Goal-oriented Autonomic BPM

The two technological traits of goal-oriented and autonomic BPM, previously described in Section IV and Section V, are leveraged by the ACM system in a practical way.

A. The ACM System Architecture

The Figure VI-1 depicts the architecture of the ACM system. There the division into Presentation, Logic and Data tiers is visible. This approach represents a standard solution more and more adopted along the past ten years, and found in many installations today.

Each tier is well separated from the others and the presentation tier does not communicate at all with the data tier. The logic tier connects the presentation and the data tiers, and it is also where the complex application behavior is defined. Therefore, a lot of infrastructural issues such as communication protocols, security and resource control belong to the middle, logic tier.

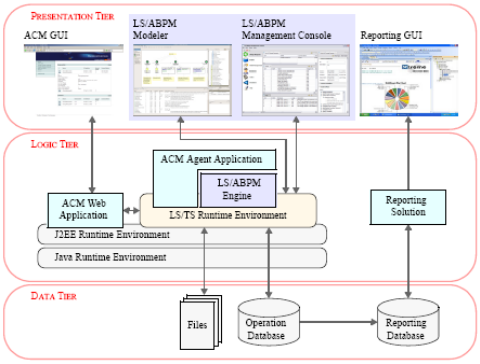


Figure VI-1. Overall architecture of the ACM system

Instead of taking care of all these complex issues within the application, a popular and more effective approach is to rely on a support layer (called middleware, broker, or application server in different situations) that provides them. This is also shown, e.g., by blocks such as J2EE Runtime Environment and is another major best practice in modern business application development.

All the above configures a state-of-the-art architecture, but it is still not enough to cope with the agility requirement and its consequences that sit at the heart of the improvement expected from the ACM system.

While the confinement of business logic to the middle tier and the reliance on a middleware are kept, a major step forward is taken when choosing the component model for the ACM application.

The chosen component model is a major way that Agent Technology contributes to the system concept and features. In particular, two kinds of components will execute within the ACM system:

- *Agents*. Autonomous and situated software components, capable of proactive and reactive behavior.
- *Services*. Non-autonomous software components, which are only capable of reactive behavior.

The interaction between two or more agents is based on asynchronous message passing, whereas the interaction between an agent and a service relies on the abstraction of operation invocation (which can itself be synchronous or asynchronous).

Both the agent-to-agent messaging and the agent-to-service invocation adopt a structured data model to express the topic of the interaction. Such topics are gathered in one or more ontologies, which are then made available at run-time providing advanced introspection on all the aspects of system behavior and give first-class status to entities such as resources, protocols, and organizations.

Multi-agent systems provide the right set of concepts to express and realize the loosely coupled, dynamically assembled and highly reflective architecture that can grant the desired agility and prompt adaptation to changes.

Moreover, in order to still achieve a satisfactory quality level, with particular respect to the non-functional qualities of software, such as modifiability, reliability and dependability, the approach of having multiple layered execution environments is followed.

The major point of the approach is to define a series of layers of abstraction, from the lower, more basic ones to the upper, more abstract ones. Each of these layers has two parts:

- *Execution environment*, which is fixed and sets the boundaries that define the abstraction layer.
- *Execution specification*, which can be changed, and defines the system behavior within the boundaries of the execution environment.

The execution environment acts as a containment envelope for the execution specification, preventing it from affecting other layers of the system.

This layering allows striking a good balance between flexibility and safety in modifying the system. Different users, with different skills and focuses, can operate at one or more of these abstraction levels. Concretely, in the ACM system these are:

- Final business process models described with the GO-BPMN graphical executable modeling language, for highest abstraction and safety. Both business and IT modelers can use the language effectively.
- GO-BPMN reusable modules, containing parametric goals and plan sets to be configured for application in several process models.
- High level scripting language to quickly express more complex behavior, within a well isolated programming environment, providing an easy to use API for most of the system functionalities.
- Java API to implement the core parts such as atomic tasks or intermediate service components (e.g., for specialized system integration needs).

B. *Process Modelling with the ACM System*

As described in [4] first experiences with the goal-oriented modeling approach where made during modeling the ECM process for the research demonstrator. It proved useful to concentrate on the "what should the process achieve", i.e. the goals with process analysts. When talking to IT-people about "how it should be done", the concrete sub-processes and all the detailed context specifications could be modeled. As a result a goal-hierarchy of the ECM process was built up. The first challenge in the ACM project was to translate this goal- and context-oriented model of the ECM process into the Whitestein platform. Since the common underlying agent technology, with the BDI agent execution engine included in the LS/TS middleware, this was rather straightforward to achieve. The Whitestein platform and idea of goal-oriented BPM turned out to be the "perfect match" for the idea of goal- and context-oriented BPM. The model could easily be built up with the modeler. Moreover, the seamless transfer from modeling into process execution was demonstrated as expected.

Also the ability to change or enhance the process model quickly and easily was used. Today, once a business process is modeled and realized as a system, it is rather hard to change the model and system. Normally bi-annual release dates exits for new system releases. This may be too long, if the process and system have to change quickly, because, e.g., a new accelerated process has to be used soon due to business requirements. Due to the seamless translation of process model to execution a changed process model can be transferred to the execution within short time, and any new processes can take the new model.

Beyond the operational goals (i.e. to realize a change in a car) several other goals where identified during process analysis. These goals are "goals to be monitored during execution" like e.g. the time of the process is below a certain limit, or cost should not increase a value. An agent, according to the ideas of autonomic BPM discussed in Section IV, can autonomously monitor these goals.

VII. CONCLUSION

Business processes are of paramount importance to the successful operation of a modern enterprise. While the field of BPM has introduced noteworthy progress in the computer support for handling business processes, more advanced approaches are necessary in order to meet the challenges of business agility.

The ACM system has to effectively deal with a complex and challenging set of requirements, bringing an agile but dependable software infrastructure to the Change Management process at Daimler. Change Management is a critical activity to keep a constant product quality and customer satisfaction level while operating in a competitive and dynamic market.

The technological leverage of Agent Technology, together with the combined concepts of goal- and context- orientation as well as Autonomic Computing, allow the conception and realization of an advanced BPM product, and of an innovative application in the Engineering Change Management domain at Daimler, such as the ACM system is going to be.

REFERENCES

[1] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, S. R. White: *A Multi-Agent Systems Approach to Autonomic Computing*. AAMAS 2004: 464-471

[2] T. Beuter: *Workflow-Management für Produkt-entwicklungsprozesse*. Dissertation Universität Ulm. (2002) (in German)

[3] N.R. Jennings, M.J. Wooldridge (Eds.): *Agent Technology – Foundations, Applications, and Markets*. Springer. (1998)

[4] B.Burmeister, H.-P. Steiert, T. Bauer, H. Baumgärtel: *Agile Processes through Goal- and Context-oriented Business Process Modeling*. in: J. Eder, S. Dustdar et al. (Eds.): BPM 2006 Workshops, LNCS 4103, Sprnger, 215 – 226, 2006.

[5] L. Braubach, A. Pokahr, W. Lamersdorf: *Jadex: A BDI-Agent System Combining Middleware and Reasoning*. In: 18.R. Umland, M. Klusch, M. Calisti (Eds.): Software Agent-Based Applications, Platforms, and Development Kits. Whitestein Series in Software Agent Technology. Birkhäuser. (2005)

[6] A.S. Rao, M.P. Georgeff: *BDI Agents: From Theory to Practice*. In V. Lesser (ed.) Proc. 1st International Conf. on Multi-Agent Systems. MIT-Press. (1995)

[7] Agentis Software: Adaptive Enterprise™ Solution Suite. http://www.agentissoftware.com

An implemetation of roles as affordances: powerJava

Erik Arnaudo, Matteo Baldoni, Guido Boella, Valerio Genovese, and Roberto Grenna

Abstract. **This document shortly describes powerJava, a Java extension which provides the instructions to manage roles. After defined the environment in which we have worked, we will discuss the language's new instructions and we will show an example.**

I. Something about roles

Object orientation is a leading paradigm in programming languages, modeling, knowledge representation and databases. When we think to an *object*, we do it in terms of *attributes* and *methods*, and if we refer to *object interaction*, we do it in terms of *public attributes* and *public methods*: these are the only ways to realize it!

In computer science literature, other kinds of interaction between entities have been proposed at levels higher than programming languages. We can properly speak about *sessions*, which contain the interaction state (as at web services level), but a very important concept is that of *role*. Steimann [1] provided an interesting role representation, giving three types of "role as": *roles as named places*, *role as specialization and/or generalization*, *roles as adjunct instances*. Our approach is to consider *roles as affordances*; we consider roles as instances having a different identity respect to the players that play them. Inspired by research in cognitive science, this view sees the properties (attributes and operations) of an object as something not independent from whom is interacting with it. In this way, an object "affords" different ways of interaction to different kinds of objects.

The notion of "affordance" has been made popular by Norman [3] (p. 9): "The term affordance refers to the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used. A chair affords ('is for') support, and, therefore, affords sitting."

How can we use the concept of "affordance" to introduce new modeling concepts in object oriented knowledge representation? The affordances of an object are not isolated, but they are associated with a given specie. So we have to consider sets of affordances. We will call a *role type* the different sets of interaction possibilities, the affordances of an object, which depend on the class of the interactant manipulating the object: the *player* of the role. To manipulate an object it is necessary to specify the role in which the interaction is made.

A given role type can be instantiated, depending on a certain player of a role (which must have the required properties), and the *role instance* represents the state of the interaction with that role player. Just to better explain the possible use of roles as affordances, we introduce the following figures, which introduce different ways to interact with an object through the roles it offers.

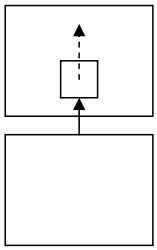


Figure 1 - An object interacts with another one by means of the role offered by it.

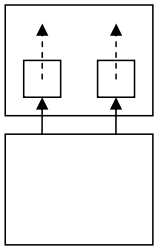


Figure 2 - An object interacting in two different roles with another

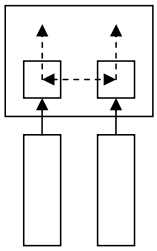


Figure 3 - Two objects which interact with each other by means of the roles of another object.

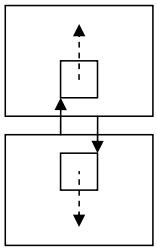


Figure 4 - Two objects interact with each other, each one playing a role offered by the other.

The idea behind affordances is that the interaction with an object does not happens directly with it by accessing its public attributes and invoking its public operations. Rather, the interaction with an object happens via a role: to invoke an operation, it is necessary first to be the player of a role offered by the object the operation belongs to. The roles which can be played depend on the properties of the role player (the requirements), since the roles represent the set of affordances offered by the object.

II. POWERJAVA AND ITS ENVIRONMENT

In order to translate the roles as affordances approach, we realize *powerJava*.

What we did, simply was an extension of Java 1.4 grammar, implemented using *JavaCC* (Java Compiler Compiler), which is a parser generator that returns in output Java code.

This parser generator is a program which reads a grammar specification and converts it in a Java program (parser) that can recognize this grammar.

JavaCC offers other tools too. For example, *JJTree*, that realizes (before obtaining the parser) a tree from the grammar written by the user. By JJTree exists the possibility of inserting code (written in Java, obviously), to "drive" the behavior of the parser in some cases. It defines the *Node* interface too. The Node interface is the interface that each node of the tree has to implement, and it offers some methods for setting the node parent or its sons.

JavaCC is a top-down parser, but JJTree generates the tree in a bottom-up mode, using a stack to contain the nodes after creating them.

The idea is that the tokens are serialized in a chain, but from each of them is possible to reach its parents (father, grandfather and so on). In this way it's possible to manage many events, and it's clear that for each new token it's possible to define the correct behavior.

The token concept (together with, for example, the *skip* concept) is something of very powerful, that collocates JavaCC at an higher level than other parsers.

JavaCC generate a LL(1) parser, but in some points of the grammar the parser can works as a LL(k) one: defining some *lookaheads* it's possible to manage expressions made by k words. In other cases, the parser works as a LL(1), with all obvious advantages in terms of performances.

The use is very simple. The grammar files have a .jj or .jjt extension. When we start from a .jjt file, we have simply to perform:

C:\>JJTree filename.jjt

The next step is to execute:

C:\>JavaCC filename.jj

which generates all the .java files needed, including our parser. All that we have to do is to compile all the .java files.

Before continuing, we have to say that we can customize a lot of the generated classes, like SimpleNode.java, or UnparseVisitor.java. "Visitor" is the pattern that JavaCC uses for visiting the nodes.

After this short description of the environment, in *Figure 5* is shortly described the extension of the grammar syntax to obtain powerJava.

The system requirements to use powerJava environment are very simple: you've only to have JDK 1.4 or later on your pc.

Please note that the name powerJava is due to the fact that we call the methods offered by roles "powers", because they offer the possibility to modify the private state and access private methods of the institution which defines them, and the state of the other roles defined in the same institution.

III. AN EXPLENATORY EXAMLPE

To make an example (see [4]), let's suppose to model a class *Printer*. The interaction possibilities offered by the class

```
rolespec ::= "role" identifier ["extends" identifier*] "playedby"
identifier interfacebody

classdef ::= ["public"|"private"|...] "class" identifier
["extends" identifier] ["implements" identifier*] classbody

classbody ::= "{" fielddef* constructors* methoddef* roleimpl* "}"

roleimpl ::= "class" identifier_1 "realize" identifier_2 rolebody

rolebody ::= "{" fielddef* constructors* methoddef* "}"

rcast ::= (expr.identifier) expr

keyword ::= that | ...
```

Figure 5 - The first extension of the Java (1.4) syntax in powerJava.

are different and depend on which objects invoke its methods. For example, some objects have more privileges than other ones, and thus they can invoke methods which are not available to other objects interacting with the same printer. Moreover, some methods keep track of the interaction with each specific object invoking them. For example, print counts the number of pages printed by each object invoking it to check whether the quota assigned to the object is respected. However, objects with more privileges do not have a quota of printed pages.

The *Printer* can be seen as an institution which supplies two different roles for interacting with it (the set of methods a caller can invoke): one role of normal *User*, and the other role of *SuperUser*. The two roles offer some common methods (roles are classes) with different implementations, but they also offer other different methods to their players (and there is no direct way to interact with the *Printer*). For example, *Users* can print their jobs and the number of printable pages is limited to a given maximum; thus, the number of pages is counted (the role associates new attributes with the player): each *User* should be associated with a different state of the interaction (the role has an instance with a state which is associated with its player). The *User* can print since the implementation of its methods has access to the private methods of the *Printer* (the methods of the *User* access the private attributes and operations of another object, the institution). *SuperUsers* have the method print with the same signature, but with a different implementation: they can print any number of pages; moreover, they can reset the page counter of *Users* (a role can access the state of another role, and, thus, roles coordinate the interaction).

A role like *SuperUser* can access the state of the other *User* roles and of the callee object (the institution *Printer*) in a safe way only if it encapsulated in the institution *Printer*. Thus the definition of the role must be given by the same programmer who defines the institution (the class of the role belongs to the same institution class namespace, or, in Java terminology, it is included in it).

In order to interact as *User* or *SuperUser* it is necessary to exhibit some requested behavior. For example, in order to be a User a caller object must have an account (it must be an *Accounted*), which is printed on the pages (returned by a method offered by the player of the role). A *SuperUser* can have more demanding requirements.

Finally, a role *User* can be played only when there is an instance of *Printer* and an instance of a class implementing *Accounted* which can play the role.

In the following figure there is the code of our example.

First, we have to import package to manage roles (Figure 6-[1]), then, we define the class for the *Login* (Figure 6-[2]) and the class for the *Job* (Figure 6-[3]).

User (Figure 6-[5]) and *SuperUser* (Figure 6-[6]) are roles, both played by an *AccountedPerson* (Figure 6-[4]).

```
import it.unito.di.javarole.*;                          [1]
class Login                                             [2]
{
        private String ID;

        public Login(String ID)
        {
                this.ID = ID;
        }
}
class Job                                               [3]
{
        private int ID;
        private int numberOfPages;

        public Job()
        {
                this.ID = -1;
                this.numberOfPages = 1;

        }

        public Job(int ID, int n)
        {
                this.ID = ID;
                this.numberOfPages = n;
        }

        public int getID()
        {
                return this.ID;
        }

        public int getNumberPages()
        {
                return this.numberOfPages;
        }
}
interface AccountedPerson                               [4]
{
        Login getLogin();
}
role User playedby AccountedPerson                      [5]
{
        int print(Job job);
        int getPrintedPages();
}
role SuperUser playedby AccountedPerson                 [6]
{
        int print(Job job);
        int getTotalPages();
}
class Person implements AccountedPerson                 [7]
{
        private Login login;

        public Login getLogin()
        {
                return login;
        }

        public void setLogin(String ID)
        {
                login = new Login(ID);
        }
}
```

Figure 6 – Our example's code – Part 1 of 3

Then we have to write the code for the *Person* class (Figure 6-[7]), which implements *AccountedPerson*; what we have now to do is to write our institution class: *Printer* (Figure 7-[9]). In *Printer* we write two inner classes, *U* (Figure 7-[9]) and *S* (Figure 7-[10]), which are the roles offered by it.

(Figure 8-[11]) showes the code for the main, in which we can also see the .transfer method (Figure 8-[12]). In last build of our environment, in fact, we have also implemented methods to transfer and to remove a role. When Sergio transfers his *SuperUser* role to chris, he will irremediably lose it. In this way, he is having no role, so that an opportune exception will be thrown if he would to play it.

Once we've written the code, we simply have to compile (better, pre-compile) it, in order to obtain a standard Java program. To do it, it's enough to run:

C:\>java JavaRolePreCompiler <MyFile.java> TargetFile.java

where MyFile.java is the source we've written few minutes ago, while TargetFile.java is the file that we will compile once we have it. It's a good choice to name the class that contain the main with TargetFile name.

Let's suppose that we saved our file as Test1.java, so we have to write:

C:\>java JavaRolePreCompiler <Test1.java> TestOne.java

Now we can compile TestOne.java, obtaining the executable:

C:\>javac TestOne.java

And, finally, we can execute:

C:\>java TestOne.java

You can download this (really working!) example (within all the work environment) from the website http://www.powerjava.org.

IV. JUST AN IDEA OF THE PRE-COMPILING RESULT

Only for give a track about the target of pre-compiling operations, we write an interesting example involving the *User* role.

In Figures 9, 10, 11 you can see the original code (before pre-compiling) and the pre-compiled code (in italic).

First, let's consider the *User* role definition (Figure 9-[13]) and it's corresponding pre-translation (Figure 9-[14]).

Following, we focus on the the class *Printer* (Figure 9-[15], 10-[16]), and in a particular way to those variables and structures added by the pre-compiler. We can see the HashTable rolelist (Figure 10-[17]), that will contain the class offering role definitions.

We defined an inner class *U* (Figure 10-[18]), realizing *User*, inside class *Printer* (which translation is in Figure 10-[19] – Figure 11).

It's very interesting to note the use of keyword *that*. It refers to *that* object is playing the role at issue, and it's used only in role implementation. An example is the invocation of *that.getLogin()* as a parameter of the print method in the previous code.

```
class Printer                                                    [8]
{
  private int totalPrintedPages = 0;
  private int MAX_PAGES_USER = 100;
  private void print(Job job, Login login)
  {
    System.out.println("Printed job " + job.getID());
    totalPrintedPages += job.getNumberPages();
  }
  class U realizes User                                          [9]
  {
    int counter = 0;

    public U(){}
    public U(int i){}
    public int print(Job job)
    {
      if (counter > MAX_PAGES_USER)
      //throw new IllegalPrintException();
      {
        System.out.println("Too many pages printed!");
        return 0;
      }
      else
      {
        counter += job.getNumberPages();
        Printer.this.print(job, that.getLogin());
        return counter;
      }
    }
    public int getPrintedPages()
    {
      return counter;
    }
  }
  class S realizes SuperUser                                     [10]
  {
    public int print(Job job)
    {
      Printer.this.print(job, that.getLogin());
      return totalPrintedPages;
    }
    public int getTotalPages()
    {
      return totalPrintedPages;
    }
  }
}
```

Figure 7 – Our example's code – Part 2 of 3

```
public class TestOne                                        [11]
{
  public static void main(String[] args)
  {
    Job j1 = new Job(1, 57);
    Job j2 = new Job(2, 160);
    Job j3 = new Job(3, 94);
    Job j4 = new Job(4, 211);
    Printer hp8100 = new Printer();
    Person chris = new Person();
    Person sergio = new Person();
    hp8100.new U(chris);
    hp8100.new S(sergio);
    ((hp8100.U)chris).print(j2);
    ((hp8100.S)sergio).print(j4);
    ((hp8100.U)chris).print(j1);
    ((hp8100.S)sergio).transfer(chris);                     [12]
    ((hp8100.U)chris).print(j1);
    ((hp8100.S)sergio).print(j3);
    System.out.println("Chris printed "
+ ((hp8100.U)chris).getPrintedPages() + " pages.");
    System.out.println("The printer printed "
+ ((hp8100.S)sergio).getTotalPages() + " pages.");
  }
}
```

Figure 8 – Our example's code – Part 3 of 3

V. Future developments

Our next goal is to implement features that make powerJava able to model all the cases represented in *Figures 1-4*; we will work to implement something of more collaborative too, and we can't exclude to try interactions between powerJava and other environments. We will surely work focusing on relations, as defined in [5] and [6], and sessions, as defined in [7].

```
role User playedby AccountedPerson          [13]
{
  int print(Job job);
  int getPrintedPages();
}

interface User                              [14]
{
  int print(Job job);
  int getPrintedPages();
}

class Printer                               [15]
{
  private int totalPrintedPages = 0;
  private int MAX_PAGES_USER = 100;
…
…
}
```

Figure 9 – Pre-compiling example – Part 1 of 3

```
class Printer implements ObjectWithRoles                   [16]
{
  private java.util.Hashtable rolelist = new java.util.HashTable();  [17]
  private int totalPrintedPages = 0;
  private int MAX_PAGES_USER = 100;
...
...
}

class U realizes User                                       [18]
{
  int counter = 0;

  public U(){}
  public U(int i){}

  public int print(Job job)
  {
    if (counter > MAX_PAGES_USER)
//throw new IllegalPrintException();
    {
      System.out.println("Too many pages printed!");
      return 0;
    }
    else
    {
      counter += job.getNumberPages();
      Printer.this.print(job, that.getLogin());
      return counter;
    }
  }

  public int getPrintedPages()
  {
    return counter;
  }
}

class U implements ObjectWithRoles, RoleInterface, User    [19]
{
  private AccountedPerson that;
  public void destroy()
  {
    ((ObjectWithRoles)this.that).removeRole(this, Printer.this);
    this.that = null;
  }
  public void transfer()
  {
    ((ObjectWithRoles)this.that).removeRole(this, Printer.this);
    this.that = (AccountedPerson)req;
    ((ObjectWithRoles)this.that).setRole(this, Printer.this);
  }
  ... // Code defining the possibility for
  ... // the class to offer roles
  int counter = 0;
```

Figure 10– Pre-compiling example – Part 2 of 3

```
  public U(AccountedPerson that)
  {
    int alreadyPresent = 0;
    try
    {
      if(((ObjectWithRoles)that).getRole(Printer.this, "U") != null)
      alreadyPresent = 1;
    }
    catch (Exception e){}
    if (alreadyPresent == 1)
      this.that = (AccountedPerson)
((ObjectWithRoles)that).getRole(Printer.this, "U");
    else
      this.that = that;
    ((ObjectWithRoles)this.that).setRoles(this.Printer, this);
  }
  public U(int i)
  {
    int alreadyPresent = 0;
    try
    {
      if(((ObjectWithRoles)that).getRole(Printer.this, "U") != null)
        alreadyPresent = 1;
    }
    catch (Exception e){}
    if (alreadyPresent == 1)
      this.that = (AccountedPerson)
((ObjectWithRoles)that).getRole(Printer.this, "U");
    else
      this.that = that;
    ((ObjectWithRoles)this.that).setRoles(this.Printer, this);
  }
  public int print(Job job)
  {
    if (this.that == null)
throw new RunTimeException("Reference to that is null");
    if (counter > MAX_PAGES_USER)
//throw new IllegalPrintException();
    {
      System.out.println("Too many pages printed!");
      return 0;
    }
    else
    {
      counter += job.getNumberPages();
      Printer.this.print(job, that.getLogin());
      return counter;
    }
  }

  public int getPrintedPages()
  {
    if (this.that == null)
throw new RunTimeException("Reference to that is null");
      return counter;
  }
}
```

Figure 11 – Our example's code – Part 3 of 3

REFERENCES

[1] F. Steimann. "On the representation of roles in object-oriented and conceptual modeling". *Data & Knowledge Engineering*, 35:1 (2000) 83-106

[2] M. Baldoni, G. Boella, and L. van der Torre. "Modelling the interaction between objects: roles as affordances". In *J. Lang, F. Lin, and J. Wang, editors, Knowledge Science, Engineering and Management: First International Conference*, KSEM, volume 4092 of LNCS, pages 42-54, Guilin City, China, August 5-8 2006. Springer.

[3] D. Norman. "The Design of Everyday Things". *Basic Books*, New York (2002)

[4] M. Baldoni, G. Boella, and L. van der Torre. "Interaction between Objects in powerjava." *Journal of Object Technology, Special Issue OOPS Track at SAC* 2006, 6(2), 2007.

[5] M. Baldoni, G. Boella, and L. van der Torre. "Relationships Define Roles, Objects Offer Them." *Roles07 workshop at ECOOP*, pages 4-14.

[6] M. Baldoni, G. Boella, and L. van der Torre. "Relationships meet their roles in object oriented programming." *Procs. of the 2nd International Symposium on Fundamentals of Software Engineering 2007 Theory and Practice* (FSEN '07).

[7] V. Genovese. "A Meta-model for Roles: Introducing Sessions." *Roles07 workshop at ECOOP*.

Eric Arnaudo is student at the Dipartimento of Informatica, Università di Torino, Corso Svizzera 185, 10149, Torino, Italy.

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.

Valerio Genovese is student at the Dipartimento of Informatica, Università di Torino, Corso Svizzera 185, 10149, Torino, Italy.

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.

ELDATool: A Statecharts-based Tool for Prototyping Multi-Agent Systems

Giancarlo Fortino, Alfredo Garro, Samuele Mascillaro and Wilma Russo

***Abstract*— This paper briefly describes the ELDATool, a Statecharts-based visual tool for the rapid prototyping of Multi-Agent Systems based on the Event-driven Lightweight Distilled Statecharts-based Agents (ELDA) model. In particular, the ELDATool, which is implemented in Java as an Eclipse plug-in, supports an iterative process involving the following phases: detailed design, automatic code generation and simulation. The high-level design, which is the input to this iterative process, can be obtained through currently available agent-oriented methodologies such as PASSI and GAIA. In order to show the main characteristics of the ELDATool, a simple case study is presented.**

Index Terms—**Visual Tools, Multi-Agent Systems, Distilled StateCharts, State-based Programming.**

I. Introduction

Agent oriented software engineering [1] aims at providing methodologies and tools for the development of complex and distributed software systems through the agent paradigm in terms of Multi-Agent Systems (MASs).

Several agent-oriented methodologies (PASSI [2], GAIA [3], SODA [4], INGENIAS [5], DSC-based [6, 7], etc.) have been to date proposed for supporting the development life-cycle of MASs. Few of them are also equipped with visual tools capable of supporting all the phases of the development life-cycle. The availability of such tools is widely considered to be strategic for supporting a rapid prototyping of the MAS under-development.

This paper introduces the ELDATool which aims at supporting the DSC-based agent-oriented methodology proposed in [6, 7, 8]. This methodology covers the modelling, implementation and simulation phases of MAS based on the ELDA (Event-driven Lightweight Distilled Statecharts-based Agents) model.

According to the ELDA model [6, 9] a multi-agent system is modelled at high-level as a set of different types of agents and a set of interaction events among the agents. An ELDA agent consists of a unique identifier, a data space, a dynamic behaviour, a single thread of control, and a queue of the received events. In particular, the dynamic behaviour of an agent is specified through the Distilled Statecharts formalism [6], derived from the well-known Statecharts [10]. Modelling an agent is basically carried out by specifying its behaviour as a hierarchical state machine compliant with the state-based template of the FIPA agent [11] and by defining the events which can be received and generated. While events are implicitly received through the event queue, they are explicitly emitted through the *generate* primitive. The received events are called IN-events, whereas the generated events are called OUT-events. In particular, events formalize three kinds of interactions [9]: (i) internal, which are sent by the agent to itself to proactively drive its activity; (ii) management, which are used to interact with the agent management system for requesting services and resources; (iii) coordination, which are exploited to interact with local or remote agents/entities through a given coordination space.

The ELDA model is implemented in the ELDA framework, an object-oriented framework which provides the programming abstractions to implement ELDA-based MAS. Currently, the ELDA framework is implemented in Java.

The ELDATool incorporates the ELDA framework and provides a graphical integrated development environment based on Eclipse [12]. The ELDATool is exemplified through a simple case study concerning with the modelling of a contractor mobile agent within an agent-based e-Marketplace.

The rest of this paper is organized as follows. Section 2 enumerates the system requirements of the ELDATool. Section 3 and section 4 respectively present the ELDATool design and implementation. Section 5 describes the use of the ELDATool through the simple case study developed. Finally conclusions are drawn and on-going work anticipated.

II. System Requirements

ELDATool aims at supporting the MAS designer for the rapid prototyping of MASs based on the ELDA model according to the iterative process shown in Figure 1.

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).

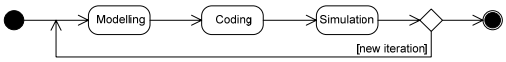


Figure1: Iterative process for prototyping ELDA-based MASs.

To support the *Modelling* phase, the tool offers the basic functionality of visual modelling of the active state of the agent behaviour through a DSC-based Hierarchical State Machine. The active state is a composite state in which the agent performs its main activity. In particular, the following modelling features are supported:

- definition of the internal states of the active state;
- definition of the events, generated (or OUT-events) and received (IN-events) by/from the ELDA agent, by extending appositely the base events provided by the Java implementation of the ELDA framework (or ELDAFramework) or events previously defined by the user;
- definition of the transitions between states which involves:
  - the use of the IN-events previously defined for labelling the transitions;
  - (possibly) the definition and the use of the guards associated to the transitions;
  - (possibly) the definition and the use of the actions associated to the transitions.

The obtained graphical modelling is serialized into XML-like files.

To support the *Coding* phase, the tool offers the functionality of automatic code generation by translating the XML-like files produced after the *Modelling* phase into Java code based on the ELDAFramework.

Finally, to support the *Simulation* phase, the tool is based on the MASSIMO framework [13] and offers the following functionalities:

- implementation of the simulator through the definition of the network topology of the agent platform, the initial location of the agents, the definition of the performance parameters, etc;
- execution of the simulator for performing the simulation;
- gathering of the values of the parameters defined for the performance measurements.

The ELDATool is implemented in Java as an Eclipse plug-in to exploit several frameworks which fully support the development of visual editors. Moreover, the high diffusion of Eclipse in the research community makes the tool immediately available to the Eclipse users and the learning process of the tool is so quicker.

III. Design

The architecture of the ELDATool is component-based; each component is responsible of the specific aspects of the *Modelling*, *Coding* and *Simulation* phases.

In particular, for each different modelling aspect the following editors have been identified and designed:

- DSCEditor, for modelling the active state of an ELDA agent;
- EventEditor, for defining the events;
- GuardEditor, for defining the guards;
- ActionEditor, for defining the actions;
- FunctionEditor, for defining the supporting functions.

Each editor is capable of handling (visually or not) the elements of its reference meta-model and producing an instance of this meta-model (or specific model) as output.

The CodeGenerator component uses the models produced by the editors as input to offer the functionalities needed for the code generation according to the classes constituting the ELDAFramework.

The Simulator component uses the code produced by the CodeGenerator component to support the *Simulation* phase.

Figure 2 shows the components, the dependence relationships among them, and their contextualization with respect to the process phases.

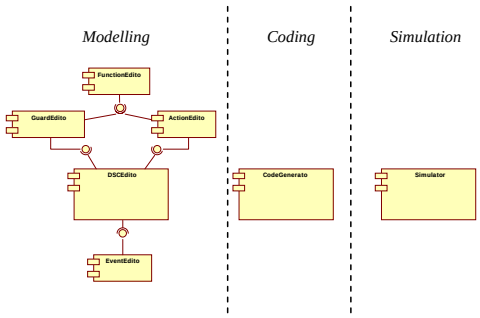


Figura 2: The ELDATool components

IV. ELDATool Implementation

Currently the ELDATool supports the first two phases of the process: *Modelling* and *Coding*. The architectural components described in section II are implemented in Java by exploiting:

- the Eclipse platform [12], which is a widely-used Integrated Development Environment (IDE) with extensible architecture based on plug-ins, i.e. independent components which can be easily installed and integrated in the IDE;
- the Graphical Editing Framework (GEF) [14] which allows for the development of visual editors in Eclipse by offering high support for the management of the user interactions;
- the Eclipse Modelling Framework (EMF) [15] which supports the modelling phase of a structural model and the automatic generation and manipulation of its Java implementation.

The editor components (see section II) are implemented according to the architectural pattern Model-View-Controller (MVC) to support the user-interaction handling (View-Controller) and the manipulation of the model in response to the generated events (Model).

In particular, user-interaction handling is implemented by extending the classes provided by GEF whereas the model manipulation is carried out by the plug-ins automatically

generated by EMF. It is worth noting that EMF generates a plug-in exposing the interfaces needed for the instantiation of the implemented meta-model. Accordingly, each editor component is constituted by an EMF-generated plug-in which manages the model and a plug-in which handles the user interaction.

In order to ease the deployment of the ELDATool the number of its constituting plug-ins was minimized. In particular, the plug-ins which manage the models are separately implemented whereas the plug-ins handling the user-interaction and supporting the code generation are integrated in a unique plug-in, the ELDAEditor.

As a consequence, the following plug-ins are implemented:

- *DSCModel*, which contains the implementation of the DSC meta-model;
- *EventModel*, which contains the implementation of the Event meta-model;
- *ActionGuardModel*, which contains the implementation of the Action and Guard meta-model;
- *FunctionModel*, which contains the implementation of the Supporting Function meta-model;
- *ELDAEditor*, which contains all the editor and the code generator.

It is worth noting that the models, obtained through instantiating the related meta-models and by using the editor made available by the ELDATool, are serialized into independent XML-like files with different extensions (see Table 1).

Table 1: Extensions of the XML-like files associated to the models

Model	File Extension
Event	event
Action	action
Guard	guard
Function	function
Active State	dsc

Figure 3 highlights and clarifies the dependence relationships among the implemented plug-ins and the GEF/EMF plug-in.

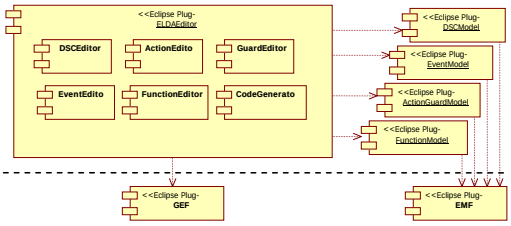


Figura 3: The ELDATool Plug-ins.

The ELDATool will be released as a set of plug-ins and a jar named ELDAFramework.jar which contains the Java implementation of the ELDA framework. It is worth noting that to install the ELDATool it is only necessary to copy the set of plug-ins and the ELDAFramework.jar into the plugins folder of Eclipse and restart Eclipse. The software requirements of the ELDATool are: Eclipse ver. 3.3, GEF ver. 3.3, EMF ver. 2.3.0 and JRE ver. 1.5.

V. A Case Study

In this section the use of the ELDATool is shown by illustrating the modelling of an ELDA agent named Contractor Mobile Agent (CMA) which operates within an agent-based e-Marketplace. After a discovery phase of the vendors offering a specific product which was carried out by another type of agent, the CMA has the goal of supporting the phase of contracting with the vendors found.

Figure 4 shows the active state of the CMA behaviour and Figure 5 reports its guards, actions, and supporting functions.

In particular, the CMA, received the identifier and the location of a given vendor and the product to buy, migrates to the vendor location (see action ac1) and starts the contracting phase (see action ac2). After obtaining the offer, if the product is immediately available (see guard productAvailable) the CMA archives the offer and comes back to the starting location (see action ac3); otherwise, the CMA waits for the product availability until a timeout expiration (see action ac5). After the timeout expiration, the CMA restarts the contracting phase if the number of trials is greater 0 (see guard NotAllTrialsDone); otherwise, the CMA archives the offer and migrates to the starting location (see action ac3). Finally, the CMA notifies the details of the contracting phase to its owner (see action ac4).

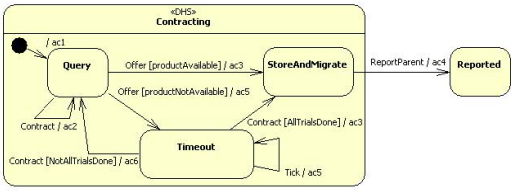


Figure 4: The active state of the CMA

```
Guards Definitions
private boolean productAvailable(ELDAEvent e){
 OfferMsg offer=(OfferMsg) e;
 if (offer.getProductAvalaible())
  return true;
 return false;
}
private boolean productNotAvailable(ELDAEvent e){
 return !productAvailable(e);
}
private boolean NotAllTrialsDone(ELDAEvent e){
 return !AllTrialsDone(e);
}
private boolean AllTrialsDone(ELDAEvent e){
 if(trials==0)
  return true;
 return false;
}
Actions Definitions
private void ac1(ELDAEvent e){
 generate(new ELDAEventMoveRequest(self(), self(),
                                   VATarget.getCurrLocation()));
 generate(new Contract(self(), self()));
}
private void ac2(ELDAEvent e){
 PriceQueryMsg priceQuery= new PriceQueryMsg(self(), VATarget, null);
 generate(new ELDAEventMSGRequest(self(), VATarget, priceQuery));
}
private void ac3(ELDAEvent e){
 OfferMsg offer=(OfferMsg) e;
 storeVAOffer(offer);
 generate(new ELDAEventMoveRequest(self(), self(),
```

```
                                        owner.getCurrLocation()));
 generate(new ReportParent(self(), self()));
}
private void ac4(ELDAEvent e){
 PPriceMsg pPrice=new PPriceMsg(self(), owner, VAOffer);
 generate(new ELDAEventMSGRequest(self(), owner, pPrice));
 generate(new ELDAEventQuitRequest(self()));
}
private void ac5(ELDAEvent e){
 if(timeout>0){
  timeout--;
  generate(new Tick(self(), self()));
 }
 else{
  timeout=100;
  generate(new Contract(self(), self()));
 }
}
private void ac6(ELDAEvent e){
 trials--;
 ac2(e) ;
}
Functions Definitions
private void storeVAOffer(OfferMsg offer){
 //omissis
}
```

Figura 5: Guards, actions and functions of the CMA behavior

To exemplify the use of the ELDATool the following activities are briefly illustrated: (A) visual definition of the active state, (B) definition of events, (C) definition of guards, (D) definition of actions, and (E) definition of supporting functions.

A. *Definition of the active state*

The result of this activity is the model of the active state of the agent behaviour; the transitions defined among the states are based on events, guards, and actions which are previously defined. Moreover, during this activity, it is possible to define local variables for each state so constituting a hierarchical data space. Figure 6 shows a snapshot of the active state of the CMA behaviour obtained through the DSCEditor.

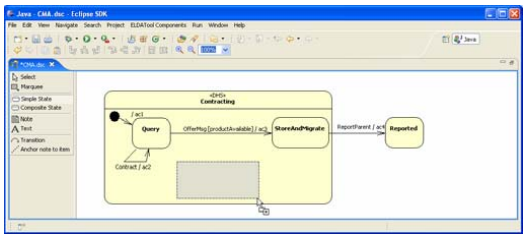


Figure 6: Definition of states

B. *Definition of events*

The EventEditor allows for the definition of new events by extending the events already offered by the ELDAFramework. In particular, for each event, the event name, the event class of the ELDAFramework to be extended, and (possibly) new parameters can be defined. Figure 7 shows the event definition dialog through which the event Tick is defined as extension of the base event ELDAEventInternal.

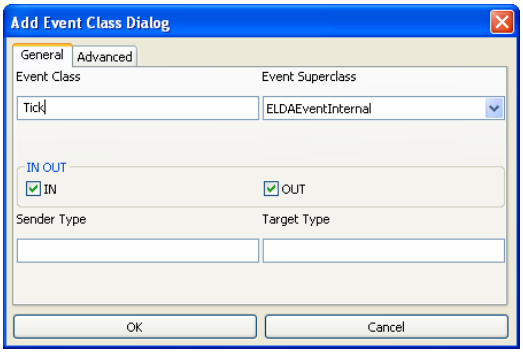


Figure 7: Definition of an event

C. *Definition of the guards*

The definition of a guard involves the definition of its name and the boolean expression associated to it (or guard body). Within a guard body it is possible to use variables belonging to the data space (e.g. the integer variable *trials*), guards previously defined and supporting functions. Figure 8 shows the definition of the guard named AllTrialsDone to be associated to the transition between the TimeOut state and the StoreAndMigrate state.

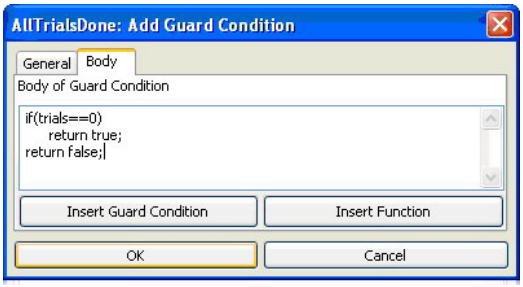


Figure 8: Definition of a guard

D. *Definition of the actions*

The definition of an action involves the definition of its name and the instructions which costitute it. In an action, it is possibile to use variables belonging to the data space, actions previously defined, and supporting functions. Figure 9 shows the definition of the action ac6 which uses the action ac2 previously defined.

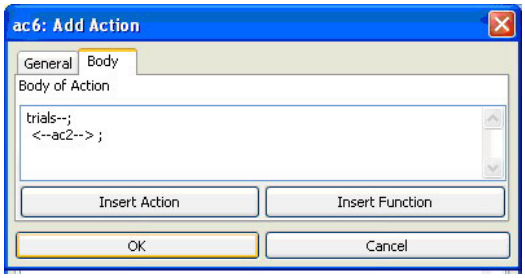


Figure 9: Definition of an action

E. Definition of supporting functions

The definition of supporting functions which can be used by actions and guards to improve design modularity, is constituted by the specification of the function name, of the type of the returned value, of the parameters and of the function body. Figure 10 shows the dialog for the definition of the supporting functions; in particular, the StoreVAOffer function is defined which returns void and has only the parameter offerMsg of the OfferMsg type.

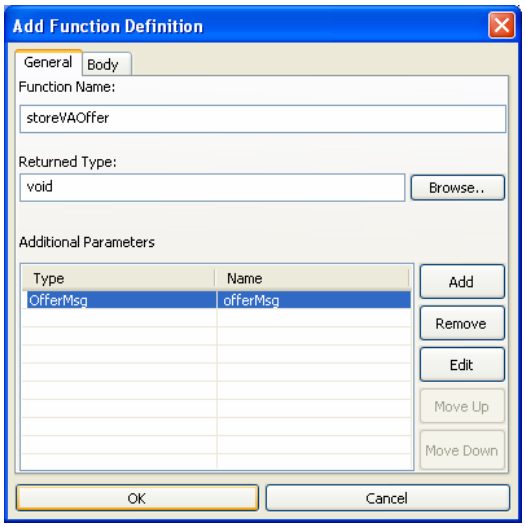


Figure 10: Definition of a supporting function

F. Code generation

After defining the agent behavior, it is possible to generate Java code through the CodeGenerator component. The code generation activity creates a new project containing the translation of specified models into Java code according to the ELDAFramework.

Figure 11 shows both the project containing the models of the whole MAS under-development (EMarketPlace) and the project structure (EMarketPlace_Implementation) generated only for the CMA which contains a package (emarketplace.cma) with the CMAActiveState class and a package (emarketplace.events) with event classes triggering the CMA (Contract, OfferMsg, ReportParent, Tick).

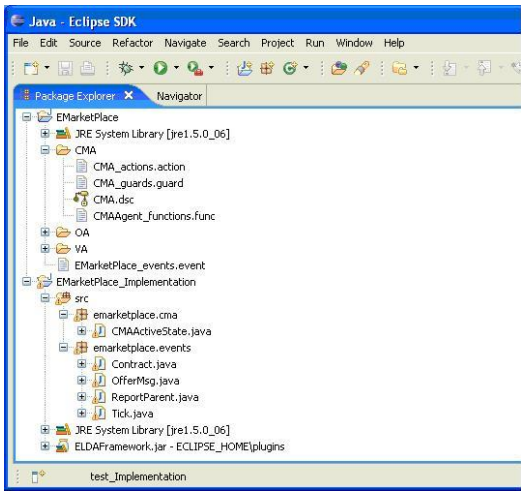


Figure 11: Structure of the generated project

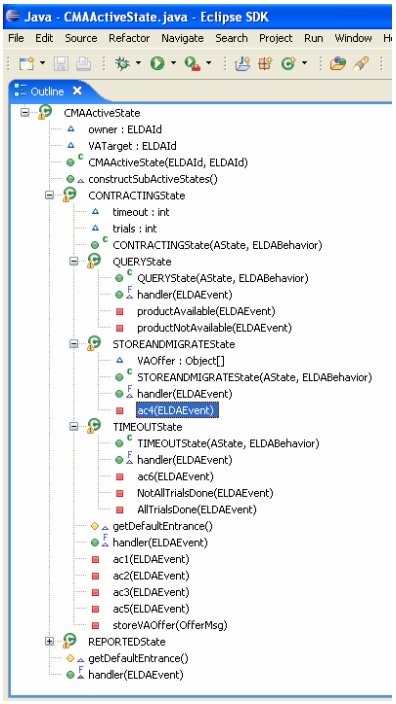


Figure 12: Active state of the CMA agent

Figure 12 shows the outline of the CMAActiveState which highlights the hierarchical dataspace of the agent and the location of guards, actions, and functions.

Figure 13 shows an excerpt of the CMA generated code related to the Timeout state.

```
// TIMEOUTState Inner Class
public class TIMEOUTState extends SimpleState implements Serializable
{

 public TIMEOUTState (AState parent, ELDABehavior ebeh) {
  super(parent,ebeh);
 }

 public final int handler(ELDAEvent evt){
 if (evt instanceof Tick){
   ac5(evt);
   return 0;
 }
 else if (evt instanceof Contract && NotAllTrialsDone(evt)){
  ac6(evt);
  ((CompositeState) parent).setActiveState(
    ((CompositeState) parent).getState("QUERY"));
  changeState(((CompositeState) parent).getActiveState());
  return 0;
 }
 else if (evt instanceof Contract && AllTrialsDone(evt)){
  ac3(evt);
  ((CompositeState) parent).setActiveState(
     ((CompositeState) parent).getState("STOREANDMIGRATE"));
  changeState(((CompositeState) parent).getActiveState());
  return 0;
 }
 else return parent.handler(evt);
 }
 // Actions Definitions Section
 private void ac6 (ELDAEvent e){
  trials--;
  ac2(e);
 }
 // Guards Definitions Section
 private boolean NotAllTrialsDone (ELDAEvent e){
  return ! AllTrialsDone(e);
 }
 private boolean AllTrialsDone (ELDAEvent e) {
  if(trials==0)
   return true;
  return false;
 }
}
```

Figure 13: The code of the Timeout state.

VI. Conclusions and Future Works

This paper has presented the ELDATool by describing its system requirements, design, implementation, and use through a simple example. The ELDATool represents a research effort aiming at supporting the rapid prototyping of MASs which is contextualized in the active research area on agent-oriented software engineering. In particular, the ELDATool gives support to a DSC-based agent-oriented methodology seamlessly covering the phases of the MAS development lifecycle from modeling to implementation.

Visual modeling and programming, and automatic code generation are very important features that any tool supporting an agent-oriented methodology should have to ease the designer tasks. The ELDATool fully provides such features and, furthermore, being based on the Eclipse platform, can be easily distributed and used by the community.

Currently, efforts are underway for (i) completing the Simulation component so allowing validation and performance evaluation of the MAS under-development; (ii) designing and implementing new components for the implementation and deployment of the prototyped MAS for a target agent platform.

References

[1] F. Zambonelli and A. Omicini, "Challenges and research directions in agent-oriented software engineering", *Autonomous Agents and Multi-Agent Systems*, 9(3), pp. 253-283, Nov. 2004.

[2] M. Cossentino, "From Requirements to Code with the PASSI Methodology," In *Agent-Oriented Methodologies*, Eds. B. Henderson-Sellers and P. Giorgini, Idea Group Inc., Hershey, PA, USA, 2005, pp. 79–106.

[3] F. Zambonelli, N. Jennings, and M. Wooldridge, "Developing multiagent systems: The Gaia methodology," *ACM Trans. Software Eng. Meth*., vol. 12, no. 3, pp.417-470, 2003.

[4] A. Molesini, A. Omicini, E. Denti, and A. Ricci, "SODA: A Roadmap to Artefacts," *6th International Workshop on Engineering Societies in the Agents World VI*, (ESAW 2005), Kusadasi, Aydin, Turkey, October 2005. *LNAI 3963*, Springer, 2006.

[5] J. Pavón, J. Gómez-Sanz, and R. Fuentes, "The INGENIAS Methodology and Tools," In *Agent-Oriented Methodologies*, Eds. B. Henderson-Sellers and P. Giorgini, Idea Group Publishing, 2005, pp. 236-276.

[6] G. Fortino, W. Russo, and E. Zimeo, "A Statecharts-based Software Development Process for Mobile Agents", *In Information and Software Technology*, 46(13), pp.907-921, Elsevier, Amsterdam, The Netherland, 2004.

[7] G. Fortino, A. Garro, and W. Russo, "An Integrated Approach for the Development and Validation of Multi Agent Systems", *In Computer Systems Science & Engineering*, 20(4), pp. 94-107, CRL Publishing Ltd., Leicester (UK), Jul. 2005a.

[8] R. Caico, M. Cossentino, G. Fortino, A. Garro, W. Russo, and F. Termine, "Simulation-driven Development of Multi-Agent Systems", *Proceedings of the EUROSIS Workshop on Multi-Agent Systems and Simulation (MAS&S'06)*, Palermo, Italy, 2006, pp. 17-24.

[9] G. Fortino and W. Russo, "Multi-coordination of Mobile Agents: a Model and a Component-based Architecture", *Proceedings of ACM Symposium on Applied Computing, Special Track on Coordination Models, Languages and Applications*, Santa Fe, New Mexico, USA, Mar. 13-17, 2005.

[10] D. Harel and E. Gery, "Executable Object Modelling with Statecharts", *IEEE Computer*, 30(7), pp. 31-42, 1997.

[11] FIPA (Foundation for Intelligent Physical Agents). 2002. FIPA Agent Management Support for Mobility Specification, Document FIPA DC00087C (2002/05/10).

[12] Eclipse - an open development platform, documentation and software, available at the World Wide Web: http://www.eclipse.org.

[13] G. Fortino, A. Garro, and Russo, W. (2005b) '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.

[14] The Graphical Editing Framework (GEF), documentation and software, available at the World Wide Web: http://www.eclipse.org/gef/.

[15] Eclipse Modeling Framework Project (EMF), documentation and software, available at the World Wide Web: http://www.eclipse.org/modeling/emf/.

The PRACTIONIST Development Tool

Fabio Centineo*, Angelo Marguglio* Vito Morreale* Michele Puccio*,

*R&D Laboratory - ENGINEERING Ingegneria Informatica S.p.A.

I. The PRACTIONIST suite

PRACTIONIST (PRACTIcal reasONIng sySTem) [1] is a suite of tools including (see figure 1): *(i)* a methodology, consisting of a UML-based modelling language (PAML) and an iterative and incremental development process, *(ii)* the PRACTIONIST runtime and framework (PRF), which defines and supports the execution logic and provides the built-in components according to such a logic to support the development of BDI agents in Java (using JADE[1]) with a Prolog belief base, and *(iii)* the PRACTIONIST Development Tool (PDT), a design and development environment which supports the methodology. The PRF also includes the PAIT, to monitor the intentional components of each agent and the PRACTIONIST Autonomic Manager (PAM) which enables PRACTIONIST applications to support the self-chop features [2] (self-configuring, self-healing, self-optimizing and self-protecting)

In this abstract we give an overview of the PDT, the modelling environment that is a part of the PRACTIONIST suite (figure 1), the metamodel it is built on, and a brief introduction of the PDT visual editors.

II. The PRACTIONIST Development Tool

The PRACTIONIST suite provides developers with the PRACTIONIST Development Tool, a tool to design and develop multi-agent systems according to the PRACTIONIST design methodology. Indeed, it supports such a methodology from the requirements analysis to the code generation of agents and artefacts (according to the A2A approach [2]), including a set of visual editors for each phase of the methodology. As an example, in figure 2 a snapshot of the Class editor is shown.

Some editors of the PDT are based on UML 2.0 meta-model[3], such as the class and use case editors, whereas the others are based on the PRACTIONIST Agent Modeling Language (PAML), which is a semi-formal UML-based visual modeling language for specifying, representing and documenting multi-agent systems designed with PRACTIONIST.

As the PAML aims to the definition of PRACTIONIST agents, its metamodel contains metaclasses to model intentional components of such agents, such as beliefs, goals and relations among them, plans and so forth.

The PDT has been developed by using several Eclipse[4] plug-ins, such as: UML2, Eclipse Modelling Framework (EMF), Graphical Editing Framework (GEF), Graphical Modeling Framework (GMF) and other Eclipse extensibility features. All the PDT editors share a common infrastructure, so that new editors can be added in it without any impact on the existing ones. Moreover, each editor inherits several features described below by the above infrastructure.

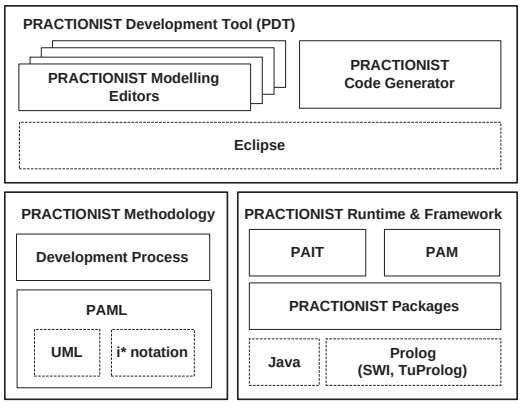


Fig. 1. The PRACTIONIST suite.

As many well-known CASE tools, the PDT editors provide all the features that support the development of complete and consistent visual models. Some of them are provided by GMF, such as the *cut and copy* and *save diagram as image* supports, the *look and feel* management, the diagram validation and so on, whereas the other ones have been built by generalizing some of the GMF project features, 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.

[1] http://jade.tilab.com
[2] http://www-03.ibm.com/autonomic/library.html
[3] UML 2.0 Superstructure Specification: formal/05-07-04
[4] http://www.eclipse.org/

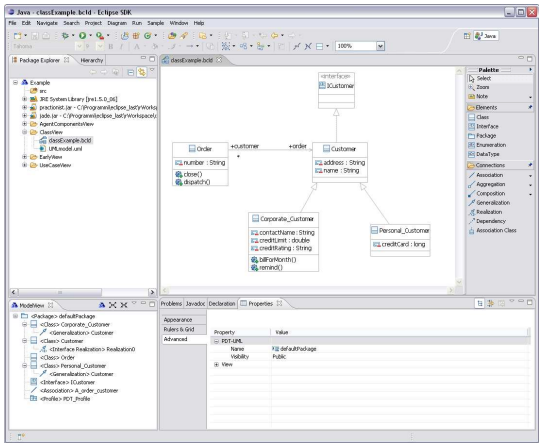


Fig. 2. A snapshot of the PRACTIONIST Development Tool.

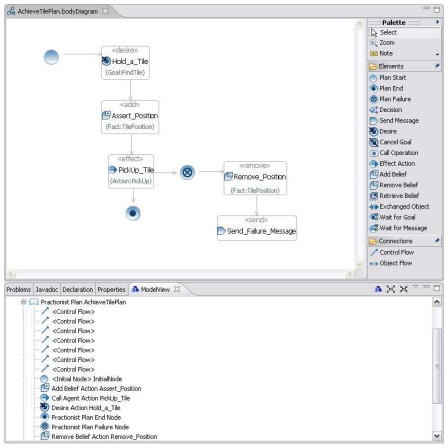


Fig. 3. A snapshot of the PDT Plan Body editor.

- *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 the PDT, 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.

As mentioned, the PDT provides PRACTIONIST developers with a rich set of visual modelling editors, as follows:

- *i*-based [3] 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 the structure of a system or of its parts for instance (see figure 2);
- *PRACTIONIST agent editors*:
  - *Agent editor*: to model agents and specify their components;
  - *Domain editor*: to model facts about the world the agent believes true, false or has no belief about;
  - *Goal editor*: to model agent goals and the relationships among them;
  - *Effector/Action - Perceptor/Perception editor*: to model the means agents interact with their environment;
  - *Plan editor*: to model the internals PRACTIONIST plans;
  - *Plan Body editor*: to model the body of PRACTIONIST plans (see figure 3).

Finally, the PDT represents a powerful visual modeling environment that supports the representation of the concepts underlying the BDI model as well as several features present in well-known UML-based CASE tools. Moreover, it let us reduce the development time of PRACTIONIST applications thanks to the code generation of agents and artefacts. The PDT aims at bridging the gap between the increasing need of development of multi-agent systems and the availability of tools for their design.

REFERENCES

[1] 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.

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

[3] 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

A Framework for Execution and Visualization of Situated Agents Based Virtual Environments

Giuseppe Vizzari[a], Giorgio Pizzi[a], Flávio Soares Corrêa da Silva[b]

***Abstract*—This document briefly describes a framework supporting the definition and implementation of virtual environment inhabited by interacting situated agents defined according to the Multilayered Multi-Agent Situated System model. The framework supports the specification and execution of visually rich 3D virtual environment endowed by the presence of mobile agents acting and interacting inside it according to a multi-agent model.**

I. Introduction

The design and realization of virtual environments inhabited by social entities is a significant application of the conjoint results of various research areas in computer science and engineering. Virtual environments have been exploited in several ways, and in particular:

- to support computer mediated forms of human interaction, characterized by the introduction of Embodied Conversational Agents facilitating users' interactions [9] or supplying awareness information in a visually effective form [13];
- to realize operational laboratories for participatory design, supporting the effective visualization of various alternative design choices to the involved stakeholders [6][9][8];
- to provide effective instruments for the modeling, simulation and visualization of the dynamics of entities situated in a representation of an existing, planned or reconstructed environment or situation [7][12];
- for sake of entertainment, in movies, computer games or in online communities (see, e.g., Second Life[1]).

While all these applications are characterized by a strong requirement for realistic and effective visualization tools (and some of them require a thorough analysis of the system usability, due to the necessary accessibility by non-technically skilled users), they also call for expressive models supporting the specification of behaviours for the entities that inhabit these environments, as well as the interaction among them and with the environment itself. The fact that the overall performance of the system is essentially dependant on the single actions and interactions that are carried out by entities inhabiting the modeled environment leads to consider that the Multi-Agent Systems approach is particularly suited to tackle the modeling issues that are posed by this scenario. This idea is also corroborated by the fact that most of the above introduced references actually describe systems based on this approach, and by specific experiences in applying MAS approaches to specific virtual environments applications such as computer games [10].

In this vein, the main aim of this document is to show the current advancement of a long term project that provides the realization of a framework supporting the development of MAS based simulations based on the Multilayered Multi-Agent Situated System model [1] provided with an effective form of 3D visualization. The main goal of the framework is to support a smooth transition from the definition of an MMASS based model of given situation to the realization of simulation systems characterized by an effective 3D user interface. One of the possible application areas of this kind of system is related to the modeling and simulation of crowds of pedestrians to support architectural design or urban planning [2][3]. In order to have information flowing appropriately from the formal model to design professionals (e.g. architects and urban planners), the MMASS-based simulator must be supported by adequate visualization and animation tools. Such supporting tools are the core issue of the present document. For sake of space, details of the MMASS are omitted, as well as a discussion of the related works; a more through discussion of these topics can be found in an extended version of this document in this volume [15].

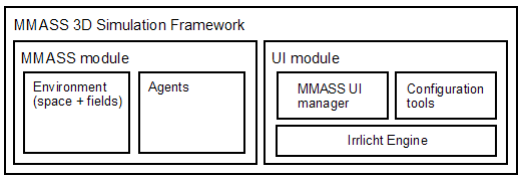


Figure 1 – Overall architecture of the framework for MMASS based virtual environments.

II. The Execution and Visualization Framework

The basic approach that was adopted for this project is to integrate an existing MAS modeling and development framework with an infrastructure supporting an effective form of 3D visualization of the dynamics generated by the model. In particular, to realize the second component we

[a]Complex Systems and Artificial Intelligence research center, University of Milano-Bicocca, via Bicocca degli Arcimboldi 8, 20126 Milano, {giuseppe.vizzari, giorgio.pizzi}@csai.disco.unimib.it
[b]Department of Computer Science, Institute of Mathematics and Statistics, Universidade de São Paulo, fcs@ime.usp.br
[1] http://secondlife.com

adopted Irrlicht[2], an open-source 3D engine and usable in C++ language. It is cross-platform, it can exploit OpenGL or DirectX libraries for 3D visualization, and it provides a performance level that we considered suitable for our requirements. It provides a high level API that was adopted for several projects related to 3D and 2D applications like games or scientific visualizations. The MAS modeling and development framework we adopted is a C++ porting and relevant refactoring of the original MMASS framework [2], aimed at adapting it to the different programming language and also at optimizing some mechanisms such as commonly adopted field diffusion algorithms. The overall framework was developed and tested in the Windows XP operating system, but it can be easily ported to MacOS X or Linux.

The overall architecture of the framework is shown in Figure 1. The following subsections will discuss the basic elements of this C++ version of the MMASS framework (MMASS module in the figure) and the infrastructure interfacing this module with the 3D visualization engine (MMASS UI manager in the figure).

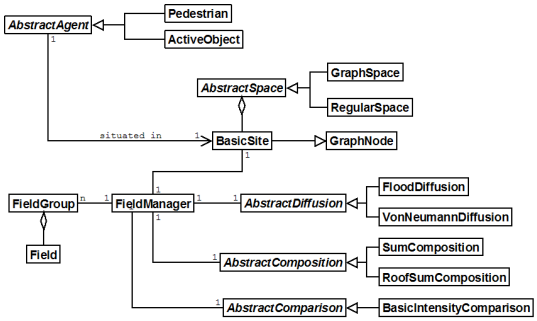


Figure 2 – Simplified class diagram of the part of the framework devoted to the realization of MMASS concepts and mechanisms.

A. *Supporting and Executing MMASS Models*

The MMASS framework adopted for this project is essentially a library developed in C++ providing proper classes to realize notions and mechanisms related to the SCA and MMASS models. In particular, a simplified class diagram of the MMASS framework is shown in Figure 2. The lower part of the diagram is devoted to the environment, and it is built around the BasicSite class. The latter is essentially a graph node (i.e. it inherits from the GraphNode class) that is characterized by the association with a FieldManager. The latter provides the services devoted to field management (diffusion, composition and comparison, defined as abstract classes). An abstract space is essentially an aggregation of sites, whose concretizations define proper adjacency geometries (e.g. regular spaces characterized by a Von Neumann adjacency or possibly irregular graphs).

An abstract agent is necessarily situated in exactly one site. Concrete agents defined for this specific framework are active objects (that are used to define concrete points of interest/reference to be adopted in a virtual environment) and pedestrians (that are basic agents capable of moving in the environment). Actual pedestrians and mobile agents that a developer wants to include to the virtual environment must be defined as subclasses of Pedestrian, overriding the basic behavioural methods and specifically the *action* method.

[2] http://irrlicht.sourceforge.net/

B. *Integrating the Models with a Realtime 3D Engine*

While the previous elements of the framework are devoted to the management of the behaviours of autonomous entities and of the environment in which they are situated, another relevant part of the described framework is devoted to the visualization of these dynamics. More than entering in the details of how the visualization library was employed in this specific context, we will now focus on how the visualization modules were integrated with the previously introduced MMASS framework in order to obtain indications on the scene that must be effectively visualized.

Figure 3 shows a simplified class diagram of the main elements of the 3D Engine Library. The diagram also includes the main classes that are effectively in charge of inspecting the state of the MMASS environment and agents, and of providing the relevant information to the SceneManager that will translate it into a scene to be visualized. The *Project* class act as a container of the 3D models providing the graphical representation of the virtual environment (*Model3D* objects), as well as the graph related to the adopted discretization of this physical space (a *Graph* object visually representing the previously discussed *physical layer*). It also includes a set of *Avatar* objects, that are three dimensional representations of *Pedestrian* objects (introduced in the previous subsection).

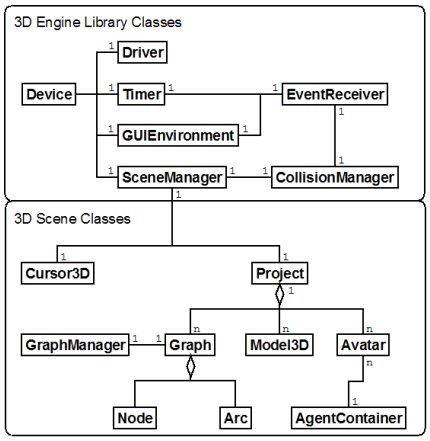


Figure 3 – Simplified class diagram of the part of the framework devoted to the management of the visualization of the dynamics generated by the model.

The framework must be able to manage in a coordinated way the execution of the model defined for the specific virtual environment and the updating of its visualization. To manage this coordinated execution of different modules and procedures three main operative modes have been defined and are supported by the framework. The first two are characterized by the fact that agents are not provided with a

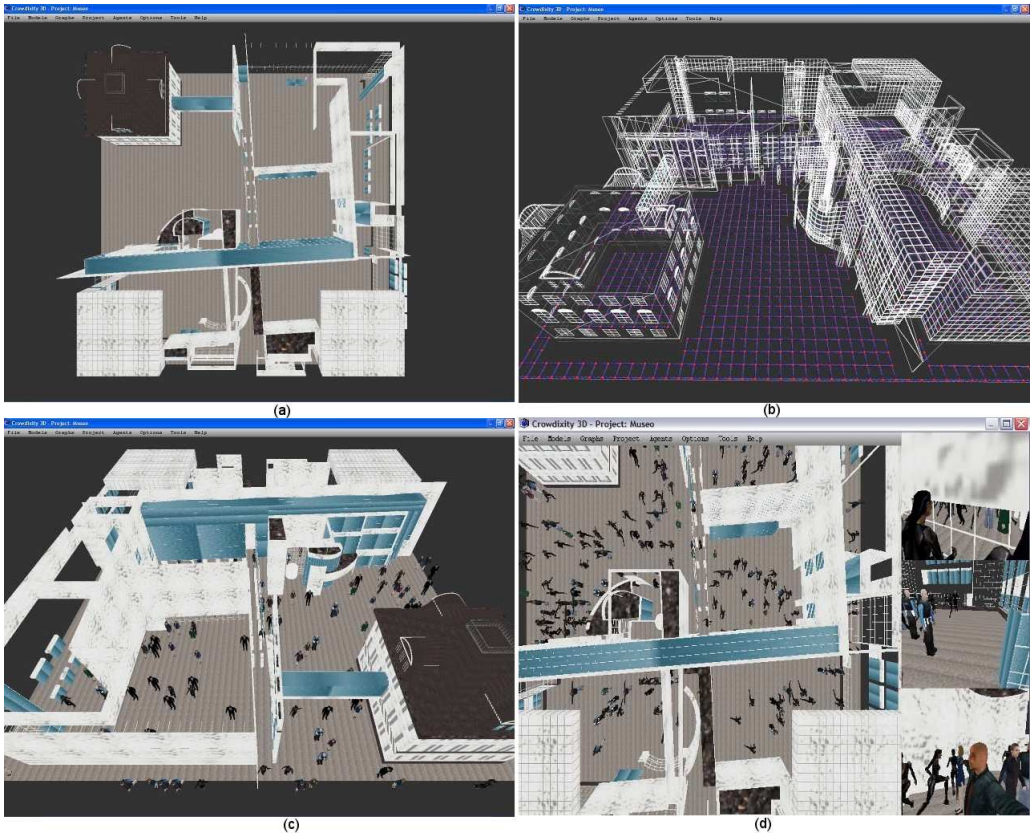


Figure 4 – Four screenshots of the virtual museum application, showing the structure of the environment - (a) and (b) – a perspective view of the evacuation and also a 'bird's eye' view of the environment coupled with three 'first-person' perspectives of agents - (c) and (d).

thread of control of their own. A notion of *turn* is defined and agents are activated to execute one action per turn, in a *sequential* way or in a conceptually *parallel* way (as for a Cellular Automaton). In this case, respectively after each agent action or after a whole turn the scene manager can update the visualization. On the other hand, agents might be *associated with a thread of control* of their own and no particular fairness policy is enforced. The environment, and more precisely the sites of the MMASS space, is in charge of managing possible conflicts on the shared resource. However, in order to support a fluid visualization of the dynamics generated by the execution of the MAS, the *Pedestrian* object before executing an action must coordinate with the related *Avatar*: if the previous movement was still not visualized, the action is temporarily blocked until the visualization engine has updated the scene. It must be noted that in all the introduced activation modes the environment is in charge of a regulation function [5] limiting agents' autonomy for sake of managing the consistency of the overall model or to manage a proper form of visualization.

III. Sample Applications

The aim of this section is to present a sample application to show how the framework supports the definition of MMASS models and the realization of an effective three dimensional visualization. The application was also chosen to show the potential of the framework in terms of execution of a large number of agents. Tests were carried out on a notebook on which the Windows XP Professional operating system was installed; the notebook was provided with an Intel Pentium IV 2.4 GHz processor, with 320 MB RAM and an ATI Raedon IGP graphic card with 128 MB (shared system memory).

The sample application is about the movement of agents inside a virtual museum; the aim of the agents in this scenario is to move outside the buildings to gather in specific areas, as in case evacuation. In this case the environment comprises around 2000 sites (a gross discretization of the represented environment) with around 6000 arcs connecting them; 500 agents were randomly positioned inside buildings, and they were provided with a thread of control of their own. Both the environment and agents were characterized by a 3D visual model, with textures; some relevant screenshots of this sample application are shown in Figure 4. Once again, the analytical results of this simulation are not relevant, since the agent models were extremely simple and they were not calibrated against real data. The simulation was executed

and visualized with a number of FPS constantly above 30.

We also executed a stress test on a different hardware configuration, to verify the scalability of the framework; the workstation was based on Windows XP Professional operating system, with an Intel Pentium Core 2 Duo 2.4 GHz, 2 GB RAM and a NVIDIA Quadro FX 3450 graphic card with 256 MB. The test environment was constituted by 11000 sites, connected by around 44000 arcs; 10000 agents, sequentially activated, were positioned in this environment. Their behaviour was simply to move towards the closest source of an 'exit' field; agents reaching the source were removed from the environment. The system was able to execute and visualize the simulation with 22 FPS, when the structure of the environment was hidden (reducing the number of displayed triangles), and with 3 FPS when it was visualized.

REFERENCES

[1] S. Bandini, S. Manzoni, C. Simone. Heterogeneous Agents Situated in Heterogeneous Spaces. Applied Artificial Intelligence, 16(9-10):831–852, 2002.

[2] S. Bandini, S. Manzoni, G. Vizzari. Situated Cellular Agents: a Model to Simulate Crowding Dynamics. IEICE - Transactions on Information and Systems: Special Section on Cellular Automata, Vol.E87-D(3):669-676, 2004.

[3] S. Bandini, S. Manzoni, G. Vizzari. Multi Agent Approach to Localization Problems: the Case of Multilayered Multi Agent Situated System. Web Intelligence and Agent Systems, IOS Press, 2(3):155-166, 2004.

[4] S. Bandini, S. Manzoni, G. Vizzari. Towards a platform for Multilayered Multi Agent Situated System based simulations: focusing on field diffusion. Applied Artificial Intelligence, Taylor & Francis, 20(4-5):327-351, 2006.

[5] S. Bandini, G. Vizzari. Regulation Function of the Environment in Agent-Based Simulation. Environments for Multi-Agent Systems III, Third International Workshop, E4MAS 2006, vol. 4389 of Lecture Notes in Computer Science, Springer-Verlag, pp. 157-169, 2007.

[6] M. Batty, A. Hudson-Smith. Urban Simulacra: From Real to Virtual Cities, Back and Beyond, *Architectural Design*, 75 (6):42-47, 2005.

[7] J. Dijkstra, H. P. J. Timmermans. Towards a multi-agent model for visualizing simulated user behavior to support the assessment of design performance. Automation in Construction 11:135-145, Elsevier, 2002.

[8] J. Dijkstra, J. Van Leeuwen, H. J. P. Timmermans. Evaluating Design Alternatives Using Conjoint Experiments in Virtual Reality. Environment and Planning B 30(3):357–367, 2003.

[9] T. Ishida, Y. Nakajima, Y. Murakami, H. Nakanishi. Augmented Experiment: Participatory Design with Multiagent Simulation. IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 1341-1346, 2007.

[10] M. Mamei, F. Zambonelli. Motion Coordination in the Quake 3 Arena Environment: A Field-Based Approach. Environments for Multi-Agent Systems, First International Workshop, E4MAS 2004, vol. 3374 of Lecture Notes in Computer Science, Springer-Verlag, pp. 264-278, 2005.

[11] H. Nakanishi, S. Nakazawa, T. Ishida, K. Takanashi, K. Isbister. Can Software Agents Influence Human Relations? - Balance Theory in Agent-mediated Communities. *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, ACM press, pp. 717-724, 2003.

[12] P. Nugues, S. Dupuy, A. Egges: Information Extraction to Generate Visual Simulations of Car Accidents from Written Descriptions. In: Computational Science and Its Applications - ICCSA 2003, vol. 2667 of Lecture Notes in Computer Science, Springer-Verlag, pp. 31-40, 2003.

[13] F. Nunnari, C. Simone. Perceiving awareness information through 3D representations. Proceedings of the working conference on Advanced Visual Interfaces, AVI 2004, ACM Press, pp. 443-446, 2004.

[14] G. Papagiannakis, S. Schertenleib, B. O'Kennedy, M. Arevalo-Poizat, N. Magnenat-Thalmann, A. J. Stoddart, D. Thalmann: Mixing virtual and real scenes in the site of ancient Pompeii. Journal of Visualization and Computer Animation 16(1):11-24, 2005.

[15] G. Vizzari, G. Pizzi, F. Soares Corrêa da Silva. A Framework for Interacting Situated Agents in Virtual Environment, WOA 2007 (this volume).

Conceptual Foundations of Interrogative Agents

Vincenzo Deufemia, Giuseppe Polese, Genoveffa Tortora, Mario Vacca
Dipartimento Matematica e Informatica
Università di Salerno
84084 Fisciano(SA), Italy
{deufemia, gpolese, tortora, mvacca}@unisa.it

***Abstract*—Reasoning by interrogation is one of the most ancient and experimented ways of reasoning. Originated by the Aristotelian *elenchus*, it has been used for many purposes, such as the resolution of mathematical and daily problems [25], [26], the discovery of new knowledge [19], [34], [36], the realization of questioning/answering processes [23]. In this paper we present the conceptual foundations of interrogative agents, a new model of BDI architecture based on interrogative logic. This model allows us to express the properties of agents in a natural way, and to use heuristics for reasoning. Finally, in order to explicate the whole approach and to highlight its main features we describe the application of interrogative agents in the context of database refactoring.**

I. Introduction

In recent years many different agent-based architectures and models have been proposed [39]. In this context, a well known architecture is the so called *Beliefs - Desires - Intentions* (*BDI*, for short), based on the concept of practical reasoning, i.e., the capability of resolving, through reflection, the problem of what to do and how to do it. Informally, this architecture is composed of four data structures: *Beliefs* representing the information that the agent has about the world, *Desires* representing the tasks that the agent has to accomplish, *Intentions* representing the sequence of actions to achieve the agent's desires, and *Plans* representing the procedural knowledge or Know-how. An *interpreter* is responsible for managing these structures.

In the last years, one of the most important issue faced by the agent community is flexibility, i.e., the agents ability to act in unknown situations or to face new situations. Franklin *et al.* defined this feature as the agents ability to have non scripted actions [13]. Many authors stressed the importance of flexibility. For instance, Barklund *et al.* took into account the problem of agent's flexibility observing that the reuse of an existing database in a different context calls for the ability of bridging the differences between the representation of input and the internal one [3]. According to the same authors, agents should be able to use different portions of knowledge depending on users or question classifications. Furthermore, they maintained the importance of using non-classical or non-deductive forms of reasoning and proposed the enrichment of agents with meta-knowledge. From the application point of view, Lin *et al.* showed that web-agents need to manage incomplete and partial information [24]. Therefore, the problem of modeling flexible agents is more a general one, involving both the way agents use knowledge and how they reason.

In this paper we face the problem of finding a BDI-like architecture for flexible agents. To this end, it is important to remark that the problem of flexibility has already been faced in the artificial intelligence field, yielding several models of intelligent systems [33], [34], [36]. These are based on problem solving and on interrogation as the underlying reasoning mechanism. The latter is considered one of the most suitable reasoning mechanisms in order to solve problems [25], [26]. Interrogation is one of the most ancient and experimented methods of reasoning [17]. It originated by the Aristotelian *elenchus* and it has been used for several purposes. This method of interrogation contains questions other than affirmations. It starts the reasoning process with a question, trying to find a plausible answer to it, after producing a sequence of questions and affirmations.

This paper is a foundational work aimed at obtaining an architecture that provides a higher degree of flexibility than other existing ones. The proposed interrogative BDI architecture containing both questions and affirmations. Questions correspond to *stimuli* (request to think) [25], [26], [36], and thinking is always an interrogation of an information source [18]. In this model desires are represented through unanswered *background questions*, intentions are more *urgent questions* to ask for, and *beliefs* contain affirmations as well as previously answered questions. Finally, heuristics, like in the Polya conception, are meta-knowledge questions to help transforming a question into another one. A processing model of the agent (usually called *cycle* in the autonomous agent literature) will be represented by a logical game [17].

The paper is organized as follows: Section 2 discusses the foundations of our proposal, and describes two models of reasoning by questioning. Section 3 introduces interrogative logic as the formalism underlying the proposed model. In Section 4 we describe the architecture of interrogative agents, whereas in Section 5 we describe the application of the proposed model to the problem of database refactoring. Finally, conclusions and further research are discussed in Section 6.

II. Conceptual Foundations of Interrogative Agents

In the last decades several intelligent systems have been presented and many models based on the interrogation paradigm have been developed [4], [19], [25], [29], [36]. According to the analysis of Jung concerning systems for scientific inquiry [19], it is possible to classify these models in two main

categories. The first one contains the models that are limited to the linguistic and logical aspects of questions and answers and to the relation question-answer (answerhood). The latter contains the models that deal with the methodological aspects of the question-answering process (*Q/A process*, for short) and, therefore, whose aim is to build interrogative-dialogical schemes. These models, named *I-D models*, are descendant from the Greek dialogical scheme and are based on four essential concepts: the dialogue is a *game*; the moves of the players are: *questioning*, *answering*, and *reasoning*; the aim of dialogic is to build well-organized *sequences of questions*; dialogical reasoning is useful to *discover* new hypothesis, theories, or general assumptions [19].

As observed by Jung "The dialogical scheme has some unique features that are ideal for the logic of discovery" [19]. The most important feature is that it can include non inferential moves (*heuristics*) in a natural way, since this kind of inferences are crucial in the context of discovery and problem solving. Another important feature is that I-D models are goal-oriented and the goal is expressed as a question to be answered. One problem with dialogical models is that dialogic is a content logic and, hence, domain-specific. In fact, the criticisms to these models in the A.I. field are related to the lack of an underlying formalism [21].

In the following we describe two main models of intelligent systems which reason by questioning: the model of Schank and the one of Polya. Both can be viewed as agents' models, and therefore they constitute a foundation for an interrogative model of agents.

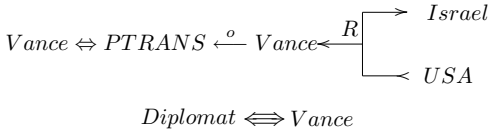
A. The Schank model of agent

The Schank group work [31]–[36] is a very large and long lasting, influencing or leading to different branches, trends and areas of research (see for example [1], [20], [22], [27]).

The Schank's theory models an intelligent (human or machine) agent able to learn occurring events and to devise plans for achieving goals. The agent is a problem solver one and its main reasoning tools are questions and interrogation. In the following, we show that the Schank's theory can be seen as a BDI model with an underlying interrogative reasoning.

1) The knowledge structures: There are different kinds of structures depending on the representation level: the actions are represented by conceptual dependencies [31], scripts and scenes are devoted to represent more complex situations [35], MOPs (Memory Organization Packets) and meta MOPs organize the high level knowledge [32], [33].

The theory of conceptual dependency (*CD theory*, for short) is a pictorial formalism developed for representing complex events through elementary ones. A *CD representation* of an event (also called *conceptualization*) is composed of objects linked together by rules. As an example, the conceptualization



means that the diplomat Vance goes from USA to Israel.

Scripts were introduced to model the daily life stereotypical situations, such as "eating in a restaurant" or "taking a plane" [35]. Scripts are frame-like knowledge structures and represent prototypical knowledge. They contain sequences of *scenes* involving a set of objects (*Props*) and a set of people (*Roles*). Scenes contain general actions aiming to reach the same goal. Entry conditions enable to activate scripts, whereas exit conditions give the status of *Roles* and *Props* after the script has been executed (used). For example, a script $DIPLOMATIC_VISIT could involve state secretaries and translators as *Roles*, and airplane and tables as *Props*. It can contain the scenes ARRIVAL and STATE_DINNER describing, respectively, what generally happens when diplomats *arrive* at a foreign state and what are the typical actions and states during a state dinner.

The model of Schank has also *goals* and *plans* [35], which are script-like structures. Indeed, both scripts and their successor MOPs can be considered as orderers of scenes and can also be used as plans [33].

In conclusion, the Schank model contains all the elements of a BDI architecture.

2) The tasks of Schank agents: Since the theory of Schank was born in the area of natural language processing, the task of the first Schank "agent" was *understanding* the occurring events. The first paradigm used was "*understanding as finding a place in the memory*" for the occurring events (e.g., see the systems MARGIE [31], SAM [35], or FRUMP [8]).

This first Schank model was very simple. In fact, in the light of theory of autonomous agents, the Schank understander was scantly autonomous, it had little or none ability to react to unknown stimuli, and it processed the same input always in the same way (i.e., it did not change through its experiences). To overcome these limitations, it was introduced the concept of memory reorganization (a form of *belief revision*), which enables the understander to learn by experience and, therefore, to react in a more timely way to external stimuli [32], [33]. In [34], [36], nevertheless, it was recognized that a serious drawback of the model were still the lack of *flexibility*, that is the attitude to change both its knowledge and its behavior to face the stimuli coming from the environment. The rigidity in the behavior was due to the systematic use of scripted activities (see Schank [36] for criticisms on script-based system). The introduction of *problem solving* in the Schank model made the systems more flexible since it enables to solve problems using the previously solved problems and stored knowledge, but also reasoning (i.e. applying both deduction and heuristics) or communication (asking other information sources for), and, hence, anomalous input or situations are treated like problems. Thus, a problem solving system trying to solve a problem will not stop the processing. In this way any activity becomes the solution to a problem which may not have a standardized solution.

3) The processing: The reasoning tools of the Schank theory are based on questions. Indeed, understanding is realized through the application of a structure of knowledge, which is a

process consisting in posing a set of questions on what could occur. The flexibility is fulfilled substituing the "filling the slots" concept of understanding by the more complex process of explanation, whose aim is to link anomalous inputs to the existent knowledge. In particular, the main phases of the explanation process [34], which embodies all the features of the questioning process, can be compared to the *cycle* of agent architectures:

a) *Finding an anomaly*
   This process starts with the application of the base questions, such as the scripts and the scenes of MOPs. If any of these questions do not obtain an answer, or the answers are different from the forecast ones, then an anomaly has been found and the system needs additional creative explanations.
b) *Posing the explanation question – Finding the explanation pattern*
   This process starts when an anomaly has been detected. The explanation question (*EQ*, for short) tries to explain the understanding failure by reformulating the question, and, if there exists an answer, it will be used as a new expectation.
c) *Reorganizing the memory*
   If there exists an answer to an EQ, the system tries to generalize and story it in memory. This is done by reminding similar cases, and by attempting to find a suitable generalization. Both reminding and generalizing are accomplished through an interrogative process.

Thus, the reasoning starts with a question and generates new ones as variations of existing questions, i.e., are obtained from another one by *transformation* which "is a way of getting an answerable question from an unanswerable one" [36], p. 287. Typical transformation mechanisms are specialization, generalization, simplification, and every one enabling to get an answer. Once an answer for the transformed question is found, the tweaking process will try to adapt it to the starting question.

Compared with the first approach based on the "question as expectation" paradigm, here the questions have an active role since they start elaboration processes and, hence, the association question-answer is dynamic.

The following very famous example about the meeting between Vance and Begin wives shows the working of the interrogative reasoning proposed by Schank:

"Q1. Did your wife ever meet Mrs. Begin?
Q2. Where would they have met?
Q3. Under what circumstance do diplomats' wives meet?
Q4. Under what circumstance do diplomats meet?
A4. On state visits to each other's countries.At international conferences.
A3. When they accompany their husbands on these visits.
Q3a. When did Vance go to Israel?
Q3b. When did Begin go to the U.S.?
A3a/A3b. Various dates can now be retrieved from memory.
Q3c. Did their wives accompany them on any of these trips?
A3c. A trip where this happened is found.
Q2a. During what part of a trip would wives meet?
A2a. During a state dinner.

Final revised question: Was there a state dinner on may 24th, 1977, during the diplomatic visit that Vance made to Israel with his wife?

Answer (A1): Probably on May 24, 1977, in Jerusalem at the state dinner at which they were both present." [35] pag. 286.

B. The Polya problem solving process

The model of problem solver proposed by Polya can also be described in terms of a BDI architecture [25], [26].

A problem solver requires two kinds of *knowledge*: one about abstract problems and the other about concrete (already solved) problems. Actually, the first one is a meta knowledge about the structure and the components of the problems.

The *cycle*, whose goal is to link the given problem to the existing knowledge in a closer and closer way, can be described by two lines going from the $Mobilization$ of knowledge to its connection to the problem ($Organization$) [26]:

1) $Mobilization \rightarrow Recognizing \rightarrow Regrouping \rightarrow Organization$
   Recognizing consists in examining the problem and *recognizing* some familiar features, whereas regrouping refers to a new way to see the existing elements after having recognized them. For example, the drawing of a bisector of the vertex angle in an isosceles triangle enables the regrouping of its elements in two equal triangles.
2) $Mobilization \rightarrow Remembering \rightarrow Supplementing \rightarrow Organization$
   When a feature has been recognized, it can allow to *remember* other problems with that feature or theorems about it. This new knowledge enriches the problem *supplementing* it by this new information.

The cycle is realized by questions. For instance, questions like "What is the unknown?", "What are the data?", "What is the condition?" can be useful for recognizing; the questions "Do you know a related problem?" or "Do you know a theorem that could be useful?" can help for remembering; "Could you restate the problem?", "Could you restate it still differently? Go back to definitions." aim to regroup (see [25] for a very rich list of questions).

However Polya warns that "Do not, however, use the checklist in a haphazard way, taking the questions at random, and do not use it mechanically, going through the questions in a fixed order. Instead, use this list of questions as an expert workman use his *tool chest*." [26].

III. The interrogative logic

A serious problem with models that reason by questioning, like the Schank's one, is the lack of underlying formalisms. In fact, it is well-known that the debate on the role of mathematical logic in artificial intelligence has characterized the developments of A.I. itself, producing currents of thought and different positions [21]. After the recent developments, we think that erotetic logic can constitute an adequate formalism for the models based on interrogation.

Erotetic logic (from the Greek word erotema meaning question) is the branch of logic studying the logic of questions and answers. According to the *received view* [5], [37], the task of erotetic logic is twofold: studying formal languages containing both questions and answers, and studying the relations between question and answer (answerhood). Nevertheless, many authors think that it is possible to reason (making inferences) by using both questions and answers [15], [17], [37], [38], yielding models close to the concept of Greek dialogic (the logic of reasoning by interrogation), as they recognize the importance of reasoning by questioning [19].

In this paper we abide by this conception of erotetic logic, which we refer to as dialogic, in order to avoid confusion with the received view. In particular, we think that the theory of Hintikka can be exploited to derive the logic formalism of our model [17].

Hintikka view of reasoning can be summerized as follows: a line of reasoning is constituted by a sequence of sentences; a new sentence in such a line is either obtained by deduction or (in the case of a rational agent) by asking for an information source (named *oracle* in the Hintikka terminology). Such rational line of reasoning is an interrogative process, which is performed by an *inquirer* [17].

The logic of Hintikka is modelled by a game played by the inquirer against one or more oracles, and the semantics used is the tableau method. Affirmations are on the left side of the tableau, the questions on the right side. The inquirer starts the moves, and the role of the oracles is to answer to the inquirer's questions. There are two kinds of moves: *logical inference moves* and *interrogative moves*. The formers are the typical deductive rules, and they are tableau-building rules (see [17] for a complete list).

Rules for questioning serve to generate questions. A rule for questioning is the following:

- If the presupposition of a question occurs on the left side of a subtableau, the inquirer may address the corresponding question to the oracle. If the oracle answers, the answer is added to the left side of the subtableau.

The system also has structural rules, which allow to manipulate the tableau.

IV. Interrogative Agents

The proposed model abides by the tradition of I-D models and is presented using some of the terminology by Jung [19].

Our ideal agent is Sherlock Holmes, whose behavior has been widely studied by philosophers and logicians (see [11] for a collection of essays). Sherlock Holmes is a problem solver with a diversified knowledge[1] and who is able to reason analytically[2]. Any activity starts with a problem (question) and it possibly ends with a plausible answer to that question. In order to solve a problem, the agent activates a *process of problem solving* which consists of linking the problem to the existing knowledge. According to the analytic method [6], a problem P is reduced to another problem that, if it is known or built, solves P. This is made trying to answer by reasoning or asking some other (also itself) for. Asking to itself means to search its own memory for some information.

Therefore, the agent has two abilities, *deducing* and *asking for/answering* (*communicating*). While deducing is a process of argumentative bridge-building, communicating serves to ask for other information sources. Also the task of an information source (named *oracle* in the Hintikka's terminology) is to answer questions, but the strategy used to accomplish it does not matter. Hence, an oracle can represent a mathematician using reasoning heuristics, an experimental physicist answering a question on the ground of experimental data, or it can implement vision recognition modules in computer systems.

A. The model

Formally, an *interrogative agent* is a quadruple

$$IA = (I, K, L, R_G)$$

where

I is an *interpreter*;

K is a set of *Information sources*. Three special sources of information are the *Problem source* P, the *Environment source* E, and the *Goal source* G;

L is the interrogative logic of the agent. It has a language constituted by two disjoint sets: Q (the set of possible *questions*) and A (the set of possible affirmations), and by rules like those in [17];

R_G is a set of rules (*guidance* rules).

The interpreter I has the goal to answer questions (called *principal questions* or *main problems*) according to the guidance rules R_G. To this aim it applies deductive rules or asks for another information source.

[1] "SHERLOCK HOLMES – his limits. 1. Knowledge of Literature. – Nil. 2. Philosophy. – Nil. 3. Astronomy. – Nil. 4. Politics. – Feeble. 5. Botany. – Variable. Well up in belladonna, opium, and poisons generally. Knows nothing of practical gardening. 6. Geology. – Practical, but limited. Tells at a glance different soils from each other. After walks has shown me splashes upon his trousers, and told me by their color and consistence in what part of London he had received them. 7. Chemistry. – Profound. 8. Anatomy. – Accurate, but unsystematic. 9. Sensational Literature. – Immense. He appears to know every detail of every horror perpetrated in the century. 10. Plays the violin well. 11. Is an expert singlestick player, boxer, and swordsman. 12. Has a good practical knowledge of British law." [10] pp. 13–14.

[2] "In solving a problem of this sort, the grand thing is to be able to reason backwards. That is a very useful accomplishment, and a very easy one, but people do not practice it much. In the every-day affairs of life it is more useful to reason forwards, and so the other comes to be neglected. There are fifty who can reason synthetically for one who can reason analytically." [10] pp. 115–116.

An information source is a couple (K_i, O_K_i), where K_i is the information store and O_K_i, named *oracle*, is the manager of the information in the store. The aim of an oracle is to answer questions and to make questions to the interpreter, by also using meta-knowledge or heuristics. In particular, an oracle

- retrieves knowledge from the source and stores new knowledge in it;
- generates questions starting from knowledge;
- revises the knowledge.

The *Goal* source contains information about the general goals of the processing agent. Typical goals are *plan coherency*, *contextual place*, *individual prediction*, *group prediction*, *new facts*, *rule copying*, *truths* [34]. Goals play an important role in our model. In fact, the model is goal-oriented, i.e., every problem the agent poses has to be filtered through the goals of the agent itself. This principle has its roots in the fact that many questions are very generic and become operative only when a goal is applied.

O_Goal is a kind of goal monitor [35], which selects the appropriate goal, generates the problem to solve, and asks for it to the interpreter (pro-activity). For instance, once selected the goal ACHIEVE(p), the oracle will generate the problem $?\exists S.(exit\_condition(S) = p)$ (that is "Is there a plan whose exit condition is p?"), and will ask the interpreter for it. The *O_Goal* could also generate new goals by unsolved problems or partially solved ones. This process can be summarized as follows:

$GOAL\ \ selection \rightarrow PROBLEM\ \ generation \rightarrow Solution \rightarrow NEW\ GOAL$

The Environment source E models the environment in which the agent operates.

As said above, the set of *guidance* rules R_G drives the interpreter in the achievement of its goals. Therefore, the R_G rules define the general behavior of the interpreter specifying the features of the game played by the interpreter.

The R_G rules are:

1) the interpreter plays a game with *O_Goal*;
   It builds a two columns tableau. Questions are placed on the right column (erotetic part of the tableau), whereas affirmations are placed on the left (assertoric part of the tableau).
2) the game starts with a question;
   This question is called *principal question*.
3) interrogative, assertoric, and communicative moves can be performed;
   Interrogative and assertoric moves are those in L. A communicative move consists in asking a question to an oracle, receiving an answer from an oracle, being asked a question by an oracle, answering back the oracle.
4) the first move is a communicative one: the interpreter asks the *O_Goal* for the principal problem (question) by asking it for "What is the problem associated with P" or, more simply, in absence of environmental commands "What is the problem?".
5) the game ends either when a conclusive answer is found (i.e., it is in the assertoric part of the tableau) [17] or when it is not possible to find it. In the first case the I wins, in the second the *O_Goal* wins.

B. The architecture

Figure 1 shows an example of architecture for an interrogative agent. It is composed of four sources of information: *Goals*, *DS-Knowledge*, *Environment*, and *Problems*, each one having associated an oracle, and an *Interpreter*.

The *Problems* source contains information about problems and the *O_Problems* is able to apply heuristics in order to transform a problem into another one (reduction method). For instance, the heuristic "generalization of individuals" could be informally described as: if a problem P contains one or more individuals a, search for a predicate F such that $F(a)$ holds and try to solve $P' \equiv P(a \leftarrow x)\&F(x)$. In order to apply such an heuristic, *O_Problems* will ask the interpreter for a feature F such that $F(a)$ holds, by a question like "$?K\exists F.F(a)$". In particular, let us consider the problem

$?\exists m.(meeting(m)\&Involve(m, X, Y)\&$
$wife(X, Vance)\&wife(Y, Begin))$

to be transformed by *O_Problems*. By applying the previously described rule, the oracle will ask the interpreter for the existence of a common feature of Vance and Begin. The interpreter, in turn, will ask this question to the *O_DS-Knowledge* which will answer that both Vance and Begin are diplomats. The interpreter will give this information to the *O_Problems* which, finally, will answer the first question

$\exists m.(meeting(m)\&Involve(m, X, Y)\&$
$wife(X, Z)\&wife(Y, T)\&diplomat(Z)\&diplomat(T))$

The *DS-Knowledge* source contains information about a specific domain. For instance, an agent able to solve the previous problem (i.e., the existence of a meeting involving the wives of Vance and Begin) should have knowledge about meetings and diplomatic meetings. In this case, the *O_DS-Knowledge* has the task to search the knowledge base in order to find information.

V. An application: the database refactoring

In this section we describe the application of the proposed model to the problem of database refactoring, which is introduced in the following.

Software refactoring is intended as the restructuring of an existing body of code, aiming to alter its internal structure without changing its external behavior [14]. It consists of a series of small behavior preserving transformations, which altogether can produce a significant software structural change. System modifications resulting in changes to the database structure are also relatively frequent [30]. These changes are particularly critical, since they affect not only the data, but also the application programs relying on them [2].

The refactoring is a very special problem, as it requires that the system coherently changes its own knowledge after a change occurred. If we look at the schema as a knowledge base, the refactoring becomes a process of changes in the

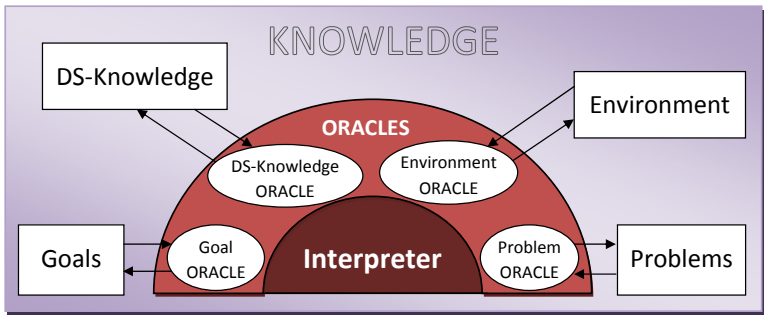


Fig. 1. The architecture of the proposed agent model.

knowledge, and hence it can be interpreted as an epistemic process. According to this view, it becomes natural to see refactoring as an agent managed process aiming to operate on the schema in order to perform the required changes, and trying to preserve original properties in terms of knowledge and queries [7].

The refactoring system can have many different goals, such as the checking of the schema consistency, supporting database administrators in the process of refactoring, or automatically apply suitable consistency maintenance changes. These examples of database refactoring support can be accomplished by three kinds of agents, each one characterized by some specific goals and abilities. For instance, the first agent needs only of deductive capabilities, the second one has a communicative nature, whereas the latter needs to know how to use some heuristics. According to the Schank classification [34], the goal of an agent performing the refactoring is *plan coherency*, i.e., the agent asks itself for the possibility to perform a certain change operation preserving coherency. Thus, the agent is modeled as a problem solver capable to perform changes which are triggered upon the detection of database schema anomalies.

More formally, a *refactoring interrogative agent* is the quadruple

$$R = (I, \{Schema, Problems, G, E\}, L, R_G)$$

Let us consider a database system storing data about employees of a company, and having a query for retrieving all employees of the Computer Science department. The *Schema* knowledge can be represented by

Attributes
$A = \{Employee\_ID, Name, Department\_ID, Salary, Address\}$

Tables
$T = \{R(Employee\_ID, Name, Department\_ID, Salary, Address)\}$

Functional dependencies
$F = \{f_1 : Employee\_ID \rightarrow Name;$
$f_2 : Employee\_ID \rightarrow Department\_ID;$
$f_3 : Employee\_ID \rightarrow Salary;$
$f_4 : Employee\_ID \rightarrow Address\}$

Queries
$Q = \{q(x, y, w, z) \equiv R(x, y, \text{“}CS\text{”}, w, z)\}$

Properties
$P = \{1) primary\_key(R, Employee\_ID)$
$2) \forall r \in T\ \exists k \subseteq Attr(r)$ such that $primary\_key(r, k)$
$3)\ key\_dep(r, k) \equiv \forall a \in (attr(r) - k)$
$(\exists f \in F$ such that $(LHS(f) = k \wedge RHS(f) = \{a\}) \wedge$
$(\neg \exists f$ such that $(LHS(f) \neq k \wedge RHS(f) = \{a\}))$
$4) \forall r \in T\ (primary\_key(r, k) \rightarrow key\_dep(r, k))\}$

The properties in P state that every relation has a primary key, and the attributes fully depend on the primary key only. $Schema$ also contains information about the ϵ operations. For instance, the splitting of a table t into two tables t' and t'', due to the introduction of a new functional dependency f and to the subsequent normalization process, can be defined as follows:

$split\_table(t, t', t'', f) \leftarrow$
$(A' = A\ \&$
$T' = (T - \{t\}) \cup \{t', t''\}\ \&$
$F' = F\ \&$
$Q' = \{q' |\ var(q') = var(q),\ body(q') = \rho(body(q), t, t' \& t'')\}\ \&$
$attr(t') = attr(t) - RHS(f)\ \&$
$attr(t'') = LHS(f) \cup RHS(f))$

An agent for the refactoring has three main problems to solve: *?Consistent(change-operation)*, *?Hold(p)*, and those generated by the predicate *Resolve(change-operation, p)*. The answer to the former is yes when the set of properties P' obtained by the application of the *change-operation* is consistent. The answer to the second one is yes when proposition p holds. Finally, the latter is true when proposition p holds after the application of *change-operation*. All of these problems can be expressed by epistemic logic using the K operator [17]. The K operator is applied to a proposition p using the expression Kp, whose meaning can be informally expressed by “it is known

that p". As a consequence, *Hold(p)* is simply expressible as Kp, *Consistent(change-operation*) as $K(\forall p \in \epsilon(P)).p$, whereas *Resolve(change-operation, p)* is nothing else that Kp applied after *change-operation*.

The Environment E is constituted by predicates modeling the ϵ operations.

The agent way of working is as follows. When E gives a command like $split\_table(T, T', T'', f_5 : Department\_ID \rightarrow Address)$ to the *O_Environment* and the latter, in turn, passes it to I, the interpreter has to decide what to do. The interpreter asks the *O_Goal* for the principal problem ("What is the problem associated with $split\_table(T, T', T'', f_5)$?").

Therefore, the *O_Goal* answers

$$Consistent(split\_table(T, T', T'', f_5))$$

and the principal problem is

$$?K(\forall p \in P).p$$

At this point the game starts. The interpreter calls the *O_DS-Knowledge* in order to answer the question "$?K(\forall p \in \epsilon(P)).p$". The answer will be communicated to the *O_Goal*. If the answer is negative, it is necessary to solve the problem trying to apply some heuristic, and the *O_Goal* will select the new goal to be achieved *fixing the incoherence* and the correspondent question:

$$?K\exists\epsilon'(\forall p \in \epsilon'(P)).p$$

A negative answer to this question means that it is not possible to find an ϵ' change operation directly, but that alternative strategies have to be used. To this end, the *O_Goal* can apply the goal *searching for the causes of incoherence* to which corresponds the question "Why is $\neg p$?" or the more concrete "What makes $\neg p$?".

In our concrete case the answer is

$$primary\_key(T'', Department\_ID) \notin P$$

The new goal is *fixing the incoherence*

$$?K\exists\epsilon' primary\_key(T'', Department\_ID) \in P$$

If the answer is negative, the *O_Problems* can suggest to generalize the question in

$$?K\exists\epsilon' p \in P$$

and

$$?K\exists\epsilon' x \in Y$$

this could require many refinement steps.

The answer is $\{add(x, Y)\}$ and therefore $\{add(p, P)\}$.

As this answer is obtained by generalization, it is uncertain (bracketed) and I can ask the environment for the possibility to apply it.

In conclusion, the process we propose is a *trial and error* one, during which the agent tries finding a solution to a given problem by deducting and/or consulting other information sources, possibly restating the problem. While doing so it is driven by its own goals.

VI. Conclusions and Future Works

In this paper we presented interrogative agents, a new model of agents that reason and communicate by using both affirmations and questions. The conceptual foundations of the model are manyfold. They can be found in the Greek dialogical, as well as in recent developments of interrogation logic (philosophy of science and logic), and in artificial intelligence (cognitive science and problem solving).

It is important to highlight both analogies and differences with respect to the classical BDI architecture. In fact, all of the structures of the BDI model are also in our interrogative model, and the functionalities of the BDI interpreter are provided through the interpreter and the oracles. On the other hand, the underlying logic of the proposed model is the logic of interrogation. Thus,our model has both deductive reasoning and heuristics.

The main advantage of the proposed approach is flexibility, which has historically been a characteristic of interrogation logic. However, several issues should further be investigated, such as the development of a logic formalism (based, for example, on the Hintikka formal logic [17]) to model the presented architecture. Moreover, we will focus our future works on the following aspects:

- the study of a logic of interrogation suitable for the interrogative architectures;
- the design of an interrogative agent-oriented language;
- the development of software tools to support the development of interrogative agent-based applications.

Finally, we plan to apply the proposed model in different application fields, such as, database refactoring [7], active video surveillance [9], e-learning of Euclidean plane geometry, and automated FAQ systems.

References

[1] A. Aamodt and E. Plaza, "Case-based reasoning: foundational issues, methodological variations, and system approaches", *Artificial Intelligence Communications*, vol. 7, no. 1, pp. 39–52, 1994.

[2] S. W. Ambler and P. J. Sadalage, *Refactoring Databases: Evolutionary Database Design*. Addison Wesley Professional, 2006.

[3] J. Barklund, S. Costantini, P. Dell'Acqua, and G.A. Lanzarone, "Metareasoning agents for query-answering systems", in T. Andreasen, H. Christiansen, and H. L. Larsen (Eds.), *Flexible Query-Answering Systems*, pp. 103–122, Kluwer Academic Publishers, 1997.

[4] S. Bromberger, *On What We Know We Don't Know: Explanation, Theory, Linguistics, and How Questions Shape Them*. The University of Chicago Press, 1992.

[5] N. D. Belnap, *The Logic of Questions and Answers*. Yale Univ. Press, 1976.

[6] C. Cellucci, *Filosofia e Matematica*. Laterza, 2002.

[7] S. K. Chang, V. Deufemia, G. Polese, and M. Vacca, "A logic framework to support database refactoring", to appear in *Proc. of 18th International Conference on Database and Expert Systems Applications (DEXA'07)*, Regensburg, Germany, 2007.

[8] G. F. De Jong, "Skimming newspaper stories by computer", in *Proc. of the 5th International Joint Conference on Artificial Intelligence (IJCAI '77)*, Cambridge, MA, 1977.

[9] V. Deufemia, M. Giordano, G. Polese, and M. Vacca, "A conceptual approach for active surveillance of indoor environments", to appear in *Proc. of International Conference on Distributed Multimedia Systems (DMS'07)*, San Francisco, USA, 2007.

[10] A. C. Doyle, *Sherlock Holmes: The Complete Novels and Stories*. Bantam Classics, 2006.

[11] U. Eco and T. A. Sebeok (eds.), *The Sign of Three: Peirce, Holmes, Dupin*, Indiana University Press, 1983.
[12] R. Fikes and N. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving", *Artificial Intelligence*, vol. 2, pp. 189–208, 1971.
[13] S. Franklin and A. C. Graesser, "Is it an agent, or just a program?: A taxonomy for autonomous agents", in *Proc. of ATAL'96*, 1996, pp. 21–35.
[14] B. Du Bois, P. Van Gorp, A. Amsel, N. Van Eetvelde, H. Stenten, S. Demeyer, and T. Mens, "A discussion of refactoring in research and practice", *Technical report*, no. 2004-03, University of Antwerp, Belgium, 2004.
[15] J. A. G. Groenendijk, "The logic of interrogation", in *Proc. of the Ninth Conference on Semantic and Linguistic Theory*, 1999.
[16] V. F. Hendricks, "Active agents", *Journal of Logic, Language and Information*, vol. 12, no. 4, pp. 469–495, 2003.
[17] J. Hintikka, I. Halonen, A. Mutanen, "Interrogative logic as a general theory of reasoning", in D. M. Gabbay, R. H. Johnson, H. J. Ohlbach, and J. Woods (eds.), *Handbook of the logic of argument and inference. The turn towards the practical*, North-Holland, Stud. Log. Pract. Reason., vol. 1, pp. 295–337, 2002.
[18] J. Hintikka, "L'épistémologie sans connaissance et sans croyance", *Journée de la Philosophie á l'UNESCO*, 2002.
[19] S. Jung, *An Interrogative Approach to Scientific Inquiry*. Peter Lang, 1996.
[20] J. L. Kolodner, "Reconstructive memory: A computer model", *Cognitive Science*, vol. 7, no. 4, pp. 281–328, 1983.
[21] R. A. Kowalski, "The limitation of logic", in *Proc. of ACM Conference on Computer Science*, 1986, pp. 7–13.
[22] W. G. Lehnert, N. G. Dyer, P. N. Johnson, C. J. Yang, and S. Harley, "BORIS - An experiment in in-depth understanding of narratives", *Artificial Intelligence*, vol. 20, no. 1, pp. 15–62, 1982.
[23] W. G. Lehnert, *The Process of Question Answering*. Erlbaum Associates, 1978.
[24] S. de Lin and C. Knoblock, "SERGEANT: A framework for building more flexible web agents by exploiting a search engine", *Journal of Web Intelligence and Agent Systems*, vol. 3, no. 1, pp. 1–15, 2005.
[25] G. Polya, *How to Solve It*. 2nd edition, Princeton University Press, 1957.
[26] G. Polya, *Mathematical Discovery: On Understanding, Learning and Teaching Problem Solving, Combined*. Wiley Press, 1981.
[27] A. Ram, "A theory of questions and question asking", *The Journal of the Learning Sciences*, vol. 1, no. 2/3, pp. 273–318, 1991.
[28] A. S. Rao, M. P. Georgeff, "Modeling rational agents within a BDI-architecture", in *Proc. of KR'91*, 473–484, 1991.
[29] N. Rescher, *Inquiry Dynamics*. Transaction Publishers, 2000.
[30] J. F. Roddick, "A survey of schema versioning issues for database systems", *Information and Software Technology*, vol. 37, no. 7, pp. 383–393, 1995.
[31] R. C. Schank, *Conceptual Information Processing*, North-Holland Publishing Co., 1975.
[32] R. C. Schank, "Language and memory", *Cognitive Science*, vol. 4, no. 3, pp. 243–284, 1980.
[33] R. C. Schank, *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, 1982.
[34] R. C. Schank, *Explanation Patterns. Understanding Mechanically and Creatively*. Lawrence Erlbaum Associates, 1986.
[35] R. C. Schank and R. Abelson, *Script Plans Goals and Understanding*. Lawrence Erlbaum Associates, 1977.
[36] R. C. Schank and P. Childers, *The Creative Attitude: Learning to Ask and Answer the Right Questions*. Macmillan, New York, 1988.
[37] A. Wisniewski, *The Posing of Questions. Logical Foundations of Erotetic Inferences*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.
[38] A. Wisniewski, "Socratic proofs", *Journal of Philosophical Logic*, vol. 33, n. 3, pp. 299–326, 2004.
[39] M. Wooldridge, "Agent-based computing", *Interoperable Communication Networks*, vol. 1, no. 1, pp. 71–97, 1998.

Declarative representation of curricula models: an LTL- and UML-based approach

Matteo Baldoni, Cristina Baroglio,
Giuseppe Berio, and Elisa Marengo
Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
{baldoni,baroglio,berio}@di.unito.it
elisa.mrng@gmail.com

***Abstract*—In this work, we present a constrained-based representation for specifying the goals of "course design", that we call curricula model, and introduce a graphical language, grounded into Linear Time Logic, to design curricula models which include knowledge of proficiency levels. Based on this representation, we show how model checking techniques can be used to verify that the user's learning goal is supplied by a curriculum, that a curriculum is compliant to a curricula model, and that competence gaps are avoided. This proposal represents the most recent advancement of a work, carried on in the last years, in which we are investigating the use of both agents and web services for building and validating curricula. We also outline future research directions.**

I. Introduction and motivations

As recently underlined by other authors, there is a strong relationship between the development of peer-to-peer, (web) service technologies and e-learning technologies [22]. The more learning resources are freely available through the Web, the more modern e-learning management systems (LMSs) should be able to take advantage from this richness: LMSs should offer the means for easily retrieving and assembling e-learning resources so to satisfy specific users' learning goals, similarly to how (web) services are retrieved and composed [17]. In [6], we have shown the possibility of automatically composing SCORM [1] courseware by exploiting semantic web technology and, in particular, LOM annotations. More rcently [3], we have developed a reasoning service that has been integrated in the Personal Reader framework, a service-oriented learning platform. The reasoning service is basically a planner, which can build curricula in a goal-driven way, where the goal is a set of desired competences. The reasoner is invoked in a service-oriented fashion to help a user and build a curriculum. To this aim, the reasoner is fed with a set of initial competences that the user has, the competences that the user would like to acquire, and the URL of a repository of descriptions of courses, given as RDF triples.

Besides building curricula, there are other interesting tasks that can be performed. Some of these concern curricula which are supplied directly by users. As in a composition of web services it is necessary to verify that, at every point, all the information necessary to the subsequent invocation will be available, in a learning domain, it is important to verify that all the *competencies*, i.e. the *knowledge*, necessary to fully understand a learning resource are introduced or available before that learning resource is accessed. The composition of learning resources, a curriculum, does not have to show any *competence gap*. Unfortunately, this verification, as stated in [15], is usually performed *manually* by the learning designer, with hardly any guidelines or support.

A recent proposal for automatizing the competence gap verification is done in [22] where an analysis of pre- and post-requisite annotations of the Learning Objects (LO), representing the learning resources, is proposed. A logic based validation engine can use these annotations in order to validate the curriculum/LO composition. Melia and Pahl's proposal is inspired by the CocoA system [12], that allows to perform the analysis and the consistency check of static web-based courses. Competence gaps are checked by a prerequisite checker for *linear courses*, simulating the process of teaching with an overlay student model. Pre- and post-requisites are represented as "concepts".

Together with the verification of consistence gaps, there are other kinds of verification. Brusilovsky and Vassileva [12] sketch some of them. In our opinion, two are particularly important: (a) verifying that the curriculum allows to achieve the users' *learning goals*, i.e. that the user will acquire the desired knowledge, and (b) verifying that the curriculum is compliant against the *course design goals*. Manually or automatically supplied curricula, developed to reach a learning goal, should match the "design document", a *curricula model*, specified by the institution that offers the possibility of personalizing curricula. Curricula models specify general rules for designing sequences of learning resources (courses). We interpret them as *constraints*, that are expressed in terms of concepts and, in general, are not directly associated to learning resources, as instead is done for pre-requisites. They constrain the process of acquisition of concepts, independently from the resources.

The availability of languages for designing curricula models, in a way that can automatically be processed by a reasoning system (be it an agent or a service) is a fundamental milestone in the development of checkers that perform the verifications described above, so to supply the the user and, when present, also the organization which supplies the courses, with a

complete set of tools to develop personalized, sound and complete curricula.

In this paper we present a constraint-based representation of curricula models. Constraints are expressed as formulas in a temporal logic (LTL, linear temporal logic [16]) represented by means of a simple graphical language that we call DCML (*Declarative Curricula Model Language*). This logic allows the verification of properties of interest for all the possible executions of a model, which in our case corresponds to the specific curriculum. Curricula are represented as *activity diagrams* [2]. We translate an activity diagram, that represents a curriculum, in a *Promela* program [21] and we check, by means of the well-known SPIN Model Checker [21], that it respect the model by verifying that the set of LTL formulas are satisfied by the Promela program. Moreover, we check that learning goals are achieved, and that the curriculum does not contain competence gaps. This work also improves the proposal of [9], where we did not consider the duration of courses and the fact that they may (partially) overlap. This leads to a different representation based on the concept of *milestones*. As in [15], we distinguish between *competency* and *competence*, where by the first term we denote a concept (or skill) while by the second we denote a competency plus the level of proficiency at which it is learnt or known or supplied. So far, we do not yet tackle with "contexts", as defined in the competence model proposed in [15], which will be part of future work.

This approach differs from previous work [7], where we presented an adaptive tutoring system, that exploits *reasoning about actions and changes* to plan and verify curricula. The approach was based on abstract representations, capturing the *structure* of a curriculum, and implemented by means of prolog-like logic clauses. Such representations were applied a procedure-driven form of planning, in order to build personalized curricula. In this context, we proposed also some forms of verification, of competence gaps, of learning goal achievement, and of whether a curriculum, given by a user, is compliant to the "course design" goals. The use of procedure clauses is, however, limiting because they, besides having a *prescriptive* nature, pose very strong constraints on the sequencing of learning resources. In particular, clauses represent what is "legal" and whatever sequence is not foreseen by the clauses is "illegal". However, in an open environment where resources are extremely various, they are added/removed dynamically, and their number is huge, this approach becomes unfeasible: the clauses would be too complex, it would be impossible to consider all the alternatives and the clauses should change along time.

For this reason we considered as appropriate to take another perspective and represent only those constraints which are strictly necessary, in a way that is inspired by the so called *social approach* proposed by Singh for multi-agent and service-oriented communication protocols [23], [24]. In this approach only the *obligations* are represented. In our application context, obligations capture relations among the times at which different competencies are to be acquired. The advantage of this representation is that we do not have to represent all that is legal but only those *necessary conditions* that characterize a legal solution. To make an example, by means of constraints we can request that a certain knowledge is acquired before some other knowledge, without expressing what else is to be done in between. If we used the clause-based approach, instead, we should have described also what can legally be contained between the two times at which the two pieces of knowledge are acquired. Generally, the constraints-based approach is more flexible and more suitable to an open environment.

II. DCML: A Declarative Curricula Model Language

In this section we describe the *Declarative Curricula Model Language* (DCML, for short), a graphical language to represent the specification of a curricula model (the course design goals). The advantage of a graphical language is that drawing, rather than writing, constraints facilitates the user, who needs to represent curricula models, allowing a general overview of the relations which exist between concepts. DCML is inspired by DecSerFlow, the Declarative Service Flow Language to specify, enact, and monitor web service flows by van der Aalst and Pesic [26]. DCML, as well as DecSerFlow, is grounded in Linear Temporal Logic [16] and allows a curricula model to be described in an easy way maintaining at the same time a rigorous and precise meaning given by the logic representation. LTL includes temporal operators such as next-time ($\bigcirc\varphi$, the formula φ holds in the immediately following state of the run), eventually ($\Diamond\varphi$, φ is guaranteed to eventually become true), always ($\Box\varphi$, the formula φ remains invariably true throughout a run), until (α U β, the formula α remains true until β), see also [21, Chapter 6]. The set of LTL formulas obtained for a curricula model are, then, used to verify whether a curriculum will respect it [5]. The adoption of a graphical language with a logical grounding allows designers, who cannot be expected to feel comfortable with the logical notation, to take advantage of automatic tools for the verification of the various kinds of properties mentioned in the introduction. As an example of curricula model, Fig. 1 shows a curricula model expressed in DCML. Every box contains at least one competence. Boxes/competences are related by arrows, which represent (mainly) temporal constraints among the times at which they are to be acquired. Altogether the constraints describe a curricula model.

A. Competence, competency, and basic constraints

The terms *competence* and *competency* are used, in the literature concerning professional curricula and e-learning, to denote the "effective performance within a domain at some level of proficiency" and "any form of knowledge, skill, attitude, ability or learning objective that can be described in a context of learning, education or training". In the following, we extend a previous proposal [5], [10] so as to include a representation of the *proficiency level* at which a competency

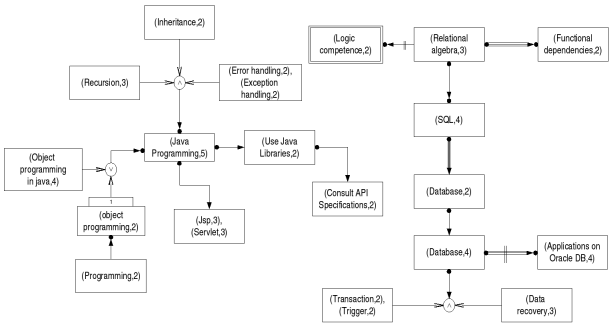


Fig. 1. An example of curricula model in DCML.

is owned or supplied. To this aim, we associate to each competency a variable k, having the same name as the competency, which can be assigned natural numbers as values. The value of k denotes the proficiency level; zero means absence of knowledge. Therefore, k encodes a *competence*, Fig. 2(a). On competences, we can define three basic *constraints*.

The "*level of competence*" constraint, Fig. 2(c), imposes that a certain competency k must be acquired at least at level l. It is represented by the LTL formula $\Diamond(k \geq l)$. Similarly, a course designer can impose that a competency must never appear in a curriculum with a proficiency level higher than l. This is possible by means of the "*always less than level*" constraint, shown in Fig. 2(d). The LTL formula $\Box(k < l)$ expresses this fact (it is the negation of the previous one). As a special case, when the level l is one ($\Box(k < 1)$), the competency k must never appear in a curriculum.

The third constraint, represented by a double box, see Fig. 2 (b), specifies that k must belong to the initial knowledge with, at least, level l. In other words, the simple logic formula $(k \geq l)$ must hold in the initial state.

To specify relations among concepts, other elements are needed. In particular, in DCML it is possible to represent *Disjunctive Normal Form* (DNF) formulas as *conjunctions* and *disjunctions* of concepts. For the sake of semplicity, in the next section we present the various constraints that can be expressed by DCML without using DNF, the interested reader can find the extension in the appendix.

B. Positive and negative relations among competences

Besides the representation of competences and of constraints on competences, DCML allows to represent *relations* among competences. For simplicity, in the following presentations we will always relate simple competences, it is, however, of course possible to connect DNF formulas. We will denote by (k, l) the fact that competence k is required to have at least level l (i.e. $k \geq l$) and by $\neg(k, l)$ the fact that k is required to be less than l.

Arrows ending with a little-ball, Fig. 2(f), express the *before* temporal constraint between two competences, that amount to require that (k_1, l_1) holds *before* (k_2, l_2). This constraint can be used to express that to understand some topic, some proficiency of another is required as precondition. It is important to underline that if the antecedent never becomes true, also the consequent must be invariably false; this is expressed by the LTL formula $\neg(k_2, l_2)\ \mathsf{U}\ (k_1, l_1)$, i.e. $(k_2 < l_2)\ \mathsf{U}\ (k_1 \geq l_1)$. It is also possible to express that a competence must be acquired *immediate before* some other. This is represented by means of a triple line arrow that ends with a little-ball, see Fig. 2(i). The constraint (k_1, l_1) *immediate before* (k_2, l_2) imposes that (k_1, l_1) holds before (k_2, l_2) and the latter either is true in the next state w.r.t. the one in which (k_1, l_1) becomes true or k_2 *never* reaches the level l_2. The difference w.r.t the *before* constraint is that it imposes that the two competences are acquired *in sequence*. The corresponding LTL formula is "(k_1, l_1) *before* (k_2, l_2)" $\wedge\Box((k_1, l_1) \supset (\bigcirc(k_2, l_2) \vee \Box\neg(k_2, l_2)))$.

Both of the two previous relations represent temporal constraints between competences. The *implication* relation (Fig. 2(e)) specifies, instead, that if a competency k_1 holds at least at the level l_1, some other competency k_2 must be acquired sooner or later at least at the level l_2. The main characteristic of the implication, is that the acquisition of the consequent is imposed by the truth value of the antecedent, but, in case this one is true, it does not specify when the consequent must be achieved (it could be before, after or in the same state of the antecedent). This is expressed by the LTL formula $\Diamond(k_1, l_1) \supset \Diamond(k_2, l_2)$. The *immediate implication* (Fig. 2(h)), instead, specifies that the consequent must *hold* in the state right after the one in which the antecedent is acquired. Note that, this does not mean that it must be *acquired* in that state, but only that it cannot be acquired after. This is expressed by the LTL implication formula in conjunction with the constraint that whenever $k_1 \geq l_1$ holds, $k_2 \geq l_2$ holds in the next state: $\Diamond(k_1, l_1) \supset \Diamond(k_2, l_2) \wedge \Box((k_1, l_1) \supset \bigcirc(k_2, l_2))$.

The last two kinds of temporal constraint are *succession* (Fig. 2(g)) and *immediate succession* (Fig. 2(j)). The *succession* relation specifies that if (k_1, l_1) is acquired, afterwards (k_2, l_2) is also achieved; otherwise, the level of k_2 is not important. This is a difference w.r.t. the *before* constraint where, when the antecedent is never acquired, the consequent

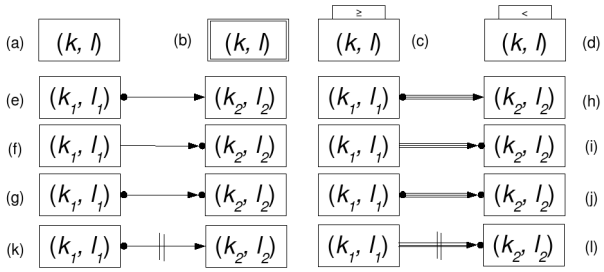


Fig. 2. Competences (a) and basic constraints (b), (c), and (d). Relations among competences: (e) implication, (f) before, (g) succession, (h) immediate implication, (i) immediate before, (j) immediate succession, (k) not implication, (l) not immediate before.

must be invariably false. Indeed, the *succession* specifies a condition of the kind *if* $k_1 \geq l_1$ *then* $k_2 \geq l_2$, while *before* represents a constraint without any conditional premise. Instead, the fact that the consequent must be acquired after the antecedent is what differentiates *implication* from *succession*. Succession constraint is expressed by the LTL formula $\Diamond(k_1, l_1) \supset (\Diamond(k_2, l_2) \wedge (\neg(k_2, l_2)\ \mathsf{U}\ (k_1, l_1)))$. In the same way, the *immediate succession* imposes that the consequent either is acquired in the same state as the antecedent or in the state immediately after (not before nor later). The immediate succession LTL formula is "(k_1, l_1) *succession* (k_2, l_2)" $\wedge\Box((k_1, l_1) \supset \bigcirc(k_2, l_2))$.

After the "positive relations" among competences, let us now introduce the graphical notations for "negative relations". The graphical representation is very intuitive: two vertical lines break the arrow that represents the constraint, see Fig. 2(k)-(l). (k_1, l_1) *not before* (k_2, l_2) specifies that k_1 cannot be acquired up to level l_1 before or in the same state when (k_2, l_2) is acquired. The corresponding LTL formula is $\neg(k_1, l_1)\ \mathsf{U}\ ((k_2, l_2) \wedge \neg(k_1, l_1))$. Notice that this is not obtained by simply negating the before relation but it is weaker; the negation of *before* would *impose the acquisition* of the concepts specified as consequents (in fact, the formula would contain a strong until instead of a weak until), the *not before* does not. The *not immediate before* is translated exactly in the same way as the *not before*. Indeed, it is a special case because our domain is monotonic, that is a competency acquired at a certain level cannot be forgotten.

(k_1, l_1) *not implies* (k_2, l_2) expresses that if (k_1, l_1) is acquired k_2 cannot be acquired at level l_2; as an LTL formula: $\Diamond(k_1, l_1) \supset \Box\neg(k_2, l_2)$. Again, we choose to use a weaker formula than the natural negation of the implication relation because the simple negation of formulas would impose the presence of certain concepts. (k_1, l_1) *not immediate implies* (k_2, l_2) imposes that when (k_1, l_1) holds in a state, $k_2 \geq l_2$ must be false in the immediately subsequent state. Afterwards, the proficiency level of k_2 does not matter. The corresponding LTL formula is $\Diamond(k_1, l_1) \supset (\Box\neg(k_2, l_2) \vee \Diamond((k_1, l_1) \wedge \bigcirc\neg(k_2, l_2)))$, that is weaker than the "classical negation" of the *immediate implies*.

The last relations are *not succession*, and *not immediate succession*. The first imposes that a certain competence cannot be acquired after another, (either it was acquired before, or it will never be acquired). As LTL formula, it is $\Diamond(k_1, l_1) \supset (\Box\neg(k_2, l_2) \vee$ "(k_1, l_1) *not before* (k_2, l_2)"). The second imposes that if a competence is acquired in a certain state, in the state that follows, another competence must be false, that is $\Diamond(k_1, l_1) \supset (\Box\neg(k_2, l_2) \vee$ "(k_1, l_1) *not before* (k_2, l_2)" $\vee\Diamond((k_1, l_1) \wedge \bigcirc\neg(k_2, l_2)))$.

In Fig. 1, some examples of constraints are represented. Conjunctions and disjunctions are represented by connecting different competences (boxes) with and/or circles. For instance, *Object programming in Java* is required at least at level 4 **or** *Object programming* is required at least at level 2, before the competence *Java Programming* can be acquired (at least at level 5).

Another example is the implication that occurs, for instance, between *Database*, at least at level 2, and *Database*, at least at level 4. This relation means that Database at level 2 is not sufficient and, when it is acquired, sooner or later the student must increase its knowledge at least at level 4.

The competence (Database,4) is also connected with an *not immediate succession* constraint to the competence (Application on Oracle DB,4). This constraint can be interpreted as the intention to let the student assimilate the knowledges on Database before applying them on a real case.

Note that this example is divided into two different areas, one concerning programming competences and one about databases. There are no connection between competences of the two parts. Anyway all the constraints must be checked on the curriculum.

III. Representing curricula as activity diagrams

Let us now consider specific curricula. In the line of [7], [4], [5], we represent curricula as sequences of courses/resources, taking the abstraction of courses as simple actions. Any action can be executed given that a set of preconditions holds; by executing it, a set of post-conditions, the effects, will become true. Specifically, courses are seen as actions for acquiring some concepts (*effects*) given that the student owns some competences (*preconditions*). So, a curriculum is seen as a sequence of actions that causes *transitions* from the initial set

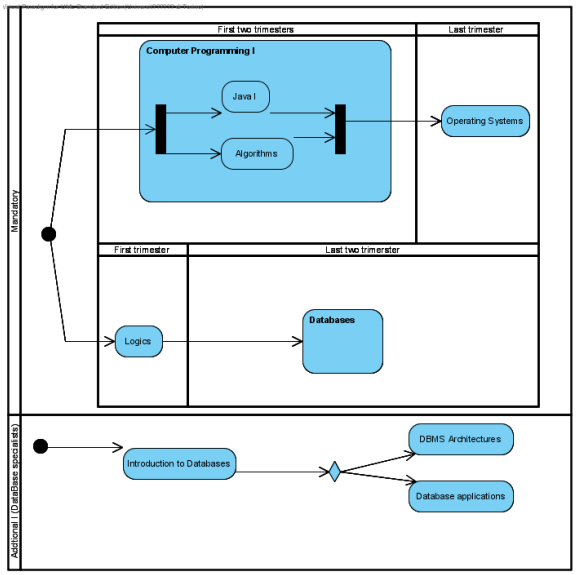


Fig. 3. Activity diagram representing a curriculum with mandatory and additional, student chosen, courses. Swimlanes represent the sequencings of courses. Vertical divisions capture the different milestones (trimesters).

of competences (possibly empty) of a user up to a final state that will contain also the acquired competences. We assume that concepts can only be added to states and competence level can only grow by executing the actions of attending courses (or more in general reading a learning material). The intuition behind this assumption is that new course do not erase the concepts acquired previously, thus knowledge grows incrementally.

Generally speaking, a curriculum may be represented with one or several sequences of courses to be attended, in alternative or as obligations. As a consequence, it seems very natural representing a curriculum by, for instance, a UML *activity diagram* [2]. The diagram represents essentially the "student personal process" to achieve the final degree. Apart its standard meaning and visualisation, a UML activity diagram may contain actions with pre- and post- conditions, combined in complex paths and possibly aggregated. Actions or activities (if further decomposed) correspond to courses or other elements, used to fundamentally build any curriculum in an organisation. Activity diagrams are rich enough to represent alternative, intermediate statuses and conditional paths.

However, we found very useful two principles when representing a curriculum: To carefully distinguish courses with distinct duration (in time); To carefully distinguish mandatory courses and additional optional courses. Modelling a curriculum with these two principles in mind introduces *(i)* a decomposition level and *(ii)* partitions among courses, being these courses from mandatory or from additional partitions.

Actvity diagrams are well suited for representing curricula under the two principle reported above. Fig. 3 reports an example with additional courses and distinguishes courses with distinct duration. The horizontal partition (*swimlanes* in UML) is corresponding to mandatory and additional courses (additional courses are for "database specialists" in this case). Vertical partitions provide information about actions and activities with distinct duration. In this case, we have used as time references the usual distinction implemented in Italian universities, in years and trimesters. The beginning/ending points of the trimesters correspond to a set of *milestones*; this temporal organization will be used to identify those states, at which the verification will be applied. In previous work, instead, courses were *atemporal* and each state was tied to the simulation of a single course. The introduction of durations allows a more realistic representation of the curricula and, especially, of the dependencies between competences. Therefore, we can easily see that the course "Logics" is delivered during the first trimester, while the course "Java I" is delivered in the first six months, being this java course part of an aggregated set of courses corresponding to the activity named "Computer Programming I". In the horizontal bottom swimlane, we are representing the fact that it is a student's choice to advance in the first year two courses of databases, once made the choice between "Database architecture" and "Database applications". The swimlanes representing additional courses can be used to represented once-time choice of the student. For instance, once the student has decided to become "database specialist", he has to complete the process represented in the swimlane. However, with additional swimlanes, we can also represent less stringent choices. In this case, however, there are typically no arrows

between courses and there is no final node. It should also be noted that processes representing a curriculum are only *views* combining activities and actions (i.e. the real taught courses). Intuitively, a course like "Network I" in a curriculum for system specialists is to be followed by "Network II"; however, the same is not required in a curriculum for "database specialists". This is very compatible with UML activity diagrams where it is possible to use reuse, in distinct contexts, activities and actions defined once.

More complex cases require special attention because hierarchical decomposition of the time-based partition does not apply directly. For instance, the case of where one two-trimester course overlaps in time with another two-trimester course. In this case, hierarchical time-based partition cannot be applied but it should be observed that the basic activity diagram is sufficient also in this case because it allows to represent the two courses in parallel. Indeed, due to overlapping, we cannot expect one to supply competences that are prerequisites for the other. Again, with reference to our example, in Fig. 3, the course "Databases" spans over the second and the third trimester, partially overlapping with the "Computer Programming 1" activity (spanning over the first and second trimester) and partly overlapping with "Operating Systems", in the third trimester.

UML 2.1.1 is extremely powerful for making partitions. Indeed, partitions apply to activities, and contain several edges and actions. This means that each activity can be independently partitioned in the diagram. However, the size of the visualised partitions does not make sense in UML (as well). Therefore, time overlapping can be shown by regulating the size and the relative position of the several visualised partitions; however, the "timed semantics" remains underspecified and may be approached in the classical way by introducing time-dependent constraints on activity edges (or, on top of the interpretation of the UML superstructure specification that often does not provide a sufficient level of detail constraints attached to the partitions themselves).

IV. Verifying curricula by means of SPIN Model Checker

In this section we discuss how to validate a curriculum. As explained, three kinds of verifications have to be performed: (1) verifying that a curriculum does not have competence gaps, (2) verifying that a curriculum supplies the user's learning goals, and (3) verifying that a curriculum satisfies the course design goals, i.e. the constraints imposed by the curricula model. To do this, we use *model checking techniques* [14].

By means of a *model checker*, it is possible to generate and analyze all the possible states of a program exhaustively to verify whether no execution path satisfies a certain property, usually expressed by a temporal logic, such as LTL. When a model checker refuses the negation of a property, it produces a *counterexample* that shows the violation. SPIN, by G. J. Holzmann [21], is the most representative tool of this kind. Our idea is to translate the activity diagram, that represents a set of curricula, in a Promela (the language used by SPIN) program, and, then to verify whether it satisfies the LTL formulas that represents the curricula model.

In the literature, we can find some proposals to translate UML activity diagrams into Promela programs, such as [18], [19]. These proposals have a different purpose than ours and they cannot directly be used to perform the translation that we need to perform the verifications we list above, however, it is possible to follow them as guidelines to perform our translation. Generally, their aim is debugging UML designs, by helping UML designers to write sound diagrams. The translation proposed in the following, instead, aims to simulate, by a Promela program the acquisition of competencies by attending courses contained into the curricula represented by an activity diagram.

Given a curriculum as an activity diagram, we represent all the competences involved by its courses as *integer variables*. In the beginning, only those variables that represent the initial knowledge owned by the student are set to a value greater than zero. *Courses* are represented as actions that can modify the value of such variables. Since our application domain is monotonic, the value of a variable can only grow.

The Promela program corresponds to a process, that contains the translation of the UML activity diagram and simulates the way competences are acquired, for *all* the curricula represented by the activity diagram, updating the set of the achieved competences at every step. Steps corrispond to the various milestones into which the curriculum is organized. For instance, in Fig. 3 we identify the initial state, a second state corresponding to the end of the first trimester, another corresponding to the end of the second trimester, and a final state, corresponding to the end of the curriculum.

```
proctype CurriculumVerification() {
  milestone_1();
  milestone_2();
  milestone_3();
  LearningGoal();
}
```

If the simulation of all its possible executions ends, then, there is no competence gap.

Each *course* is represented by its preconditions and its effects. For example, the course "Databases" is as follows:

```
inline preconditions_course_databases() {
  assert(logical_reasoning >= 4);
}
inline effects_course_databases() {
  SetCompetenceState(database, 2);
  SetCompetenceState(relational_algebra, 4);
  SetCompetenceState(ER_language, 4);
}
```

assert verifies the truth value of its condition, which in our case is the precondition to the course. If violated, SPIN interrupts its execution and reports about it. *SetCompetenceState* increases the level of the passed competence if its current level is lower than the second parameter. If all the curricula represented by the translated activity diagram have *no competence gaps*, no assertion violation will be detected. Otherwise, a counterexample will be returned that corresponds to an effective sequence of courses leading to the violation, giving a precise feedback

to the student/teacher/course designer of the submitted set of curricula.

Generally speaking, a milestone implements the act of adding to the state all the competencies that have been acquired within itself, plus the act of checking the applicability of the subsequent courses (those that will lead to the next milestone). Since each curriculum contains both mandatory and additional courses, the latter depending on a student's choice, every milestone verifies, by default, the mandatory courses and simulates the different alternatives concerning additional courses, which the student might has chosen. This is done by means of the introduction of a variable that is used to discriminate among the alternative paths. *Decision and merge nodes* can be used to represent such altenatives.

```
inline milestone() {
  atomic {
    preconditions_course_java_programming_II();
    if
    :: (path == 1) ->
       preconditions_course_logic();
    :: (path == 2) ->
       precondition_course_physics();
    :: else -> skip;
    fi;
    effects_course_java_programming_II();
    if
    :: (path == 1) ->
       effects_course_logic();
    :: (path == 2) ->
       effects_course_physics();
    :: else -> skip;
    fi;
  }
}
```

The test of the preconditions and the update of the state are performed as an atomic operation.

The last instruction of the process *CurriculumVerification*, which is applied only if all the curricula can be executed to their end, is *LearningGoal*. *LearningGoal* performs the check of the user's learning goal. This just corresponds to a test on the knowledge in the ending state. For example, a student interested in web and databases could have the goal:

```
inline LearningGoal()
{ assert(advanced_java_programming>=5
  && N_tier_architectures >= 4
  && relational_algebra>=2
  && ER_language>=2); }
```

To check if the curriculum complies to a curricula model, we check if every possibly sequence of execution of the Promela program satisfies the LTL formulas, now transformed into *never claims* directly by SPIN. The assertion verification is not computationally expensive. The automata generated from the Promela program encoding the first three years of courses at our University is still tractable. Also the verification of the temporal constraints is not hard if we check the constraints one at the time.

V. Conclusions

In this paper we have introduced a graphical language to describe curricula models as temporal constraints posed on the acquisition of competences (supplied by courses), therefore, taking into account both the concepts supplied/required and the proficiency level. We have also shown how model checking techniques can be used to verify that a curriculum complies to a curricula model, and also that a curriculum both allows the achievement of the user's learning goals and that it has no competence gaps. This use of model checking is inspired by [26], where LTL formulas are used to describe and verify the properties of a composition of Web Services. Another recent work, though in a different setting, that inspired this proposal is [25], where medical guidelines, represented by means of the GLARE graphical language, are translated in a Promela program, whose properties are verified by using SPIN. Similarly to [25], the use of SPIN, gives an *automa-based semantics* to a curriculum (the automaton generated by SPIN from the Promela program) and gives a declarative, formal, representation of curricula models (the set of temporal constraints) in terms of a LTL theory that enables other forms of reasoning. In fact, as for all logical theories, we can use an inference engine to derive other theorems or to discovery inconsistencies in the theory itself.

The presented proposal is an evolution of earlier works [8], [4], [7], where we applied semantic annotations to learning objects, with the aim of building compositions of new learning objects, based on the user's learning goals and exploiting planning techniques. That proposal was based on a different approach that relied on the experience of the authors in the use of techniques for reasoning about actions and changes which, however, suffers of the limitations discussed in the introduction. We are currently working on the automatic translation from a textual representation of DCML curricula models into the corresponding set of LTL formulas and from a textual representation of an activity diagram, that describes a curriculum (comprehensive of the description of all courses involved with their preconditions and effects), into the corresponding Promela program. We are also going to realize a graphical tool to define curricula models by means of DCML. We think to use the Eclipse framework, by IBM, to do this. In [3], we discuss the integration into the Personal Reader Framework [20] of a web service that implements an earlier version of the techniques explained here, which does not include proficiency levels. Last but not least, if in a University framework the notion of competence that we have used is sufficient to represent and reason about curricula, in business organizations this notion usually requires more complex models. As future work, we mean to integrate the proposed approach with the CRAI competence model [13] and with competence management information systems [11].

Acknowledgements.

The authors would like to thank Viviana Patti for the helpful discussions. This research has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net), and it has also been supported by MIUR PRIN 2005 "Specification and verification of agent interaction protocols" national

project.

REFERENCES

[1] ADL Technical Team. SCORM XML controlling document - SCORM CAM version 1.3 navigation XML XSD version 1.0, 2004. http://www.adlnet.org/.

[2] Unified Modeling Language: Superstructure, version 2.1.1. OMG, February 2007.

[3] M. Baldoni, C. Baroglio, I. Brunkhorst, E. Marengo, and V. Patti. Curriculum Sequencing and Validation: Integration in a Service-Oriented Architecture. In *Proc. of EC-TEL'07*, *LNCS*, 2007. Springer.

[4] M. Baldoni, C. Baroglio, and N. Henze. Personalization for the Semantic Web. In *Reasoning Web*, *LNCS 3564 Tutorials*, pp. 173–212. Springer-Verlag, 2005.

[5] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and L. Torasso. Verifying the compliance of personalized curricula to curricula models in the semantic web. In *Proc. of Int.l Workshop SWP'06, at ESWC'06*, pp. 53–62, 2006.

[6] M. Baldoni, C. Baroglio, V. Patti, and L. Torasso. Reasoning about learning object metadata for adapting SCORM courseware. In *Proc. EAW'04*, 2004.

[7] M. Baldoni, C. Baroglio, and V. Patti. Web-based adaptive tutoring: an approach based on logic agents and reasoning about actions. *Artificial Intelligence Review*, 22(1):3–39, 2004.

[8] M. Baldoni, C. Baroglio, V. Patti, and L. Torasso. Reasoning about learning object metadata for adapting SCORM courseware. In *Proc. of Int.l Workshop EAW'04, at AH 2004*, pp. 4–13, Eindhoven, The Netherlands, August 2004.

[9] M. Baldoni, C. Baroglio, and E. Marengo. Curricula model checking. In *Proc. of AIIA'07*. To appear.

[10] M. Baldoni and E. Marengo. Curricula model checking: declarative representation and verification of properties. In *Proc. of EC-TEL'07*, *LNCS*, 2007. Springer.

[11] G. Berio and M. Harzallah. Knowledge Management for Competence Management. *J. of Universal Knowledge Management*, 1:21–28, 2005.

[12] P. Brusilovsky and J. Vassileva. Course sequencing techniques for large-scale web-based education. *Int. J. Cont. Engineering Education and Lifelong learning*, 13(1/2):75–94, 2003.

[13] M. Harzallah and F. Vernadat. IT-based Competency Modeling and Management: from theory to practice in enterprise engineering and operations. *Computers in industry*, 48:157–179, 2002.

[14] O. E. M. Clarke and D. Peled. *Model checking*. MIT Press, 2001.

[15] J. L. De Coi, E. Herder, A. Koesling, C. Lofi, D. Olmedilla, O. Papapetrou, and W. Sibershi. A model for competence gap analysis. In *Proc. of WEBIST 2007*.

[16] E. A. Emerson. Temporal and model logic. In *Handbook of Theoretical Computer Science*, volume B, pages 997–1072. Elsevier, 1990.

[17] R. Farrell, S. D. Liburd, and J. C. Thomas. Dynamic assebly of learning objects. In *Proc. of WWW 2004*, New York, USA, May 2004.

[18] M. del Mar Gallardo, P. Merino, and E. Pimentel. Debugging UML Designs with Model Checking. *Journal of Object Technology*, 1(2):101–117, July-August 2002.

[19] N. Guelfi and A. Mammar. A Formal Semantics of Timed Activity Diagrams and its PROMELA Translation. In *Proc. of APSEC'05*, pp. 283–290. 2005.

[20] N. Henze and D. Krause. Personalized access to web services in the semantic web. In *The 3rd Int.l Workshop SWUI, at ISWC 2006*, 2006.

[21] G. J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2003.

[22] M. Melia and C. Pahl. Automatic Validation of Learning Object Compositions. In Proc. of *IT&T'2005: Doctoral Symposium*, Carlow, Ireland, 2006.

[23] M. P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.

[24] M. P. Singh. A social semantics for agent communication languages. In *In Issues in Agent Communication*, number 1916 in LNCS, pages 31–45. Springer, 2000.

[25] P. Terenziani, L. Giordano, A. Bottrighi, S. Montani, and L. Donzella. SPIN Model Checking for the Verification of Clinical Guidelines. In *Proc. of ECAI 2006 Workshop on AI techniques in healthcare*, Riva del Garda, August 2006.

[26] W. M. P. van der Aalst and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In *Proc. of WS-FM'06*, LNCS, 2006. Springer.

APPENDIX

Let k be a competence, we denote by (k, l) the constraint $k \geq l$ and by $\neg(k, l)$ the constraint $k < l$. A *conjuctive competence formula* cf is a conjuction of atomic competence constraints $cf = (k_1, l_1) \wedge \cdots \wedge (k_n, l_n)$. A conjunction can also be interpreted as the set of constraints $cf = \{(k_1, l_1), \ldots, (k_n, l_n)\}$. We can extend the definition of *negation*, *level of competence*, *always less than level*, and *next* to a conjunctive competence formula as follow:

- $\mathsf{negation}(cf) = \bigwedge_{(k_i,l_1)\in cf} \neg(k_i, l_i)$;
- $\mathsf{existence}(cf) = \bigwedge_{(k_i,l_i)\in cf} \Diamond(k_i, l_i)$;
- $\mathsf{absence}(cf) = \bigwedge_{(k_i,l_i)\in cf} \Box\neg(k_i, l_i)$;
- $\mathsf{possibility}(cf) = \bigwedge_{(k_i,l_i)\in cf} (\Diamond(k_i, l_i) \vee \Box\neg(k_i, l_i))$.
- $\mathsf{next}(cf) = \bigwedge_{(k_i,l_i)\in cf} \bigcirc(k_i, l_i)$.

A *disjunctive normal competence formulae* dnf is a disjunction of conjunctive competence formulas, $dnf = cf_1 \vee \cdots \vee cf_n$. Again, we also denote a disjunctive normal competence formula as a set of conjuctive competence formulas $dnf = \{cf_1, \ldots, cf_n\}$. Therefore, a disjunctive normal competence formula is a set of sets of atomic competences.

The positive relations presented in Section II-B can be generalised to a DNF formula as follows:

- dnf_1 *before* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} \mathsf{negation}(cf_j)\ \mathsf{U}\ cf_i$;
- dnf_1 *immediate before* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} cf_i$ *before* $cf_j \wedge \Box(cf_i \supset (\mathsf{next}(cf_j) \vee \mathsf{absence}(cf_j)))$;
- dnf_1 *implies* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} \mathsf{existence}(cf_i) \supset \mathsf{existence}(cf_j)$;
- dnf_1 *immediate implies* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} cf_i$ *implies* $cf_j \wedge \Box(cf_i \supset \mathsf{next}(cf_j))$;
- dnf_1 *succession* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} \mathsf{existence}(cf_i) \supset (\mathsf{existence}(cf_j) \wedge cf_i$ *before* $cf_j)$;
- dnf_1 *immediate succession* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} cf_i$ *succession* $cf_j \wedge \Box(cf_i \supset \mathsf{next}(cf_j))$.

The negative relations presented in Section II-B can be generalised to a DNF formula as follows:

- dnf_1 *not before* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} \mathsf{negation}(cf_i)\ \mathsf{U}\ (cf_j \wedge \mathsf{negation}(cf_i))$;
- dnf_1 *not immediate before* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} \mathsf{negation}(cf_i)\ \mathsf{U}\ (cf_j \wedge \mathsf{negation}(cf_i))$;
- dnf_1 *not implies* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} \mathsf{existence}(cf_i) \supset \mathsf{absence}(cf_j)$;
- dnf_1 *not immediate implies* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} \mathsf{existence}(cf_i) \supset (\mathsf{absence}(cf_j) \vee \Diamond(cf_i \wedge \mathsf{next}(\mathsf{negation}(cf_j))))$;
- dnf_1 *not succession* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} \mathsf{existence}(cf_i) \supset (\mathsf{absence}(cf_j) \vee cf_i$ *not before* $cf_j)$;
- dnf_1 *immediate succession* dnf_2: $\bigvee_{cf_i\in dnf_1, cf_j\in dnf_2} \mathsf{existence}(cf_i) \supset (\mathsf{absence}(cf_j) \vee cf_i$ *not before* $cf_j \vee \Diamond(cf_i \wedge \mathsf{next}(\mathsf{negation}(cf_j))))$.

Semantic Resource Management in MAS

Nicola Cannata, Flavio Corradini, Francesca Piersigilli, Emanuela Merelli and Leonardo Vito
Dipartimento di Matematica e Informatica
Università di Camerino, Via Madonna delle Carceri 9, 62032 Camerino (MC), Italy
Email: name.surename@unicam.it

***Abstract*—In highly dynamic environments like academy and industry it is becoming essential the need of efficient systems for resources organization and discovery. In this paper we describe a semantic resources manager, called Resourceome. This system allows both to discover and organize resources for agents' goals achievement. The ontological descriptions of resources and of domains allow to contextualize a resource instance in its domain through a *concern* relation. The proposed model supports the navigation from domain to resource concepts and vice versa. Resourceome represents our proposal for describing the particular vision of the world perceived by multi-agent systems.**

I. Introduction

The wide use of distributed systems led to the design and implementation of middlewares. Corba, RMI, Web Services, and FIPA are the most important standard specifications, which gave rise to successful middlewares in both business and academic environment. Even if actual middlewares allow developers of distributed applications to overcome the interconnection and integration troubles, it still remains present the need to have suitable support to organize and discover resources to fully support the systems interoperability. For *resources* we mean web services, persons, tools, databases, files and others available either on the web or locally. Concerning pharmaceutical industry and bioinformatics research we are witnessing a growing number of published resources, if properly organized could be the basic knowledge of artificial systems in life science [1].

In any distributed system, likewise an agents community, the organization of the resources plays a very important role. In fact, it is often difficult for a software agent to look for the right resource in an unexplored open environment [2]. Agents generally do neither know what kind of resources are available, nor if a certain resource is still existing in their environments; even playing a specific role. An organization of contextualized resources in their domains can help their discovery (in particular at run-time) by a resource manager. This could also replaced a resource with another one when the original one is not available or when an equivalent or better one is found. The dynamic and distributed scenario is the natural environment for multi-agent systems (MAS). In such context, where the aggregation of new communities of agents is possible, the semantic discovery of resources would be very useful.

To this end, we propose a model for a semantic resources manager, called Resourceome. System that allows both to discover and organize resources contextualized in their domains. Resources are described by suitable ontology whose instance resources are related to their domain concepts by a specific relation, that in the proposed model, is called *concern*. The proposed model supports the navigation from domain to resource concepts and vice versa.

Resourceome model differs from existing models, as OWL-S and WSDL-S, in two main features: 1) Resourceome allows the description of any kind of resources, e.g. web service and ; 2) it allows to add new resources and to contextualize them on appropriate domains.

Resourceome with its model, thus, represents our proposal for describing the particular vision of the world perceived by multi-agent systems.

The rest of this papers is organized as follows. Section II introduces the concept of Resourceome. In Section III the Resourceome management system is described. Section IV shows how the Resourceome can help software agents to discover resources. A case study of resources discovery with Resourceome is presented in Section V.

II. What is a Resourceome?

In this section we describe the basic model of Resourceome. The basic idea behind Resourceome is that of a formal, machine understandable description of resources through two ontologies: the domain ontology and the resources ontology [3].

The main purpose of the resources ontology is to organize the concepts, properties and relationships through which a resource can be classified. Whereas the domain ontology organizes the topics the users are interested in. We can foresee general purpose resourceomes to organize resources in industry, computer science, bioinformatics or something else. For example, industrial topics could be represented by concepts like *electricity*, *energetic saving*, *electronics* and *staff management*. The domain ontology defines the resources context, allowing humans and software agents to easily understand what a resource is about.

Thus, a resource context is also given in terms of concepts of the domain: in our representation resources are linked to the concepts of the domain ontology through a specific relation called *concern*. Figure 1 describes the Resourceome model in terms of concepts (ovals) and *is-a* relationships (arrows) between concepts, and *concern* relationships between ontologies (dark arrows). Resourceome can manage any ontology provided there exists at least one *concern* relation. In the sequel of the paper, for sake of simplicity, we have only used, as examples, ontologies with *is-a* relation.

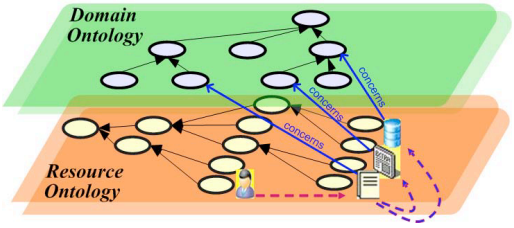


Fig. 1. The Resourceome model

To define the *concern* relation we aimed to use OWL-Full, but actually it doesn't exist a reasoning software able to support complete reasoning for every feature of OWL Full. Consequently, we used OWL-DL, even if its expressiveness was not sufficient - OWL-DL can be used only under certain restrictions, for example, a class cannot be in relation with an instance of class-. To overcome the OWL-DL restrictions we have introduced an *hidden* instance for any ontological concept.

A domain could eventually be specified through more than one ontology, e.g. for interdisciplinary domains. Furthermore and more concretely, resources are univocally defined through their metadata and in particular through their URIs [4], that can be LSIDs [5], DOIs [6], URLs, and so on.

A. An example of Resources Ontology

The notion of resource is fundamental in current networked information systems. The term is used often, specifically in relation to World Wide Web and Semantic Web activity, e.g. in standards such as RDF [7]. This term masks an exceptional amount of ambiguity. Although a stated definition of a resource in the URI RFC [4] exists, it is in many respects vague: "A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings and books in a library can also be considered resources [...] a resource can remain constant even when its content, the entities to which it currently corresponds, changes over time, provided that the conceptual mapping is not changed in the process. [...]". In an industrial scenario, the types of related resources depend on the production domain, even though we can affirm that there are some resources common to every sphere. Just to give an idea, Figure 2 shows some concepts that can represent a fragment of knowledge in an industrial scenario. In particular, *warehouseman*, *firm employee*, *person agent*, *software agent*, *human agent* are concepts defined by the following relationships:

- a Warehouseman is-a Firm Employee
- a Firm Employee is-a Person
- a Software Agent is-a Agent
- a Human Agent is-a Agent

Also a *service agent* [8] can be considered a resource, in fact, in our proposal - as it is described in the sequel - a service agent has the possibility to register itself as a resource in the Resourceome of the running platform by reporting its particular features and concerning one or more domain topics.

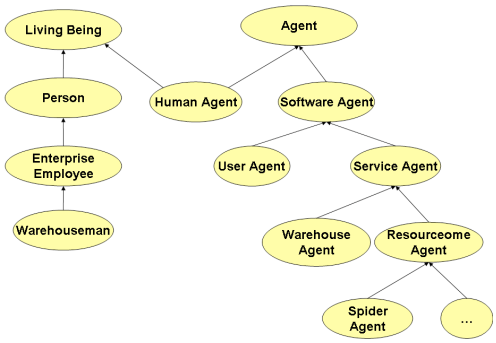


Fig. 2. A fragment of an industrial Resources Ontology

In addition, we can establish more explicit relations between resources like: *uses*, *describes*, *is author of*, *is executed by*, and so forth. And concepts can have many attributes such as *name*, *description*, *URI*, etc.. These relations and attributes allow software agents to obtain information and knowledge about a certain resource, so that they can choose between apparently equivalent resources.

We have developed some examples of resources ontology, however being Resourceome a customizable system, resource ontologies can be built at will.

B. Resources instances

Instances of resources can be considered as an *instances ontology*. Although this component extends the resources ontology, it is more appealing to see it as an independent data set. In the present prototype, it is composed of distributed OWL-DL [9] files referring to a specific resources ontology and concerning a particular domain ontology. The instances files refer to both the resource and domain ontologies by importing them through the definition of their *namespaces*. Individuals can have limited life span, or can be irregularly available. Therefore there is the problem of how to manage no more or temporary not reachable resources. As we will deep in Section III, this task is performed by a pool of specific software agents that periodically check for resources and their status. When a not reachable anymore resource is discovered, it is marked in a different way, in order to allow the user to be aware of it. The ontologies grow receiving concepts, relationships and instances in two manners: they can be added manually by users or automatically by the software agents.

Resources could be available through the web or could represent local resources in a remote machine.

III. AGENTS MAKE "ALIVE" THE RESOURCEOME

The management of the Resourceome is performed by a multi-agent system whose organization allows to carry out also the interactions with the final users.

Software agents are autonomous computational entities operating in an open dynamic environment [10]. Agents generally interact with each other, also collaborating to achieve common goals. Agents can be seen as a design metaphor for constructing complex systems around autonomous, communicating entities [11]. In this context the multi-agent system allows to consider the Resourceome like a living organism that is composed of a static component formally described (i.e. the ontology) and a set of proactive components that maintain updated the ontology (i.e. the multi-agent system). In order to achieve this goal, we introduce the following specialized agents:

- Query/Answer agent: its role is that of taking any user query, and translate it in SPARQL [12]- the language recommended by the W3C and adopted in this solution -. After having queried the ontologies, the results are translated back in the user language.
- Spider agent: its role is that of surfing Internet to find new resources (and also new concepts) concerning a particular domain of interest. The research is based on some parameters such as URLs and keywords. The outputs provided by the Spider Agent are new instances of resources and eventually new concepts and relations to add to the Resourceome.
- Parser agent: its role is that of parsing a flat file prepared to include a set of resources in the Resourceome. The file, e.g. an XML file, is created by users to index the proper user resources (more details are provided in the Section III-A).
- Text mining agent: its role is that of automatically "annotate" the resources described in a text document. It looks for relations between a certain text document and a set of given concepts and resources by following these steps:
  1) it tries to find the best fitting concept belonging to a resources ontology for classifying the resource described by the document;
  2) it tries to find the best fitting topics belonging to a domain ontology for providing semantics to the resource described by the document;
  3) it tries to enrich the semantics of the described resources adding their relationships with other resources. In particular if the resource to be annotated is an article then an instance of the relationship *cites* can be created for every other literature resource cited in the text.
- Matcher agent: it concerns the matching between the new knowledge eventually found by the spider or parser agent, and the actual Resourceome content.
- Session agent: it is the sessions responsible. In fact, more than one session can be opened by users to access the Resourceome, or by the agents that manage the Resourceome, for read/write operations.
- Monitor agent: it tracks the resources signaling if a resource is currently reachable or not. When no longer available after a reasonable observation time, it can be considered definitively unreachable. The use of URN (like LSID and DOI) instead of URL and the resources monitoring, partially solves the "404 not found" problem [13].

We are aware that many of the proposed agents' roles are still open issues. Nevertheless, we developed those basic features already proposed in literature.

A. How to "feed" Resourceome with new resources

Excluding the monitor and query/answer agents, all the others collaborate, by interacting with each other, to automatically add knowledge to the Resourceome. We aim to formalize, in the next future, the communication protocol by using a MAS methodology.

The instances of the resources ontology could be enriched by the spider agent "walking" across the Internet. It should try to individuate the metadata describing interesting resources, adding this information to the current instances "ontology". The task just described is rather complicated. It is performed by a pool of software agents entrusted - by spider agents - to continuously effect searches on Internet, and look for new available interesting resources. The basic knowledge of the spider agent consists on a set of URLs - that acts as the spider starting points - and a list of keywords - concepts of the domain and resources ontologies -.
When a spider finds a new resource to be added, its metadata are converted in OWL-DL, if specified in a different language. Then the matcher agent imports them into the ontology files. The metadata are connected to the suitable concepts, if they are available.
If the right concepts to identify the resource and/or its domain are not yet provided, they are automatically added by the matcher agent supported by the new resource metadata. The metadata and a vocabulary with the use of a similarity algorithms, help the matcher to reach its goal. When the Resourceome does not contain adequate concepts to describe the new resource, new ones can be added from the remote ontology. Also relationships and properties are imported, if possible.

We have also implemented another kind of search based on a particular flat file that Resourceome users should prepare and publish. This file contains the metadata of the users' resources, which must be indexed by the Resourceome. In this case the work carried on by the Parser Agent is indispensable. The flat file should be built in a standard format in order to let the parser identify the elements of the resource description.

IV. THE RESOURCEOME ROLE IN A MAS

Considering multi-agent systems as virtual extension of the human reality, the agent society, at least partially, should replicate some of its specific aspects [14]. A software agent is basically characterized by the goals it must achieve, by the

roles it can play in its existence, by its attitudes and by the contexts in which it lives. A context defines also the conditions by which an agent plays a role and has some goals [15]. A context can change during an amount of time and it is very important that an agent has the ability of understanding its context and how it evolves.

The Resourceome allows agents to understand part of the environmental context. In particular the use of the resources, which of course characterize the context. When an agent realizes to be in a certain context, where it can use some resources, it can decide to acquire a new role. The Resourceome logically represents a knowledge-base accepted by the agents community which shares it. It represents also a system that allows every agent to share its knowledge, in order to enrich the intellectual capital of the organization.

There are some objects, such as resources, that have features dependent from observers. In Searle [16] every institutional fact is defined by constitutive rules, like *X counts as Y in context C*. We think that the Resourceome is able to provide this kind of characterization. In fact a resource identifies a certain entity, i.e a resource instance for a precise context of use bind to a specific domain. When it is communally recognized that a resource *X identifies Y in a context C*, the resource *X* assumes a particular *status*. The status indicates the functionality that the resource *X* can play in the context *C*.

Thanks to this characterization also agents (which in turn represent resources) can be contextualized and semantically discovered through the combination of the Resourceome and of the Directory Facilitator.

A. The broker role of the Resourceome

The Resourceome can be considered a data structure representing formally part of shared knowledge among an agents community. The pool of agents for the Resourceome management, described in Section III, can be seen as an organization whose members have been created to hold specific roles and to interact with each other by respecting the organization laws. Indeed not all agents must necessarily have the ability to directly navigate the ontologies of the Resourceome to gain knowledge about context resources. Any agent can be helped by specialized agents which are the promoters of its requests execution. These agents specialized to query the Resourceome give to the Resourceome system the role of resource broker.

In such a view, the system can find one or more kinds of resources concerning a particular domain, or alternatively, from a given domain concept it retrieves all resources concerning it. It follows a list of some search examples supported by the actual prototype implementation:

- based on a list of resources ontology concepts, select all resources having a "concerns" relation with a given domain ontology concept;
- taking as starting point a domain concept, search all resources concerning that concept;
- search all resources having a relation with a particular resource.

The presence of relations between resources allows the broker agents to dig deeper into a resource knowledge and possibly to infer on it and deduce new knowledge. Knowledge that can improve the description of the resource itself for further search.
Every resource, besides being characterized by its hierarchy and relations with other resources, can be described also by some properties, such as *name*, *description*, *location* and others. For example, a resource such as a web service, can be characterized by the WSDL service descriptor, or the OWL-S file and by a relation like "stub with". The "stub with" relation can in addition specify that the service stub needs the "WSDL2Java", which is a web service resource instance.

Besides query functionality the system allows to add new resources in order to enrich the knowledge base behind Resourceome.

B. The zooming-in zooming-out in the Resourceome

A MAS is generally formed by an agents community operating in one or more distributed platforms. In a distributed environment resources discovery is a very important. To such purpose, maintaining a unique centralize Resourceome acting as resources provider, would be very an impracticable solution as the community growths. On the other hand, a distinct Resourceome for each place, managing only local resources should be reductive, while aligning them would result extremely redundant.

Thanks to the ontologies distribution, we can consider the domain as organized in a top-down model. The resources discovery can be done through several "zoom-in" or "zoom-out" in the different domain ontologies as shown in Figure 3. In the top level of the domain hierarchy can be found one or more upper ontologies concerning very generalized concepts [17]. Zooming-in the domain ontologies, we can navigate the ontologies through the platforms and have a more and more specialized view of our interest domain. These ontologies are in local platforms. The zoom-in is implemented through a *part-of* relations, connecting several domain ontologies. The resources discovery can be executed starting from a general view at a more specific view. In this way it will be possible to discover the needed resources concerning a particular topic of the domain.

V. Case study

The production process of a manufacturing company is usually performed by executing a set of distinct activities, sequential or not. Performing an activity often requires the use of resources. As a case study we have chosen a simply supply chain to build and test electrical domestic appliances (see Figure 5). The supply chain consists of federated enterprises: several suppliers, a production plant, a distribution center and a technical service center. Each enterprise is characterized by a specific role and carries out a set of specific tasks in the virtual organization. Usually, tasks need specific resources like files describing particular tests, components for controlling the machine status during the production process, databases for

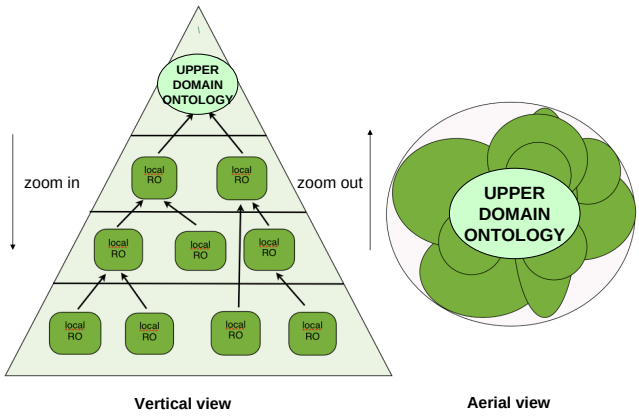


Fig. 3. Domain ontology view

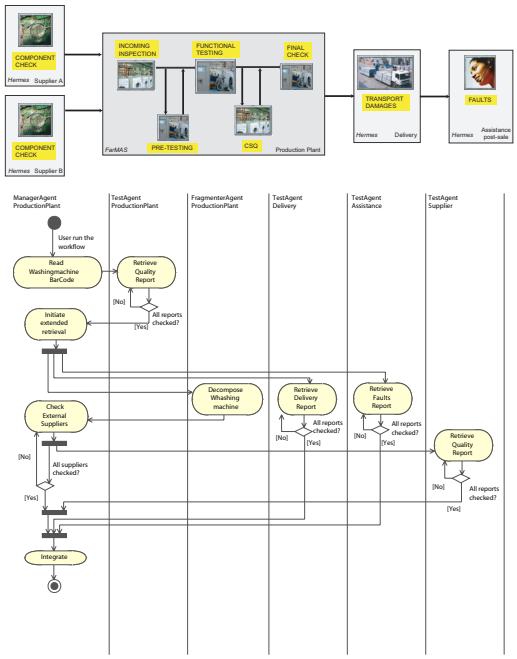


Fig. 4. Case study: the MAS supply chain [18]

managing stores, providers of machines components, software agents to assist the orders management and so forth.

In this context, these and other resources are distributed in the production branches and described by Resourceome (see Figure 2).

In our previous work we have defined a society of autonomous agents created to support the traceability of components information in a federated enterprises environment [18]. A simple supply chain described in Figure 4 for the production of electrical appliances such as washing machines, refrigerators or dishwashers whose componentsÕ traceability is defined in terms of a kind of workflow extended for quality control.

During the testing phase, it could happen that a test shows a malfunction of a particular component of a washing machine, elsewhere assembled. If the same component is not available, it is necessary to find a similar one. Discovering a similar component in the federated enterprises is not an easy task if the semantic description of the component (for us a resource) is not available. In the hypothesis that Resourceome provides the domain ontology for dry washing and the resource ontology for the component of a washing machine, the discovering of a new component could be performed by the cooperation of the enterprise employee, the warehouse agent, the query/answer agents and the store agent.

If the failure occurs frequently, besides blocking the production, this circumstance will cause also inconsistency in the databases distributed in the federation. Also this problem is easily predictable and solvable by a semantic resource management system .

VI. Conclusion

Multi-agent systems can be understood as complex entities where a multitude of agents interact, within a structured environment aiming at some global purpose. The Resourceome system is used to discover the resources necessary for the agents goals achievement. In fact, agents organizations often need heterogeneous resources, such as files, persons, web services or agents providing some services (*service agents*), not of easy to discover. An agent could be blocked because of its

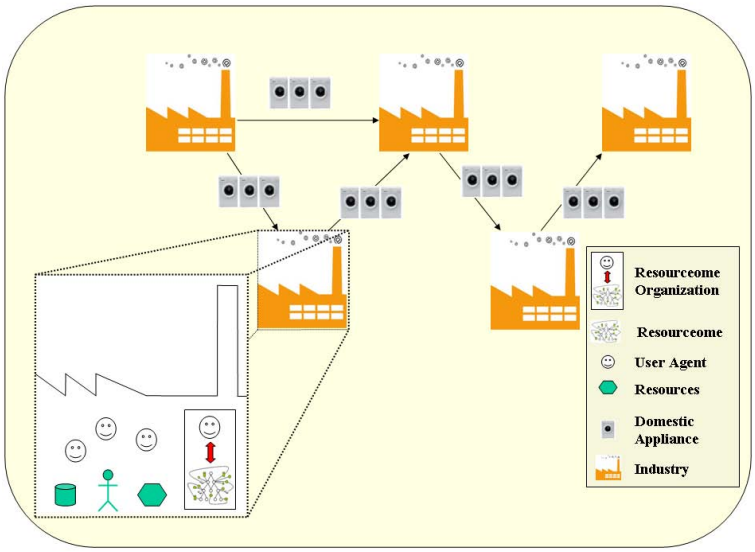


Fig. 5. Case study: the Resourceomic MAS in a supply chain

inability of retrieving a resource or of resource unavailability. The use of a semantic system for resources organization avoids this kind of problem providing analogous resources, replacing the ones searched by the agent, but not available at that moment. So the resources contextualization in a precise domain inside a multi-agent system, represents a very important aspect for the characterization of the environment and its evolutions. In this paper we presented the Resourceome both as semantic system for resources management and contextualization, and as resource broker. The use of ontologies organized as domain, resources and instances ontologies, besides resources discovery, also allows to express roles and conventions of the organization representing the multi-agent system. Since a service agent is itself a resource, the Resourceome can give semantics to Directory Facilitators.

ACKNOWLEDGMENT

The authors would like to thank the Loccioni Group and the *Italian Investment Funds for Basic Research* (MIUR-FIRB) project, *Laboratory of Interdisciplinary Technologies in Bioinformatics* (LITBIO) for supporting this work.

REFERENCES

[1] E. M. N. Cannata and R. B. Altman, "Time to organize the bioinformatics resourceome," *PLoS Comput Biol.*, vol. 1, no. 7:e76, 2005.

[2] M. Klusch and K. Sycara, "Brokering and matchmaking for coordination of agent societies: A survey," in *Coordination of Internet Agents: Models, Technologies, and Applications*, A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, Eds. Springer-Verlag, Mar. 2001, ch. 8, pp. 197–224.

[3] N. Cannata, F. Corradini, S. Gabrielli, L. Leoni, E. Merelli, F. Piersigilli, and L. Vito, "Intuitive and machine-understandable representation of the bioinformatics domain and of related resources with resourceomes," in *NETTB: A Semantic Web for Bioinformatics: Goals, Tools, Systems, Applications*, June 2007.

[4] T. B.-L. et al., "Rfc 2396 - uniform resource identifiers (uri): Generic syntax," 1998, `www.faqs.org/rfcs/rfc2396.html`.

[5] T. Clark, S. Martin, and T. Liefeld, "Globally distributed object identification for biological knowledgebases." *Briefings in Bioinformatics*, vol. 5, no. 1, pp. 59–70, 2004.

[6] "The digital object identifier system," `http://doi.org`.

[7] L. O. and S. R. R., "Resource description framework (rdf): Model and syntax specification," 1999, `www.w3.org/TR/REC-rdf-syntax`.

[8] `www.fipa.org/specs/fipa00001/SC00001L.pdf`.

[9] "OWL web ontology language guide," `www.w3.org/TR/owl-guide/`.

[10] M. J. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.

[11] N. R. Jennings, "An agent-based approach for building complex software systems." *Commun. ACM*, vol. 44, no. 4, pp. 35–41, 2001.

[12] "Sparql query language for rdf," `www.w3.org/TR/rdf-sparql-query/`, 2007.

[13] J. D. Wren, "404 not found: the stability and persistence of urls published in medline." *Bioinformatics*, vol. 20, no. 5, pp. 668–672, 2004.

[14] M. V. M. Colombetti, N. Fornara, "Linguaggio e realtà sociale nei sistemi di agenti artificiali," *Networks*, no. 1, pp. 48–67, 2003.

[15] K. P. Sycara, J. A. Giampapa, B. K. Langley, and M. Paolucci, "The RETSINA MAS, a case study." in *SELMAS*, 2002, pp. 232–250.

[16] J. R. Searle, *The construction of social reality*, simon & schuster ed., 1995.

[17] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider, "Sweetening ontologies with DOLCE." in *EKAW*, 2002, pp. 166–181.

[18] D. Bonura, F. Corradini, E. Merelli, and G. Romiti, "Farmas: a MAS for extended quality workflow," in *2nd IEEE International Workshop on Theory and Practice of Open Computational Systems*. IEEE Computer Society Press, 2004.

News Retrieval through a MultiAgent System

Andrea Addis, Giuliano Armano, Francesco Mascia, and Eloisa Vargiu
University of Cagliari
Piazza d'Armi
I-09123, Cagliari, Italy
Email: {addis,armano,f.mascia,vargiu}@diee.unica.it

***Abstract*— The continuous growth of information sources on the web, together with the corresponding volume of daily-updated contents, makes the problem of finding news and articles a challenging task. This paper presents a multiagent system aimed at creating press reviews from online newspapers by progressively filtering information that flows from sources to the end user, so that only relevant articles are retained. Once extracted, newspaper articles are classified according to a hierarchical text categorization approach. Moreover, an optional feedback provided by the user is exploited to improve the overall performances. The system is built upon a generic multiagent architecture that supports the implementation of personalized, adaptive and cooperative multiagent systems devised to retrieve, filter and reorganize information in a web-based environment.**

I. Introduction

The World Wide Web offers a growing amount of information and data coming from different and heterogeneous sources. As a consequence, it becomes more and more difficult for Internet users to select contents according to their interests, especially if contents are frequently updated (e.g., news, newspaper articles, reuters, RSS (Really Simple Syndication) feeds, and blogs). Supporting users in handling the enormous and widespread amount of web information is becoming a primary issue. To this end, several online services have been proposed (for instance Google News [1] and PRESSToday [2]). Unfortunately, they allow users to choose their interests among macro-areas (e.g. *economics*, *politics*, and *sport*), which is often inadequate to express what the user is really interested in. Moreover, existing systems typically do not provide a feedback mechanism able to allow the user to specify non-relevant items –with the goal of progressively adapting the system to her / his actual interests.

In this paper, we propose a multiagent system devised to handle the task of generating press reviews. To this end, the system (i) extracts articles from online newspapers, (ii) classifies them using hierarchical text categorization, and (iii) provides suitable feedback mechanisms to the end user. The motivation for adopting a multiagent system lies in the fact that a centralized classification system might be quickly overwhelmed by a large and dynamic document stream, such as daily-updated online news [18]. Furthermore, Internet is intrinsically a distributed system and offers the opportunity to take advantage of distributed computing paradigms and distributed knowledge resources.

[1] http://news.google.com/
[2] http://www.presstoday.com/

The remainder of the paper is organized as follows: Section II recalls some relevant related work. Section III describes the proposed multiagent system. In Section IV the underlying motivation in adopting a multiagetn system are briefly pointed out. Section V illustrates the experiments that has been performed during the training phase. Section VI shows the main functionalities of the system. Section VII draws conclusions and points to future work.

II. Related Work

This section is two-tiered, being aimed at recalling and summarizing relevant topics on multiagent systems used in information retrieval and on hierarchical text categorization.

A. MultiAgent Systems for Information Retrieval

In the literature, several centralized agents architectures aimed at performing information retrieval tasks have been proposed. Among others, let us recall NewT [38], Letizia [29], WebWatcher [4], and SoftBots [16].

NewT [38] has been designed as a society of information-filtering interface agents, which learn user preferences and act on their behalf. To filter information agents use a keyword-based filtering algorithm, whereas the adopted adaptive techniques are relevance feedback and genetic algorithms. Letizia [29] is an intelligent user interface agent able to assist a user while browsing the Web. The search for information results as a cooperative venture between the user and the software agent: both browse the same search space of linked web documents, looking for interesting ones. WebWatcher [4] is an information search agent that follows web hyperlinks according to user interests, returning a list of links deemed interesting to the user. In contrast to systems for assisted browsing or information retrieval, SoftBots [16] accept high-level user goals and dynamically synthesize the appropriate sequence of Internet commands according to a suitable ad-hoc language.

Despite the fact that a centralized approach could have some advantages, in information retrieval tasks it may encompass several problems, in particular how to scale up the architectures to large numbers of users, how to provide high availability in case of constant demand of the involved services, as well as how to provide high trustability in case of sensitive information, such as personal data. To this end, in the literature, suitable multiagent systems devoted to perform information retrieval tasks have been proposed. For the sake of

brevity, let us recall here CEMAS [8], IR agents [24], and the cooperative multiagent system for web information retrieval proposed in [37].

In CEMAS (Concept Exchanging Multi-Agent System) the basic idea is to have specialized agents for each main task, the main tasks being: (i) exchanging concepts and links, (ii) representing the user, (iii) searching for new relevant documents matching existing concepts, and (iv) agent coordination. IR agents implement an XML-based multiagents model for information retrieval. The corresponding framework is composed of three kinds of agents: (i) managing agents, aimed at extracting the semantics of information and at performing the actual tasks imposed by coordinator agents, (ii) interface agents, devised to interact with the users, and (iii) search agents, aimed at discovering the information on the web. Finally, in [37] the underlying idea is to adopt intelligent agents that mimic everyday-life activities of information seekers. To this end, agents are also able to profile the user in order to anticipate and achieve her/his preferred goals.

B. Hierarchical Text Categorization

Research interest in text categorization has been growing in machine learning, information retrieval, computational linguistics, and other fields. This reflects the importance of text categorization as an application area of machine learning, also facilitated by the availability of several document collections [28], [43] to which domain experts have assigned categories from a predefined (flat) set. These collections are in fact a benchmark that allow researchers to test their approaches while comparing the corresponding results.

Hierarchical text categorization deals with problems where categories are organized in form of a hierarchy. Many information sources are organized as hierarchies, e.g. web repositories, digital libraries, patent libraries, email folders, product catalogs. In particular, several web repositories encompass an underlying taxonomy, such as DMOZ [3] and the Google directory [4]. Taxonomies are also very useful in the field of news categorization, such as the one provided by the International Press Telecommunications Council [5] and the RCV-taxonomy (proposed by Lewis [27] to perform hierarchical text categorization on the Reuters standard document collection).

Until the mid-1990s researchers mostly ignored the hierarchical structure of categories that occur in several domains. In 1997, Koller and Sahami [22] carried out the first proper study of a hierarchical text categorization problem on the Reuters-22173 collection. First, a small hierarchical subset of Reuters-22173 has been generated by identifying labels that subsume other labels. Then, experiments have been performed by comparing a Naive Bayes classifier with two limited-dependency Bayes net classifiers –both on flat and hierarchical models. Documents were classified into the hierarchy by first filtering them through the single best-matching first-level class and then sending them to the appropriate second level. This approach showed that hierarchical models perform well when a small number of features per class is used. No advantages were found using the hierarchical model for large numbers of features. After this work several approaches to hierarchical text categorization have been proposed (see for instance [10], [30], [42], [14], [40], [34], [9]).

[3]http://www.dmoz.org/
[4]http://www.google.com/dirhp?hl=eng
[5]http://www.iptc.org/

III. The Proposed MultiAgent System for News Retrieval

Generally speaking, a system devoted to perform information retrieval tasks might encompass three main steps: (i) extract the required information from web sources, (ii) categorize items according to a given taxonomy, and (iii) provide suitable feedback mechanisms. The proposed multiagent system is organized in three layers, each aimed at performing one of the above information-retrieval steps, as sketched in Figure 1.

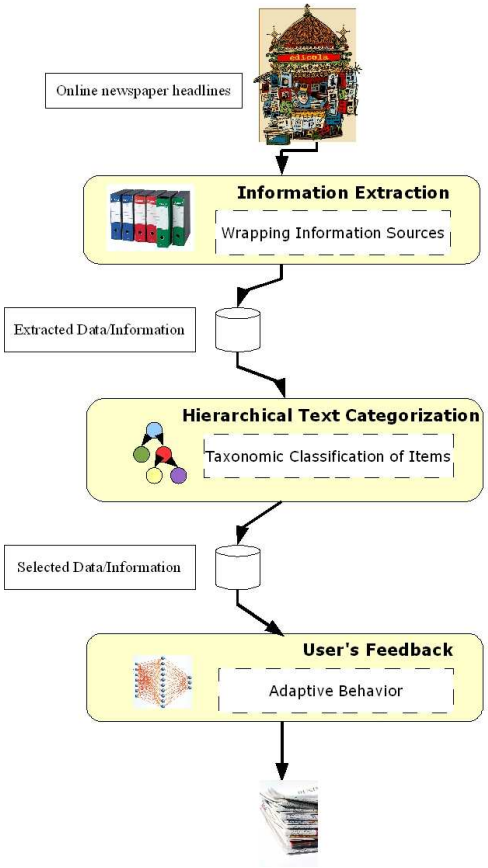


Fig. 1. A functional view of the proposed multiagent system

A. Information Extraction

The information extraction activity is essential to retrieve documents provided by heterogeneous and distributed sources, such as web sites, digital archives, and web services.

In the literature, several tools have been proposed to better address the issue of generating wrappers for web data extraction [25]. Such tools are based on distinct techniques, such as declarative languages [19], [12], HTML structure analysis [13], [35], natural language processing [17], [39], machine learning [20], [23], data modeling [1], [7], and ontologies [15].

To perform information extraction, we use several wrapper agents, each associated with a specific information source. In particular, three wrappers able to deal with the RSS format have been implemented so far. These wrappers are devoted to process the news feed provided by the Reuters portal [6], The Times [7], and The New York Times [8]. Other wrappers, able to embed the Reuters document collection used to train the system and to embed the adopted taxonomy have also been implemented.

Once extracted, all the information is suitably encoded to facilitate the text categorization task. To this end, all non-informative words such as prepositions, conjunctions, pronouns and very common verbs are removed using a stop-word list. After that, a standard stemming algorithm [33] removes the most common morphological and inflectional suffixes. Then, for each category of the taxonomy, feature selection, based on the information-gain heuristics [26], has been adopted to reduce the dimensionality of the feature space.

B. Hierarchical Text Categorization

A main issue in news categorization is how to deal, for each category, with an unbalance between relevant and non-relevant items. In particular, one may expect that most documents are non relevant to the user, the ratio between negative (e.g., non-relevant) and positive (e.g., relevant) examples being high (typical orders of magnitude are 10^2 - 10^3). Unfortunately, this aspect has a very negative impact on the precision of a text-categorization system. With the aim of coping with this phenomenon, we adopted a solution that exploits the ability of a pipeline of classifiers to progressively filter out non relevant information.

To better illustrate the underlying mechanism, let us consider the adopted taxonomy, i.e., the RCV1-taxonomy (Figure 2 reports part of the branch corresponding to the *economics* topic). Each node of the taxonomy represents a classifier entrusted with recognizing all corresponding relevant inputs. Any given input traverses the taxonomy as a "token", starting from the root (in the example in Figure, ECAT). If the current classifier recognizes the token as relevant, it passes it on to all its children (if any). The typical result consists of activating one or more pipelines of classifiers within the taxonomy. As an example, let us consider Figure 3 that illustrates the pipeline activated by an input document, which encompasses the categories economics (ECAT), government finance (E21), and expenditure/revenue (E211). This means that *all* involved classifiers recognize the input as relevant.

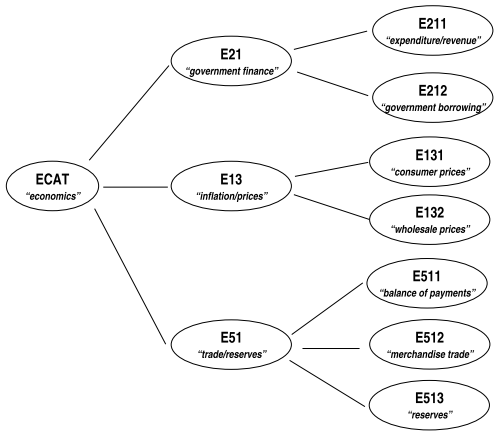


Fig. 2. A portion of the RCV1-taxonomy.

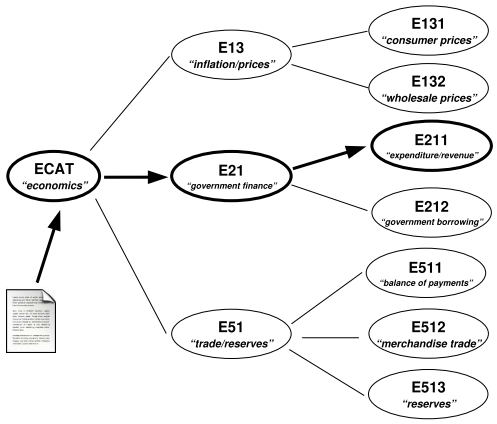


Fig. 3. An example of pipeline (highlighted in bold).

Each item in the taxonomy is implemented by an agent that embeds the corresponding classifier, in the current implementation the underlying classification technique being *k*-NN [44] –in its "weighted" variant [11]. The motivation for adopting this particular technique stems from the fact that it does not require specific training and is very robust with respect to noisy data.

C. User's Feedback

So far, a simple solution based on the *k*-NN technology has been implemented and experimented to deal with the problem

[6] http://www.reuters.com
[7] http://www.the-times.co.uk/
[8] http://www.nytimes.com/

of supporting the user's feedback. When a non-relevant article is evidenced by the user, it is immediately embedded in the training set of a *k*-NN classifier that implements the user feedback. A suitable check performed on this training set after inserting the negative example allows to trigger a procedure entrusted with keeping the number of negative and positive examples balanced. In particular, when the ratio between negative and positive examples exceeds a given threshold (by default set to 1.1), some examples are randomly extracted from the set of "true" positive examples and embedded in the above training set.

IV. Underlying Motivation in Adopting a MAS

An information retrieval system must take into account several issues, the most relevant being: (i) how to deal with different information sources and to integrate new information sources without re-writing significant parts of it, (ii) how to suitably encode data in order to put into evidence the informative content useful to discriminate among categories, (iii) how to control the unbalance between relevant and non relevant articles (the latter being usually much more numerous than the former), (iv) how to allow the user to specify her / his preferences, and (v) how to exploit the user's feedback to improve the overall performance of the system.

The above problems are typically strongly interdependent in state-of-the-art systems. To better concentrate on these aspects separately, we adopted a layered multiagent architecture, able to promote the decoupling among all aspects deemed relevant. In particular, the proposed system has been built upon PACMAS (Personalized Adaptive and Cooperative MultiAgent System), a generic multiagent architecture aimed at retrieving, filtering, and reorganizing information according to the users' interests [2]. The adoption of PACMAS is motivated by the willing of better concentrating on the above aspects separately, as it is in fact a layered architecture capable of promoting the decoupling among all relevant aspects of a complex task aimed at performing information retrieval. The PACMAS generic architecture has been implemented on to of the well known JADE [6] agent-based infrastructure. PACMAS encompasses four main levels:

- *information level*, aimed at wrapping information sources. The ability of the system to deal with new information sources affects only this level (i.e., a corresponding adapter or wrapper agent must be devised and implemented for each new kind of information source to be processed);
- *filter level*, devoted to suitably encode the text content according to an information-gain heuristics. Agents belonging to this architectural level encode (and embed) the text content of an article into a vector of words, which in turn is used to discriminate among existing categories;
- *task level*, devoted to identify relevant articles depending on the user interests. Agents belonging to this architectural level are aimed at performing two-tiered action: first the input is classified in accordance with the existing taxonomy, then the intended category (defined by composing existing categories with *and*, *or*, and *not* operators) is used to decide whether it interests the user or not. The former action embeds suitable policies aimed at controlling the negative impact of the unbalance between relevant and non-relevant articles, whereas the latter allows the user to explicitly specify her / his preferences about the set of relevant vs. non-relevant articles;
- *user interface level*, agents belonging to this level are aimed at performing the last check in order to decide whether the given input is of interest for the user and –optionally– at providing a feedback by the user, which can be exploited to improve the overall ability of discriminating relevant from non relevant inputs.

Finally, let us put into evidence that the adoption of a multiagent system allows to distribute the computation among several nodes. More the number of involved classifiers grows, more the distribution becomes an important issue to be taken into account. To this end, let us note that the adopted RCV1 taxonomy is composed of 103 classes and each node of the taxonomy represents a classifier entrusted with recognizing all corresponding relevant inputs.

V. Training the System

The system has been trained using RCV1-v2, the standard document collection proposed in [27], which is organized in four hierarchical groups: CCAT (Corporate/Industrial), ECAT (Economics), GCAT (Government/Social), and MCAT (Markets).

Before studying the progressive filtering technique, several experiments devoted to set the system parameters have been performed. In particular, experimentally we found that the optimal number of features is 200, and that the number of nearest neighbors to be taken into account by the *wk*-NN (e.g., the value of *k*) is in the range 7..11.

To assess the capabilities of the proposed progressive filtering technique, suitable pipelines composed of three classifiers [9] have been considered. First, each node of the pipeline is trained with a balanced data set by using 200 features (TFIDF) selected according to the information gain method. Then, for each node of the taxonomy, a learning set of 500 articles, with a balanced set of positive and negative examples, has been selected to train a classifier based on the *wk*-NN technology.

A complete discussion on the progressive filtering technique being out of the scope of this paper (see [3] for a detailed discussion), let us briefly summarize our experimental results.

As for testing, several randomly selected sets of 1000 documents have been generated –characterized by a different ratio between relevant and non-relevant inputs. In particular, the ratio between positive and negative examples has been set to 1:2, 1:10, 1:20, and 1:100 (50%, 10%, 5%, and 1%), say TS_{50}, TS_{10}, TS_5, and TS_1, respectively. To study the impact of progressively filtering information with pipelines of *wk*-NN classifiers (denoted as PIPE), we tested with the

[9] We considered pipelines of only three classifiers due to the limited depth of the adopted RCV1-taxonomy.

TABLE I

MICRO- AND MACRO-AVERAGING.

pos	$\mathbf{f_1^{\mu}WKNN}$	$\mathbf{f_1^{M}WKNN}$	$\mathbf{f_1^{\mu}SVM^1}$	$\mathbf{f_1^{M}SVM^1}$	$\mathbf{f_1^{\mu}SVM^2}$	$\mathbf{f_1^{M}SVM^2}$	$\mathbf{f_1^{\mu}Pipe}$	$\mathbf{f_1^{M}Pipe}$
50	0,883	0,883	0.831	0.832	0,898	0,897	0.905	0.905
10	0,646	0,647	0.507	0.521	0,719	0,722	0.721	0.720
5	0,513	0,514	0.412	0.428	0,535	0,543	0.683	0.682
1	0,165	0,169	0.169	0.190	0,344	0,349	0.412	0.431

above test sets some relevant pipelines, each concerning three nodes of the taxonomy ($k = 3$). Results have been compared with those obtained by running the same tests on stand-alone classifiers based on the following technologies: *wk*-NN (denoted as WKNN), [10] linear SVM (denoted as SVM^1), and RBF-SVM (denoted as SVM^2).

Table I summarizes the experimental results illustrating the micro- and macro-averaging of F_1 obtained by moving the acceptation threshold of the classifier(s) under investigation over the range $[0, 1]$. A concise recall of the corresponding definitions follows (the interested reader may consult the corresponding literature, e.g. [36]).

As for micro- and macro-averaging, they are aimed at obtaining estimates of precision (*P*) and recall (*R*) relative to the whole category set. In particular, micro-averaging evaluates the overall *P* and *R* by globally summing over all individual decisions. In symbols:

$$P^{\mu} = \frac{TP}{TP + FP} \quad (1)$$

$$R^{\mu} = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m} (TP_i + FN_i)} \quad (2)$$

where the "μ" superscript stands for microaveraging. On the other hand, macro-averaging first evaluates *P* and *R* "locally" for each category, and then "globally" by averaging over the results of the different categories. In symbols:

$$P^{M} = \frac{\sum_{i=1} mP_i}{m} \quad (3)$$

$$R^{M} = \frac{\sum_{i=1} mP_i}{m} \quad (4)$$

where the "M" superscript stands for macroaveraging.

As for F_1 [41], it is obtained from a more general definition by imposing that *P* and *R* have the same degree of importance. In symbols:

$$F_1 = \frac{2PR}{P + R} \quad (5)$$

Table I highlights that, in all selected samples, the solution based on multiple classifiers has reported better results than those obtained with flat models. Summarizing, experimental results show that in presence of unbalanced inputs, a pipeline of three classifiers is able to counteract an unbalance of up to 100 non relevant articles vs. one relevant article.

[10]The technique based on *wk*-NN has been used with both the hierarchical classification (PIPE) and the flat model (WKNN).

VI. THE CURRENT PROTOTYPE OF THE SYSTEM

Figure 4 illustrates the current user interface of the system. Through it, the user can set (i) the source from which news will be extracted, and (ii) the topics s/he is interested in. As for the newspaper headlines, the user can choose among the Reuters portal, The Times, and The New York Times. As for the topics of interest, the user can select one or more categories in accordance with the given RCV1 taxonomy.

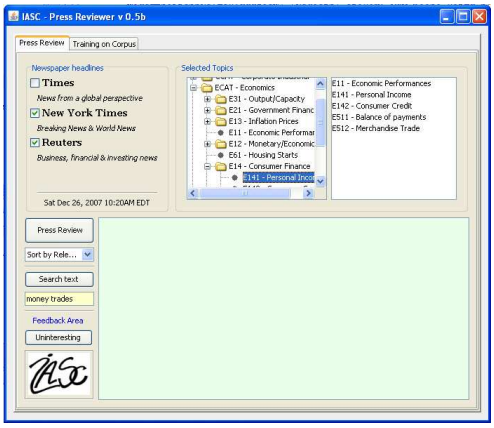


Fig. 4. The current user interface of the system.

The search for relevant news is activated by clicking on the *press review* button. First, information agents able to handle the selected newspaper headlines extract the news. Then, all agents that embody a classifiers trained on the selected topics are involved to perform text categorization. Finally, the system supplies the user with the selected news through suitable interface agents (see Figure 5).

Let us recall that the user can provide a feedback to the system by selecting all non-relevant news (i.e false positives). This feedback is important to let the system adapting itself to the actual interests of the corresponding user.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, a multiagent system devised to generate press reviews has been presented. The system encompasses three main tasks: (i) extracting articles from online newspapers, (ii) classifying them using hierarchical text categorization, and (iii) providing suitable feedback mechanisms.

Fig. 5. An example of results provided by the system.

As for the future work, more sophisticated strategies to provide personalization are currently under investigation. Moreover, we are implementing a new release of the system with improved text categorization functionalities by adopting Support Vector Machines.

ACKNOWLEDGMENTS

This work has been supported by the Italian Ministry of Education, under the project "DART - Distributed Architecture for Semantic Search and Personalized Content Retrieval".

REFERENCES

[1] B. Adelberg, "NoDoSEa tool for semi-automatically extracting structured and semistructured data from text documents", in *Proceedings of the 1998 ACM SIGMOD international Conference on Management of Data* (Seattle, Washington, United States, June 01 - 04, 1998). A. Tiwary and M. Franklin, Eds. SIGMOD '98. ACM Press, New York, NY, 1998, pp. 283-294.

[2] G. Armano, G. Cherchi, A. Manconi, and E. Vargiu, "PACMAS: A Personalized, Adaptive, and Cooperative MultiAgent System Architecture", in *Workshop dagli Oggetti agli Agenti, Simulazione e Analisi Formale di Sistemi Complessi (WOA 2005)*, 2005, pp. 54–60.

[3] G. Armano, F. Mascia, and E. Vargiu, "Using Taxonomic Domain Knowledge in Text Categorization Tasks", *International Journal of Intelligent Control and Systems, special issue on Distributed Intelligent Systems*, 2007, in press.

[4] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell, "Webwatcher: A learning apprentice for the world wide web", in *AAAI Spring Symposium on Information Gathering*, 1995, pp. 6–12.

[5] M. Ashburner, C.A. Ball, J.A. Blake, D. Botstein, H. Butler, J.M. Cherry, A.P. Davis, K. Dolinski, S.S. Dwight, J.T. Eppig, M.A. Harris, D.P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J.C. Matese, J.E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock, "Gene Ontology: Tool for the Unification of Biology", *Nature Genetics*, 25(1), 2000, pp. 25–29.

[6] F.L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*, John Wiley and Sons, 2007.

[7] B. Ribeiro-Neto, A.H. Laender, and A.S. da Silva, "Extracting semi-structured data through examples", in *Proceedings of the Eighth international Conference on information and Knowledge Management* (Kansas City, Missouri, United States, November 02 - 06, 1999), S. Gauch, Ed. CIKM '99. ACM Press, New York, NY, 1999, pp. 94–101.

[8] M. Bleyer, "Multi-Agent Systems for Information Retrieval on the World Wide Web", Diploma Thesis, University of Ulm, Germany, 1998.

[9] L. Cai, and T. Hofmann, "Hierarchical Document Categorization with Support Vector Machines", in *Proceedings of the ACM Conference on Information and Knowledge Management*, 2004, pp. 78–87.

[10] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan, "Using Taxonomy, Discriminants, and Signatures for Navigating in Text Databases", in *Proceedings of the 23rd international Conference on Very Large Data Bases* (August 25 - 29, 1997), M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, Eds. Very Large Data Bases, Morgan Kaufmann Publishers, San Francisco, CA, 1997, pp. 446–455.

[11] W. Cost, S. Salzberg, "A weighted Nearest Neighbor Algorithm for Learning with Symbolic Features", *Machine Learning*, Vol. 10, 1993, pp. 57–78.

[12] V. Crescenzi and G. Mecca, "Grammars Have Exceptions", *Information Systems*, Vol. 23 (8), 1998, pp. 539–565.

[13] V. Crescenzi, G. Mecca, and P. Merialdo, "Roadrunner, Towards Automatic Data Extraction from Large Web Sites", in *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 109–118.

[14] S. Dumais, and H. Chen, "Hierarchical Classification of Web Content", in *Proceedings of the ACM International Conference on Research and Development in Information Retrieval (SIGIR)*, 2000, pp. 256–263.

[15] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y. K. Ng, D. Quass, and R. D. Smith, "Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages", in *Data Knowledge Engineering*, Vol. 31(3), 1999, pp. 227–251.

[16] O. Etzioni and D. Weld, "Intelligent agents on the internet: fact, fiction and forecast", *IEEE Expert*, Vol. 10 (4), 1995, pp. 44–49.

[17] D. Freitag, "Machine Learning for Information Extraction in Informal Domains", Ph.D. dissertation, Carnegie Mellon University, 1998.

[18] Y. Fu, W. Ke, and J. Mostafa, "Automated text classification using a multi-agent framework", Proceedings of JCDL, 2005, pp. 157–158.

[19] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos, "Template-Based Wrappers in the TSIMMIS system", in *Proceedings of the 1997 ACM SIGMOD international Conference on Management of Data* (Tucson, Arizona, United States, May 11 - 15, 1997), J. M. Peckman, S. Ram, and M. Franklin, Eds. SIGMOD '97. ACM Press, New York, NY, 1997, pp. 532–535.

[20] C. N. Hsu and M. T. Dung, "Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web", *Information Systems*, Vol. 23(8), 1998, pp. 521–538.

[21] B.L. Humphreys, D.A. Lindberg, H.M. Schoolman, and G.O. Barnett, "The Unified Medical Language System: an informatics research collaboration", *Journal of the American Medical Informatics Association*, 5(1) (Jan-Feb 1998), 1998, pp. 1–11.

[22] D. Koller, and M. Sahami, "Hierarchically Classifying Documents Using Very Few Words", in *Proceedings of the International Conference on Machine Learning (ICML)*, 1997, pp. 170–178.

[23] N. Kushmerick, "Wrapper Induction: Efficiency and Expressiveness", *Artificial Intelligence*, Vol. 118 (1-2), 2000, pp. 15–68.

[24] W. Jirapanthong and T. Sunetnanta, "An XML-Based Multi-Agents Model for Information Retrieval on WWW", in *Proceedings of the 4th National Computer Science and Engineering Conference (NCSEC2000)*, Bangkok, Thailand, 2000.

[25] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and Juliana S. Teixeira, "A Brief Survey of Web Data Extraction Tools", *SIGMOD Rec.*, Vol. 31 (2), 2002, pp. 84–93.

[26] D. D. Lewis. "An evaluation of phrasal and clustered representations on a text categorization task", in *Proceedings of SIGIR-92, 15th ACM International Conference on Research and Development in Information Retrieval*, (Kobenhavn, DK, 1992), 1992, pp. 37–50.

[27] D.D. Lewis, Y. Yand, T. Rose, F. Li, "Rcv1: A New Benchmark Collection for Text Categorization Research", in *Journal of Machine Learning Research*, Vol. 5(Dec.2004), 2004, pp. 361–397.

[28] D.D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka, "Training Algorithms for Linear Text Classifiers", in *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, (Zurich, Switzerland, August 18 - 22, 1996). SIGIR '96. ACM Press, New York, NY, 1996, pp. 298–306.

[29] H. Lieberman. "Letizia: An agent that assists web browsing", in C.S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Quebec, Canada, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995, pp. 924–929.

[30] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Ng, "Improving Text Classification by Shrinkage in a Hierarchy of Classes", in *Proceedings of the International Conference on Machine Learning (ICML)*, 1998, pp. 359–367.

[31] S. Nelson, M. Schopen, A. Savage, J. Schulman, N. Arluk, "The MESH translation maintenance system: Structure, interface design, and implementation", in Fieschi, M.e.a., ed., *Proceedings of the 11th World Congress on Medical Informatics*, IOS Press, 2004, pp. 67–69.

[32] H.T. Ng, W.B. Goh and K.L. Low, "Feature Selection, Perceptron Learning, and a Usability Case Study for text Categorization", in *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, July 27-31, Philadelphia, 1997, pp. 67–73.

[33] M. Porter, "An Algorithm for Suffix Stripping", *Program*, Vol. 14(3), 1980, pp. 130-137.

[34] M. Ruiz, and P. Srinivasan, "Hierarchical Text Categorization Using Neural Networks", *Information Retrieval*, 5, 2002, 87–118.

[35] A. Sahuguet and F. Azavant, "Building Intelligent Web Applications Using Lightweight Wrappers", *Data Knowledge Engineering*, Vol. 36(3), 2001, pp. 283–316.

[36] F. Sebastiani, "Machine learning in automated text categorization", *ACM Computing Surveys*, 34(1) (Mar. 2002), 2002, pp. 1–47.

[37] K. Shaban, O. Basir, M. Kamel, "Team Consensus in Web Multi-agents Information Retrieval System", World Automation Congress, Vol. 17, 2004, pp. 68–73.

[38] B. Sheth and P. Maes, "Evolving agents for personalized information filtering", In I. Press, editor, *9th Conference on Artificial Intelligence for Applications (CAIA-93)*, 2003, pp. 345–352.

[39] S. Soderland, "Learning Information Extraction Rules for Semi-Structured and Free Text", *Machine Learning*, Vol. 34(1-3), 1999, pp. 233–272

[40] A. Sun, and E.P. Lim, "Hierarchical Text Classification and Evaluation", in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2001, pp. 521–528.

[41] C. van Rijsbergen, *Information Retrieval*, Butterworths, London, 1979.

[42] K. Wang, S. Zhou, and S.C. Liew, "Building hierarchical classifiers using class proximity", in M.P. Atkinson, M.E. Orlowska, P. Valduriez, S.B. Zdonik, and M.L. Brodie, eds, *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*, Morgan Kaufmann, 1999, pp. 363–374.

[43] Y. Yang, "An Evaluation of Statistical Approaches to Text Categorization", in *Information Retrieval*, Vol. 1(1-2), 1999, pp. 69–90.

[44] Y. Yang and X. Liu, "A re-examination of text categorization methods", in *Proceedings of the 22nd Annual international ACM SIGIR Conference on Research and Development in information Retrieval*, (Berkeley, California, United States, August 15 - 19, 1999). SIGIR '99. ACM Press, New York, NY, 1999, pp. 42–49.

A Comparison of Upper Ontologies

Viviana Mascardi, Valentina Cordì
DISI, Università degli Studi di Genova,
Via Dodecaneso 35, 16146, Genova, Italy
E-mail: {cordi,mascardi}@disi.unige.it

Paolo Rosso
DSIC, Universidad Politécnica de Valencia,
Camino de Vera s/n, 46022, Valencia Spain
E-mail: prosso@dsic.upv.es

***Abstract*—Upper Ontologies are quickly becoming a key technology for integrating heterogeneous knowledge coming from different sources. In fact, they may be exploited as a "lingua franca" by intelligent software agents in all those scenarios where it is impossible (or there is no will) for an agent to disclose its own entire ontology to other agent, despite the need to communicate with it. This paper represents the very preliminary step towards the exploitation of Upper Ontologies as bridges for allowing intelligent software agents to align heterogeneous ontologies in an automatic way, where we analyse the most up-to-date state-of-the-art. In this paper we analyse 7 Upper Ontologies, namely BFO, Cyc, DOLCE, GFO, PROTON, Sowa's ontology, and SUMO, according to a set of standard software engineering criteria, and we synthesise our analysis in form of a comparative table. A summary of some existing comparisons drawn among subsets of the 7 Upper Ontologies that we deal with in this document, is also provided.**

I. Introduction

The increasing pressing need that human and software agents have to retrieve and exchange knowledge in a precise and efficient way, have caused ontologies, web services, and the combination of both, i.e., semantic web services, to be more and more exploited for sharing knowledge within and outside the boundaries of companies and other organisations. Intelligent software agents are recognised by both researchers and practitioners from the industry as one of the most suitable means for mediating among the heterogeneity of applications working within open, distributed, concurrent systems, and for this reason they find application in many commercial projects.

However, there are still many unsolved issues for developing and deploying multi-agent systems (MASs) in open, distributed, concurrent scenarios. One of them is how to find mappings between concepts belonging to different ontologies (in technical word, finding an *alignment* between these different ontologies) in an automatic way. We are considering the adoption of "Upper Ontologies" as bridges for making this alignment possible.

Upper ontologies are quickly becoming a key technology for integrating heterogeneous knowledge coming from different sources. In fact, they may be used by different parties involved in a knowledge integration and exchange process as a reference, common model of the reality. In particular, they may be exploited as a "lingua franca" by intelligent software agents in all those scenarios where it is impossible (or there is no will) for an agent to disclose its own entire ontology to other agent, despite the need to communicate with it.

The definition of upper ontology (also named top-level ontology, or foundation ontology) given by Wikipedia [22] is "*an attempt to create an ontology which describes very general concepts that are the same across all domains. The aim is to have a large number on ontologies accessible under this upper ontology*".

This paper represents the very preliminary step towards the exploitation of Upper Ontologies as bridges for allowing intelligent software agents to align heterogeneous ontologies in an automatic way, where we analyse the most up-to-date state-of-the-art. In fact, in this paper we have described 7 upper ontologies along different criteria that include dimension, implementation language(s), modularity, developed applications, alignment with the WordNet lexical resource, and licensing. We have chosen these criteria for three reasons:

- They are software engineering criteria useful for the developer of a knowledge-based system that has to choose the most suitable Upper Ontology for his/her needs, among a set of existing ones. Since all of us have a computer science background, these criteria are more familiar to us than philosophical ones.
- They take into account some of the evaluation questions proposed by the IEEE Standard Upper Ontology Working Group (`http://suo.ieee.org/SUO/Evaluations/`), and they also extend the criteria considered in an existing comparison among SUMO, Cyc, and DOLCE [18], thus allowing us to "reuse", and to be consistent with, the results already obtained there.
- They are not (easily) scientifically falsifiable.

The choice of the 7 upper ontologies we have described, namely BFO, Cyc, DOLCE, GFO, PROTON, Sowa's ontology, and SUMO, is based on how much, to the best of our knowledge, they are visible and used inside the research community. For example, we have discussed all the Upper Ontologies referenced by Wikipedia, apart from WordNet that we consider a lexical resource rather than an Upper Ontology, and from the Global Justice XML Data Model and National Information Exchange Model, that addresses the specific application domain of justice and public safety. We have reported alignments between the Upper Ontologies and WordNet, when existing. To the 5 Upper Ontologies considered by Wikipedia, we have added PROTON and Sowa's ontology. We have also cited three attempts to merge existing Upper Ontologies, namely COSMO, MSO, and OntoMap, although we have not

described them in detail since the first two ones are still work in progress, and the last one is over since four years.

The methodology followed to draw this paper consisted in checking the existing literature, producing a first draft of the comparison based on the retrieved literature, submitting it to the attention of the developers of all the 7 upper ontologies under comparison, and integrating the obtained answers and suggestions into the current version of the paper. Due to time constraints, we were not able to experiment with the upper ontologies by our own. This "on the field" experimentation is part of our near future work.

The paper is organised in the following way: Section II provides a description of the 7 upper ontologies, and Section III surveys some existing, partial comparisons drawn in the past years among subsets of the Upper Ontologies that we describe in Section II, and provides a synthesis of the results of our comparison among them.

II. Description

a) Basic Formal Ontology (BFO):

- **Status of this description.** Validated by H. Stenzhorn, research associate and doctoral student at the IFOMIS and the University Hospital Freiburg - Medical Informatics Department, and one of BFO's developers.
- **Home page.** http://www.ifomis.org/bfo.
- **Developers.** B. Smith, P. Grenon, H. Stenzhorn, A. Spear (IFOMIS, Saarland University).
- **Description.** BFO consists in two sub-ontologies: SNAP – a series of snapshot ontologies (O_{ti}), indexed by times – and SPAN – a single videoscopic ontology (O_v). An O_{ti} is an inventory of all entities existing at a time, while an O_v is an inventory of all processes unfolding through time. Both types of ontology serve as basis for a series of sub-ontologies, each of which can be conceived as a window on a certain portion of reality at a given level of granularity.
- **History.** The theory behind BFO has been developed and formulated by Smith and Grenon in a series of publications starting in 1998. Its current implementation in OWL has been developed by Stenzhorn with contributions from Spear.
- **Dimensions.** BFO contains 1 top connecting class ("Entity"), 18 SNAP classes, and 17 SPAN classes for a total of 36 classes which are, in version 1.0 of the implementation, connected via the *is_a* relation. The forthcoming version of BFO will incorporate relations between classes too.
- **Implementation language(s).** OWL [21].
- **Modularity.** BFO is divided into the SNAP and SPAN modules.
- **Applications.** BFO has been applied to the biomedical domain [8] and it is currently used in building an ontology for clinic-genomic trials on cancer (http://www.acgt-eu.org).
- **Alignment with WordNet.** Not supported.
- **Licensing.** BFO is freely available; its OWL implementation may be downloaded from http://www.ifomis.org/bfo/1.0.

b) Cyc:

- **Status of this description.** Validated by L. Lefkowitz, executive director for business solutions at Cycorp.
- **Home page.** http://www.cyc.com/.
- **Developers.** Cycorp.
- **Description.** The Cyc Knowledge Base (KB) is a formalised representation of facts, rules of thumb, and heuristics for reasoning about the objects and events of everyday life. The KB consists of terms and assertions which relate those terms. These assertions include both simple ground assertions and rules. The Cyc KB is divided into thousands of "microtheories" focused on a particular domain of knowledge, a particular level of detail, a particular interval in time, etc.
- **History.** The Cyc project was founded in 1984 by D. Leant as a lead project in the Microelectronics and Computer Technology Corporation (MCC). In 1994, Cycorp was founded to further develop, commercialize, and apply the Cyc technology. Cycorp offers a no-cost license to its semantic technologies development toolkit to the research community (ResearchCyc). Additionally, it has placed the core Cyc ontology (OpenCyc) into the public domain.
- **Dimensions.** The Cyc KB (including Cyc's microtheories) contains more than 300,000 concepts and nearly 3,000,000 assertions (facts and rules), using more than 15,000 relations.
- **Implementation language(s).** Cyc is represented in the CycL formal language (http://www.cyc.com/cycdoc/ref/cycl-syntax.html). The latest release of Cyc includes an Ontology Exporter that allows to export specified portions of Cyc to OWL files.
- **Modularity.** The "microtheory" approach supports modularity.
- **Applications.** Cyc has been used in the domains of natural language processing, in particular for the tasks of word sense disambiguation [4] and question answering [5], of network risk assessment [19], and of representation of terrorism-related knowledge [6].
- **Alignment with WordNet.** The last release of Cyc (as well as of OpenCyc and ResearchCyc) includes links between Cyc concepts and about 12,000 WordNet synsets.
- **Licensing.** Cyc is a commercial product, but Cycorp also released OpenCyc (http://www.opencyc.org/), the open source version of the Cyc technology, and ResearchCyc (http://research.cyc.com/), namely the Full Cyc ontology, but with a research-only license.

c) DOLCE (a Descriptive Ontology for Linguistic and Cognitive Engineering):

- **Status of this description.** Not validated by the ontology developer(s).
- **Home page.** http://www.loa-cnr.it/DOLCE.html.

- **Developers.** Researchers from the Laboratory for Applied Ontology, headed by N. Guarino.
- **Description.** DOLCE is the first module of the WonderWeb Foundational Ontologies Library. DOLCE has a clear cognitive bias, in the sense that it aims at capturing the ontological categories underlying natural language and human commonsense. According to DOLCE, different entities can be co-located in the same space-time. DOLCE is described by its authors as an "ontology of particulars", which the authors explain as meaning an ontology of instances, rather than an ontology of universals or properties. The taxonomy of the most basic categories of particulars assumed in DOLCE includes, for example, abstract quality, abstract region, agentive physical object, amount of matter, non-agentive physical object, physical quality, physical region, process, temporal quality, temporal region.
- **History.** DOLCE has been developed as part of WonderWeb, a project funded as a shared-cost RTD under the European Commission information society technologies (IST) programme. WonderWeb started in 2002 and ended in 2004. Although the project has already ended, DOLCE is actively maintained and used.
- **Dimensions.** Around one hundred of terms, and a similar number of axioms.
- **Implementation language(s).** First Order Logic, KIF [1], OWL.
- **Modularity.** The intended use of DOLCE is within a modular library of foundational ontologies, but it is not currently divided into modules.
- **Applications.** According to the "DOLCE around the world" web page (http://www.loa-cnr.it/dolcevar.html), there are many projects that use DOLCE, including the LOIS Project – an international research project on multilingual information retrieval from legal databases –, SmartWeb – a centre of excellence in research on intelligent computing technologies and their application to web-based systems and services –, Language Technology for eLearning – a project funded by the EC, and using multilingual language technology tools and semantic web techniques for improving the retrieval of learning material –, AsIsKnown – a semantic-based knowledge flow system for the European home textiles industry, also funded by the EC –, and the Projects of the Laboratory for Applied Ontology.
- **Alignment with WordNet.** The OntoWordNet Project aims at aligning the top-level of WordNet to DOLCE, in order to obtain an "ontologically sweetened" lexical resource, meant to be conceptually more rigorous, cognitively transparent, and efficiently exploitable in several applications. The beta version (v0.72) of the OWL alignment of WordNet 1.6 Noun Synsets to the DOLCE-Lite-Plus ontology library consists of an alignment between DOLCE-Lite-Plus and about 100 Wordnet sysnsets, and can be downloaded from http://www.loa-cnr.it/ontologies/OWN/OWN.owl.
- **Licensing.** The OWL version of DOLCE can be freely downloaded from http://www.loa-cnr.it/ontologies/DLP3971.zip.

d) GFO (General Formal Ontology):

- **Status of this description.** Validated by F. Loebe, PhD student at the University of Leipzig under the supervision of H. Herre and M. Löffler, members of the scientific board of Onto-Med.
- **Home page.** http://www.onto-med.de/ontologies/gfo.html.
- **Developers.** The Onto-Med Research Group (http://www.onto-med.de/).
- **Description.** GFO includes elaborations of categories like objects, processes, time and space, properties, relations, roles, functions, facts, and situations. Work is in progress on an integration with the notion of levels of reality in order to more appropriately capture entities in the material, mental, and social areas.
- **History.** Work on GFO has started in 1999 in the context of the GOL project (General Ontological Language). Meanwhile, several directions of research have been recognised and divided the initial project, such that GFO is now one component of a larger framework. Work on GFO remains in progress, because the development of top-level ontologies is a long-term research effort.
- **Dimensions.** The OWL version of GFO consists of 79 classes, 97 subclass-relations, and 67 properties.
- **Implementation language(s).** The FOL axiomatization of GFO and a KIF implementation of it are forthcoming. An OWL-DL version also exists.
- **Modularity.** GFO exhibits a three-layered meta-ontological architecture consisting of an abstract top level, an abstract core level, and a basic level. The foundational ontology GFO is structured into several ontological modules including a module for functions and a module for roles.
- **Applications.** One of the aims of the group Onto-Med is the application of the GFO in the field of biomedical science. GFO has been used to represent knowledge about biological functions in the Gene Ontology, the Celltype Ontology, and the Chemical Entities of Biological Interest (ChEBI) Ontology [2], and GFO-Bio (http://onto.eva.mpg.de/gfo-bio.html) is based on GFO and is a core ontology for biology. Another area of application is the ontological foundation of conceptual modelling. First examples of applying GFO to UML are demonstrated in [9].
- **Alignment with WordNet.** Not supported.
- **Licensing.** The OWL version of GFO is released under the modified BSD Licence (http://www.opensource.org/licenses/bsd-license.php) and can be downloaded from http://www.onto-med.de/ontologies/gfo.owl.

e) PROTON (PROTo ONtology):

- **Status of this description.** Validated by A. Kiryakov, head of Ontotext Lab, member of the board.
- **Home page.** http://proton.semanticweb.org/

- **Developers.** Ontotext Lab, Sirma (http://www.ontotext.com/).
- **Description.** PROTON (PROTo ONtology) is a basic upper-level ontology providing coverage of the general concepts necessary for a wide range of tasks, including semantic annotation, indexing, and retrieval of documents. The design principles can be summarized as follows (i) domain-independence; (ii) light-weight logical definitions; (iii) alignment with popular standards; (iv) good coverage of named entities and concrete domains (i.e. people, organizations, locations, numbers, dates, addresses).
- **History.** PROTON has been developed in the scope of SEKT, a project co-funded by the EU 6th Framework programme. SEKT started the 1st of January, 2004 and will conclude at the end of 2006. PROTON is a development of the KIMO ontology, which had been created and used in the scope of the KIM platform for semantic annotation, indexing, and retrieval (http://www.ontotext.com/kim/). Currently, KIMO does not exist anymore; it is replaced by PROTON, KIMLO (http://www.ontotext.com/kim/2005/04/kimlo#) and KIMSO (http://www.ontotext.com/kim/2005/04/kimso#).
- **Dimensions.** PROTON contains about 300 classes and 100 properties.
- **Implementation language(s)** A fragment of OWL Lite.
- **Modularity.** PROTON is organized in three levels including four modules.
  The System module ontology module occupies the first, basic layer. It defines several notions and concepts of a technical nature that are substantial for the operation of any ontology-based software, such as semantic annotation and knowledge access tools. The Top ontology module occupies the second layer and includes basic philosophically-reasoned distinctions between entity types, such as Object, Happening, Abstract. Further uplevel, PROTON extends into its third layer, where either of two independent ontologies, which defines much more specific classes, can be used: PROTON Upper module or PROTON KM (Knowledge Management) module. Examples of concepts belonging to these modules are Mountain, as a specific type of Location, and ResourceCollection as a sub-class of InformationResource.
- **Applications.** As witnessed by a large number of publications (http://www.ontotext.com/publications/), PROTON has been used in different domains and for different purposes, including semantic annotation within the KIM platform, and knowledge management systems in legal and telecommunications domain [3]. It has also been used as a basis for a domain ontologies in media research and analysis (project MediaCampaign) and research intelligence (project IST World), and a basis for Business Data Ontology for Semantic Web Services [13].
- **Alignment with WordNet.** Not supported.
- **Licensing.** The four modules that compose PROTON are freely accessible via Web: System module (http://proton.semanticweb.org/2005/04/protons); Top module (http://proton.semanticweb.org/2005/04/protont); Upper module (http://proton.semanticweb.org/2005/04/protonu); Knowledge Management module (http://proton.semanticweb.org/2005/04/protonkm).

f) Sowa's Ontology:

- **Status of this description.** Not validated by the ontology developer(s).
- **Home page.** http://www.jfsowa.com/ontology/.
- **Developers.** J. F. Sowa.
- **Description.** Sowa's ontology is based on [20]. The basic categories and distinctions have been derived from a variety of sources in logic, linguistics, philosophy, and artificial intelligence. To keep the system open-ended, Sowa's ontology is not based on a fixed hierarchy of categories, but on a framework of distinctions, from which the hierarchy is generated automatically. For any particular application, the categories are not defined by drawing lines on a chart, but by selecting an appropriate set of distinctions. These categories include Object, Process, Schema, Script, Juncture, Participation, Description, History, Structure, Situation, Reason, and Purpose. Each of these categories may be either Physical or Abstract (and in both cases, it can be either Continuant or Occurrent), and it may also be either Independent, Relative, or Mediating. For example, Process is Physical, Occurrent and Independent.
- **History.** Sowa's ontology dates back to 1999. The two major influences on it are the semiotics of C. Sanders Peirce and the categories of existence of A. North Whitehead.
- **Dimensions.** The KIF encoding of Sowa's ontology contains about 30 classes, 5 relationships among classes, and among classes and instances (has, instance-of, subclass-of, temp-part-of, spatial-part-of), about 30 axioms.
- **Implementation language(s).** Sowa's ontology uses a first-order modal language, i.e., a first-order language with the modal operators "nec" and "poss". A version written in KIF also exists.
- **Modularity.** Sowa's ontology is not explicitly divided into modules, although each of the top level categories can be intended as a module by its own, connected to the other ones by means of relations.
- **Applications.** Sowa's ontology inspired many existing implemented upper ontologies, and thus its exploitation in the development of "second-generation" upper ontologies may be seen as one, and probably the most relevant, of its practical applications.
- **Alignment with WordNet.** Not supported.
- **Licensing.** The KIF encoding of Sowa's upper ontology can be freely downloaded from http://suo.ieee.org/SUO/ontologies/Sowa.txt.

g) SUMO (Suggested Upper Merged Ontology):

- **Status of this description.** Validated by A. Pease, current Technical Editor of SUMO.

- **Home page.** http://www.ontologyportal.org/.
- **Developers.** The SUMO starter document was created at Teknowledge by I. Niles and A. Pease, with a contribution by C. Menzel.
- **Description.** SUMO and its domain ontologies [14] form one of the largest formal public ontology in existence today. They are being used for research and applications in search, linguistics and reasoning. SUMO is extended with many domain ontologies and a complete set of links to WordNet, and is freely available.
- **History.** SUMO was first released in December 2000. It was created at Teknowledge Corporation and it was proposed as a starter document for the Standard Upper Ontology Working Group (http://suo.ieee.org/), an IEEE-sanctioned working group of collaborators from the fields of engineering, philosophy, and information science. SUMO was created by merging publicly available ontological content into a single, comprehensive, and cohesive structure. This content included the ontologies available on the Ontolingua server (http://www.ksl.stanford.edu/software/ontolingua/), Sowa's upper level ontology, and various mereotopological theories, among other sources.
- **Dimensions.** SUMO contains about 1000 terms and 4000 axioms; if we consider also the terms and axioms of its domain ontologies, however, it reaches the dimension of 20,000 terms and 60,000 axioms.
- **Implementation language(s).** The first-order logic language SUO-KIF (http://suo.ieee.org/SUO/KIF/suo-kif.html), OWL.
- **Modularity.** SUMO consists of SUMO itself (the official latest version on the IEEE web site can be downloaded from http://suo.ieee.org/SUO/SUMO/SUMO_173.kif), the MId-Level Ontology (MILO), and ontologies of Communications, Countries and Regions, Distributed Computing, Economy, Finance, Engineering Components, Geography, Government, Military, North American Industrial Classification System, People, Physical Elements, Transnational Issues, Transportation, Viruses, World Airports. Additional ontologies of terrorism are available on request.
- **Applications.** The applications of SUMO are documented by the almost one hundred published papers describing its use (http://www.ontologyportal.org/Pubs.html). The largest user community is in linguistics, but other classes of applications include "pure" representation, and reasoning. Applications range from academic to government, to industrial ones.
- **Alignment with WordNet.** SUMO has been mapped to all of Wordnet v2.1 by hand. The mappings can be downloaded from `http://sigmakee.cvs.sourceforge.net/sigmakee/KBs/WordNetMappings/`.
- **Licensing.** SUMO is free and owned by the IEEE. Its SUO-KIF implementation can be downloaded from `http://sigmakee.cvs.sourceforge.net/*checkout*/sigmakee/KBs/Merge.kif`, while the OWL implementation can be downloaded from http://www.ontologyportal.org/translations/SUMO.owl.txt. The ontologies that extend SUMO are available under GNU General Public License.

h) Merging Upper Level Ontologies.: Three attempts to merge some of the upper level ontologies, thus leading to an "upper-upper level ontology", are COSMO (COmmon Semantic MOdel, `http://colab.cim3.net/cgi-bin/wiki.pl?CosmoWG/-TopLevel`), MSO (Multi-Source Ontology, http://www.webkb.org/doc/MSO.html), and the OntoMap Project [11].

COSMO results from the efforts of the COSMO working group (COSMO-WG) and its parent group, the Ontology and Taxonomy Coordinating Working Group (ONTACWG). COSMO is viewed as consisting of a lattice of ontologies which will serve as a set of basic logically-specified concepts (classes, relations, functions, instances) with which the meanings of all terms and concepts in domain ontologies can be specified. The use of a common set of defining concepts will permit accurate interoperability of knowledge-based systems using the logical relations of their ontologies as the basis for reasoning in the system. Currently, COSMO integrates concepts from the OpenCyc and SUMO ontologies, with some classes from DOLCE and BFO. The work on COSMO is in progress.

MSO is the Multi-Source Ontology of WebKB-2, a knowledge server that permits Web users to browse and update private knowledge bases on their machines, or alternatively, a large shared knowledge base on the server machine. The ontology of the shared knowledge base is currently an integration of various top-level ontologies and a lexical ontology derived from an extension and correction of the noun-related part of WordNet 1.7. The semantics of some categories from WordNet has been modified in order to fix inconsistencies, while the semantics of categories from other sources (e.g. Sowa, DOLCE) has been kept. Also the work regarding the MSO is still in progress. In particular, the integration of the SUMO is still far from being complete. This integration links the SUMO categories to the existing categories of the MSO, adds some structure when needed, adds equivalent categories the names of which are better suited for knowledge representation conventions that are "common" in the communities using graph-based or frame-based notations, and finally translates the axioms from KIF to more intuitive notations that permit people to more easily understand the meanings of the categories and their relationships.

Finally, OntoMap was a project with the goal to facilitate the access, understanding, and reuse of such resources. A semantic framework on conceptual level was implemented that was small and easy enough to be learned on-the-fly. Technically, OntoMap was implemented as a web-site providing access to several upper-level ontologies and manual mapping between them. OntoMap was similar in spirit to COSMO and MSO, but only the very top concepts of each of the Upper Ontologies considered there were aligned. Unfortunately, OntoMap was over 4 years ago, and no maintenance was guaranteed to it. The

web-portal which was allowing online browsing is no longer available, but the stand-alone viewer may be downloaded from http://www.ontotext.com/projects/OntoMapViewer/install.htm.

III. Comparison

Some partial comparisons exist among subsets of the Upper Ontologies that we have considered in Section II. In the next paragraphs, we have summarised them in the most faithful way. The interested reader should go to the source, always cited, in order to have a complete picture of the conclusions reached by the comparisons' authors. The last paragraph, instead, provides a synthesis of the description we have given in Section II.

i) Pease's comparison of DOLCE and SUMO.: In [15] and [16], Pease draws a comparison between DOLCE and SUMO. His conclusions are that DOLCE has a similar purpose and business process to SUMO in that it is a free research project for use in both natural language tasks and inference. DOLCE has been carefully crafted with respect to strong principles. DOLCE is an "ontology of particulars"; it does have universals (classes and properties), but the claim is that they are only employed in the service of describing particulars. In contrast, SUMO could be described as an ontology of both particulars and universals. It has a hierarchy of properties as well as classes. This is a very important feature for practical knowledge engineering, as it allows common features like transitivity to be applied to a set of properties, with an axiom that is written once and inherited by those properties, rather than having to be rewritten, specific to each property. Other differences include DOLCE's use of a set of meta-properties as a guiding methodology, as opposed to SUMO's use and formal definition of such meta-properties directly in the ontology itself. With respect to SUMO, DOLCE does not include such items as a hierarchy of process types, physical objects, organisms, units and measures, and event roles.

j) Onto-Med's comparison of GFO, DOLCE, and Sowa's ontology.: In [10], informal mappings from GFO to DOLCE and from GFO to Sowa's ontology, and viceversa, are specified. The authors of the comparison observe that all of Sowa's categories except for three can be reinterpreted in GFO. However, mapping in the opposite direction seems to be more problematic. For many of GFO categories, the corresponding notions in Sowa's ontology has not been found. Neither a space-time model nor a property model is included in Sowa's ontology, and the construction method of GFO is not as strictly combinatorial as is Sowa's ontology. In DOLCE, levels of reality are not introduced explicitly, while in GFO the authors explicitly distinguish three levels of reality. Universals are excluded from DOLCE, which supports neither the distinctions provided in GFO concerning sets and items, nor concerning the typology of categories. A time or a space model is not built directly into DOLCE. Instead, the representation of various models of space and time is permitted, which can be introduced by means of qualities and their associated "qualia" (the latter are similar to GFO's quality values). In the GFO, spatial location is modelled in terms of spatial regions and relations, like occupation and location; temporal location is based on time regions and projection relations. In addition, presently the GFO provides a model for time and space. The DOLCE distinction between endurant and perdurant is based on the behavior of entities in time. Endurants are entities that can change in time, are wholly present at any time of their existence, and have no temporal parts but their parts are time-indexed, and participate in perdurants. GFO distinguishes between persistence through time and being wholly present at a time-boundary. This has produced two GFO categories instead of endurant alone: persistants and presentials. GFO persistant refers to the idea of persistence through time as attributed to DOLCE's endurant, although persistants are not considered in GFO as individuals but as universals[1]. GFO presentials can be generally interpreted as DOLCE endurants, but without temporal extension. Intuitively, DOLCE notion of perdurant corresponds to GFO notion of occurrent. Moreover, it seems that the GFO notions of process, state and change can be interpreted in DOLCE as stative, state and event, respectively. Finally, the GFO categories that concern properties and their values correspond rather well to DOLCE qualities, qualia and quality spaces.

k) MITRE's comparison of SUMO, Upper Cyc, and DOLCE.: In [18], Semy, Pulvermacher and Obrst compare SUMO, Upper Cyc, and DOLCE according to the existence of an open license, modularity and evidence of use. We have adopted these criteria inside our analysis, which thus subsumes Semy, Pulvermacher and Obrst's one.

l) Grenon's comparison of DOLCE and BFO.: Grenon made a careful comparison between DOLCE and BFO [7]. The conclusion is that both ontologies contain a category of endurants and perdurants and an eternalist stance, and that the theory of parthood and the theory of dependence are similar in the two ontologies. Despite these similarities, there are also many differences, including:

- DOLCE is methodologically fundamentally conceptualist while BFO is methodologically fundamentally realist;
- DOLCE seems to be oriented toward commonsense, and BFO's naïve realism is in the same spirit. However, DOLCE distinguishes between abstract and concrete entities, and it includes agents and intentionality. BFO is deliberately not committed to these distinctions. In particular, the physical / non-physical endurants distinction in DOLCE is absent in BFO.
- As already mentioned, DOLCE is intended as an ontology of particulars. BFO is intended to be an ontology of both universals and particulars.
- In DOLCE, qualities are abstract entities which may not be found in space or time, and do not have parts. For BFO, the proxies of DOLCE's qualities ("tropes") are located in space and exist at a time in the very same way that the entities in which they inhere.

[1]The forthcoming release of GFO, expected by early 2007, will include some refinements of the notion of persistence which will make this statement no longer valid.

Another source of information about the similarities and differences between DOLCE and BFO is [12], where Masolo, Borgo, Gangemi, Guarino, and Oltramari of the Laboratory For Applied Ontology (LOA) compare DOLCE and BFO (besides the OCHRE object-centered ontology, [17], that we did not consider in our analysis) by representing the assertion "A statute of clay exists for a period of time going from t_1 to t_2. Between t_2 and t_3, the statue is crashed and so ceases to exists although the clay is still there." in both of them.

m) Other existing sources of comparison.: Evaluations of three Candidate Common Upper Ontologies, including SUMO and MSO, can be found at http://suo.ieee.org/SUO/Evaluations/. The criteria considered there include maturity, robustness, potential for broad acceptance, language flexibility, ownership/cost, and domain friendliness. These evaluations are not comparative: each Upper Ontology is evaluated (usually, by its creator) according to the above metrics.

n) Our comparison.: The description of the 7 Upper Ontologies given in Section II is synthesised here in Tables I and II.

IV. Conclusions

This paper represents a preliminary step towards the exploitation of upper ontologies as the means for allowing intelligent software agents to integrate heterogeneous sources of information, respecting privacy issues that are more and more commong in many scenarios, such as virtual enterprises and e-commerce. In fact, this paper provides an original and unpublished analysis of the state-of-the-art in the field of upper ontologies. This analysis is a necessary activity before starting to think how upper ontologies may be actually exploited as a bridge among two or more ontologies to be integrated. If the original ontologies cannot be disclosed for privacy issues, each agent involved in the application and responsible for accessing and integrating one ontology, may "align" (i.e. find a mapping between concepts) its own, private ontology, with the upper ontology, and refer to the latter one in all its communicative acts. At the time of writing, the design of an algorithm for aligning ontologies using upper ontologies as a bridge is under way. As soon as we will be able to implement and test it, we will obtain results that will give us an important help in understanding under which conditions the exploitation of upper ontologies is feasible, and which upper ontologies are better for being used as a bridge in the alignment process. Our current and future work is entirely aimed at completing the design and implementation of the algorithm and systematically describing our experimental results.

Acknowledgments

We want to acknowledge all the researchers that helped in drawing this comparison with their constructive comments and useful advices. In particular, many thanks go to J. Euzenat, A. Kiryakov, L. Lefkowitz, F. Loebe, A. Pease, J. Schoening, P. Shvaiko, and H. Stenzhorn.

We also acknowledge the research projects TIN2006-15265-C06-04 and "Iniziativa Software" CINI-FINMECCANICA that partially funded this work.

References

[1] American National Standard. KIF Knowledge Interchange Format – draft proposed American National Standard (dpANS) NCITS.T2/98-004, 1998.

[2] P. Burek, R. Hoehndorf, F. Loebe, J. Visagie, H. Herre, and J. Kelso. A top-level ontology of functions and its application in the open biomedical ontologies. In *ISMB (Supplement of Bioinformatics)*, volume 22, pages 66–73, 2006.

[3] 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, *Proceedings of the 3rd International Workshop on Regulatory Ontologies (WORM 2005)*, volume 3762 of *Lecture Notes in Computer Science*, pages 846–855. Springer, 2005.

[4] J. Curtis, D. Baxter, and J. Cabral. On the application of the Cyc ontology to word sense disambiguation. In *Proceedings of the 19th International Florida Artificial Intelligence Research Society Conference*, pages 652–657, 2006.

[5] J. Curtis, G. Matthews, and D. Baxter. On the effective use of Cyc in a question answering system. In *Proceedings of the IJCAI Workshop on Knowledge and Reasoning for Answering Questions (KRAQ'05)*, 2005.

[6] C. Deaton, B. Shepard, C. Klein, C. Mayans, B. Summers, A. Brusseau, and M. Witbrock. The comprehensive terrorism knowledge base in Cyc. In *Proceedings of the 2005 International Conference on Intelligence Analysis*, 2005.

[7] P. Grenon. BFO in a nutshell: A bi-categorial axiomatization of BFO and comparison with DOLCE. Technical Report 06/2003, IFOMIS, University of Leipzig, 2003.

[8] 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.

[9] G. Guizzardi, H. Herre, and G. Wagner. On the general ontological foundations of conceptual modeling. In S. Spaccapietra, S. T. March, and Y. Kambayashi, editors, *Proceedings of the 21st International Conference on Conceptual Modeling (ER 2002)*, volume 2503 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2002.

[10] H. Herre, B. Heller, P. Burek, R. Hoehndorf, F. Loebe, and H. Michalek. General formal ontology (GFO) – part I: Basic principles. Technical Report 8, Onto-Med, University of Leipzig, 2006.

[11] A. Kiryakov, K. Ivanov Simov, and M. Dimitrov. OntoMap: portal for upper-level ontologies. In *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS 2001)*, pages 47–58. ACM, 2001.

[12] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Ontology library (final). IST Project 2001-

33052 WonderWeb Deliverable D18, 2003.

[13] G. Nagypál and J. Lemcke. A business data ontology. Data, Information and Process Integration with Semantic Web Services Project, FP6 Ű 507483, Deliverable D3.3, 2005.

[14] I. Niles and A. Pease. Towards a standard upper ontology. In C. Welty and B. Smith, editors, *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, pages 2–9. ACM Press, 2001.

[15] A. Pease. Formal representation of concepts: The Suggested Upper Merged Ontology and its use in linguistics. In A. C. Schalley and D. Zaefferer, editors, *Ontolinguistics. How Ontological Status Shapes the Linguistic Coding of Concepts*. Mouton de Gruyter, 2006.

[16] A. Pease and C. Fellbaum. Formal ontology as interlingua: The SUMO and WordNet linking project and GlobalWordNet. To appear.

[17] L. Schneider. How to build a foundational ontology: The object-centered high-level reference ontology OCHRE. In A. Günter, R. Kruse, and B. Neumann, editors, *Proceedings of the 26th Annual German Conference on AI, KI 2003: Advances in Artificial Intelligence*, volume 2821 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2003.

[18] S. K. Semy, M. K. Pulvermacher, and L. J. Obrst. Toward the use of an upper ontology for U.S. government and U.S. military domains: An evaluation. Technical Report MTR 04B0000063, The MITRE Corporation, 2004.

[19] B. Shepard, C. Matuszek, C. B. Fraser, W. Wechtenhiser, D. Crabbe, Z. Güngördü, J. Jantos, T. Hughes, L. Lefkowitz, M. J. Witbrock, D. B. Lenat, and E. Larson. A knowledge-based approach to network security: Applying Cyc in the domain of network risk assessment. In M. M. Veloso and S. Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference*, pages 1563–1568. AAAI Press AAAI Press / The MIT Press, 2005.

[20] J. F. Sowa. In *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing, 1999.

[21] W3C. OWL Web Ontology Language Overview – W3C Recommendation 10 February 2004 , 2004.

[22] Wikipedia. Upper ontology – wikipedia, the free encyclopedia, 2006. [Online; accessed 15-December-2006].

	Home page	Developers	Dimensions	Language(s)
BFO	http://www.ifomis.org/bfo	Smith, Grenon, Stenzhorn, Spear (IFOMIS)	36 classes related via the is_a relation	OWL
Cyc	http://www.cyc.com/	Cycorp	About 300,000 concepts, 3,000,000 assertions (facts and rules), 15,000 relations (these numbers include microtheories)	CycL, OWL
DOLCE	http://www.loa-cnr.it/DOLCE.html	Guarino and other researchers of the LOA	About 100 concepts and 100 axioms	First Order Logic, KIF, OWL
GFO	http://www.onto-med.de/ontologies/gfo.html	The Onto-Med Research Group	79 classes, 97 subclass-relations, 67 properties	First Order Logic and KIF (forthcoming); OWL
PROTON	http://proton.semanticweb.org/	Ontotext Lab, Sirma	300 concepts and 100 properties	OWL Lite
Sowa's	http://www.jfsowa.com/ontology/	Sowa	30 classes, 5 relationships, 30 axioms	First Order Modal Language, KIF
SUMO	http://www.ontologyportal.org/	Niles, Pease, and Menzel	20,000 terms and 60,000 axioms (including domain ontologies)	SUO-KIF, OWL

Table I
COMPARISON, PART I

	Modularity	Applications	Alignment with WordNet	Licensing
BFO	SNAP and SPAN modules	Mainly in the biomedical domain	Not supported	Freely available
Cyc	"Microtheory" modules	Natural language processing, network risk assessment, terrorism management	Cyc is mapped to about 12,000 WordNet synsets	Commercial product; ResearchCyc and OpenCyc are instead freely available (ResearchCyc for research purposes only)
DOLCE	Not divided into modules	Multilingual information retrieval, web-based systems and services, e-learning	DOLCE-Lite-Plus has been aligned with about 100 Wordnet sysnsets	Freely available
GFO	Abstract top level, abstract core level, basic level	Mainly in the biomedical domain	Not supported	Released under the modified BSD Licence
PROTON	Three levels including four modules	Semantic annotation within the KIM platform, knowledge management systems in legal and telecommunications domain, media research and analysis, research intelligence, Business Data Ontology for Semantic Web Services.	Not supported	Freely available
Sowa's	Not divided into modules	No documented applications have been developed, but Sowa's ontology inspired the creation of many implemented Upper Ontologies	Not supported	Freely available
SUMO	Divided into SUMO itself, MILO, and domain ontologies	Linguistics, representation, reasoning	SUMO has been mapped to all of Wordnet v2.1 by hand	Freely available

Table II
COMPARISON, PART II

A Swarm Intelligence Method Applied to Manufacturing Scheduling

Davide Anghinolfi, Antonio Boccalatte, Alberto Grosso, Massimo Paolucci, Andrea Passadore, Christian Vecchiola, *DIST – Department of Communications Computer and System Sciences, University of Genova*

Abstract—In this paper we present a multi-agent search technique to face the NP-hard single machine total weighted tardiness scheduling problem in presence of sequence-dependent setup times. The search technique is called Discrete Particle Swarm Optimization (DPSO): differently from previous approaches the proposed DPSO uses a discrete model both for particle position and velocity and a coherent sequence metric. We tested the proposed DPSO over a benchmark available online. The results obtained show the competitiveness of our DPSO, which is able to outperform the best known results for the benchmark, and the effectiveness of the DPSO swarm intelligence mechanisms.

***Index Terms*—Particle Swarm Optimization, Swarm Intelligence, Scheduling**

I. Introduction

In this paper we propose a new DPSO approach to face the single machine total weighted tardiness scheduling with sequence-dependent setup times (STWTSDS) problem. Scheduling with performance criteria involving due dates, such as (weighted) total tardiness or total earliness and tardiness (E-T), and that takes into account sequence-dependent setups, is a reference problem in many real industrial contexts. Meeting due dates is in fact recognized as the most important objective in surveys on manufacturing practise, e.g., in [1]. The objective of minimizing the total weighted tardiness has been the subject of a very large amount of literature on scheduling even if sequence-dependent setups have not been so frequently considered. Setups usually correspond to preparing the production resources (e.g., the machines) for the execution of the next job, and when the duration of such operations depends on the type of last completed job, the setups are called sequence-dependent. The presence of sequence-dependent setups greatly increases the problem difficulty, since it prevents the application of dominance conditions used for simpler tardiness problems [2]. The choice of the STWTSDS problem as reference application for the proposed DPSO approach has then two main motivations: first the fact that the solution of single machine problems is often required even in more complex environments [3], and second the absence, to the best authors' knowledge, of any other DPSO approach in literature for the STWTSDS problem. Regarding the latter point, note that the approach in [4] seems to be the only previous DPSO application to the single machine total weighted tardiness (STWT) problem.

The rest of the paper is organized as follows. Section 2 introduces a formal problem definition and provides a general review of the relevant literature for it. Section 3 illustrates the basic aspects of the PSO algorithm, analysing in particular the DPSO approaches previously proposed in the literature. Section 4 then describes the proposed DPSO approach, discussing how it can be applied to the STWTSDS problem and highlighting the new features introduced. Section 5 presents the experimental campaign performed, which is mainly based on the benchmark set generated by Cicirello in [5] and available on the web. Finally, Section 6 draws some conclusions.

II. The STWTSDS Problem

The STWTSDS problem corresponds to the scheduling of n independent jobs on a single machine. All the jobs are released simultaneously, i.e., they are ready at time zero, the machine is continuously available and it can process only one job at a time. For each job j=1,..., n, the following quantities are given: a processing time p_j, a due date d_j and a weight w_j. A sequence-dependent setup time s_{ij} must be waited before starting the processing of job j if it is immediately sequenced after job i. The tardiness of a job j is defined as $T_j=max(0, C_j-d_j)$, being C_j the job j completion time. The scheduling objective is the minimization of the total weighted tardiness expressed as $\sum_{j=1}^{n} w_j T_j$. This problem, denoted as $1/s_{ij}/\sum w_j T_j$, is strongly NP-hard since it is a special case of the $1//\sum w_j T_j$ that has been proven to be strongly NP-hard in [6]. In the literature both exact algorithms and heuristic algorithms have been proposed for the STWTSDS problem or for a slightly different version disregarding the job weights. However, since only instances of small dimensions can be solved by exact approaches, recent research efforts have been focused on the design of heuristics. The apparent tardiness cost with setups (ATCS) heuristic [7] is currently the best *constructive* approach for the STWTSDS problem. Constructive heuristics require a small computational effort, but they are generally outperformed by *improvement*

approaches, based on local search algorithms, and *metaheuristics*, which on the other hand are much more computational time demanding. The effectiveness of such approaches has been largely demonstrated: for example, Potts and van Wassenhove [8] show as simple pair-wise interchange methods outperform dispatching rules for the STWT problem, as well as more recently constructive heuristics appear dominated by a memetic algorithm in [9] or by a hybrid metaheuristic in [10] where a similar parallel machine case is considered. The effectiveness of stochastic search procedures for the STWTSDS is shown in [11], where the authors compare a value-biased stochastic sampling (VBSS), a VBSS with hill-climbing (VBSS-HC) and a simulated annealing (SA), to limited discrepancy search (LDS) and heuristic-biased stochastic sampling (HBSS) on a 120 benchmark problem instances for the STWTSDS problem defined by Cicirello [5]. The literature about applications of metaheuristics to scheduling is quite extended. In [12] an ant colony optimization (ACO) algorithm for the STWTSDS is proposed, which is able to improve about 86% of the best known results for the Cicirello's benchmark previously found by stochastic search procedures in [11]. Recently the Cicirello's best known results have been further independently improved in [13] by means of a GA approach, in [14] with three SA, GA and tabu search (TS) algorithms, and in [15] using an ACO approach; in particular, the new set of best known results established by Lin and Ying [14], which improved more than 71% of the previous best known solutions, was lastly updated by the ACO by Anghinolfi and Paolucci [15] that was able to improve 72.5% of the Lin and Ying solutions.

III. Overview of the Basic PSO Algorithm and its Discrete Variant

Particle Swarm Optimization (PSO) algorithm is a recent metaheuristic approach motivated by the observation of the social behaviour of composed organisms, such as bird flocking and fish schooling, and it tries to exploit the concept that the knowledge to drive the search for optimum is amplified by social interaction. PSO executes a population-based search procedure in which the exploring agents, called *particles*, adjust their positions during time (the particles *fly*) according not only to their own experience, but also to the experience of other particles: in particular, a particle may modify its position with a velocity that in general includes a component moving the particle towards the best position so far achieved by the particle itself to take into account its personal experience, and a component moving the particle towards the best solution so far achieved by any among a set of neighbouring particles (*local neighbourhood*) or by any of the exploring particles (*global neighbourhood*). PSO is based on the *Swarm Intelligence* (SI) concept [16]. This is a particular agent-based modelling technique which mostly relies on the cooperation among large number of simple agents in order to model an autonomous self-organizing system for solving optimization problems. The agents are able to exchange information in order to share experiences, and the performance of the overall multi-agent system (the swarm) emerges from the collection of the simple agents' interactions and actions. PSO has been originally developed for continuous nonlinear optimization ([17]; [18]). The basic algorithm for a global optimization problem, corresponding to the minimization of a real objective function $f(x)$, uses a population (*swarm*) of m particles. Each particle i of the swarm is associated with a position in the continuous n-dimensional search space, $x_i=(x_{i1},\ldots, x_{in})$ and with the correspondent objective value $f(x_i)$ (*fitness*). For each particle i, the best previous position, i.e. the one where the particle found the lowest objective value (*personal best*), and the last particle position change (*velocity*) are recorded and represented respectively as $p_i=(p_{i1},\ldots, p_{in})$ and $v_i=(v_{i1},\ldots, v_{in})$. The position associated with the current smallest function value is denoted as $g=(g_1,\ldots, g_n)$ (*global best*). Denoting with x_i^k and v_i^k respectively the position and velocity of particle i at iteration k of the PSO algorithm, the following equations are used to iteratively modify the particles' velocities and positions:

$$v_i^{k+1} = w \cdot v_i^k + c_1 r_1 \cdot (p_i - x_i^k) + c_2 r_2 \cdot (g - x_i^k) \quad (1)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (2)$$

where w is the *inertia* parameter that weights the previous particle's velocity; c_1 and c_2, respectively called *cognitive* and *social* parameter, multiplied by two random numbers r_1 and r_2 uniformly distributed in [0, 1], are used to weight the velocity towards the particle's personal best, $(p_i - x_i^k)$, and the velocity towards the global best solution, $(g - x_i^k)$, found so far by the whole swarm. The new particle position is determined in (2) by adding to the particle's current position the new velocity computed in (1). The PSO parameters that must be fixed are the inertia w, the cognitive and social parameters c_1 and c_2, and finally the dimension of the swarm m.

In recent years several studies applying the PSO approach to discrete combinatorial optimization problems appeared in the literature; however, to the best authors' knowledge, none of them faced the STWTSDS problem. PSO has been applied to combinatorial optimization problems, as travelling salesman problem (TSP) [19], vehicle routing problem [20], and scheduling problems ([4]; [12]; [21]; [22]; [23]; [24]). DPSO approaches differ both for the way they associate a particle position with a discrete solution and for the velocity model used; in particular, we could classify DPSO approaches in the literature according to three kinds of solution-particle mapping, i.e., binary, real-valued and permutation-based, and three kinds of velocity model used, i.e., real-valued, stochastic or based on a list of moves. The first DPSO algorithm proposed in [25] is characterized by a binary solution representation and a stochastic velocity model since it associates the particles with n-dimensional binary variables and the velocity with the probability for each binary dimension to take value one. A variation of this DPSO to face flow shop scheduling problems is defined in [26]. A different model is used in [4] to develop a PSO algorithm for the STWT problem and in [27] for the total flowtime

minimization in permutation flow shop problems: using a technique similar to the *random key representation* [28], real values are associated with the particle dimensions to represent the job place in the scheduling sequence and the *smallest position value* (SPV) rule is exploited to transform the particle positions into job permutations. Permutation-based solution-particle mappings are used in [29] for the n-queens problem together with a stochastic velocity model, representing the probability of swapping items between two permutation places, and a mutation operator, consisting of a random swap executed whenever a particle coincides with the *local* (*global*) *best* one.

The velocity models used in all the DPSO approaches above mentioned are either stochastic or real-valued. To the best authors' knowledge the unique example of velocity model based on a list of moves can be found in the DPSO approach for the TSP in [30]. The reason why this kind of model has not been investigated in the scheduling literature may be explained by the main difficulty of defining new appropriate sum and multiplication operators for equations (1) and (2) to make them work in a discrete solution space. Nevertheless, in the following section we propose a new DPSO approach to single machine scheduling based on both a permutation solution-particle representation and on a list-of-moves velocity model.

IV. The Proposed DPSO Approach

To pursuit our purpose we will need to redefine all the arithmetical operators involved in equations (1) and (2). This redefinition will lead in general to build unfeasible sequences (*pseudo-sequences*) that will be fixed and converted in feasible sequences with a procedure called *sequence completion procedure*.

Let us first introduce some notation. In general, a solution x to the problem of scheduling n independent jobs on a single machine is associated with a sequence $\sigma=([1],...,[n])$. In addition we denote with $\varphi_\sigma:\{1,..., n\}\rightarrow\{1,..., n\}$, the mapping between the places in a sequence σ and the indices of the sequenced jobs; for example, if job j is sequenced in the h-th place of σ we have $j= \varphi_\sigma(h)$. In the proposed DPSO we consider a set of m particles; each particle i is associated with a sequence σ_i, i.e., a schedule x_i, and it has a *fitness* given by the cost value $Z(x_i)$. Thus, the space explored by the *flying* particles is the one of the sequences. In the following we introduce a metric for such a space, called *sequence metric*, that is, a set of operators to compute velocities and to update particles' positions consistently.

A. The particle velocity and the sequence metric operators

Given a pair of particles p and q, we define the distance between them as the difference between the associated sequences (*position difference*), i.e., $\sigma_q-\sigma_p$, which corresponds to a list of moves that we call *pseudo-insertion* (PI) *moves*. We denote a PI move as (j, d), where d is the integer displacement that must be applied to job j to *direct* the particle p toward q. Roughly speaking, assuming for example that $j=\varphi_\sigma(h)$, a positive displacement d corresponds to a towards-right move that extracts job j from its current place h and reinserts it in place $\min(h+d, n)$ in the sequence, so generating a new sequence σ' such that $j=\varphi_{\sigma'}(\min(h+d, n))$, and a corresponding solution x'; analogously, a negative displacement $-d$ corresponds to a towards-left extraction and reinsertion move that generates a new sequence such that $j=\varphi_{\sigma'}(\max(h-d, 0))$. The difference between the positions of two particles p and q defines a *velocity* v, which consequently is a set of PI moves; then, applying the PI moves in v to p we can move this particle to the position of particle q. The following example would simply illustrate this concept. Let the number of jobs n=4 and the sequences corresponding to the positions of two particles p and q respectively σ_p=(1,2,3,4) and σ_q=(2,3,1,4); then, the velocity associated with the difference between the two positions is $v=\sigma_q-\sigma_p=\{(1,2),(2,-1),(3,-1)\}$; here the PI move (1, 2) denotes that job 1 must be delayed (moved towards-right) of 2 places in the sequence to direct particle p towards q. Note that a velocity can include at most a single PI move for a given job. The reason why we denote as "pseudo-insertion" such kind of moves is that, as detailed in the following, in general the rule used to apply the PI moves in a velocity to a sequence may fail to produce a feasible sequence, but it may produce a so-called *pseudo-sequence*, and we need to introduce a final *sequence completion procedure* to correctly implement equation (2) in the sequence metric.

The position-velocity sum operator applies one PI move composing the velocity at a time, first to the initial sequence and then to the *pseudo-sequences* successively obtained, hereafter denoted by π.

B. The sequence completion procedure

In general, the pseudo-sequences produced do not correspond to feasible sequences since some sequence places may be left empty whereas some others may contain a list of jobs. If, for example, we apply the move (1,2) to σ_p=(1,2,3,4), we obtain the pseudo-sequence π=(-,2,[3,1],4), where "- " denotes that no job is assigned to the first place of π, whereas [3,1] represents the ordered set of jobs assigned to the third place of π. Let us denote with $\pi(h)$ the ordered set of items in the h-th place of the pseudo-sequence π; with *pull*(s) the function that extracts the first element from an ordered set s, and with *push*(i, s) the function that inserts the element i at the bottom of the set s. Then, in order to convert a pseudo-sequence into a feasible sequence, the sequence completion procedure manages $\pi(h)$ as a first-in-first-out (FIFO) list, as reported in Fig. 1.

As an example, the behaviour of such a procedure for a pseudo-sequence π=([1,3],-,-,[4,2]) is shown in Fig. 2.

```
Input: π a pseudo-sequence
Output: σ a feasible sequence
for each h=1,...,n
{
   if |π(h)|=1 skip;
   else if |π(h)|=0
   {
      repeat
         k=h+1;
      while k<n and |π(k)|=0
      π(h)=pull(π(k));
   }
   else if |π(h)|>1
   {
      while |π(h)|>1
         push(pull(π(h), π(h+1));
   }
}
σ=π;
```

Fig. 1: The sequence completion procedure.

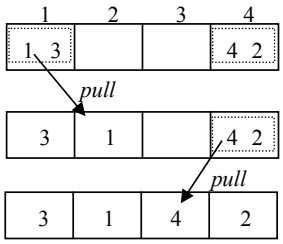


Fig. 2: An example of sequence completion procedure execution.

The procedure considers one place at a time of π starting from the first one on the left; since an ordered set of jobs is encountered in place h=1, then the first job is extracted and reinserted in the first following empty position (h=2), thus, the pseudo-sequence is updated as (3,1,-,[4,2]); then place h=2 is skipped because it contains just one job. In h=3 an empty place is encountered, so the procedure extracts a job from the next not empty place, here place 4 containing the FIFO list [4,2], and reinserts it there; after this step the final feasible sequence (3,1,4,2) is obtained.

It is easy to verify that the iterated application of the extract-reinsert operator in (3) to compute σ_p+v in the case of the first example where σ_p=(1,2,3,4), σ_q=(2,3,1,4), and v={(1,2),(2,-1),(3,-1)} directly gives the target sequence σ_q since π_0=(1,2,3,4), π_1=(-,2,[3,1],4), π_2=(2,-,[3,1],4) and finally π_3=(2,3,1,4).

A velocity v can be summed to another velocity v' producing a new velocity w. This is a different sum operator (*velocity sum*) that generates the resulting velocity as the union of the moves in v and v'. Any job can appear only once in the set of pseudo-moves defining a velocity; therefore, if v and v' include respectively (j, d) and (j, d'), then the resulting sum w must include the pseudo-insertion move (j, $d+d'$). Note that if $d+d'=0$ the move is removed from the list.

Finally, a velocity v can be multiplied by a real positive constant c (*constant-velocity multiplication*) generating a new velocity $w=c \cdot v$. We devised the following constant-velocity multiplication rule according to which the constant c modifies the displacement values of the pseudo-moves included in $v=\{(j_1,d_1),...,(j_s,d_s)\}$; in particular, this rule produces a velocity $w=\{(j_1, c \cdot d_1),...,(j_s, c \cdot d_s)\}$.

C. The overall DPSO algorithm

The very high level structure of the developed DPSO algorithm is given in Fig. 3. In the following we will describe each step in the detail.

```
Initialization;
While <termination condition not met>
{
   For each particle p belonging to P
   {
      Update particle velocity;
      Update particle position;
      Compute particle fitness;
   }
   Intensification phase;
   Update best references;
}
```

Fig. 3: The overall D-PSO algorithm.

Initialization. An initial sequence σ_i^0, i=1,..., m, (i.e., an initial solution x_i^0) is assigned to each of the m particles. In particular, we use three different constructive heuristics, the earliest due date (EDD), the shortest processing time (SPT), and the apparent tardiness cost with setups (ATCS) to generate three different starting sequences. Then, a set of v_i^0, i=1,...,m initial velocities is randomly generated and associated with the particles. Finally, the initial position for each particle i is produced first randomly selecting one among the first three starting sequences and then summing the correspondent initial velocity v_i^0.

Velocity and position update. At iteration k, each particle i first computes the following components: the inertial velocity (iv), the directed to personal best velocity (pv), and the directed to global best velocity (gv), according to the following equations:

$$iv_i^k = w \cdot v_i^{k-1} \tag{3}$$

$$pv_i^k = c_1 r_1 \cdot (p_i - \sigma_i^{k-1}) \tag{4}$$

$$gv_i^k = c_2 r_2 \cdot (g - \sigma_i^{k-1}) \tag{5}$$

where r_1 and r_2 are independent random numbers extracted from U[0,1].

Then the particle velocity at iteration k is updated with a procedure that separately sums to the current particle position

each velocity component one at a time (in the *iv*, *pv*, *gv* order), thus moving the particle through a set of intermediate sequences. For example, denoting with *is* the intermediate sequences, we must execute three sums, $is_1 = \sigma_i^{k-1} + iv_i^k$, $is_2 = is_1 + pv_i^k$ and $\sigma_i^k = is_2 + gv_i^k$ in order to update the position of a particle.

Finally, the schedule x_i^k associated with the updated particle position σ_i^k is determined by a straightforward timetable procedure, and the fitness $Z(x_i^k)$ is computed. Note that if the velocity for a particle becomes null then it is reinitialized by a random restart.

Intensification phase. After all the particles have updated their position and computed their fitness at an iteration *k*, an intensification phase is performed consisting of a local search (LS) exploration that starts from the best solution found by the particles in the current iteration. We adopt a stochastic LS (S-LS) algorithm similar to the one in [4], which in turn is based on a variant of the variable neighbourhood search (VNS) [31].

The S-LS algorithm performs a random neighbourhood exploration allowing an alternation of random insert and swap moves with a maximum number of random restarts bounded by n/5; thus the overall complexity of the LS algorithm is $O(n^3)$. After the intensification phase, the solution obtained by the S-LS algorithm is substituted to the starting x_{i*}^k for the particle i^*, whose position and fitness are updated accordingly.

Update of the best references. After the completion of the intensification phase, the global and the personal best position for the particles may be updated.

V. Experimental Analysis of the Algorithm

We coded the DPSO algorithm in C++ and implemented it on a Pentium IV, 2.8 GHz, 512 Mb PC. We extensively tested the behaviour of the proposed DPSO through an experimental campaign mainly based on the benchmark due to Cicirello [5], which is available on the web at http://www.cs.drexel.edu/~cicirello/benchmarks.html. This benchmark is made of a set of 120 STWTSDS problem instances with 60 jobs. We compared our DPSO algorithm with the following three sets of best reference solutions for the considered benchmark: (a) a set including the overall aggregated best known results, denoted with OBK, mostly composed by the solutions yielded by the SA, GA and TS algorithms in [14] with the addition of few best solutions from the ant colony optimization (ACO) algorithm in [12], and taking also into account the best solutions from the GA in [13]; (b) the set of best results obtained by the ACO algorithm in [12], denoted with ACO-LJ; (c) the most recent set of the best results produced by the authors with an ACO approach denoted with ACO-AP [15].

During all the experimental campaign, we set the number of particles m=120 and we adopted the same termination criterion used in [14] fixing the maximum number of fitness function evaluation = 20,000,000. After a preliminary experiment campaign we also fixed c_1=1.5, c_2=2.0 and w=1.0. Please note that these parameters seemed to be not much sensitive, as also other configurations gave results statistically not different. We executed 10 runs for each instance and then we computed the best results, summarized in Table 1. This table reports the average percentage deviations (*Avg dev*), the related 95% confidence (*Conf*), the percentage number of improved (*Impr sol*) and identical (*Ident sol*) solutions found by DPSO with respect to the three sets of reference solutions. Table 1 clearly shows that the best DPSO solutions outperform on the average the OBK and the ACO-LJ ones, while they are substantially equivalent to the ones in ACO-AP approach.

	Avg dev	Conf	Impr sol	Ident sol
OBK	-2.80	1.72	70.83	18.33
ACO-LJ	-4.60	1.91	65.00	13.33
ACO-AP	-0.24	1.42	31.67	26.67

Table 1: The best results of the DPSO with respect to three solution sets (%)

The dominance of DPSO in the first two comparisons, also witnessed by the 95% confidence results, was also confirmed by statistically significance tests.

At the website http://www.discovery.dist.unige.it/DPSO_best.html the complete best results for each instance can be found with every objective function value and sequence of jobs.

A. The evaluation of the importance of the swarm intelligence mechanisms

In order to finally verify the effectiveness of using swarm intelligence mechanisms in exploring the solution space, we developed a modified version of our DPSO, denoted as Random Particle Search (RPS), removing from the algorithm every memory and particle interaction mechanism. The RPS, starting from the set of solutions initially associated with the *m* particles, executes at each iteration a random position update for each particle and an intensification step with the S-LS for the particle position correspondent to the best solution found in the iteration. Differently from the DPSO, the RPS updates the particle positions computing a random velocity as follows: for each particle dimension, i.e., job *j* in the sequence of the associated solution, a pseudo-insertion move (*j*, *d*) is determined by stochastically generating the job displacement *d* from a normal distribution $N(\mu, \sigma^2)$, with mean μ=0 and standard deviation σ fixed as algorithm parameter. The developed RPS can be viewed as a sort of multiple iterated local search method that uses the velocity concept from DPSO in order to perturb the current solutions, but that does not include any "swarm" interaction mechanism as well as PSO memory structures (personal or global best). We tested three RPS configurations characterized by a different value for the parameter σ, fixing $\sigma \in \{4, 6, 18\}$, executing 10 runs for each configuration on the Cicirello's benchmark, then computing for each instance the best average result over the three RPS configurations. Then we compared the RPS results with the average DPSO solutions finding that the RPS produced an

average percentage deviation from the DPSO of 12.15%, with a 95% confidence of 9.45%. From such results RPS appears dominated by the DPSO and this fact clearly confirms the fundamental role of the DPSO swarm intelligence mechanisms.

VI. Conclusions

In this paper we describe a new DPSO algorithm that we used to face the NP-hard STWTSDS problem. To our best knowledge, this should be the first application of a discrete PSO metaheuristic to this class of scheduling problem. Differently from previous approaches in the literature where PSO has been applied to scheduling problems, our DPSO adopts a discrete model both for particles and velocities, respectively corresponding to job sequences and list of so-called pseudo-insertion moves.

The experimental tests performed on the Cicirello's benchmark demonstrate the competitiveness of the proposed DPSO; in particular, we can highlight the ability of the DPSO of generating excellent average results, as well as its very limited dependency from the parameter values, which makes the algorithm tuning not critical. Finally, we showed the effectiveness of this swarm intelligence method, since turning off interaction and memory mechanisms of agents the performance of the algorithm deteriorates significantly.

References

[1] Wisner, J., & Siferd, S. (1995). A Survey of U.S. Manufacturing Practices in Make-to-Order Machine Shops. Production and Inventory Management Journal , 36, 1-7.

[2] Rubin, P., & Ragatz, G. (1995). Scheduling in a sequence dependent setup environment with genetic search. Computers & Operations Research , 22, 85–99.

[3] Pinedo, M. (1995). Scheduling: Theory, Algorithms, and Systems. Englewood Cliffs, NJ: Prentice Hall.

[4] Tasgetiren, M., Sevkli, M., Liang, Y., & Gencyilmaz, G. (2004). Particle swarm optimization algorithm for single machine total weighted tardiness problem. Proceedings of the IEEE congress on evolutionary computation, vol.2, p. 1412–1419. Portland.

[5] Cicirello, V. (2003). Weighted tardiness scheduling with sequence-dependent setups: a benchmark library. Carnegie Mellon University, USA, Technical Report of Intelligent Coordination and Logistics Laboratory, Robotics Institute.

[6] Lawler, E. (1997). A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics , 1, p. 331–342.

[7] Lee, Y., Bhaskaran, K., & Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transaction , 29, 45-52.

[8] Potts, C., & van Wassenhove, L. (1991). Single machine tardiness sequencing heuristics. IIE Transactions , 23, 346–354.

[9] França, P., Mendes, A., & Moscato, P. (2001). A memetic algorithm for the total tardiness single machine scheduling problem. European Journal of Operational Research (132), 224-242.

[10] Anghinolfi, D., & Paolucci, M. (2007). Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. Computers & Operations Research (34), 3471-3490.

[11] Cicirello, V., & Smith, S. (2005). Enhancing stochastic search performance by value-based randomization of heuristics. Journal of Heuristics (11), 5–34.

[12] Liao, C., & Juan, H. (2007). An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers & Operations Research (34), 1899-1909.

[13] Cicirello, V. (2006). Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respects Absolute Position. Proceeding of GECCO'06 Conference, (p. 1125-1131). Seattle, Washington, USA.

[14] Lin, S., & Ying, K. (2006). Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. The International Journal of Advanced Manufacturing Technology .

[15] Anghinolfi, D., & Paolucci, M. (2007). A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. accepted for publication on International Journal of Operations Research .

[16] Kennedy, J., & Eberhart, R. (2001). Swarm Intelligence. San Francisco: Morgan Kaufmann Publishers.

[17] Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. Proceeding of the 1995 IEEE International Conference on Neural Network (p. 1942-1948). IEEE Press.

[18] Abraham, A., Guo, H., & Liu, H. (2006). Swarm Intelligence: Foundations, Perspectives and Applications. Swarm Intelligence in Data Mining, Studies in Computational Intelligence (series) .

[19] Pang, W., Wang, K., Zhou, C., & Dong, L.-J. (2004). Fuzzy discrete particle swarm optimization for solving traveling salesman problem. Proceedings of the 4th International Conference on Computer and Information Technology (p. 796 – 800). IEEE CS Press.

[20] Chen, A., Yang, G., & Wu, Z. (2006). Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. Journal of Zhejiang Univ. SCIENCE A (7), 607-614.

[21] Lian, Z., Gu, X., & Jiao, B. (2006). A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. Applied Mathematics and Computation (175), 773-785.

[22] Lian, Z., Gu, X., & Jiao, B. (2006). A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. Applied Mathematics and Computation (183), 1008-1017.

[23] Allahverdi, A., & Al-Anzi, F. (2006). A PSO and a Tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. Computers & Operations Research (33), 1056–1080.

[24] Parsopoulos, K., & Vrahatis, M. (2006). Studying the Performance of Unified Particle Swarm Optimization on the Single Machine Total Weighted Tardiness Problem. Lecture Notes in Artificial Intelligence (4304), 760-769.

[25] Kennedy, J., & Eberhart, R. (1997). A discrete binary version of the particle swarm algorithm. Proceedings of the International Conference on Systems, Man, and Cybernetics. vol. 5, p. 4104–4108. IEEE Press.

[26] Liao, C.-J., Tseng, C.-T., & Luarn, P. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. Computers & Operations Research (34), 3099-3111.

[27] Tasgetiren, M., Liang, Y.-C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research (177), 1930-1947.

[28] Bean, J. (1994). Genetic algorithm and random keys for sequencing and optimization. ORSA Journal on Computing (6), 154-160.

[29] Hu, X., Eberhart, R., & Shi, Y. (2003). Swarm intelligence for permutation optimization: a case study of n-queens problem. Proceedings of the 2003 IEEE Conference on Swarm Intelligence Symposium (SIS '03) (p. 243-246). IEEE Press.

[30] Clerc, M. (2004). Discrete Particle Swarm Optimization. Onwubolu GC, Babu BV (Eds), New Optimization Techniques in Engineering , 219-240.

[31] Mladenovic, N., & Hansen, P. (1997). Variable Neighbourhood Search. Computers & Operations Research (24), 1097-1100.

An Agent Based Solution for Dispatching Items in a Distributed Environment

Christian Vecchiola, Alberto Grosso, Andrea Passadore, Davide Anghinolfi, Antonio Boccalatte, Massimo Paolucci, *DIST – Department of Communications Computer and System Sciences, University of Genova*

Abstract—This paper describes Herald, an agent based toolkit for dispatching and processing items in a distributed environment. Herald is suitable for scenarios where the process could be modeled as a tree: starting from the root node the collection of items is distributed along the nodes where they can be processed, forwarded to other nodes, and duplicated if necessary. Herald assigns a specific software agent to each node of the tree which participates into the dispatching process according to the knowledge base of the multi-agent system. Herald works as a general infrastructure for simulating, testing and executing dispatching algorithms that can be easily integrated into the system by changing the decision making process of the agents composing the architecture. A prototypal implementation, based on the AgentService programming framework, is then presented as a proof of its applicability in industrial scenarios.

***Index Terms*—Agent Oriented Software Engineering, Agent-based coordination, software agents for logistics.**

I. INTRODUCTION

The problem of distributing items on a somehow hierarchical structure is common to many different application contexts, for example logistics [1], routing [2], and scheduling [3]. In all the previously cited contexts a collection of elements (packets, orders, or simply items) has to be dispatched according to a certain strategy. Moreover, the distribution takes place on a structure exposing a sort of hierarchical organization: routing algorithms generally operate on graphs while in the case of scheduling orders are allocated to production area and then assigned to specific machines. The nature of items and the use of dispatching strategies, normally being aware of the structure, are what specializes the problem in each context. For these reasons, providing a general solution to the dispatching problem is limiting: it would not be possible to consider the specific issues of each scenario for effectively optimizing the dispatching process. A better idea could be providing a general framework for creating a dispatching system that is easily customizable for each specific context. In order to provide such a flexible structure the use of the agent-oriented technology could be an interesting approach. Multi-agent systems are flexible and dynamic software systems [4]. Software agents natively adopt high-level interaction patterns [5] and this is a relevant aspect for application interoperability and component coordination. By using software agent we can either provide the general structure of the dispatching or leave room for the specializations.

In this paper we will present Herald a toolkit for dispatching items in a distributed environment: the system is composed by a collection of agents, which manages the dispatching process, and a collection of additional components used to integrate the multi-agent system with the existing software. The main idea behind Herald is not to provide a ready to used product but a toolkit which is customizable to different scenarios with little effort. Herald provides a collection of agents implementing the dispatching infrastructure and describes a methodology to customize the toolkit for the different application contexts. The strength of Herald resides in exploiting the flexibility of agents for implementing the hierarchical dispatching infrastructure according to the structure required by the real application scenario. The multi-agent system, developed with the AgentService programming framework [6], is the core component of Herald along with a base class library defining the common data structures defined by these agents. Developers have to provide an application-based version of items, the elements to be dispatched, and custom dispatching strategies if required by the application scenario. Developers can also integrate external components for driving the dispatching activities since the systems allows callbacks at each stage of the process.

This paper is organized as follows: in Section II we will introduce the key elements of agent-based dispatching; in Section III we will present a selected collection of the most representative work in the field; Section IV describes in detail the architecture of Herald while in Section V we will present a practical application of the toolkit by describing the prototype developed in collaboration with Siemens A&D for a real scheduling scenario. Conclusions and final remarks follow.

II. AGENT-BASED DISPATCHING

The relatively large number of solutions based on multi-agent systems, demonstrates the usefulness of an agent based

infrastructure for the development of distributed and hierarchical applications managing logistic, routing and dispatching issues.

Since the operative management of resources is a critical aspect of the business activity of an enterprise, there exist several industrial solutions, involving multi-agent systems too. On the other hand, the research activity in this sector is lively and ready to exploit every new approach. The aim is to provide systems with an intrinsic intelligence, then denoting a particular adaptiveness to the environment changes and unexpected events. Multi-agent systems seem to satisfy these requirements. The typical socio-technical functions that we find in an enterprise can be easily modeled by means of a software agent playing a management role. For this reason an agent denotes a natural predisposition to collaborate with peers in order to achieve a common goal, through cooperation protocols. It could have skills and behaviours that can implement different solving strategies [7]. As we will see in the following, multi-agent systems represent a basis on which different and original solutions can be implemented.

A. *Industrial solutions*

In the panorama of industrial solutions regarding the management of resources, two strengthened products are on the market: Magenta and LS/ATN (Living System Adaptive Transportation Network).

Magenta [8] is a MAS framework for the development of ad hoc applications focused on the design, planning, scheduling, and management of enterprise resources. Typical examples of Magenta applications are the supply chain management, enterprise resource planning, transportation logistics, crew scheduling and knowledge management. Magenta integrates three crucial technologies: multi-agent systems, semantic web and J2EE. One of the main features of Magenta is the strong support for modeling the relevant entities of the enterprise through an ontology representation. Ontologies can be modified and updated online, during the execution of the application.

LS/ATN [9] is a comprehensive solution for optimization and dispatching of full and part truck loads including tracking and real-time event handling. It is produced by Whitestein Technologies Inc. and it is oriented to the European logistics companies. LS/ATN is based on the LS/TS (Living System Technology Suite) agent development framework. The implementation of the agent-based solution takes into account the geographically dispersed nature of transportation. For this reason agents represent the geographical regions and they exchange objects representing the cargo loads. The transportation operations are allocated to the different dispatching regions which are managed by an agent region manager; a broker agent named agent distributor deals with incoming transportation requests. The optimization process involves two steps: a first local optimization within the region (it involves the broker agent and the agent region manager) and the global optimization through the collaboration of different agent region manager.

B. *Research solutions*

Different proposals involve the agent technology, in order to implement original solutions: from the imitation of social insects to genetic algorithms. A more pragmatic approach is the identification of relevant entities involved in the dispatching process (machines, raw materials, tools, management units, etc.), establishing a direct correspondence among these entities and software agents. The significant observation is that every approach can be implemented by a multi-agent platform.

There is an immediate similarity between agents and ants: the power of an ant colony is not the single individual but the cooperation of every insect. In the solution presented in [10] the agents-ants collaborate in order to provide a heuristic scheduling solution in a parallel machine environment. The proposed solution is applied in a complex context regarding a manufacturing company.

Another solution [11] to the dispatching problem involves genetic algorithms and an agent community particularly reactive and adaptive to the environment changes. The agents do not have a predefined set of rules or instruction to reach their goal. In particular, an agent is defined by the knowledge about the environment condition (i.e. processing resources and the status of other agents), the agent status (position, process plan status, completed and remaining operations), and a tuning vector (to weight the decision rules). There are two types of agent: the part agent able to select the machine which will process the part and the workstation agent, which select the part to process on the basis of different parameters as: processing time, deadlines, and setup times. These agents adapt their actions to the plant status using a multi-criteria decision-making algorithm that encompasses multiple weighted dispatching rules, in particular using fuzzy set concepts to implement a trade-of among different decision rules. The performance of each agent is evaluated by a performance indicator. Cyclically, every agent is replaced by another one with different tuning parameters. The “natural selection” detects the best agent; this approach leads up to two considerations: in case of stationary conditions of the plant, the dynamic optimization acts as an online strategy that improves the whole MAS performance. In case of unexpected perturbations (e.g. a machine failure), the genetic adaptation allows searching of more effective agents for the new operating context.

A more conventional solution [12] follows the usual methodology to create agents representing the main entities involved in the dispatching of resources. A set of highly specialized agents is provided in order to adapt the system to the perturbations and disturbances. A system editing agent (SEA) is in charge of the composition of the whole system, as the agent instantiation and the centralized database filling. It is an interface between the human user and the multi-agent platform. The manufacturing management agent (MMA) is associated to a production area, receives orders (composed by a set of operations), instantiates a job order agent (JOA) monitoring its activity. The JOA is able to evaluate the

feasibility of an order and communicates with the production area. A logistic management agent (LMA) holds the logistic data of a production area, while the logistic agent (LA) helps the JOA and the agents associated to the machines (machine agent, MA), tools (tool agent, TA), and jigs (jig agent, JA) to evaluate the feasibility of an order or operation.

All the proposed solutions show how a hierarchical structure of cooperative agents is helpful in the resolution of dispatching problems. The variety of agents which belong to the tree hierarchy suggests that a toolkit for the implementation of a custom project is an interesting approach. All the analyzed solutions propose specific algorithms for driving agents' behaviors, i.e. swarm intelligence and genetic algorithms; on the other hand, the aim of the toolkit proposed in this paper is to provide an high level infrastructure for developing distributed systems where different kinds of algorithms can be applied.

III. The Herald Toolkit

In the Middle Age the term herald was used to identify those people which acted like factotums at the king's court. Among the other tasks, heralds were in charge of dispatching messages, managing negotiations, and accomplishing missions. As the ancient herald, the Herald toolkit is aimed to resolve logistic and decision problems.

A. Introduction

Herald is a toolkit that implements agent based software systems and aims to act as a generic distribution network. Herald proposes an approach which is independent from the specific nature either of the items dispatched or of the problem domain. Hence, it proposes a general infrastructure that needs to be further customized in order to be effectively applied to real-life scenarios. The core features of Herald are its agent based architecture and the protocol adopted to dispatch and assign items to the processing units. External software components customizing the dispatching infrastructure are integrated into the system and their activities mainly cover the collection of the items to dispatch, the selection of the dispatching policies, and the management of results. The external components are also responsible of activating the multi-agent system and controlling the dispatching process if needed. In this section we will discuss the architecture of the system and the details of the protocol proposed by Herald.

B. Architecture

The core of Herald is based on a protocol execution engine constituted by a collection of software agents which create the dispatching process by interoperating with external software modules driving the protocol. Figure 1 gives an overview of the organization of the components constituting an implementation of Herald.

The architecture proposed by Herald is based on the assumption that the dispatching process takes place within a hierarchical structure that can be easily represented by a tree: the process originates from one root node and at each node items are distributed among the child nodes. Such a structure is re-created within the multi-agent system by three different types of agents representing the different roles which nodes have in the dispatching process. Additional agents take care of the interaction with external world. The multi-agent system is implemented with the AgentService programming framework which provides advanced features for the creation of multi-agent systems and their integration with non-agent based software.

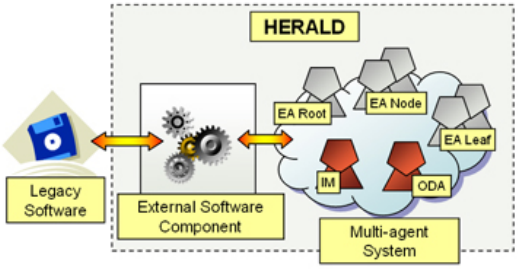


Fig. 1 – View of the System

1) Multi-agent System

The dispatching process is mostly performed by the collection of agents which constitute the multi-agent system designed in Herald. Herald proposes two different kinds of software agents: logical agents and physical agents. Physical agents represent physical entities existing in the hierarchy of the real structure, while logical agents are mostly related with the elaboration of the input and the output data of the whole process.

There are three different types of physical agents – also called entity agents – they are:

- EA Root (Entity Agent Root): commonly, there is only one agent of this type since it abstracts the root of the tree. This agent receives the whole collection of items and distributes them among its child nodes. It can also perform additional operations specific to the role of the root in the real scenario.
- EA Node (Entity Agent Node): these agents represent all the intermediate levels of the hierarchy and dispatch items among to their children nodes.
- EA Leaf (Entity Agent Leaf): they represent the leaf of the tree structure. These agents process the items and send up to the hierarchy the data collected during their activity on the items.

Tree structure having more than one intermediate level are modeled by introducing the required number of *EA Node* agents representing each node in these levels.

During the dispatching process items can be manipulated at each level: items can be aggregated, duplicated, or split. The nature of these operations strictly depends on the application context and Herald gives the opportunity to external software

components to control the dispatching process.

The multi-agent system is completed by two logical agents that are the Item Manager Agent and the Output Data Agent:

- Item Manager (IM) Agent: the IM Agent collects the items that have to be processed and distributed along the hierarchy. As happens, for the EA Root it can pre-elaborate the collection of items but its main role is to represent the access point to the dispatching process by the external software;
- Output Data Agent (ODA): the ODA is activated at the end of the dispatching process; it receives the mapping between the items and the EA Leaf agents composing the system. Additional information, specific to the application context, can decorate this mapping.

All the agents defined in the system mostly manipulate items (split, duplicate, or change properties): these are abstract entities that can be further customized with additional properties.

2) AgentService

Even though the practical implementation of Herald does not strictly require a specific agent programming framework, its canonical model adopts the AgentService framework for which we have developed all the required class libraries and software agents. AgentService [6] is an agent programming framework built on top of the Common Language Infrastructure [13], whose .NET Framework is the most popular implementation. The framework provides the programmers with the following features:

- definition of autonomous, independent, and persistent agents;
- concurrent execution of agents and their multi-behavior activity;
- persistent shared data structures within a single agent;
- transactional agent communication based on message exchange;
- access to the FIPA service components (AMS, DF, MTS).

One of the key features of AgentService is its agent model that is not particularly tied to specific agent architectures, but is flexible enough to implement different ones. Within AgentService, an agent is constituted by a set of knowledge objects and a set of behaviour objects. Knowledge objects define the agents' knowledge base while behaviour objects define the activities that an agent can perform and the services it offers to the others. Knowledge objects are shared among the different behaviours that are scheduled in a concurrent manner. The definition of a new type of agent leads to the definition of a template which specifies the behaviour and knowledge objects that characterize it. It also leads to the definition of these behaviour and knowledge types.

AgentService supports mobile agents [15] and provides programming tools for managing ontologies and interaction protocols [16]. AgentService also allows an easy integration of multi-agent systems with external applications by using agent avatars: agent avatars are application stubs that are seen as software agents inside multi-agent system. By using avatars external software modules can interact with agents and access platform services as if they were like real agents.

3) External Software Components

External software components connect the multi-agent system with the external world and the problem domain. These components have the following responsibilities:

- they provide the collection of items to the MAS;
- they activate the system;
- they control the execution of the protocol;
- they elaborate results.

These components manage all the customization aspects of the system: they create the items, they add the required additional properties, and they provide the custom dispatching policies when needed. Within a default installation of Herald external software components constitute the software environment into which the multi-agent system is created and executed. By using an agent avatar they interoperate with the MAS, drive the dispatching protocol, and get the results.

C. Dispatching Protocol

Herald uses a flexible dispatching structure in which the decisions concerning the distribution of items can be taken in cooperation with external software entities providing either the complete set of allocations or just simple indications.

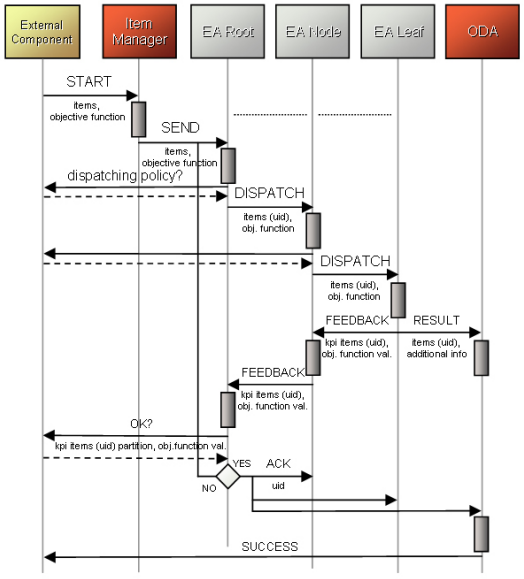


Fig. 2 – Protocol execution step

This gives a high degree of flexibility since it allows

developers to modify the course of actions with the highest level of detail and it makes Herald suitable for many different scenarios. The only requirement of Herald is the hierarchical structure of the distribution process which strongly characterizes the architecture of Herald.

As previously said in section III.B.3 the process is activated by an external software component which has to create the collection of agents described in section III.B.1. Once the multi-agent system is activated by the external software components the following steps apply:

1. the external software component creates/retrieves the collection of items that have to be dispatched and send them to Item Manager agent along with additional ordering criteria and an objective function that has to be minimized or maximized;
2. the Item Manager pre-elaborates the items, sorts them according to the suggested criteria and communicates to the EA Root the collections of sorted items;
3. the EA Root creates a unique dispatching identifier that will be tagged to the collection of items while they flow through the child nodes. If necessary, the EA Root performs additional operations on the items (i.e. aggregation of items) before sending them to child nodes. The EA Root asks to the external software module for a dispatching strategy of the collections of items: if the external software component gives suggestions (i.e. by providing a mapping from items to child nodes or by explicitly giving a partition algorithm to apply) these suggestions are applied otherwise the default partitioning is applied. Then the items are sent to the child nodes according to the partition previously performed;
4. each EA Node which does not receive an empty list of items executes the dispatching. As happened for the EA Root the EA Node can, if required, manipulate items by aggregating or splitting them according to the requirements of the possible final targets (the subset of EA Leaf agents that can be reached from this node) of the items;
5. the EA Node asks to the external software component indications about the partition of items by sending the request along with some context information (i.e.: the identifier of the node and the collections of items). If there are any suggestions they are applied otherwise the default distribution takes place. The items are then sent to child nodes;
6. steps 4 and 5 are repeated for each level of the tree until leaf nodes are reached;
7. each EA Leaf that receives a collection of items elaborates them and eventually computes some key performance indexes as suggested by the external software component and its contribute for the objective function;
8. each EA Leaf sends a feedback – composed by the key performance indexes and the value of the objective function – to the parent EA Node and the list of assigned items along with additional information to the ODA;
9. each EA Node aggregates the feedbacks received from the child nodes and compute the value of the objective function for the node, then sends back to the parent node the data;
10. the EA Root aggregates the feedback received by the child nodes, computes the final value of the objective function and, according to these data, decides if the current partition of the items is acceptable, by eventually asking to the external component. In case of successful partition the EA Root broadcasts an acknowledge message to all the entity agents and the ODA by specifying the dispatching identifier. If the partition is not acceptable the entire process starts again from step 3 by using different criteria;
11. the entity agents – and the ODA – receiving an acknowledge message keep track of the partition related to the acknowledged dispatching identifier and delete all the other partitions;
12. the external component is notified by the ODA the successful termination of the dispatching process.

Figure 2 gives a graphical representation of the steps described above. We can observe that the protocol simplifies the introduction of on-line decisions. On-line decisions are taken while the system is running and they can modify its course of action. This is accomplished by letting the entity agents interact with the external software components which should maintain an updated view of the state of the problem domain. Finally, keeping separate the general infrastructure of the protocol and all those aspects customizing the algorithm for a specific problem we are able to obtain a very flexible structure. Such structure can be easily applied to different scenarios and problem domains. In each customization what really changes is the external software component driving the protocol and what this module provides to the multi-agent system which remains almost the same.

Another important aspect of this system is the ability of executing multiple dispatching processes in parallel without increasing the number of agents. Each Entity Agent instantiates a behaviour for executing a new incoming dispatching strategy: item partitions belonging to different strategies are identified by different unique dispatching identifiers. Thanks to the natural and effective multi-threading management system of the AgentService framework the execution of multiple dispatching processes comes with no additional cost. The ability of executing dispatching processes in parallel allows implementations of Herald to try more different solutions for the same problem at the same time and then select the best one.

D. Application Scenarios and Customization

There are many different domains in which there is the need of using a dispatching policy for some kind of items. If we

have a network the routing of packets is a common application of dispatching policies. Other examples involve logistics and scheduling: in the case of logistics there is the need to select the best route for a given item while in the case of scheduling we have to process a collection of orders in a given time constraint. The execution of orders eventually results in a set of tasks that have to be assigned to a set of machines according to a given algorithm. These are only the most evident domains in which the problem of dispatching items has to be managed. Not all the possible instances of these domains are eligible as case studies for Herald. In particular those exposing a structure that is inherently a graph and that cannot be reduced as a tree cannot be considered. Fortunately there are many cases in which the original hierarchy is a tree or can be reduced to a tree: these are the instances eligible for Herald and now we will see what is required to customize Herald for a practical application.

The first thing that needs to be customized is the item: in order to fully represent the entities of the problem domain, items need to be enriched with additional properties. The Herald Toolkit provides a library defining all the data structures required to apply the dispatching process: by sub-classing the Item type developers can enrich the item class with all the required properties. In order to fully exploit the personalization applied to items all the data structures which operate on item have to be specialized: in particular, algorithms for distributing items and those evaluating the key performance indexes have to take into account the value of additional properties. Herald defines interfaces and delegates for these objects and developers just have to adhere to these type contracts while implementing the specialization. The last component that needs to be implemented is the external software module which drives the protocol and connects the multi-agent system with the problem domain. The implementation of this component is a common activity when designing multi-agent systems with AgentService: developers are normally required to create a batch which sets up the multi-agent system and interact with it if necessary. The implementation of a protocol driver for Herald does not take any additional burden.

In the next section we will see a practical example of the customization described here by describing a case study where Herald as been effectively applied.

IV. Case Study

A. The Problem Context

In order to test its feasibility we implemented a prototype of Herald in the field of production scheduling as a result of collaboration with Siemens Automation & Drive (A&D). Siemens A&D is a leading firm in the field of MES (Manufacturing Execution System) production and scheduling. In order to satisfy the customers' needs Siemens A&D offers the SimaticIT Production Suite which is a suite of cooperating applications which takes care of production process by starting from the Enterprise Resource Planning (ERP) and by reaching the machine level. The Production Suite controls many different tasks: low level plant monitoring, resources allocation, logistics, and scheduling. In this context we developed a prototype based on Herald for scheduling a collection of production orders into a production plant. The Agent Based Detailed Production Scheduler, which is the name of the prototype, relies on the AgentService programming framework for the design and the implementation of software agents constituting the architecture defined by Herald.

B. The Multi-agent System

The structure of plant, which is defined by the S-95 standard [14], naturally resembles a hierarchical tree: the standard defines a production site as a collection of areas which are subsequently partitioned into production cells constituted by units. Production units host production machines which execute real production operations. Hence, mapping such a hierarchy into a tree structure – having the root node in the site and developing till the production units – has been a natural and seamless task. In order to complete the case study in the scope of this paper we will consider a simplified example of this hierarchy that is the sub-tree representing a production area. The hierarchy of the system is mapped onto collection of three different entity agents: EA Area, EA Cell, EA Unit. Moreover, a collection of specific logical agents, namely the WOM (Work Order Manager) and the OSA (Output Schedule Agent), are provided to retrieve data and present results of the scheduling process.

The EA Area agent is a specialization of the EA Root agent and basically dispatches the production orders to the EA Cell agents, waits for the KPI indexes, and eventually sends the acknowledge to terminate the process. The dispatching strategy is selected by the external component. EA Cell agents specialize the EA Node agent. EA Cell agents dispatch the entry received by the EA Area to the EA Units according to some dispatching strategy decided by the external component. They wait for the KPI from the units and assemble them before sending them to EA Area. EA Unit agents are EA Leaf agents: they process the entries, execute the scheduling algorithm, register the KPI, and send them back to EA Cell agent.

The Work Order Manager agent is the specialization of the IM agent. It receives the collection of orders to be scheduled and sets up the multi-agent system constituting the production area to which these orders have been dispatched. Finally, the Output Schedule Agent specializes the ODA and has been introduced to the system to organize and present the schedule data collected from the units in a more convenient format: the OSA uses these data and organizes them into a Gantt diagram.

C. The Protocol

Due to the requirements provided by Siemens A&D the entire dispatching process must be controlled by the Production Modeler (PM) which is the component of the Production Suite actively controlling the physical plant. For

this reason the dispatching protocol is said to be PM driven: the Production Modeler is the component which starts the scheduling process and it is also the one which is constantly queried by physical agents during the process. Finally, the results collected by the OSA agent are returned back to the PM which evaluates the performance indexes computed by the multi-agent system and chooses the best schedule. For all these reasons, the PM represents the external component which is involved in the dispatching process as described in Herald. Thanks to the flexible structure of Herald and advanced features of AgentService the customization has been simple and quick.

We decided to centralize the interaction with the PM by using a specific software module implemented as an agent avatar and that we called PM Gateway: the PM Gateway represents the PM in the multi-agent systems and the other agents interact with the PM by exchanging messages with this agent. This design decision represents a slight variation of the architecture proposed by Herald but it is mostly an implementation issue which helped us while testing the system off-line. The use of the PM Gateway allowed us to deploy the multi-agent system by only changing the implementation of the agent avatar and without modifying the code of the other software agents.

V. Conclusions

In this paper we presented an agent-based toolkit for implementing flexible dispatching infrastructures. Flexibility is given by the adoption of agent technology as a core component of Herald and by the high degree of customization offered to end users who can drive the process at each stage. Software agents are adopted to replicate the hierarchical structure of the real life system where the dispatching process takes place. Users just have to develop the software interconnection layer between the multi-agent system and the software legacy system requiring the support of Herald. The features provided by the AgentService programming framework make the interaction with multi-agent system an easy task and allow developers to quickly customize the model provided by Herald to their own problem context.

As noticed in the introduction, even though the model proposed by Herald is based on a tree hierarchy for dispatching the items it remains general enough to be applicable to a wide range of scenarios. As a future improvements we consider the adoption of a more flexible holonic architecture. We developed a case study in the field of production scheduling in collaboration with Siemens A&D and the results have been interesting. The case study showed also that the use of parallel dispatching strategies is valuable since this features makes easier comparing the application of different strategies side by side during their execution.

The Herald toolkit is actually developed in its core components and it has been tested on a real case study, we are now working on creating libraries of dispatching strategies and on providing a graphical user interface for using Herald as a standalone tool. In particular, in order to definitely improve Herald usability, a “drag and drop” interface for graphically model the hierarchical structure of the system and a tool for monitoring the dispatching process at runtime have to be implemented.

References

[1] M. Walliser, M. Calisti, T. Hempfling, S. Brantschen, F. Klügl, A. Bazzan, and S. Ossowski, “Agent-Based Approaches to Transport Logistics, Applications of Agent Technology in Traffic and Transportation, Birkhäuser Basel, pp. 1-15, March 30, 2006.

[2] I. Kassabalidis, A.K. Das, M.A. El-Sharkawi, R.J. Marks II, P. Arabshahi, and A. Gray, “Intelligent routing and bandwidth allocation in wireless networks”, Proc. NASA Earth Science Technology Conf. College Park, MD, August 28-30, 2001.

[3] D. Ouelhadj, C. Hanach, and B. Bouzouia, “Multi-agent system for dynamic scheduling and control in manufacturing cells”, Robotics and Automation, 1998, Proceedings, 1998 IEEE International Conference on V. 3, pp. 2128–2133, 1998.

[4] H. Nwana, “Software agents: An Overview”, Knowledge and Engineering Review, November, Vol. 11, No 3, 1996.

[5] K. Sycara, and D. Zeng, “Coordination of Multiple Intelligent Software Agents”, International Journal of Cooperative Information Systems Vol. 5, 1996.

[6] C. Vecchiola, A. Grosso, A.Gozzi, and A. Boccalatte, “AgentService”, Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE04), Banff, Alberta Canada, KSI Publisher, 2004.

[7] M. Woolridge, “Intelligent Agents”, Multiagent Systems: A modern Approach to Distributed Artificial Intelligence, Weiss, MIT Press, 1999.

[8] Magenta Technology, “Software Platform v2.1”, Magenta Technology Whitepaper, [Online document] 2005, Available at HTTP: www.magenta-technology.com/

[9] Whitestein Technologies, “LS/TS – Living Systems ® Technology Suite”, [Online document], Available at HTTP: http://www.whitestein.com/resources/products/whitestein_lsts_flyer.pdf

[10] C.A. Silva, J.M. Sousa, T.A. Runkler, and J.M. Sà da Costa, “A Multi-Agent Dispatching Heuristic for Manufacturing Systems Using Ant Colonies”, Proceedings of the European Network of Excellence on Intelligent Technologies for Smart Adaptive Systems (EUNITE 2002), 2002.

[11] B. Maione, and D. Naso, “Evolutionary adaptation of dispatching agents in heterarchical manufacturing systems”, International Journal of Production Research, Vol. 39, N. 7, pp. 1481-1503(23), 2001.

[12] S. Heinrich, H. Durr, T. Hanel, and J. Lassig, “An Agent-based Manufacturing Management System for Production and Logistics within Cross-Company Regional and National Production Networks”, International Journal of Advanced Robotic Systems, Vol. 2, N. 1, pp. 7-14, 2005.

[13] Standard ISO/IEC 23271:2003: Common Language Infrastructure, ISO, 2003.

[14] ISA, S95—Enterprise-Control System Integration, Part 1: Models and Terminology, Instrumentation, Systems and Automation Soc., 2000. ISA, S95—Enterprise-Control System Integration, Part 2: Object Model Attributes, Instrumentation, Systems and Automation Soc., 2001 [Online]. Available: http://www.isa.org.

[15] A. Boccalatte, A. Grosso, C. Vecchiola, “Implementing a Mobile Agent Infrastructure on the .NET Framework”, 4th International Conference in Central Europe on .NET Technologies, Plzen, 2006.

[16] A. Passadore, C. Vecchiola, A. Grosso, and A. Boccalatte, “Designing agent interactions with Pericles”, ONTOSE 2007, Second International Workshop on Ontology, Conceptualization and Epistemology for Software and Systems Engineering, Milan, Italy, Giugno 2007.

Agents and Security in a cultural assets transport scenario

Stefania Costantini, Arianna Tocchio, Panagiota Tsintza
Dipartimento di Informatica
Università degli Studi di L'Aquila
Via Vetoio, Loc. Coppito,L'Aquila - Italy
Email: {stefcost,tocchio,panagiota.tsintza}@di.univaq.it

Leonardo Mostarda
Department of Computing
Imperial College London,
London, UK SW7 2AZ
Email: lmostard@doc.ic.ac.uk

***Abstract*— Museums and exhibitions represent a relevant contribution to the economy all over the world. In Italy, in the year 2006 the 400 national museums, monuments and archaeological sites have been visited by 34.492.875 people with an average entrance fee of 6,64 euro for person while in France 18.367.000 people decided to dedicate some time for visiting the national museums. Considered the increasing relevance of the cultural and economical level of museums, several works in Artificial Intelligence (AI) have proposed new methodologies for supporting the users during their visits. However, few research groups have faced the problem of the cultural assets transport. This paper pays attention to a particular aspect of the museum activities: how to identify and to transport in a secure way the cultural assets. In fact, a higher security in transport among museums may increase the exchanges and, consequently, the cultural offer. For reaching this goal we exploited the Galileo Satellite services and the Intelligent Agents technology and we experimented the system in a real scenario.**

I. Introduction

People often believes that transport of a cultural asset is a simple process. Instead, the act of moving a work of art from a place to another one requires attention, experience and competence. In fact, the transport phase hides risks, delays, anxieties and difficulties due to the unpredictability of the events. The quest for perfection in this field is unavoidable, as slight differences may determine a success or a failure. Every day, several companies in the world try to reach the goal of moving cultural assets while reducing as much as possible the risks. This process often involves an accurate packaging phase, an escort service from the origin to the destination and an insurance policy, as criminal actions are always possible.

A relevant role in the prevention of the criminal activities could be assigned to the Galileo satellite, a "big brother" capable not only of precisely identifying the position of a cultural asset but also of following it during the journey. The context in which technology can increase security involves both the packaging and the transport monitoring phase. In the packaging phase, particular devices and sensors are put in the cultural assets packs. In the transport phase, these devices receive the Galileo satellite signal and allow one to track the cultural assets movements.

In previous work, we have combined the Galileo infrastructure with the common security mechanisms to build the Geo Time Authentication system (GTA) [9]. The GTA provides the following services: (i) cultural asset identification and authentication; (ii) integrity of cultural assets information; (iii) secure transport of cultural assets. Secure transport is achieved by means of the GTA monitoring component. This component is wrapped in each cultural asset package and is connected to several sensors (i.e., temperature, humidity and light sensors). At run time, the GTA monitoring component controls the sensor data variation to detect package opening. It also checks the mutual position among packages to detect possible thefts and uses the Galileo signal to check the correct transport routing.

However, in real experiments [10] that we have performed we have noticed that the GTA monitoring component can rise false alarms. These are consequence of unexpected environmental conditions (e.g., quickly weather breaks, sudden track breaking) that require some intelligent deductions missing in the GTA implementation. In this paper, we present recent developments aimed at enhancing the GTA system by means of the deduction capabilities of intelligent agents. There are good reasons for adopting agents in order to improve GTA capabilities. Agents offer autonomy, reactivity, pro-activity, social ability, very useful for all applications where some degree of autonomy is needed. There are application contexts that actually offer no alternative to autonomous software. Agents provide a tool for structuring an application so as to support the design metaphor in a direct way. In this sense, they offer an appropriate support to the development of complex systems.

Agents and multi-agent systems (MAS) have emerged as a powerful technology for facing the complexity of a variety of ICT scenarios. There are now several industrial applications that demonstrate the advantage of using agents. However, agent systems have yet to achieve widespread deployment in operating environments, as technology has to move from pure research to development. In our context, intelligent agents have been used to discriminate between motivated and unmotivated warnings signaled by the GTA in the transport scenario. The improvement due to introducing agents is based on the intelligent and cooperative analysis of particular events like, for example, the change of the external temperature or a grinding halt.

Consider a possible scenario for the first case. The truck is

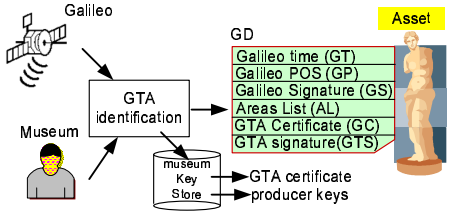


Fig. 1. The identification of a cultural asset

going from Rome to Naples. It contains several packs, each one controlled by the GTA, an Intelligent Agent and some sensors. At a certain moment, a pile-up blocks the traffic along the highway. We are in July, the external temperature is high and the sensor in a pack measures a temperature degree beyond the prefixed threshold. The GTA, after giving the alert, waits for the MAS evaluation. The Agents in the MAS compare their information about the temperature and, through a reasoning process (by evaluating the information about the season, the position of the truck, the kind of freezing plant and so on) try to deduce if the alert is motivated. In fact, if the temperature is high in all packs as well as in an external device, there are good motivations for supposing that no person opened one or more pack but, rather, that either the external temperature influenced the internal one of the packs or the freezing plant in the truck is malfunctioning. Consequently, a message is sent to the route supervisor about the false alarm and the temperature threshold is incremented. An unmotivated alert has been avoided. We are aware that in real applications in a first stage the agents behavior must be monitored for avoiding that actual alerts could be underrated: however, after an initial verification phase, the agent can be trained to efficiently support traditional techniques in security scenarios.

This paper is organized as follows. Section II shortly describes the main characteristics of the GTA component. Section III presents the MAS structure, paying a particular attention to the environment role, while Section IV explores the Agents activities in the monitoring scenario. Section V relates about some aspects of the MAS implementation while Section VI introduces the main motivations for this choice. Finally, Sections VII and VIII conclude the paper.

II. The Geo Time Authentication system at a glance

The Geo Time Authentication (GTA) [9] is a prototype system that provides authenticity and integrity of cultural assets information. It has been conceived in the context of the CUSPIS project [10] and, afterwards, it has been generalized to the context of assets and goods where relevant problems of counterfeiting and thefts exist. To prevent these crimes, the GTA system provides the following services: (i) identification; (ii) authentication; (iii) access control; (iv) integrity; (v) privacy and confidentiality; (vi) non repudiation; (vii) secure transport of assets. As we are going to see in the rest of this section, each service is based on traditional cryptography mechanisms enhanced with the satellite information (i.e., latitude, longitude and time).

The GTA identification service (see Figure 1) permits to uniquely identify a cultural asset. To this aim, each museum is equipped with a *GTA identification component*. This component takes as input the Galileo signal and a key store that contains the museum private key. The component output is a so-called "GD" (GTA Data) for each asset.

A GD contains the following information: (i) the Galileo time (GT); (ii) the Galileo geographical coordinates (GP); (iii) the Galileo Signature (GS); (iv) the Areas List (AL); (v) the GTA certificate (GC); (vi) the GTA signature (GTS). The Galileo geographical coordinates (GP) are the longitude and the latitude of the geographical area where the cultural asset identification takes place.

The Galileo time (GT) refers to the time when the identification is performed. The concatenation of both GP and GT is referred to as Galileo Identifier (GAID) and is used to uniquely identify the cultural assets. In fact, each museum is assigned its own identification area and thus, at any time, exactly one GD can be produced. The Areas List (AL) refers to the list of geographical areas where the cultural asset will be exposed. The GTA certificate (GC) contains the identity of the museum that has generated the GD. Finally, the GTA signature (GTS) is the signature of all fields for ensuring GD integrity.

The GTA authentication service guarantees the authenticity of a GD. This ensures that the cultural assets authenticated in the museum are the ones indicated on the GD and that the GD was indeed generated in the origin indicated on it. In particular, GD authentication prevents an attacker from masquerading as a legitimate museum. In this way, introducing counterfeit objects in the market becomes more difficult. For instance, in the CUSPIS case study that we have worked out, the GTA has been used to guarantee that an ancient sculpture was indeed catalogued by the Greek museum in Athens.

The GTA access control service is the GTA ability to limit and control the access to the GTA Data (GD) information. To this aim, each entity must be authenticated, so that access rights can be tailored to the individual case. The GTA integrity service ensures that a GD is received as sent, i.e, no duplication, reuse, destruction can be performed. The GTA privacy and confidentially services guarantee that GD information is provided only to authorized people. The GTA non repudiation service prevents a producer to deny a generated GD. The GTA secure transport of assets ensures that assets are not stolen or substituted during the transport phase. In particular, this service is performed by monitoring that (i) the transport is routed along the correct path; (ii) the variation of temperature, humidity and light inside the cultural assets package does not exceed a threshold (i.e., packages are not opened); (iii) mutual position among packages do not change over the time (i.e., packages are not stolen).

However, the GTA transport security measures can be too strict and sometimes can rise false alarms that need to be handled by the security system manager. For instance, if the

environmental conditions quickly change (e.g., the weather quickly breaks, the track is inside a tunnel) then the thresholds can be subject to a considerable variation. A track braking can cause all packages to change their mutual positions and an alarm is raised. To address such unexpected situations, in this paper we enhance the GTA monitoring system through the deduction capabilities of intelligent agents. We have experimented the cooperation between the GTA and the intelligent agents in order to build a more flexible transport monitoring system that is able to recognize false alarms in the transport of cultural assets.

III. Environment and MAS structure

After explaining the main characteristics and the role of the GTA, in this section we formalize the infrastructure of the MAS environment by means of the approach of Viroli et al. ([25]). We examine the role of agents and environment in the monitoring scenario. The above-mentioned paper proposes to analyze the roles and the features of a MAS environment by decomposing it into basic bricks called *environment abstractions*. Each agent perceives the existence of such abstractions and interacts with them in order to achieve individual or social goals. Figure 2 provides an overview of the infrastructure of the MAS application.

At the bottom level, the physical support specifies the hardware components of the system whose data the MAS is interested in. In particular, we mention the ABUs (Asset Board Unit) which are satellite signal receivers contained in the packs, the Galileo infrastructure, the sensors used to capture the changes in the environmental transport conditions (variation of the temperature or of the light intensity, ...) and the generic communication infrastructure. Each ABU receives the Galileo signal and transmits the pack's position to the MAS.

The execution platform includes the operating systems, the virtual machine and other middlewares. The interpreter of the DALICA MAS agents has been written in Sicstus Prolog [22] while other functionalities have been implemented in Java. Jasper [22] allowed us to interface the prolog language with the Java one. The execution platform also includes the Agent/Environment communication middlewares: the Linda Tuple Space [22] and the Event-based communication support. Each information coming from external sources (ABUs, sensors,...) is transformed into an event and put in the Tuple Space to be received by the agents. When the agents needs to communicate with the external world, its messages are transformed into events through the Jasper interface and delivered to the corresponding devices.

Components external to the MAS communicate by means of an event-based mechanism. Figure 3 synthesizes the communication infrastructure. Messages coming from the MAS and addressed to the external components are received by the Output Agent and, via the Jasper Interface and the Server RMI, are sent to the Messaging Component. The Messaging Component implements a plug-in architecture for supporting run-time registration of both server and client components. After the registration process, each client interacts with the messaging component and receives an ID that will be used to send and receive events. In other words, after the registration process, each client can build and send to the messaging component an event containing its ID and the server component ID. It is worth noticing that a basic event does not provide meaningful information since it only contains routing information (i.e., the receiver and the sender IDs). Therefore, a basic event can be (if needed) specialized into a new one that contains an additional field. This field can be used for instance by a hardware component in order to send detected data to a server component. The MAS component uses it for receiving environmental data and for replaying to external components such as as VCC, GTA and so on.

At the top layer, we find the MAS application. The Application Agents area contains the three kinds of agents composing the MAS:

Control Device Agent: The role of this agent is to monitor the condition of the assigned pack, by checking both the position through the Galileo satellite signal and the sensors conditions.

Transport Device Agent: This agent has the role of coordinating several Control Device agents. It is capable of integrating the warnings coming form the Control Device Agents. It attempts to deduce what is happening and to evaluate the alert degree. The role of this agent will be explained at more length in the next sections.

Output Agent: Manages communications between the MAS and the external infrastructures such as VCC (Virtual Control Center), Sensor interface, GTA and so on.

Finally, the Application Environment contains the Sensors Interface that allows Control Device agents to get information about the temperature, humidity or light in the packs; the GTA component and the VCC components. The former one receives the results of the MAS deduction process while the latter one manages the communication between the MAS and the security control center where human operators monitor the transport conditions.

IV. Monitoring the Cultural Assets transport

In the transport scenario, Control Device Agents are used for checking several transport parameters and have the responsibility of informing the authorities in case of tampering or theft. In the following, we describe all phases involved in the transport of cultural assets (see Figure 4) by emphasizing the agents roles.

In the transport planning phase, the owner of the cultural assets, the renter (i.e., the entity who wishes to take the cultural assets) and third-part entities (i.e., those who vouches for the content and the routing of transport) cooperate to produce different certificates. In this paper, we focus on the authorization and the transport certificates since they are used by the MAS.

Each package of the transport is equipped with an authorization certificate that contains the list of all cultural assets inserted into the package. This certificate is used to check the presence of the cultural assets inside the package.

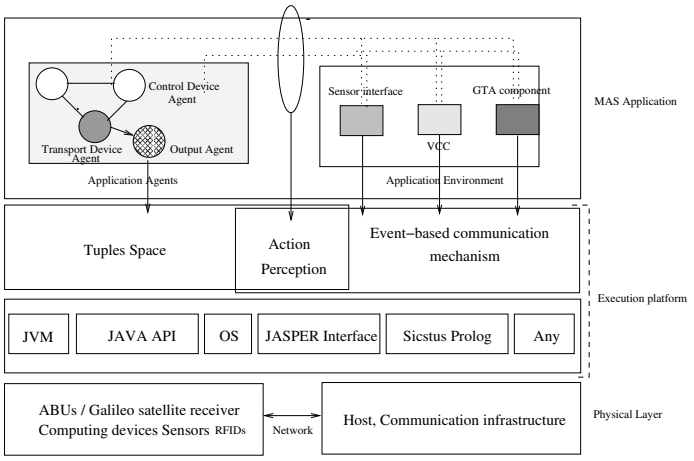


Fig. 2. MAS Environment Application Layers

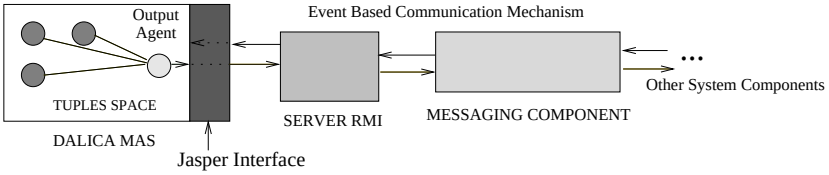


Fig. 3. Communication Infrastructure

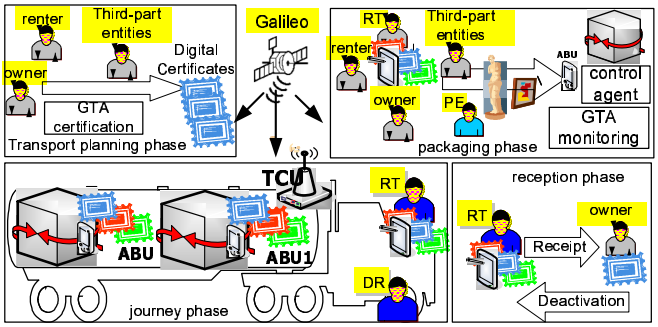


Fig. 4. The CUSPIS system

Each transport has exactly one transport certificate that contains the correct routing. The routing is defined in terms of a list of couples $\{(A_s, T_{A_s}), (A_1, T_{A_1}) \ldots (A_i, T_{A_i}) \ldots (A_n, T_{A_n}) (A_d, T_{A_d}) \}$ where A_s is the starting transport area and T_{A_s} the related date (i.e., day and hour), A_i an area that the transport has to pass and T_{A_i} the related date; (A_d, T_{A_d}) is the destination area and its date. It is worth noticing that for specific assets transport other certificates (e.g., insurance certificates) can be added and are signed by a certain number of entities.

In the packaging phase, the above entities in cooperation with the responsible of transport (RT) and the packaging expert (PE) supervise the packaging of assets. Each package is filled with: (i) a set of assets, each of them identified by an RFID tag, which is an object that can be attached to or incorporated into a product, animal, or person for the purpose of identification using radiowaves; (ii) a Control Device (ABU); (iii) a sensor of humidity; (iv) a sensor of light; (v) a sensor of temperature. The ABU unit contains the authorization and the transport certificate. Moreover, it hosts both the GTA monitoring component and the control agent. While the GTA monitoring component provides security by means

of traditional mechanisms (i.e., cryptography, certificates and detection algorithm) the control agent enhances security issues through intelligent deductions.

During the journey, each Control Device Agent, which is capable of acting in a proactive way, performs the following activities: (i) correct routing checking; (ii) cultural assets verification; (iii) sensor data checking; (iv) package position verification.

The Correct routing checking is performed by the Control Device Agent by using both the Galileo signal and the transport certificate. In particular, the agent uses the Galileo satellite to check that each area is passed at the right time. The cultural assets verification is an activity in which the control agent loads all cultural assets IDs contained in the authorization certificate and checks their presence inside the package. It is worth noticing that this activity is performed repeatedly over time. Both correct routing checking and cultural assets verification are simultaneously performed also by the GTA monitoring component. Therefore, the agents contribution is a redundant check that enhances the system fault tolerance.

The sensor data checking verifies the variation of the sensor data over time. This variation must not exceed a given threshold that, as we are going to see in the following, is dynamically adapted by means of the agents cooperation. This check ensures that a package is neither opened nor kept in a dangerous environment. In this case, the agent contribution is indeed needed since temperature checking must involve some intelligent reasoning. For instance, temperature may change for environmental reasons so that the GTA may raise a false alarm. If the temperature of the environment changes for all packs because of a natural process and overcomes the threshold, the agents in the ABUs can activate a communication process and reach the conclusion that no package has been tampered because each of them signals the same temperature.

The package position verification ensures that all packages are in the correct position. In this process the agents exploit their cooperation capabilities. In fact, from time to time each Control Device Agent sends a message to the other ones asking their position. Then, it computes the distance and verifies that the positions do not vary over time. In fact, a variation of the package mutual position can imply a package theft. In this case, the agent reasoning is indeed needed to enhance the whole system effectiveness. In fact, due e. g. to a quick break, the packages mutual position can change. In this case, the agent contribution can detect and avoid GTA false alarms.

When a Control Device Agent detects some anomaly, it sends a warning message to the Transport Device Agent whose role is to cross the information with those coming by the other entities and to verify the seriousness of the warning. Elaborated the degree alert, the Transport Device Agent sends a message to the VCC terminal. Moreover, each Control Device Agent maintains a direct communication channel with the GTA component. In fact, if the GTA identifies a suspicious situation, it contacts the agent in order to trigger a checking process of the package status.

V. The approach in more detail

Control Device, Transport Device and Output Agents have been implemented in the DALI language. DALI has a Sicstus Prolog interpreter, while the other components, like the GTA, have been implemented in Java. The interface between the two languages has been provided by the Jasper Sicstus Prolog library. In the rest of this section, we illustrate the main features of the DALI language and explain how DALI reactivity and pro-activity has been used for implementing the agents behavior.

A. The DALI language in a nutshell

DALI [5] [6] [24] [7] [8] is an Active Logic Programming language designed in the line of [16] for executable specification of logical agents. DALI is a prolog-like logic programming language with a prolog-like declarative and procedural semantics [17]. In order to introduce reactive and proactive capabilities, the basic logic language has been syntactically, semantically and procedurally enhanced by introducing several kinds of *events*, managed by suitable *reactive rules*. All the events and actions are time-stamped, so as to record when they occurred. These features are summarized very shortly below.

An *external event* is a particular stimulus perceived by the agent from the environment. We define the set of external events perceived by the agent from time t_1 to time t_n as a set $E = \{e_1 : t_1, ..., e_n : t_n\}$ where $E \subseteq S$ and S is the set of environment states. The e_i's are atoms indicated with postfix E in order to be distinguished from both plain atoms and other DALI events. External events allow an agent to react through a particular kind of rules, reactive rules, aimed at interacting with the external environment. When an event comes into the agent from its external world, the agent can perceive it and decide to react. The reaction is defined by a reactive rule which has in its head that external event. The special token $:>$, used instead of $:-$, indicates that reactive rules performs forward reasoning.

A reactive rule has the form:
$ExtEvent_E :> Body$ or
$ExtEvent_{1E}, ..., ExtEvent_{nE} :> Body$
where $Body$ has the usual (logic programming) syntax and intended meaning except that it may contain the DALI event and action atoms.

The *internal event* concept allows a DALI agent to be proactive independently of the environment by reacting to its own conclusion (notice that this feature can be considered as a form of introspection). More precisely: An internal event is syntactically indicated by postfix I and implies the definition of two rules. The first one contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction (in the second rule) may happen:
$IntEvent :- Conditions$
$IntEvent_I :> Body$

Internal events are automatically attempted with a default frequency customizable by means of user directives in the initialization file that can tune also other parameters such as how many times an agent must react to the internal event

(forever, once, twice,...) and when (forever, when triggering conditions occur, ...); how long the event must be attempted (until some time, until some terminating conditions, forever).

Actions are the agent's way of affecting the environment, possibly in reaction to either an external or internal event. An action in DALI can also be a message sent by an agent to another one. An action atom is syntactically indicated by postfix A. Clearly, when an atom corresponding to an action occurs in the inference process, the action is supposed to be actually performed by suitable "actuators" that connect the agent with its environment. In DALI, actions may or may not have preconditions: in the former case, actions are defined by actions rules, in the latter case they are just action atoms. An action rule is a plain rule, but in order to emphasize that it is related to an action, we have introduced the new token $:<$. External and internal events that have happened (i.e., that have been reacted to) and actions that have been performed are recorded as past events, that represent the agent's memory, and the basis of its "experience".

B. Reactivity and proactivity in DALICA MAS

In this section, we present a snapshot of the Control Device Agent, paying a particular attention to some reactive and proactive capabilities of the agent implemented in DALI. The signal of the Galileo satellite is received by the agent by means of a DALI reactive rule:

$posE(Lat, Lng, Time, Date, Integrity, \_) :>$
$def\_position(Lat, Lng, Time, Date, Integrity).$
$def\_position(\_, \_, \_, \_, Integrity) : -$
$Integrity = 0, no\_correct\_signalA.$
$def\_position(Lat, Lng, Time, Date, Integrity) : -$
$Integrity = 1, positionA(Lat, Lng, Time, Date, 1).$
$def\_position(Lat, Lng, Time, Date, Integrity) : -$
$Integrity = 2, positionA(Lat, Lng, Time, Date, 2).$

where Lat and Lng are, respectively, the latitude and the longitude of the pack position, while Time and Date have the obvious meaning. This reactive rule "filters" the Galileo signal according to its integrity value. Only if the integrity is different from 0, the signal is accepted and, by means of the action $positionA(Lat, Lng, Time, Date, \_)$, enables related pro-active rules for the needed inferential activities. In fact, the action $positionA(Lat, Lng, Time, Date, \_)$ is transformed into the past event $positionP(Lat, Lng, Time, Date, \_)$. This transformation, managed by the DALI interpreter, records the past event into the agent's memory. Then, the past event can trigger proactive inference as a condition of an internal event.

Proactive and reactive capabilities are adopted by the Control Device Agent for monitoring the cultural assets transport. We propose a snapshot of the DALI code that controls and manages the temperature threshold.

The internal event $monitor\_temperature$ checks wether the temperature of the cultural assets in the pack is greater than the prefixed threshold $threshold\_temp(Y)$. If so, then the Control Device Agent sends a message to the other Control Devices Agents (cda's) in order to know if the increase of the temperature is general or has happened only in its pack.

$monitor\_temperature(X) : -sensor\_data(X),$
$clause(threshold\_temp(Y), \_), X > Y.$
$monitor\_temperatureI(X) :> clause(agent(A), \_),$
$messageA(cda, send\_message(give\_temperature(X, A), A)).$

The content of the message requires that only those agents that detect in their packs a temperature greater than the threshold will reply. In particular, they will send the external event $value\_temp\_highE(Ag)$ where Ag is the name of the sender agent. After receiving this external event, the agent sends a message to the Transport Device Agent (tda) in order to check its sensors values. In fact, if the increase of the temperature is justified by a natural motivation as, for example, the sun or the stop in a gallery, an alert is not in order.

$value\_temp\_highE(Ag) :> clause(agent(A), \_),$
$messageA(tda, send\_message(give\_temp(A), A)).$

If the temperature X communicated by the Transport Device Agent via the external event $value\_temp\_tda(X)$ is under the threshold but some parameters are evaluated negatively, then the Control Device Agent sends an alert message to all authorities who are in charge of the transport process. The $reasoning\_module(X, Y, R)$ is the responsible of the environmental parameters evaluation process. It takes as input the temperatures and other available parameters for evaluating if the temperature change is motivated. If the response R is negative, then the change is not motivated and an alert is sent. In this case, the increase of the temperature in the pack does not correspond to that of the external environment. If instead the external temperature overcomes the threshold (like for the packs), then there is no clear evidence of risk. However, the agent informs the responsible of the transport (tr) about the higher temperature and asks for the explicit authorization to increase the related threshold.

$value\_temp\_tdaE(X) :> once(evaluate(X)).$
$evaluate(X) : -clause(threshold\_temp(Y), \_),$
$X < Y, reasoning\_module(X, Y, no),$
$clause(agent(A), \_), messageA(authorities,$
$send\_message(alert\_temperature(A), A)).$
$evaluate(X) : -clause(threshold\_temp(Y), \_),$
$X >= Y, reasoning\_module(X, Y, yes),$
$clause(agent(A), \_), messageA(tr,$
$send\_message(alert\_temperature(A), A)), messageA(tr,$
$send\_message(new\_temperature\_threshold(X, A), A)).$

The monitoring of the packs distances is based on the ABUs positions. When packs are charged into the truck, the initial distance among packs is recorded. For preventing a thief to shift a pack unobserved, agents in the ABUs start, at the beginning of the journey, a cooperative activity with the objective of checking the relative distances among their packs. If the monitoring activity of an agent detects that a distance has been modified, it starts an interaction phase involving all agents in the ABUs. The distances among the packs are verified, the difference between the right distance and the new one is computed for each couple of packs and a specific

algorithm evaluates wether the movement has been collective. If the global movement is coherent, then the Transport Device Agent sends an information message to the VCC and resets the packs distances; else, it sends an alert message. For lack of space, we just propose an example of the proactive Control Device Agent behavior. The following internal event allows the agent within a pack to check the distance among it and the other Agents.
$monitor\_dist(Lat1, Lng1, Date, Ag_1) : -messageP(Ag,$
$send\_message(my\_position(Lat1, Lng1, \_, Date, \_), Ag_1)).$
$monitor\_distI(Lat1, Lng1, Date, Ag_1) :>$
$positionP(Lat, Lng, \_, Date, \_), clause(agent(Ag), \_),$
$clause(default\_distance(Ag, Ag_1, D), \_),$
$verify\_distance(Lat, Lng, Lat1, Lng1, D).$

where $verify\_distance(Lat, Lng, Lat1, Lng1, D)$ computes the distance between the Galileo coordinates (Lat, Lng) and $(Lat1, Lng1)$ and compares it with the default distance D. Every time the agent receives a position from another one, the internal event is triggered and the check is performed.

VI. Motivation: why DALI logical agents

Nowadays, many kinds of applications need some degree of autonomy. There are application contexts that actually offer no alternative to autonomous software. Agents provide a tool for structuring an application in a way that supports its design metaphor in a direct way. In this sense, they offer an appropriate support to the development of complex systems.

Platforms for building autonomous software require dedicated basic concepts and languages. At the level of individual agents, representational elements such as observations, actions, beliefs and goals are required. Reactivity is the ability of an agent to perceive its external environment and take appropriate measures in response to perception. Proactivity is the ability of an agent to take initiatives based on its own evaluation of relevant conditions. Going further on the line of autonomous software, new applications need "intelligence", in the sense of the ability to exhibit, compose and adapt behaviors, and being able to learn the appropriate way of performing a task rather than being instructed in advance.

A multi-agent system (MAS) is a collection of software agents that work in conjunction to each other. They may cooperate or they may compete, or they may adopt a behavior that combines cooperation and competition. In order to form a MAS, agents must have communication abilities together with an internal mechanism for deciding when social interactions are appropriate, both in terms of generating requests and of judging incoming requests.

Among the potential applications, distributed monitoring/control systems (DCMS's for short) appear to be a natural application for agents, by virtue of controllers being in principle autonomous entities. This kind of application implies measuring a system, so as to verify whether specific measurable values are within a pre-defined range, and acting on the system so as to keep specific observable values within the range characterizing an acceptable behavior of the system itself. Measuring a system implies selecting the correct checks to perform at each stage. Controlling the system implies being able to either modify or restore its operational parameters as behavior requires. Agents can replace a human operator in this kind of task. If the controlled system is composed of several parts, single agents can control the various parts, and can cooperate so as to enforce the system overall behavior.

The DALICA system can be seen as a distributed monitoring system. The "objects" of the agents monitoring activity are the packs containing cultural assets. DALICA in fact has been designed as a cultural assets DCMS based on the Galileo platform. An agent-based solution has been chosen for the following reasons. Each pack has to be supervised individually in autonomous way, reacting to every stimulus but also to every relevant exogenous state change: these changes must be detected and appropriate measures must be taken.

Typical events that may happen in a DCMS such as the DALICA system are highly asynchronous. The reactive nature of agents allow asynchronous events to be coped with in a natural way. Moreover, proactivity allows system parameters to be observed and tuned whenever necessary, thus potentially preventing the occurrence of critical situations. The distributed nature of this DCMS implies the need for each component to possibly communicate the events occurred to other components. Then, agents supervising single components form a MAS with the overall objective of coordinating activities also in case of critical situations. Agents communicate to the others that some potentially unwanted change has occurred, and the MAS cooperatively establish which are the necessary actions to be undertaken. However, no agent is allowed to directly force other agents to behave in a certain way. This improves the system reliability also in presence of software malfunctioning.

An Object-Oriented solution has not been applied as in the DALICA scenario it appeared less suitable and more difficult to implement. In fact, in the Object-Oriented paradigm message exchange means methods invocation, i.e., synchronous procedure call, which means that autonomy and privacy of components are hard to reach, especially whenever "public" methods are allowed.

In the particular DCMS that we have considered, some kind of "intelligence" is needed so as to interact with the environment in a flexible customizable evolving way, rather than through predefined rigid unalterable patterns. In our DALI implementation, computational logic has taken a relevant role in this sense, as a good tool for building intelligent agents. The DALI logic language in fact, due to the traditional "fast prototyping" character of logic languages in general, to the new efficient implementation and to the new concepts that it embodies, has proven to be suitable for implementing such an advanced application.

Moreover, in a complex distributed environment, rule-based logical languages allow behaviors to be defined by means of independent sets of rules. These behaviors will be triggered in any order by what happens in the environment, without a complex control structure that should foresee all cases or

combination of cases.

Logic languages in general are evolving from static to "active", and are being enriched with new capabilities based on the "agency" metaphor. In fact, the application presented in this paper practically demonstrates that logic agent-oriented language may provide an affordable way of introducing the engineering of intelligent behaviors into software engineering and development practice. In addition, the clear semantics of such languages allows formal properties of an implemented system to be proved, which is relevant in critical application domains.

VII. Related work

To the best of our knowledge, no other systems exist capable of monitoring the transport of cultural assets by exploiting the Galileo satellite signal and agents technology. However, agents have been adopted for monitoring activities in several contexts.

The work of Bunch et al. in [4] is centered on a particular experimentation scenario: the monitoring of chemical processes by means of software agents. In particular, the KARMEN multi-agent system monitors specific combinations of process conditions of interest to individual plan operators, supervisors and other personnel and notifies them through modes that the individuals select in accordance with corporate policy and personal preferences. The monitoring of Medical Protocols by means of agents is described in the work of Alsinet et al. in [1]. Specialized domain agents assist and supervise the execution of medical protocols in hospital environments. As in the DALICA system, in this sensible context authors have introduced privacy, integrity and authentication methods for guaranteeing a secure process of information exchange among agents.

Finally, we mention the works by Muller [18] and Ramasubramanian et al. [21]. In the former, Intelligent Agents improve network operations by identifying, for example, the overall load of network traffic, what times of the day certain applications load the network, which servers may be over-utilized and so on. Instead, the latter integrates intelligent agents in an Intrusion Detection System. The critical context where agents are put at work (a Corporate Bank, Chennai, India) proves their reliability in facing complex and critical tasks. In order to reduce the points of failures which are unavoidable in centralized security systems, the authors designed a dynamic distributed system in which the security management task is distributed across the network by using intelligent agents.

If we concentrate our attention on the available systems capable of exploiting the satellite signal for monitoring activities, an interesting tool is Track-King [12]. It supervises the temperature of the goods during the transport phase. In particular, the system provides continuous verification that the refrigerator equipment is operating at the right set point. The satellite system has a wide coverage and allows remote intervention if necessary. Track-King does not use intelligent agents for reasoning activities. Other systems exploit the GPS signal for tracking purposes, among which are those proposed by several companies ([13], [14], [15]) for protecting the cars from thefts.

In general, they advice the car owner when the vehicle is moving or when it goes out the prefixed route. The reaction often involves the car halt by means of particular mechanic devices or the sending of an alert text message to the cell phone of the owner. These systems imply reactive capabilities only and no proactivity or learning methods are applied. Moreover, the car monitoring process does not require the adoption of cooperation strategies because each monitoring system in a car works independently of others.

Finally, several works in literature have proposed systems for supporting the users during their visits to museums ([19],[20],[2],[11]). We cite in particular the KORE system [3] where agents have been adopted for reasoning about the visitors profiles. The architecture of KORE is based on a distributed system composed of some servers, installed in the various areas of museums, which host specialized agents. The KORE system practically demonstrates that intelligent agents can have a relevant role in capturing the user profile by observing the visitor behavior. They possess the capability to be autonomous and to remain active while the visitor completes her/his visit; they can percept through the sensors all choices performed by the user and, consequently, activate a reasoning process.

In summary however, to the best of uor knowledge the GTA and the DALICA system jointly applied to a cultural assets transport scenario constitute a novelty in the field of AI.

VIII. Concluding Remarks

In this paper, we have presented the DALICA MAS architecture applied in a transport monitoring scenario. Considering the risks involved in the cultural assets transport, we think that the GTA and the DALICA MAS can be profitably exploited in this context, thus making the whole process more secure. As discussed in the previous section, the role of the agents in monitoring activities is increasingly effective in time. Their reasoning capabilities are relevant in all those cases where it is necessary to discriminate the events and to correctly interpret the reality. In the future, we will enhance DALICA system by introducing new features for reasoning on a wider range of environmental factors. We also intend to improve the learning capabilities of agents both in general and in this scenario.

References

[1] Alsinet, T., Bjar, R., Fernanadez, C., and Many, F. *A Multi-agent system architecture for monitoring medical protocols*, In Proc. of the Fourth international Conference on Autonomous Agents (Barcelona, Spain, June 03 - 07, 2000). AGENTS '00. ACM Press, New York, NY, 499-505. URL=http://doi.acm.org/10.1145/336595.337580

[2] Amigoni, F., Della Torre, S., and Schiaffonati, V., *Yet Another Version of Minerva: The Isola Comacina Virtual Museum*, In Proc. of the First European Workshop on Intelligent Technologies for Cultural Heritage Exploitation, at The 17th European Conference on Artificial Intelligence, 1-5, 2006.

[3] Bombara, M., Cal, D., and Santoro, C., *Kore: A multi-agent system to assist museum visitors*, Proc. of the Workshop on Objects and Agents (WOA2003), URL=http://citeseer.ist.psu.edu/708002.html, 2003

[4] Bunch, L., Breedy, M., Bradshaw, J. M., Carvalho, M., Suri, N., Uszok, A., Hansen, J., Pechoucek, M., and Marik, V. *Software agents for process monitoring and notification*, In Proc. of the 2004 ACM Symposium on Applied Computing (Nicosia, Cyprus, March 14 - 17, 2004). SAC '04. ACM Press, New York, NY, 94-100. URL= http://doi.acm.org/10.1145/967900.967921.

[5] Costantini, S., and Tocchio, A. *A logic programming language for multi-agent systems*, In Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002, LNAI 2424, Springer-Verlag, Berlin, 2002.

[6] Costantini, S., and Tocchio, A. *The DALI logic programming agent-oriented language*, In Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004,LNAI 3229, Springer-Verlag, Berlin, 2004.

[7] Costantini, S., Tocchio, A., and Verticchio, A., *A Game-Theoretic Operational Semantics for the DALI Communication Architecture*, In Proc. of WOA04, Turin, Italy, December 2004, ISBN 88-371-1533-4,2004.

[8] Costantini, S., and Tocchio, A. *About declarative semantics of logic-based agent languages*, In Post-Proc. of DALT 2005, LNAI 3229, Springer-Verlag, Berlin, 2006.

[9] Costantini, S., Inverardi, P., Mostarda, L., and Tocchio, A. *A Geo Time Authentication System* In IFIPTM 2007, Joint iTrust and PST Conferences on Privacy, Trust Management and Security, in print.

[10] *CUSPIS PROJECT WEB SITE* URL=http://www.cuspis-project.info/demonstrations.htm

[11] Damiano, R., Galia, C., and Lombardo, V., *Virtual tours across different media in DramaTour project*, In Proc. of the First European Workshop on Intelligent Technologies for Cultural Heritage Exploitation, at The 17th European Conference on Artificial Intelligence, 2006

[12] Fick, T. *Remote monitoring: a logistical solution for profiting in a changing world*, In Food Safety, GDS Publishing URL=http://www.gdspublishing.com/ic_pdf/usfs/thki.pdf.

[13] GPS car monitoring system. *SAT track GPS Vehicle Tracking System*, URL=http://www.pcmobile.ca/PC-CDN-Inside.pdf.

[14] GPS car monitoring system. *LoJack - GPS Vehicle Tracking*, URL=http://www.lojack.com/gps-vehicle-tracking.html.

[15] GPS car monitoring system. *Millennium - Advanced Internet Based GPS Vehicle Tracking*, URL=http://www.vehicle-tracking-gps.com.

[16] Kowalski, A. *How to be Artificially Intelligent - the Logical Way*, Draft, revised February 2004, Available on line, URL=http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html, 2006.

[17] Lloyd, J. W. *Foundations of Logic Programming (Second, Extended Edition)*, Springer-Verlag, Berlin, 1987.

[18] Muller, N. J. *Improving Network Operations With Intelligent Agents*, Int. J. Netw. Manag. 7(3) (Jul.1997), 116-126. $URL = http : //dx.doi.org/10.1002/(SICI)1099 - 1190(199705/06)7 : 3 < 116 :: AID - NEM239 > 3.0.CO; 2 - Q$

[19] Park, D., Nam, T., Shi, C., Golub, G.H., and Van Loan, C.F., *Designing an immersive tour experience system for cultural tour sites*, In CHI '06 Extended Abstracts on Human Factors in Computing SystemsACM Press, 2006.

[20] Pilato, G., Augello, A., Santangelo, A., Gentile, A. and Gaglio, S., *An Intelligent Multimodal Site-guide for the Parco Archeologico della Valle dei Templi in Agrigento*, In Proc. of First European Workshop on Intelligent Technologies for Cultural Heritage Exploitation, at The 17th European Conference on Artificial Intelligence, 2006.

[21] Ramasubramanian, P., and Kannan, A. *Intelligent Multi-agent Based Database Hybrid Intrusion Prevention System*, In ADBIS, 2004, URL=http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=3255&spage=393, DBLP:conf/adbis/2004, DBLP, URL=http://dblp.uni-trier.de.

[22] *SICSTUS PROLOG HOME PAGE* URL=http://www.sics.se/isl/sicstuswww/site /index.html.

[23] Stallings, W. *Cryptography and network security: Principles and Practice*, Prentice Hall, 2006.

[24] Tocchio, A., *Multi-Agent sistems in computational logic*, Ph.D. Thesis, Dipartimento di Informatica, Universitá degli Studi di L'Aquila, 2005.

[25] Viroli, M., Holvoet, T., Ricci, A., and Schelfthout, K., and Zambonelli, F., *Environments in multiagent systems*, The Knowledge Engineering Review, 20(2), 2005.

A Multi-Agent Platform Supporting Maintenance Companies on the Field

Andrea Passadore, Giorgio Pezzuto, *D'Appolonia S.p.A. Via San Nazaro 19, 16145 Genova, Italy*

Abstract—In this paper we present the European project named E-Support, aimed to the maintenance companies which work on the field, away from the central headquarters. The main goal of E-Support is to help the field engineers and technicians to access the knowledge base of the company. They will connect to remote servers by using mobile devices in order to get information about vendors, customers, plants, parts and download technical documents. The whole system will be implemented by a multi-agent platform running agents on mobile devices and server agents that provide the services. A particular emphasis will be placed on the contribution of D'Appolonia regarding the document retrieval system.

***Index Terms*—Enterprise, multi-agent, document, indexing, clustering, ontology.**

I. Introduction

E-SUPPORT is a collective European project which involves RTD (Research and Technology Development) companies, SMEs (Small and Medium Enterprises), and associations of maintenance companies. The project is addressed to these maintenance companies which send small teams of engineers and technicians at the customer's, in order to fix, upgrade or simply manage a plant. Often the personnel which works away from the headquarters needs information in real-time regarding the customers, vendors, parts to replace and any type of data useful during the everyday work activity. The aim of E-Support is to provide these data on the field, directly on the plant, connecting a mobile device to the remote server containing all the information owned by the company and the information offered by the associations (as regulations, norms, standards, etc.) that reunite maintenance companies. E-Support enables technicians to use every type of mobile device: mobile phones, smart phones, PDAs (Personal Digital Assistant), and notebooks. It is a task of the remote system to display on the mobile device screen only the information that the device is able to visualize. The entire system runs over a multi-agent platform. Considering the need to run remote agents representing the mobile devices, we chose the well-known multi-agent system (MAS) JADE [1], with the extension for mobile devices named LEAP [2].

A field service engineer (FSE) which wants to access the enterprise knowledge base, connects his mobile device to the available network infrastructure (GPRS, UMTS, Wi-Fi, Wi-Max) and contacts (through his mobile agent) a remote agent running on the server platform. This remote agent (called *interface agent*) processes the query and forwards the related tasks to the agents in charge of the databases management and the documents collection. The interface agent then collects the results of these tasks, generates an html page considering the display capabilities of the mobile device, and sends it to the FSE.

The role of D'Appolonia in E-Support is to manage the documents collection providing advanced tools able to classify them by using clustering techniques over the usual indexing of the documents corpus.

In the following sections are illustrated the E-Support project ensemble (section II), the MAS solution (section III), and the documents retrieval system (section IV), highlighting the contribution of the agents in the project success.

II. The E-Support project

A. Project context

The first task of the project was the analysis of a market study addressed to 65 European maintenance companies from Romania, Italy, Holland, Spain, and Slovakia. Most of the maintenance companies have less than 20 workers (35.5%) or 21 to 100 workers (38.7%). The survey shows that the 47.6% of the companies covers different maintenance sectors. The 23.8% operates in the chemical sector, the 22.2% in the construction field, the 15.9% in automotive, the 15.9% in the alimentary field. Few companies work in the ICT sector (4.8%). The 89.8% handles technical data in electronic format: especially by using office tools, cad, and CMMS (Computerized Maintenance Management System). No one seems to use databases.

The communication devices used by technicians are mainly cell phones (83% of companies), notebooks (55.4%), smart phones (15.4%), PDAs (10.8%) and tablet PCs (3.1%).

The interviewed companies assert that, by using the existing tools, the detected difficulties concern the trouble to find accurate (55.4%) and complete (33.8%) information, to file the info on paper (32.3%), the lack of immediate responses from the manufacturers (30.8%), to get updated information (24.6%), and the waste of money (24.6%) and time (20.0%).

Considering the emerging context, the maintenance companies which fill out the questionnaire, judge the E-Support project useful (30.6%) or quite useful (56.9%).

On the basis of these results, the E-Support tool is conceived as to limit the costs incurred by the SMEs, to easily find significant information and especially to reach the field

service engineers on the field. For these reasons, the E-Support system is designed as a service provider managed by maintenance associations and distributed to the subscribing SMEs. Figure 1 shows the high level architecture of the system. Four major challenges are recognized, in order to provide a useful tool:

- The *document retrieval system* (named FSE-Assistant) for the management of documents and the easy discovery of them.
- The *knowledge sharing and learning* (FSE-Master) for the management of databases and the training of the workers.
- *Wireless mobile client infrastructure*: it regards the network infrastructure, the security, and the compression of transmitted data.
- The *multi-agent platform* which hosts the whole service, mobile devices included.

B. *Document retrieval system*

The aim of the document retrieval system is to allow technicians on the field to find information contained in textual documents, datasheets, cad files and pictures. This large amount of electronic documents is hosted in a file server staying at the provider server farm. The tool the E-Support project wants to delivery is not a mere search engine, able to index textual documents and to systematically return a list of files containing a word: the tool is also able to order the documents into clusters [3] namely categories (and sub-categories) which contain documents belonging the same topic. These topics are automatically selected by advanced clustering algorithms. An interesting objective is the indexing of non-textual documents. In this case is useful to manually label these files with tags. The implementation of the document retrieval system is deeply discussed in section III.

C. *Knowledge sharing and learning*

The goal is to develop an intelligent open knowledge sharing and learning system able to provide a technical training to FSEs who work on site. The FSE-Master comprehends intelligent user-profiles and personalization capabilities, in a user-friendly web-based context. The learning audience benefits from a modular step-by-step approach, following several training processes: *learning*, *practicing*, *testing*, and *assessing*. According to the E-Support essence, the field service "students" can access to the "classroom", when and where they need, through their mobile devices. The e-learning tool is enriched with the possibility to share a personal knowledge with other colleagues, offering own experiences in helping them to solve a problem.

D. *Wireless mobile client infrastructure*

The main problem in reaching the workers on the field is the network infrastructure. For this reason the E-Support system must be versatile, offering different media to connect a mobile device to the central server. Considering the low costs level of the E-Support product, oriented towards the SMEs, the mobile device is able to manage different standards, choosing among those that are available on site. The E-Support tool must tolerate several bandwidths and rate profiles, self-adapting the throughput to the current context. Several wireless technologies have been evaluated considering their speed, price, compatibility with the devices, and coverage. The E-Support project took into account existing and incoming standards [4] as: Wi-Fi, Wi-Max, UMTS, GPRS, and Edge. The main considerations regard the coverage and the speed. Wi-Fi is quite widespread in offices, factories and production areas and it has an adequate bandwidth. UMTS and GPRS cover every populated area of Europe but they denote a low bandwidth. The Wi-Max technology represents the best solution, but, at now, is not commercialized and not at all diffused. Considering these points for the E-Support system, the main wireless medium is the Wi-Fi where possible, switching to GSM standards otherwise. The Wi-Max will be monitored in order to introduce it in the system as soon as possible.

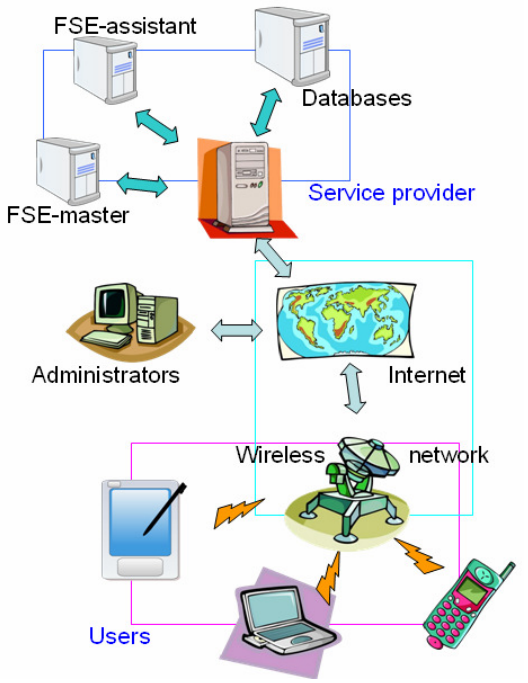


Fig. 1: the whole E-Support service.

Other issues related to the communication layer are the security of transactions and the compression of transmitted data [5]: S-HTTP and DES (Data Encryption Standard) ensure safe communications and the ZIP method to compress the exchanged files.

The connection switching and security functionalities are hidden to the end-user who can use the E-Support device without caring the communication status.

E. *Multi-agent architecture*

The motivations that encourage us to adopt a multi-agent platform concern especially:

- The dynamicity of the system, considering the mobile

devices, the different data sources, the needed intelligence and pro-activity of the requested software components.
- The scalability of the system, which has to be adapted to different scenarios, with enterprises having different sizes and requirements.
- The naturally distributed environment.

To implement the E-Support multi-agent society, the JADE platform has been chosen, due to its good reputation, stability, and mainly the possibility to distribute agents and agent containers over a network of mobile devices, by using the JADE LEAP extension. Another consideration is the fact that other E-Support components are written in Java and therefore they are easily mixable with the multi-agent platform.

The multi-agent platform is deeply investigated in the next section.

III. THE MULTI-AGENT PLATFORM

A. Introduction

In order to design a multi-agent system which accomplishes the main requirements of E-Support, some considerations are reported:

- The connection among mobile devices and the core platform is wireless.
- The connection could be slow, considering the use of different wireless technologies (GPRS, UMTS, Wi Fi, and Wi-Max).
- The connection could be affected by line losses.
- Some mobile devices could have strict restrictions to display downloaded files and information and to execute weighty programs.
- The access to centralized information is concurrent.
- The platform must be scalable and adaptable to different kind of companies, also by using pre-existent tools and databases.
- Different programmer teams will work on the system. Agent roles help the teams to integrate their modules. Ontologies modeling agent interactions are a tool that improves the integration of these service modules.

For these reasons, the architecture of the system takes into account that every hard computation is in charge of the server side, letting the mobile device free (especially if it is a PDA or any other device with limited resources). The mobile device hosts only one agent, able to connect to the remote platform, sends queries to the system, receives the results and browses them. This agent, called *front-end* agent, represents the end-user, namely the technician on the field who wants to exploit the functionalities of the E-Support platform.

The front-end agent is able to interact with *back-end agents* hosted on the server side. Back-end agents provide differentiated services to manage databases, the indexing engine, the clustering engine, and the authentication service. These agents can be cloned and the multi-agent system server can be split into different computers, in order to increase performances or to adapt to extended companies with particular requirements. The JADE cloning function and the agent containers are useful tools, in this sense.

The front-end agent does not interact directly with the back-end agents, but leans on the *interface agent*, which offers an interface of the server side services. It is a sort of mediator which collects the queries coming from the end user and reroute them to the back-end agents. Details about the interface agent and other agents are shown in the next paragraphs.

B. Front-end agent

A front-end agent is merely a browser which displays results sent by the interface agent which is talking with the agent. Every heavy computation is on the back of core agents and especially on the back of an interface agent.

When a user wants to connect to the E-Support platform through a mobile device, the corresponding front-end agent contacts the *Directory Facilitator* (DF, the yellow pages service, embedded in JADE) of the platform to discover the name of a free interface agent. The DF responds and the front-end agent opens a conversation with the suggested interface agent. Then, the user must authenticate himself, sending to the interface agent his username and password. Automatically, the front-end agent communicates the mobile device type too, in order to send, in response to the incoming queries, only the displayable information.

The user is now able to query the system. Every communication is in charge of the interface agent, which routes requests to the core agents. Every front-end agent has different behaviours, differentiated by user privileges. The intention is to consider every mobile device at disposal of each technician on the field, regardless of his rank. Every agent has a complete set of behaviours, raised only if the current user has the required rights.

Regarding the front-end agent, JADE LEAP allows programmers to split the related agent container to a front-end, hosted on the mobile device and a back-end hosted on the server, in order to move the infrastructure of the agent container on the server side, avoiding an overload of the mobile device.

C. Interface agent

The main task of an interface agent is the mediation among front-end agents and core agents. The aim of interface agents is to book core agents just the time necessary to serve out the query, without waste of time due to slow connections and line losses. The interface agent collects the queries of the front-end agent and redirects them to the core agents, then it gets the query result and releases the core agent; achieved the information, the interface agent composes a presentation using html tags, considering only those records that can be displayed by the mobile device. The information about the type of the mobile device involved in the communication can be retrieved, asking the authentication agent.

Every interface agent has more behaviours, every one specialized to interact with a core agent. Depending on the

company size and the number of field technicians, the interface agent can be configured to serve only one front-end agent at a time or to serve a strict number of front-end agents at the same time.

D. Authentication agent

It manages user's credentials and maintains his status (i.e. if he is online and his mobile device type). The information about the agent status is useful to compose a presentation page taking into account the display capabilities of the mobile device. For this reason, the interface agent contacts the authentication agent to get these data, at the moment of the composition of an html page.

E. Database agents

They manage the knowledge base of the company. The database agents interrogate the database using SQL queries.

Implementing different behaviours, it is possible to manage different types of database, considering the pre-existent tools at the company's disposal (e.g. Oracle, SQL Server…). The database contains information about customers, vendors, suppliers, parts, components and data supporting e-learning tools.

F. Text indexing agents

They index large amount of documents (*doc*, *pdf*, *txt*, *html*, etc.) reading selected folders and downloading updated versions of manuals from suggested web sites.

These agents (one or more, depending on the company size and possible specializations) write the indexing results to a centralized index, managed by a specialized agent, with the main task to coordinate the concurrent access to the centralized index. We distinguish among agents that read text documents hosted in a file server and that read web pages.

G. Clustering agent

The clustering agent processes the index created by indexing agents, in order to execute the query sent by remote users (via the interface agent). It analyzes the index and returns a list of clusters, also considering an ontology of relevant terms concerning a certain domain. The returned list of clusters and documents is raw and must be processed by the interface agent, to erase unreadable files (considering the mobile device type) and to present the result in human readable template adaptable to different kind of displays. More details about clustering agent and its services are reported in section IV.

H. Web agent

The web agent is in charge to manage interactions between the platform and the customer, who can get information about the status of the maintenance process or submit a new fix request.

The agent submits queries to the interface agent, as a normal end-user. The customer can also interact with the maintenance company through usual channels, as the telephone or email. In this case is the human operator to insert o to communicate data about a fix action.

I. Other agents

The previous agents are the most significant ones, but we can consider the introduction of other agents in order to manage integration with existent CMS (Content Management System) and other management tools. These agents denote behaviours which allow conversion among the E-Support internal knowledge representation and third-parts knowledge bases. They are easy to introduce in the MAS, if we consider them as a sort of "special" end users interacting with the interface agents and then the core agents.

J. Related works

Due to the complex nature of the whole E-Support project, it is difficult to find and describe similar works that involve every aspect of this project. Relating the document retrieval system and multi-agent technology, several proposals are described in literature. The management of electronic documents is in turn, a complex problem that involves issues as scalability, high dimensionality, content meaning, etc.

The split of the entire complex problem into circumscribed aspects is a common approach and the multi-agent system technology is a valid help to solve and manage these contexts.

A solution aimed to multimedia documents is proposed in [13] where the main goal is to efficiently accesses a scene of a video without a brute force approach (by using forward and rewind functions). As an example is reported the recording of a conference event; the system, based on specialized agents, will be able to select a scene given a query like ("give me the scene where T presents its paper). The provided solution is to define agents specialized to process single media (video, audio, text) and other agents able to locate a face in a frame, to identify the selected face, and to check if the person is speaking. An agent is able to analyse the audio in order to extract intelligible words (or, more easily, to analyse a text describing a scene provided by a human operator). The orchestration of these agents allows the system to perceive the goal.

Another solution aimed to the indexing of document in a team of users (both end-users and owners of documents) is described in [14]. An agent runs on every team member PC; it is able to maintain an updated index of data stored in the PC and to reply to the queries given by the team member. This agent contacts its counterparts running on the other members' PCs, in order to support a parallel document search. Considering the architecture of the team network, the agent running on a PC interacts with peers synchronously where possible, asynchronously (via mail) where firewalls or other systems inhibit the direct communication.

In [15] is illustrated a prototype which is quite similar with the E-Support document retrieval system. The system offers facilities to index and share information both in local repositories and in the WWW. Every user can organize the retrieved information in hierarchical categories (a sort of static cluster) with the help of a personal agent which is able to interact with other personal agents in order to share local data among end-users. Other agent roles are defined: the matchmaker agents help personal agent to find the peers that manage significant information for a particular end-user query.

Group agents are particular personal agents that manage common repositories, not directly linkable to a specific end-user. Finally the knowledge agent (that is under development) has the main feature to manage knowledge bases related to particular domains in order to improve the results of end-user queries.

Regarding the clustering techniques involving multi-agent systems, a proposed solution [9] split the clustering process, usually centralized, in a distributed environment, where clustering agents manage local repositories and learn from the results of cooperative peers.

Another proposed system involves swarm intelligence and in particular the self-organization behaviour of ants applied to large amount of documents.

K. *A demonstrative scenario*

In order to explicate the main functions of these agents, a simple interaction of a remote user is reported (figure 3).

A technician on the field must access to the company knowledge base to search information about a *hot-water heater* to repair.

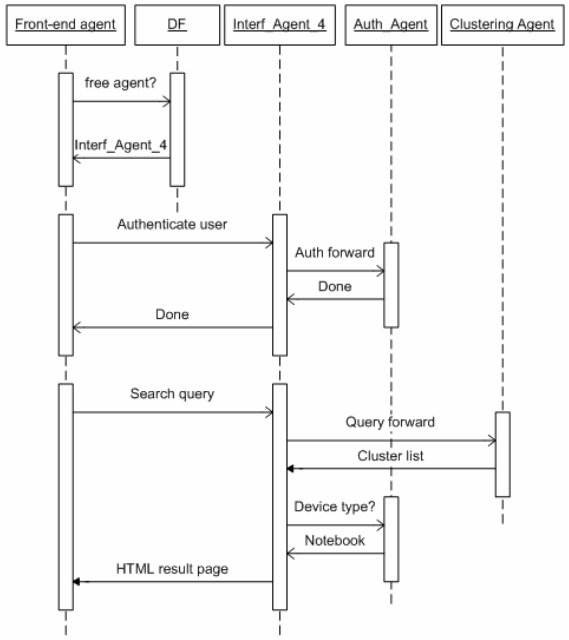


Fig. 2: an example of agent interactions.

He is on the field, namely in the boiler room, so he can connect to the Wi-Fi LAN of the office. Through Internet, the front-end agent contacts the remote server and asks the DF in order to find a free interface agent (see in figure 2 the agent interactions for a search query). Established the channel between front-end agent and interface agent, the end-user must authenticate himself. The front-end agent sends username, password and mobile device type to the interface agent which reroutes the information to the authentication agent. Once the user is logged, he can send queries to the remote system. As an example he wants to find all the documents containing the word “heater”. He types the word “heater” and its front-end agent sends the query to the interface agent.

The interface agent forwards the query to the clustering agent. Ignoring for the moment the detailed functioning of the information retrieval system, the clustering agent reads the index of documents owned by a particular indexing agent and then returns a hierarchical list of categories containing all the documents belonging to knowledge base of the company, in which is contained the word “heater”. The categories represent different topics regarding a heater, depending on the occurrence of most significant words in every document. If the mobile device used by the user does not support a type of document (for example *ps* files), the interface agent prunes from the clustering list all the *ps* files. Therefore, the cluster list is converted by the interface agent in an html page, ready to be sent to the front-end agent.

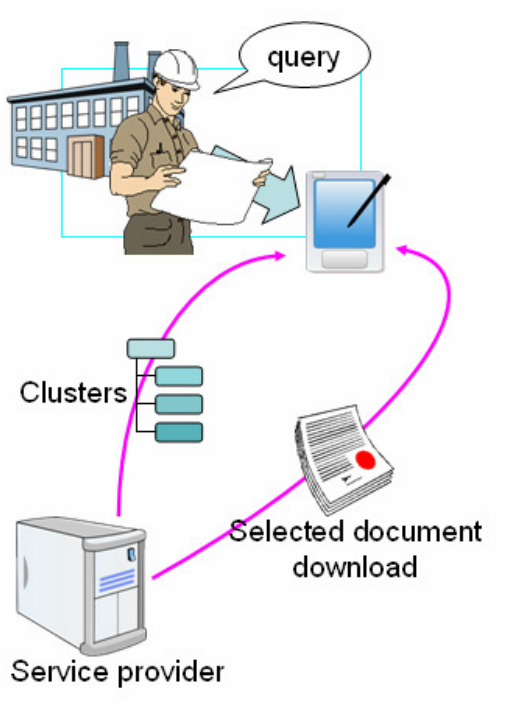


Fig. 3: the remote connection of a worker.

The technician is now able to navigate the clusters, in order to select the most relevant documents and download them.

Read the documents, for example technical manuals, the user decides that the heater must be repaired, replacing a particular component. In this case, the technician can interrogate the central database to discover the closest supplier of this component, its price, and relative instructions. As usual, the front-end agent contacts the interface agent, which reroutes the queries to the apposite database and composes the html page containing the results.

IV. DOCUMENT RETRIEVAL SYSTEM

A. Introduction

The contribution of D'Appolonia in the E-Support project concerns the *document retrieval system*. As mentioned in the previous sections, this system is able to receive the queries of the user, in order to find documents and every other useful file for the everyday work activity. The goal is to provide a smart system which allows the indexing of the entire document corpus of the company and the automatic clustering of documents on the basis of the searched word. E-support wants to ease the work of technicians and engineers on the field and this essence is also applied to the document retrieval system. The use of clustering techniques represents a first automatic partition of documents in the most relevant topics concerning a particular maintenance field. The solution is reported in the next paragraphs.

B. Clustering techniques

The document clustering is an unsupervised learning technique that furnishes a hierarchy of document which facilitates the browsing of a collection of documents. Documents belonging the same cluster have a high degree of similarity. In general, during the clustering generation process, a document is represented by a vector that contains the most significant words of the selected document. A pre processing process could be useful to remove stop words, forbidden words, etc. Some clustering techniques allows the extrapolation of the most significant words considering the entire document set, in order to focus a set of expressions with a significant discriminating power. Meaningful elements to classify a clustering algorithm are the following:

- High dimensionality: usually, in a document there are thousands or tens of thousands relevant terms. Each term represents a dimension of the document. Natural clusters are not selected in the full dimensional space, but in subspaces formed by a set of correlated dimensions. The identification of these subspaces is often difficult.
- Scalability: the algorithms must work fine with both limited and large sets of documents.
- Accuracy: documents belonging the same cluster must be similar. External evaluation methods for the accuracy measure are developed [10].
- Meaningful cluster description: every cluster must be described with a significant label which helps the user to browse the cluster hierarchy.
- Prior domain knowledge: several algorithms can be tuned with input parameters. Oftentimes, the user is unable to set up these parameters. In this case the algorithms should not sensitively decrease the performances.

Clustering algorithms can be classified into main categories:

- Hierarchical clustering methods: they can have a bottom-up approach, i.e. they generate a set of clusters and then they merge the most similar clusters; the top-down approach, on the other hand, divides every cluster in sub-clusters, until an end condition is reached.[11]
- Partitional clustering methods: they consist in the k-means methods and variants. Every document is associated to the closest centroid (considering the similarity), starting from a set of k random centroids. The algorithm selects a centroid to split and repeats the process, until the number of k cluster is reached. [11]
- Frequent item-set based methods: the idea is that many frequent items (namely terms) should be shared within a cluster and different cluster should have more or less different frequent items. The result is not a hierarchical clustering; in order to introduce this feature, the notion of item-set is created. [12]

C. Architecture

The figure 4 shows the proposed architecture of the system. The whole system is based on the collection of textual documents and other type of files (as pictures, cad files, datasheets, etc.). These documents can be tagged, in order to add useful information to the single file. This is helpful for textual documents, in order to add detailed information about the nature of the document; this option becomes necessary in case of non-textual documents. These tags are defined in specific ontologies: one for the description of document types and one containing the relevant entities of a particular maintenance sector (for example computer maintenance, electrical, heating, building, etc.). The administrator of the system or a skilled employee can create an instance of a particular concept and then attach it to the selected document.

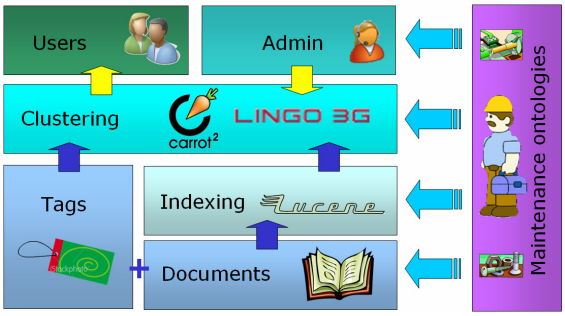


Fig. 4: the document retrieval system architecture.

The textual documents are indexed by the *Apache Lucene* indexing engine. An indexing agent continuously reads the document corpus and updates the generated index. Another similar agent supervises a group of selected web sites and the result of the web indexing is merged with the main one.

The clustering engine taps into the Lucene index and the set of tags in order to provide a more efficient clustering. For the E-Support we select the *Carrot²* clustering engine, which implements the most advanced clustering algorithms.

Authenticated users exploit the functionalities of Carrot² and administrators are able to tune the parameters of Carrot² and configure the whole system.

D. The document corpus

The work of a maintenance technician is not only based on the hint of textual documents. Maintenance companies provide precise requirements regarding the sharing of pictures of plants and parts, with attached comments containing the experiences of the colleagues. These companies consider helpful the indexing of cad files containing the technical schema of the plants, datasheet with details of the components, and maps of buildings, compounds and cities.

E. Tags and ontologies

An ontology containing the different types of documents is going to be developed. The aim of this ontology is to provide a complete catalogue of the possible documents owned by a maintenance enterprise. For example, the document ontology contains the *textualDocument* class, with properties *hasTitle*, *hasTopics*, and *hasComment*. The textual document class is specialized in different subclasses, e.g. the *book* class with properties *hasTitle*, *hasTopics*, *hasComment* (all inherited from the textual document class), *hasISBN*, *hasEditor*; the webpage class with properties *hasTitle*, *hasTopics*, *hasComment*, *hasURL* etc. Another class (disjoint from the *textualDocument* class) could be the *picture* class, with a property for the technician comments, the location, the date etc.

Regarding the maintenance ontologies, a generic maintenance ontology will be delivered and different domain specific ontologies will be built on the basic one. The survey involving European maintenance companies shows that these companies cover different maintenance sectors and often the same enterprise manages several fields. Therefore, specific ontologies are the solution in order to deeply customize the functioning of the document retrieval system, especially increasing the performances of the clustering engine.

For example, if the company does maintenance in the computer field, an ontology based on Information Technology will be built. The ontology will contain the most relevant entities of the computer discourse domain. The components of a PC will be described. As an example a class describing a chip of RAM, handles properties like *hasManufacturer*, *hasCapacity*, *hasDimension* etc.

Both the ontologies are used to catalog a document. For instance, we have a web page containing specific information about RAM chips and then the human operator can add a series of tags describing the page: a tag that instances the concept of the *web page*, one or more tags instantiating the concept RAM, and so on. The ontologies are developed by using the Protégé ontology editor [13] and the OWL language [14].

F. The indexing engine

To furnish a valid tool for the document clustering, the whole system must be based on a robust indexing engine, able to index efficiently a large amount of files, customizable, open source and possibly written in Java. The Apache Lucene project corresponds perfectly to these requirements. Lucene offers good performances in terms of RAM and CPU usage and disk space occupation. It allows the definition of customized file parsers in order to read every type of file. The complete API of Lucene allows the implementation of an agent able to manage the indexing process and the management of the resulting index. The indexing engine provides functions to merge different indexes, to build an index in incremental mode or batch mode; it allows the simultaneous searching and updating and the research through several fields as the title, the author, the content, etc.

G. The clustering engine

It is the core of the document retrieval system. The selected tool is the Carrot² clustering engine. It is a Java open source software that automatically organizes the search results into thematic categories. At now Carrot² supports 5 algorithms: Fuzzy Ants [15], HAOG-STC, Lingo [16], Rough K-Means [17], STC [18]. The clustering engine can exploit different indexing engines both online as Google, MSN, Yahoo, and offline as Lucene.

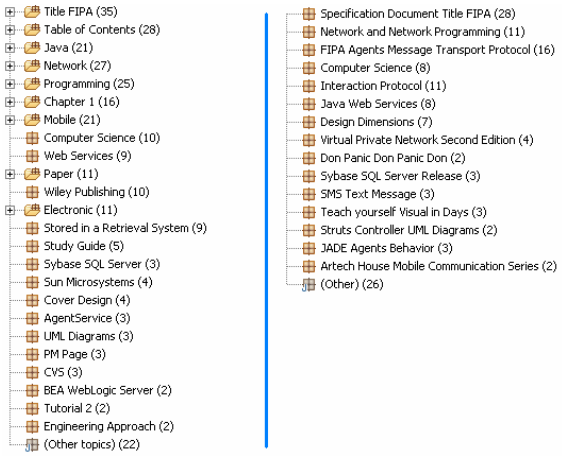


Fig. 5: the clusters generated by Lingo3G compared with the Carrot² ones.

There exists a commercial version of Carrot², named Lingo3G, which provides improved features and performances. It is able to get hierarchical clusters, to filter or boost suggested cluster names, supports the definition of synonyms and multilingual clustering. The APIs of the free version and the commercial one are quite similar, then, in the E-Support project, we intends to provide both the services and allow the companies to chose the level of accuracy they need buying the license of the commercial version or using the open source one. In parallel with the Carrot²/Lingo3G clustering we execute a sort of clustering based on the existing tags. The clustering based on tags is more accurate, because it derives from a human classification. Even if the automatic clustering is less accurate and smart, the results of the tests we did are encouraging. The results of the two clustering processes are merged, obviously highlighting the ones coming from the tag clustering.

Another interesting feature connected to the clustering is the extension of the thematic classification to the terms belonging to the maintenance ontology which are related to the current searched keyword (only if the keyword belongs to the ontology) through usual ontological relations as: *is subclass of*, *is a part of*, *same as* etc.

Fig. 6: sub-clusters for the "Java" cluster (Lingo3G).

All these features are managed by the clustering agent which runs different behaviours that implement the aforementioned methodologies of clustering.

Some testing results are shown in figures 5 and 6: the clustering engine prototype ran on an index containing about 800 indexed documents (pdf and doc file). The system returns a collection of clusters for the keyword "message". Figure 5 reports a comparison between the Lingo3G results and the Carrot² ones. Figure 6 reports a detail regarding a sub-cluster generated for the "Java" cluster. Considering these results, the Lingo3G engine returns a high number of significant clusters. The cluster labels appear more accurate and sub-clusters are on topic with the parent cluster.

H. The user functionalities

As emerges from the requirements analysis, the user (regarding the document retrieval system) is able to access the knowledge base of the company represented by the collection of documents and other files, through different ways:

- Simple search: the user enters a keyword and the system returns a set of clusters (figure 7).
- Conceptual search: the user instantiates a concept coming from the document or the maintenance ontology. The system returns another set of clusters based on the instance of the concept.
- Directories: they are a sort of static clusters: typical topics related to the specific maintenance sector. They are set *a priori*.
- Documents chronology: the list of recent documents read by the user.
- New and updated documents: a list of the new documents and a report of the updated ones.

I. The administrator features

Regarding the document retrieval system, the administrator manages the entire documents corpus. He can add, update and modify documents, and he can manage the tags linked to every file. A particular function of the administrator is the possibility to create user profiles. A user's profile describes a particular category of end users. In a maintenance enterprise the workers are specialized in a maintenance sector, therefore for each worker profile, the administrator can select a particular maintenance ontology, a particular set of documents directories, a certain list of preferred cluster labels, and a customized Lingo3G/Carrot² setup. The administrator can link every user to a profile in order to improve the performances of the document retrieval system.

Fig. 7: a demo for the simple search function.

V. CONCLUSIONS

The E-Support project is currently under development. At now, the European partners are going to validate the final architecture proposal and the implementation of the system will start as soon as possible. Regarding the document retrieval system, both the relative agents and the architecture seems to be consolidated.

Concerning the performances of the clustering engine, we are stressing the two versions: Carrot² and Lingo3G. Obviously the two tools are not a magic box and the results are less efficient than the human classification. Nevertheless the results are encouraging and fairly positive: the test tools we have developed works on thousands of technical documents and scientific papers and they have become an utility for the everyday activity. We think that through an opportune calibration, and with the help of ontologies the developed tools supporting the front-end agent, will make it a very smart agent: a useful partner for each technician on the field.

REFERENCES

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

[2] 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.

[3] S. Landau, M. Leese, "Cluster Analysis", Oxford University Press US, 2001.

[4] A. Tanenbaum, "Computer Networks", Prentice Hall, 2002.

[5] D. Salomon, "Data Privacy and Security: Encryption and Information Hiding", Springer-Verlag, New York 2003.

[6] F. Dubois, B Mérialdo, "A framework for multi-agent multimedia indexing", *In Proceedings of workshop on intelligent multimedia information indexing*, August 1995 - Montreal, Canada.

[7] C. N. Linn, "A multi-agent system for cooperative document indexing and queryingin distributed networked environments", *In Proceedings of International Workshops on Parallel Processing*, 1999.

[8] J. Chen, S. Wolfe, S. Wragg, "A distributed multi-agent system for collaborative information management and sharing", *In Proceedings of the ninth international conference on Information and knowledge management***,** McLean, Virginia, United States, 2000.

[9] K. Hammouda and M. Kamel, "Collaborative Document Clustering*", In Proceedings of Conference on Data Mining (SDM06)*, pp. 453-463, Bethesda, Maryland, April 2006.

[10] C. J. van Rijsbergen, "Information Retrieval" Butterworth Ltd., second edition, London, 1979.

[11] L. Kaufman, P. J. Rousseeuw, "Finding Groups in Data: An Introduction to Cluster Analysis", New York: John Wiley & Sons, 1990.

[12] K. Wang., C. Xu, B. Liu, Clustering transactions using large items. *In Proceedings of International Conference on Information and Knowledge Management, CIKM'99*, Kansas City, Missouri, United States, 483–490, 1990.

[13] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy, S. W. Tu, "The Evolution of Protégé: An Environment for Knowledge-Based Systems Development", 2002.

[14] D. L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview", Feb. 2004, [Online document], Available at HTTP: http://www.w3.org/TR/owl-features/

[15] S. Schockaert, M. de Cock, C. Cornelis, E. E. Kerre, "Efficient Clustering with Fuzzy Ants", In Proceedings of Ant Colony Optimization and Swarm Intelligence (ANTS 2004), pages 342-349, 2004.

[16] S. Osinski, D. Weiss, "A Concept-Driven Algorithm for Clustering Search Results", IEEE Intelligent Systems, 3 (vol. 20), 2005, pp. 48-54.

[17] C.L.Ngo, H.D.Nguyen, A Tolerance Rough Set Approach to Clustering Web Search Results*, in Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2004)*, Italy, September 2004.

[18] J. Stefanowski, D. Weiss, "Carrot and Language Properties in Web Search Results Clustering", *In Proceedings of the First International Atlantic Web Intelligence Conference*, Madrid, Spain, 2003, pp. 240-249.

A Framework for Interacting Situated Agents in Virtual Environments

Giuseppe Vizzari[a], Giorgio Pizzi[a], Flávio Soares Corrêa da Silva[b]

***Abstract*—This paper presents a framework supporting the definition and implementation of virtual environment inhabited by interacting situated agents defined according to the Multilayered Multi-Agent Situated System model. The framework supports the specification and execution of visually rich 3D virtual environment endowed by the presence of mobile agents acting and interacting inside it according to a multi-agent model. The paper briefly describes the related works and possible application scenarios for the framework, then it introduces the multi-agent model underlying the framework and its basic architecture. Sample applications are also described so as to show the potential of the framework in executing models comprising several hundreds of agents producing an effective visualization of the generated dynamics.**

***Index Terms*— multi-agent systems, virtual environments, simulation, 3D visualization**

I. Introduction

The design and realization of virtual environments inhabited by social entities is a significant application of the conjoint results of various research areas in computer science and engineering. Virtual environments have been exploited in several ways, and in particular:

- to support computer mediated forms of human interaction, characterized by the introduction of Embodied Conversational Agents facilitating users' interactions [18] or supplying awareness information in a visually effective form [20];
- to realize operational laboratories for participatory design, supporting the effective visualization of various alternative design choices to the involved stakeholders [8][13][11];
- to provide effective instruments for the modeling, simulation and visualization of the dynamics of entities situated in a representation of an existing, planned or reconstructed environment or situation [10][19];
- for sake of entertainment, in movies, computer games or in online communities (see, e.g., Second Life[1]).

While all these applications are characterized by a strong requirement for realistic and effective visualization tools (and some of them require a thorough analysis of the system usability, due to the necessary accessibility by non-technically skilled users), they also call for expressive models supporting the specification of behaviours for the entities that inhabit these environments, as well as the interaction among them and with the environment itself. The fact that the overall performance of the system is essentially dependant on the single actions and interactions that are carried out by entities inhabiting the modeled environment leads to consider that the Multi-Agent Systems [12] approach is particularly suited to tackle the modeling issues that are posed by this scenario. This idea is also corroborated by the fact that most of the above introduced references actually describe systems based on this approach, and by specific experiences in applying MAS approaches to specific virtual environments applications such as computer games [16].

In this vein, the main aim of this paper is to show the current advancement of a long term project that provides the realization of a framework supporting the development of MAS based simulations based on the Multilayered Multi-Agent Situated System model provided with an effective form of 3D visualization. The main goal of the framework is to support a smooth transition from the definition of an MMASS based model of given situation (in terms of environment, relevant entities and their behaviours, expressed as individual actions interactions) to the realization of simulation systems characterized by an effective 3D user interface. One of the possible application areas of this kind of system is related to the modeling and simulation of crowds of pedestrians to support architectural design or urban planning [3][4]. In order to have information flowing appropriately from the formal model to design professionals (e.g. architects and urban planners), the MMASS-based simulator must be supported by adequate visualization and animation tools. Such supporting tools are the core issue of the present paper.

The paper breaks down as follows: the following section discusses related works in different application scenarios, while section III briefly introduces the MMASS model and its application to model pedestrians situated and moving in representations of physical spaces. Section IV discusses the architecture of the proposed framework, its main components and the tools supporting developers adopting it. Section V presents two sample applications aimed at showing the potential of the framework in executing models comprising several hundreds of agents producing an effective visualization of the generated dynamics. Conclusions and future developments end the paper.

[a]Complex Systems and Artificial Intelligence research center, University of Milano-Bicocca, via Bicocca degli Arcimboldi 8, 20126 Milano, {giuseppe.vizzari, giorgio.pizzi}@csai.disco.unimib.it

[b]Department of Computer Science, Institute of Mathematics and Statistics, Universidade de São Paulo, fcs@ime.usp.br

[1] http://secondlife.com

II. Related works and Application scenarios

The realization of virtual environments inhabited by autonomous entities and characterized by a realistic three-dimensional form of visualization was the goal of several projects, both commercial and academic, with different aims, features and available documentation. A complete and thorough description of the state of the art in this area, besides being extremely difficult to realize, is out of the scope of this paper; we will rather briefly report the survey activity that was carried out before starting the project and that motivated the effort related to the design and realization of the framework.

The main aim of the research effort is to realize an instrument that, on one hand, supports an effective form of visualization of a virtual environment and, on the other, allows the specification of the behaviours of the autonomous entities that inhabit it in terms of an expressive agent based model. To this purpose, we considered several possible supporting instruments, both commercial and open source, both providing a basic enabling technology and also some projects providing a support to the phases of modeling and definition of agents behaviours.

Some relevant representatives of the category of commercial instruments that can support the design and development of virtual environments are Quadstone Paramics[2] and Massive[3]. Paramics is a traffic micro-simulation software, that is able to generate realistic 3D visualizations of the simulated dynamics. Massive is instead an application specifically devoted to the generation of photorealistic animation of crowd-related visual effects (it was adopted for several films – e.g. to generate the battles of the Lord of the Ring – and commercials – e.g. to create the audience in a stadium). These instruments are generally extremely focused, very powerful, but their internal mechanisms are not well documented. In general they cannot be adapted to tackle application scenarios different from those they were originally conceived for; therefore they do not provide the degree of flexibility required by our project.

Some open source efforts were also considered, and two relevant representatives of the analyzed platforms are Mason[4] [15] and Breve[5] [14]. Mason is a discrete-event multi-agent simulation library, designed to be the foundation for custom-purpose Java simulations. It also includes an optional suite of visualization tools in 2D and 3D. However, this suite does not represent a proper support to the realization of a virtual environment, but rather a library for realizing simple 3D visualization of the simulated system. Breve, on the other hand, is a software package enabling the definition of 3D simulations of multi-agent systems and artificial life. It adopts a specific ad-hoc language (called "steve") for the specification of agents' behaviours. However, it is more focused on providing an abstract and extremely simplified 3D environment for specifying and testing multi-agent models and artificial life models, rather than supporting the realization of a detailed virtual environment.

The analyzed system that was closest to meet our requirements is Freewalk[6], an application that was adopted in some of the previously cited applications of virtual environments. It adopts a scenario specification language, called Q, that supports the specification of the environment (that must be a VRML model) and the behaviours of agents (through the notion of *scenario*). Freewalk, however, is not an open-source project and the Q language is not very focused on the interaction among the agents and the environment (that can be conceived as an element having an influence on their behaviour that goes beyond the fact that it provides obstacles to their movement). These considerations lead us to consider the possibility to adopt a basic enabling infrastructure for an effective visualization of system dynamics and an existing model - MMASS [1] - and platform for the specification of situated MAS supporting the definition of virtual environments. The MMASS model is in fact characterized by the fact that agents' environment is a first class element of the model, and it deeply influences agents' perceptions and actions, supporting forms of interactions that are particularly suited to represent the movement of pedestrians in physical spaces [2].

III. Multilayered multi-agent situated system model

This section will introduce the basic elements of the MMASS model and its application to represent and manage mechanisms of interaction between the environment and active autonomous entities that are useful for the specification of dynamic virtual environments. We will start discussing the elements of a single layered model, a Situated Cellular Agents model, then we will show how it can be applied to represent physical environments and active entities situated and moving in it. The last subsection will discuss how a multilayered structure can be adopted to enhance the model and support more autonomous forms of agents' behaviours.

A. Situated Cellular Agents

A system of Situated Cellular Agents can be denoted by the three-tuple $\langle Space,F,A\rangle$ where *Space* is a single layered environment where agents are situated, act autonomously and interact by means of reaction or through the propagation of fields belonging to the set *F*. Field based interaction is an indirect interaction mechanism that provides a modification of agents' environment that can be perceived by agents according to their context and state; a more thorough description of field based interaction can be found in [17], whereas the specific SCA field-based interaction model is discussed in [6]. Agents belong to *A*, a finite set of agents, each characterized by a type determining their state, perceptive capabilities and behavioural specification. The elements of this three tuple will now be formally described.

Space - The *Space* consists of a set *P* of sites arranged in a network (i.e. an undirected graph of sites). Each *site* $p \in P$ can contain at most one agent and is defined by $\langle a_p,F_p,P_p\rangle$

[2] http://paramics-online.com/
[3] http://www.massivesoftware.com/
[4] http://cs.gmu.edu/~eclab/projects/mason/
[5] http://www.spiderland.org/
[6] http://www.ai.soc.i.kyoto-u.ac.jp/freewalk/

where $a_p \in A \cup \{\perp\}$ is the agent situated in p ($a_p = \perp$ when no agent is situated in p, in other words p is empty); $F_p \subseteq F$ is the set of fields active in p ($F_p = \emptyset$ when no field is active in p); and $P_p \subset P$ is the set of sites adjacent to p. Edges connecting sites represent a constraint to the movement of agents situated in the environment and also on the diffusion of fields, which only propagate through these connections.

Fields - A field $f_\tau \in F$ that can be emitted by agents of type τ is denoted by the four-tuple $\langle W_\tau, Diffusion_\tau, Compare_\tau, Compose_\tau \rangle$ where:

- $W_\tau = S \times N$, where $S \subseteq \Sigma_\tau$ denotes the set of values that the field can assume; given $w_\tau \in W_\tau$, $w_\tau = \langle s_\tau, i_\tau \rangle$, where $s \in S$ represents information brought by the field (i.e. the field payload) and $i_\tau \in N$ represents its intensity.
- $Diffusion_\tau : P \times W_\tau \times P \rightarrow W_\tau$ is the diffusion function for field type τ; $Diffusion_\tau(p_s, w_\tau, p_d)$ computes the value of a field on a given destination site (p_d) taking into account in which site it was emitted (p_s) and with which initial value ($w_\tau \in W_\tau$).
- $Compare_\tau : W_\tau \times W_\tau \rightarrow \{True, False\}$ is the function that compares field values. It is used by the perceptive system of agents to evaluate if the value of a certain field type is such that it can be perceived.
- $Compose_\tau : (W_\tau)^+ \rightarrow W_\tau$ expresses how field values of the same type have to be combined in order to obtain the unique value of a field type at a given site.

Agent Types - The possibility to define different agent types introduces heterogeneity, in other words the chance to define different abilities and perceptive capabilities. Defining T the set of types, it is appropriate to partition the set of agents in disjoint subsets corresponding to different types. The set of agents can thus be defined as $A = \bigcup_{\tau \in T} A_\tau$ where $A_{\tau_i} \cap A_{\tau_j} = \emptyset$ for $i \neq j$. An agent type τ is defined by the three tuple $\langle \Sigma_\tau, Perception_\tau, Action_\tau \rangle$ where:

- Σ_τ defines the set of states that agents of type τ can assume;
- $Perception_\tau : \Sigma_\tau \rightarrow [N \times W_{f_1}] ... [N \times W_{f_{|F|}}]$ is a function associating to each agent state the vector of pairs representing respectively a receptiveness coefficient modulating the intensity of that kind of field and a sensitivity threshold represented by a specific field value; these functions represent the perceptive capabilities specification for that type of agent and their usage will be clarified in the description of agents and their behaviours. Formally, this vector of pairs is defined as

$$\left(c_\tau^1(s), t_\tau^1(s)\right), \left(c_\tau^2(s), t_\tau^2(s)\right), ..., \left(c_\tau^{|F|}(s), t_\tau^{|F|}(s)\right)$$

where for each i ($i = 1...|F|$), $c^i_\tau(s)$ and $t^i_\tau(s)$ express respectively a receptiveness coefficient to be applied to the field value f_i and the agent sensibility threshold to f_i in the given agent state s.

- $Actions_\tau$ denotes the set of actions that agents of type τ can perform, and will be described in the following.

Agents and their Behaviours - An agent $a \in A$ is defined by the three–tuple $\langle s, p, \tau \rangle$, where:

- $s \in \Sigma_\tau$ denotes the *agent state* and can assume one of the values specified by its type;
- $p \in P$ is the site of the *Space* where the agent is situated;
- τ is the *agent type*, which provides the allowed states, perceptive capabilities and behavioural specification for that type of agents.

The first two elements were previously introduced, we will now focus on $Action_\tau$, which is made up of a set of actions and an action selection strategy. Actions can be selected from a set of primitives which include *reaction* (synchronous interaction among adjacent agents), field *emission* (asynchronous interaction among at–a–distance agents through the field diffusion–perception–action mechanism), *trigger* (change of agent state as a consequence of a perceived event) and *transport* (agent movement across the space). The two interaction mechanisms provided by the SCA model (i.e. reaction and field–based interaction) are also described by the diagram in Figure 1. Every primitive will be now briefly described specifying preconditions and effects. It must be noted that an action selection strategy is invoked when the preconditions of more than one action are verified; several possible strategies can be defined, but in this context a non–deterministic choice among possible action was adopted.

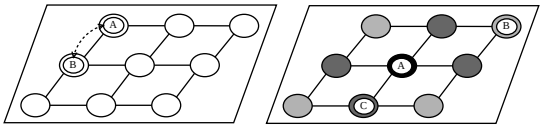


Figure 1 – A diagram showing the two interaction mechanisms provided by the SCA model: two reacting agents on the left, and a field emission on the right.

The behavior of Situated Cellular Agents is influenced by agents situated on adjacent positions and, according to their type and state agents are able to synchronously change their states. Synchronous interaction (i.e. reaction) is a two–step process. Reaction among a set of agents takes place through the execution of a protocol introduced in order to synchronize the set of autonomous agents. When an agent wants to react with the set of its adjacent agents since their types satisfy some required condition, it starts an *agreement* process whose output is the subset of its adjacent agents that have agreed to react. An agent agreement occurs when the agent is not involved in other actions or reactions and when its state is such that this specific reaction could take place. The agreement process is followed by the synchronous reaction of the set of agents that have agreed to it. Reaction of an agent a situated in site $p \in P$ can be specified as:

action:*reaction*($s, a_{p1}, a_{p2}, ..., a_{pn}, s'$)
condition:*state*(s),*position*(p),*agreed*($a_{p1}, a_{p2}, ..., a_{pn}$)
effect:*state*(s')

where *state*(s) and *agreed*($a_{p1}, a_{p2}, ..., a_{pn}$) are verified when the state of agent a is s and agents situated in sites $\{p_1, p_2, ..., p_n\} \subseteq P_p$ have previously agreed to undertake a synchronous reaction. The effect of a reaction is the synchronous change in state of the involved agents; in particular, agent a changes its state into s'.

Other possible actions are related to the indirect interaction mechanism, related to field emission and to the perception–deliberation–action mechanism. Agent emission can be defined as follows:

action:emit(s,f,p)
condition:state(s),position(p)
effect:added(f,p)

where *state*(*s*) and *position*(*p*) are verified when the agent state is *s* and int position is *p*. The effect of the emit action is a change in the active fields related to sites involved in the diffusion, according to $Diffusion_f$. One of the possible effects of an agent perception of a certain field f_i can be defined as

action:trigger(s,f_i,s')
condition:state(s),position(p),perceive(f_i)
effect:state(s')

where *perceive*(f_i) is verified when $f_i \in F_p$ and $Compare_\tau(c^i_\tau \cdot i_{f_i}, t^i_\tau)=true$ (in other words, field intensity modulated by a receptiveness coefficient exceeds the sensitivity threshold for that field). The coefficients c^i_τ and t^i_τ are those determined by the perception function for that type of agent in the state *s*. The effect of the trigger action is a change in agent's state according to the third parameter.

The last possible action for an agent causes a change in its position and can be specified as follows:

action:transport(p,f_i,q)
condition:position(p),empty(q),near(p,q),perceive(f_i)
effect:position(q),empty(p)

where *empty*(*q*) and *near*(*p*,*q*) are verified when $q \in P_p$ and $q=\langle \perp, F_q, P_q \rangle$ (*q* is adjacent to *p* and it does not contain agents). The effect of a transport action is thus to change the position of the related agent.

B. Modeling Crowds with SCAs

The basic idea underlying the application of the SCA model to represent environments and entities situated and moving in it is that this kind of movement can be generated by means of attraction and repulsion effects (as also suggested in [9]). These effects are generated by means of fields that can be emitted by specific point of the environment, and that can be perceived as attractive/repulsive or that can even be simply ignored by different types of moving entities in specific states. Also pedestrians themselves are able to emit fields and thus, in turn, they can generate attraction/repulsion effects, and what is called an 'active walker' model.

A thorough discussion of this modeling approach is out of the scope of this paper and it can be found in [2], we will now just give some indications of the main steps that must be followed to define a SCA model starting from an abstract description of a given scenario.

Definition of the **spatial infrastructure** of the environment – a SCA space can represent a discrete abstraction of a physical environment, in which a site corresponds to a portion of space that can be occupied by a pedestrian. For instance, a corridor and the rooms having a door on it could be discretized in 40cm^2 cells characterized by a Von Neumann adjacency.

Definition of **points of interest/reference** in the environment – specific spots of the environment can represent elements of interest, reference points or constraints (e.g. gateways, doorways) influencing pedestrian movements. These elements must be associated with immobile agents (e.g. door jambs) able to emit fields indicating the presence of the point of interest/reference to pedestrians. For instance, considering a corridor the exits should be associated to suitable fields able to guide agents towards them, but also possible doorways leading to rooms should be provided with agents emitting proper fields.

Definition of **mobile entities** of the environment (pedestrians) – the different types of mobile entities, agents representing pedestrians, can be now defined in terms of attitudes towards the movement in the environment (sort of states indicating how an agent interprets fields in choosing where to move). For instance, in the corridor example, agents in different states could be attracted by different exits of the corridors and thus could be attracted by the related field, ignoring fields generated by doors leading to internal rooms. This attitude could change according to internal decisions of the agent, or to an external event perceived by it. Of course, different agent types can have different attitudes; summarizing, different agent types can interpret fields in a different way, and agents of the same type, according to their state, can also react in different way to the perception of the same kind of signal.

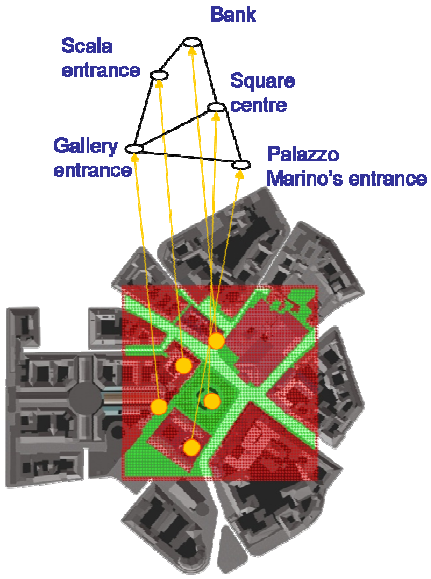


Figure 2 – A diagram illustrating a multilayered environment specification: the bottom layer is a fine grained discretization of Scala Square and the top layer represents its points of interest.

C. From a Single Layer to Multiple Layers

The previously introduced representation of the environment can be enhanced by introducing additional representations, for instance representing a different abstraction of the physical space related to the virtual environment. In particular, the different points of interest/reference might be represented on a graph whose links represent proximity or direct reachability relations among the related points, realizing a sort of *abstract map* of the environment. This layer might be interfaced to the previously introduced finer representation of the environment (i.e. the *physical layer*), and it could be the

effective source of fields generated by infrastructural elements, that are diffused to the physical layer by means of interfaces. A sample diagram illustrating this approach to the modeling of a physical environment is shown in Figure 2: the bottom layer is a fine grained discretization of Scala Square and the top layer represents its points of interest, that are associated with agents emitting a proper distinctive presence field.

The abstract map could also be (at least partly) owned by an agent, that could thus make decisions on what attitude towards movement should be selected according to its own goals and according to the current context by reasoning on/about the map, instead of following a predefined script. This kind of considerations do not only emphasize the usefulness of a multiple layered representation of the environment, but they also point out the possibility to enhance the current agents (that are characterized by a reactive architecture) by endowing them with proper forms of *deliberation*, towards a hybrid agent architecture. A complete definition of these deliberative elements of the situated agents is object of current and future works.

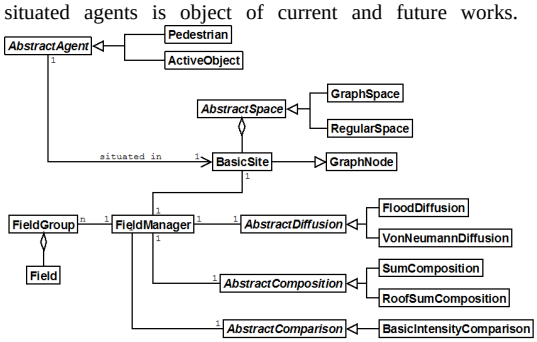


Figure 3 – Simplified class diagram of the part of the framework devoted to the realization of MMASS concepts and mechanisms.

IV. The Execution and Visualization Framework

As discussed in section II, the basic approach that was adopted for this project is to integrate an existing MAS modeling and development framework with an infrastructure supporting an effective form of 3D visualization of the dynamics generated by the model. In particular, to realize the second component we adopted Irrlicht[7], an open-source 3D engine and usable in C++ language. It is cross-platform and it provides a performance level that we considered suitable for our requirements. It provides a high level API that was adopted for several projects related to 3D and 2D applications like games or scientific visualizations. The MAS modeling and development framework we adopted is a C++ porting and relevant refactoring of the original MMASS framework [2], aimed at adapting it to the different programming language and also at optimizing some mechanisms such as commonly adopted field diffusion algorithms. The following subsections will discuss the basic elements of this C++ version of the MMASS framework and the infrastructure interfacing this module with the 3D visualization engine.

[7] http://irrlicht.sourceforge.net/

A. Supporting and Executing MMASS Models

The MMASS framework adopted for this project is essentially a library developed in C++ providing proper classes to realize notions and mechanisms related to the SCA and MMASS models. In particular, a simplified class diagram of the MMASS framework is shown in Figure 3. The lower part of the diagram is devoted to the environment, and it is built around the BasicSite class. The latter is essentially a graph node (i.e. it inherits from the GraphNode class) that is characterized by the association with a FieldManager. The latter provides the services devoted to field management (diffusion, composition and comparison, defined as abstract classes). An abstract space is essentially an aggregation of sites, whose concretizations define proper adjacency geometries (e.g. regular spaces characterized by a Von Neumann adjacency or possibly irregular graphs).

An abstract agent is necessarily situated in exactly one site. Concrete agents defined for this specific framework are active objects (that are used to define concrete points of interest/reference to be adopted in a virtual environment) and pedestrians (that are basic agents capable of moving in the environment). Actual pedestrians and mobile agents that a developer wants to include to the virtual environment must be defined as subclasses of Pedestrian, overriding the basic behavioural methods and specifically the *action* method.

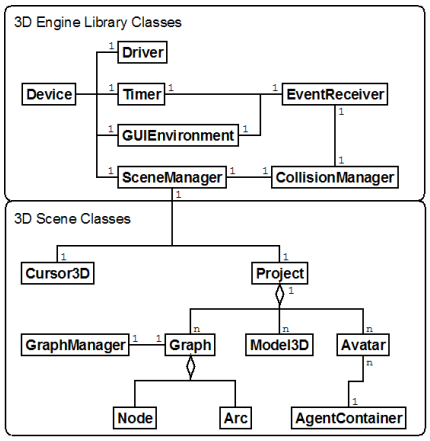


Figure 4 – Simplified class diagram of the part of the framework devoted to the management of the visualization of the dynamics generated by the model.

B. Integrating the Models with a Realtime 3D Engine

While the previous elements of the framework are devoted to the management of the behaviours of autonomous entities and of the environment in which they are situated, another relevant part of the described framework is devoted to the visualization of these dynamics. More than entering in the details of how the visualization library was employed in this specific context, we will now focus on how the visualization modules were integrated with the previously introduced MMASS framework in order to obtain indications on the scene that must be effectively visualized.

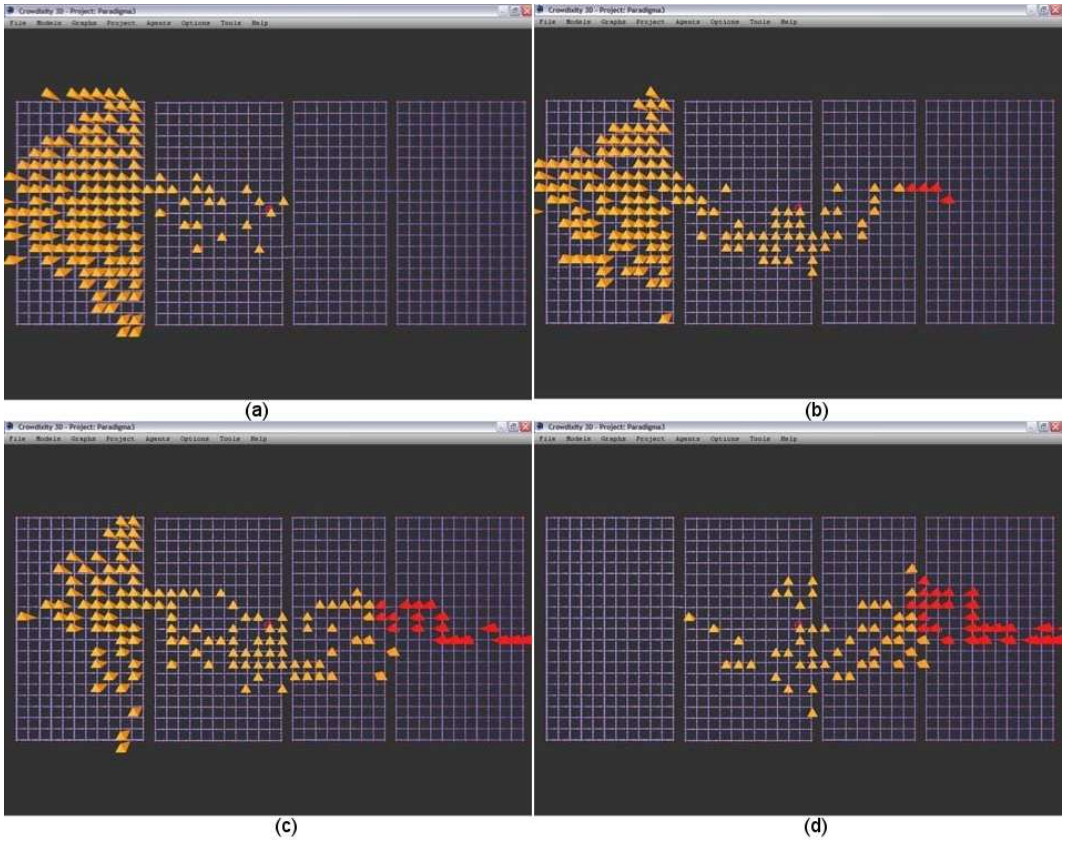


Figure 5 – Four screenshots of the first sample application, showing the movement of very simple agents from a starting room on the left, to an exit in the rightmost room.

Figure 4 shows a simplified class diagram of the main elements of the 3D Engine Library. The diagram also includes the main classes that are effectively in charge of inspecting the state of the MMASS environment and agents, and of providing the relevant information to the SceneManager that will translate it into a scene to be visualized. The *Project* class act as a container of the 3D models providing the graphical representation of the virtual environment (*Model3D* objects), as well as the graph related to the adopted discretization of this physical space (a *Graph* object visually representing the previously discussed *physical layer*). It also includes a set of *Avatar* objects, that are three dimensional representations of *Pedestrian* objects (introduced in the previous subsection).

The framework must be able to manage in a coordinated way the execution of the model defined for the specific virtual environment and the updating of its visualization. To manage this coordinated execution of different modules and procedures three main operative modes have been defined and are supported by the framework. The first two are characterized by the fact that agents are not provided with a thread of control of their own. A notion of *turn* is defined and agents are activated to execute one action per turn, in a *sequential* way or in a conceptually *parallel* way (as for a Cellular Automaton). In this case, respectively after each agent action or after a whole turn the scene manager can update the visualization. On the other hand, agents might be *associated with a thread of control* of their own and no particular fairness policy is enforced. The environment, and more precisely the sites of the MMASS space, is in charge of managing possible conflicts on the shared resource. However, in order to support a fluid visualization of the dynamics generated by the execution of the MAS, the *Pedestrian* object before executing an action must coordinate with the related *Avatar*: if the previous movement was still not visualized, the action is temporarily blocked until the visualization engine has updated the scene. It must be noted that in all the introduced activation modes the environment is in charge of a regulation function [7] limiting agents' autonomy for sake of managing the consistency of the overall model or to manage a proper form of visualization.

V. Sample Applications

The aim of this section is to present some sample applications to show how the framework supports the definition of MMASS models and the realization of an effective three dimensional visualization. The applications were also chosen to show the potential of the framework in terms of execution of a large number of agents. Tests were

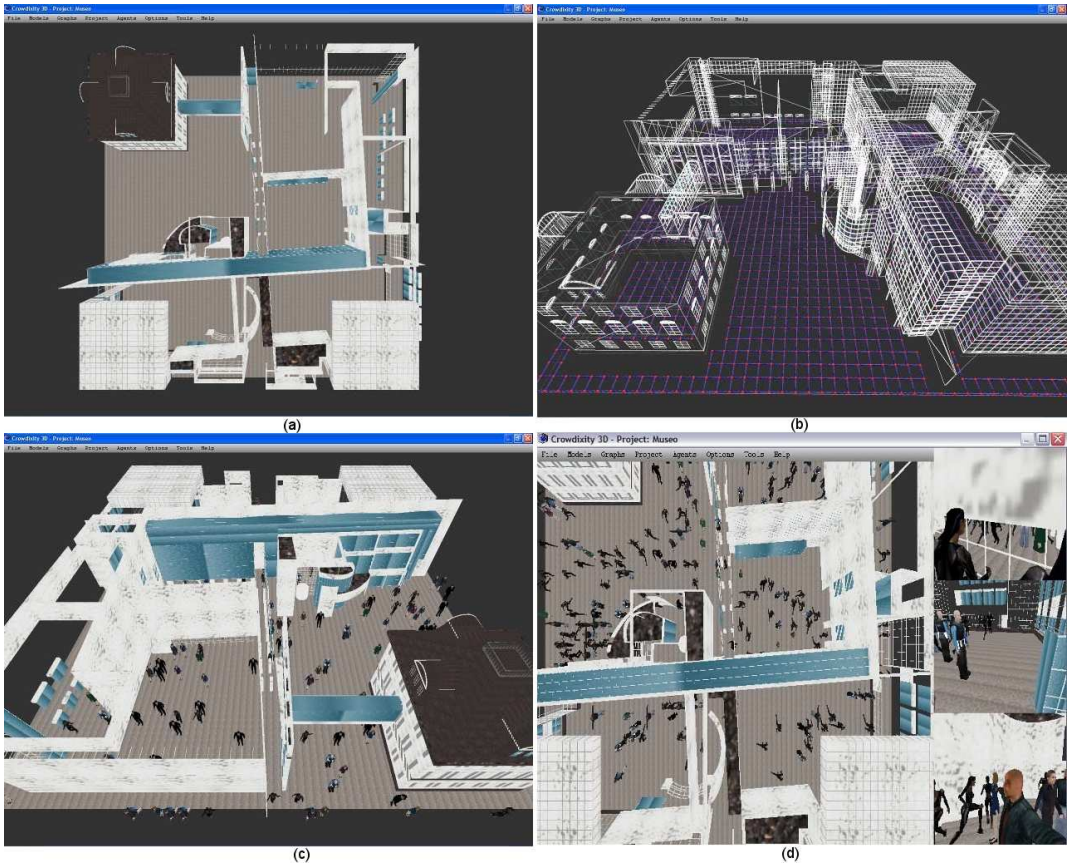


Figure 6 – Four screenshots of the virtual museum application, showing the structure of the environment - (a) and (b) – a perspective view of the evacuation and also a 'bird's eye' view of the environment coupled with three 'first-person' perspectives of agents – (c) and (d).

carried out on a notebook on which the Windows XP Professional operating system was installed; the notebook was provided with an Intel Pentium IV 2.4 GHz processor, with 320 MB RAM and an ATI Raedon IGP graphic card with 128 MB (shared system memory).

The first application is about the simulation of the evacuation of a section of a building, comprising several rooms connected by doors. In this specific scenario agents' behaviours are very simple, and only provide the movement towards specific exits. Agents reaching these exits are simply eliminated from the scenario; some screenshots of this example are shown in Figure 5. In this scenario the environment comprises a graph of around 1000 sites, connected by more than 3500 arcs; 150 agents are situated in the scenario and they are activated according to sequential activation strategy. The analytical results of the simulation are not relevant in this context, also because the agent models were not calibrated against real data; the simulation was executed and visualized with a number of frames per second (FPS) constantly above 60. The speed of the simulation was in fact actually limited to achieve a smooth form of visualization of the system dynamics.

The second example is about the movement of agents inside a virtual museum; the aim of the agents in this scenario is to move outside the buildings to gather in specific areas, as in case evacuation. In this case the environment comprises around 2000 sites (a gross discretization of the represented environment) with around 6000 arcs connecting them; 500 agents were randomly positioned inside buildings, and they were provided with a thread of control of their own. Both the environment and agents were characterized by a 3D visual model, with textures; some relevant screenshots of this sample application are shown in Figure 6. Once again, the analytical results of this simulation are not relevant, since the agent models were extremely simple and they were not calibrated against real data. The simulation was executed and visualized with a number of FPS constantly above 30.

We also executed a stress test on a different hardware configuration, to verify the scalability of the framework; the workstation was based on Windows XP Professional operating system, with an Intel Pentium Core 2 Duo 2.4 GHz, 2 GB RAM and a NVIDIA Quadro FX 3450 graphic card with 256 MB. The test environment was constituted by 11000 sites, connected by around 44000 arcs; 10000 agents, sequentially activated, were positioned in this environment. Their behaviour was simply to move towards the closest source of an 'exit' field; agents reaching the source were

removed from the environment. The system was able to execute and visualize the simulation with 22 FPS, when the structure of the environment was hidden (reducing the number of displayed triangles), and with 3 FPS when it was visualized.

VI. Conclusions and Future Developments

The paper has presented a framework supporting the definition and realization of virtual environment inhabited by interacting situated agents modeled according to the Multilayered Multi-Agent Situated System. The framework supports the specification and execution of visually rich 3D virtual environment characterized by the presence of situated agents acting and interacting inside it. The paper briefly introduced some relevant related works, then it presented the multi-agent model underlying the framework and its basic architecture (with specific reference to the integration of computational support to the formal model and the visualization components). Sample applications were also described in order to show the potential of the framework in executing models comprising several hundreds of agents producing an effective visualization of the generated dynamics.

Future works are aimed, on the one hand, at improving the set of support instruments, both methodological and computational, supporting for instance the definition of the spatial structure of the virtual environment. Some support instruments, such as a tools for a semi-automatic realization of discrete abstractions of an existing 3D model (e.g. a 3D Studio design of an architectural space) was already realized, but it must still undergo a thorough testing phase. Additional relevant future works are instead aimed at providing a more expressive modeling framework, as briefly discussed in Section III-C.

References

[1] S. Bandini, S. Manzoni, C. Simone. Heterogeneous Agents Situated in Heterogeneous Spaces. Applied Artificial Intelligence, 16(9-10):831–852, 2002.

[2] S. Bandini, M. L. Federici, S. Manzoni, G. Vizzari. Towards a methodology for SCA based crowd simulations. In: VI International Workshop Engineering Societies in the Agents' World, vol. 3963 of Lecture Notes in Artificial Intelligence, Springer-Verlag, pp. 203–220, 2006.

[3] S. Bandini, S. Manzoni, G. Vizzari. Situated Cellular Agents: a Model to Simulate Crowding Dynamics. IEICE - Transactions on Information and Systems: Special Section on Cellular Automata, Vol.E87-D(3):669-676, 2004.

[4] S. Bandini, S. Manzoni, G. Vizzari. Multi Agent Approach to Localization Problems: the Case of Multilayered Multi Agent Situated System. Web Intelligence and Agent Systems, IOS Press, 2(3):155-166, 2004.

[5] S. Bandini, S. Manzoni, G. Vizzari. Towards a platform for Multilayered Multi Agent Situated System based simulations: focusing on field diffusion. Applied Artificial Intelligence, Taylor & Francis, 20(4-5):327-351, 2006.

[6] S. Bandini, G. Mauri, G. Vizzari. Supporting Action-At-A-Distance in Situated Cellular Agents. Fundamenta Informaticae, 69(3):251-271, 2006.

[7] S. Bandini, G. Vizzari. Regulation Function of the Environment in Agent-Based Simulation. Environments for Multi-Agent Systems III, Third International Workshop, E4MAS 2006, vol. 4389 of Lecture Notes in Computer Science, Springer-Verlag, pp. 157-169, 2007.

[8] M. Batty, A. Hudson-Smith. Urban Simulacra: From Real to Virtual Cities, Back and Beyond, *Architectural Design*, 75 (6):42-47, 2005.

[9] M. Batty. Agent-based pedestrian modeling. In *Advanced Spatial Analysis: The CASA Book of GIS*, pp. 81-106, 2003.

[10] J. Dijkstra, H. P. J. Timmermans. Towards a multi-agent model for visualizing simulated user behavior to support the assessment of design performance. Automation in Construction 11:135-145, Elsevier, 2002.

[11] J. Dijkstra, J. Van Leeuwen, H. J. P. Timmermans. Evaluating Design Alternatives Using Conjoint Experiments in Virtual Reality. Environment and Planning B 30(3):357–367, 2003.

[12] J. Ferber. Multi-Agent Systems. Addison-Wesley, 1999.

[13] T. Ishida, Y. Nakajima, Y. Murakami, H. Nakanishi. Augmented Experiment: Participatory Design with Multiagent Simulation. IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 1341-1346, 2007.

[14] J. Klein. Breve: a 3D simulation environment for the simulation of decentralized systems and artificial life. In Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems. The MIT Press, pp. 329–334, 2002. http://www.spiderland.org/breve/breve.pdf.

[15] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, G. Balan. MASON: A Multi-Agent Simulation Environment. In Simulation 81(7) :517-527, 2005.

[16] M. Mamei, F. Zambonelli. Motion Coordination in the Quake 3 Arena Environment: A Field-Based Approach. Environments for Multi-Agent Systems, First International Workshop, E4MAS 2004, vol. 3374 of Lecture Notes in Computer Science, Springer-Verlag, pp. 264-278, 2005.

[17] M. Mamei, F. Zambonelli. Field-Based Coordination for Pervasive Multiagent Systems, Springer-Verlag, 2006.

[18] H. Nakanishi, S. Nakazawa, T. Ishida, K. Takanashi, K. Isbister. Can Software Agents Influence Human Relations? - Balance Theory in Agent-mediated Communities. *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, ACM press, pp. 717-724, 2003.

[19] P. Nugues, S. Dupuy, A. Egges: Information Extraction to Generate Visual Simulations of Car Accidents from Written Descriptions. In: Computational Science and Its Applications - ICCSA 2003, vol. 2667 of Lecture Notes in Computer Science, Springer-Verlag, pp. 31-40, 2003.

[20] F. Nunnari, C. Simone. Perceiving awareness information through 3D representations. Proceedings of the working conference on Advanced Visual Interfaces, AVI 2004, ACM Press, pp. 443-446, 2004.

[21] G. Papagiannakis, S. Schertenleib, B. O'Kennedy, M. Arevalo-Poizat, N. Magnenat-Thalmann, A. J. Stoddart, D. Thalmann: Mixing virtual and real scenes in the site of ancient Pompeii. Journal of Visualization and Computer Animation 16(1):11-24, 2005.

Expectations driven approach for Situated, Goal-directed Agents

Michele Piunti
Institute of Cognitive Sciences
and Technologies, ISTC - CNR
Università degli studi di Bologna
DEIS - Bologna, Italy
Email: michele.piunti@istc.cnr.it

Cristiano Castelfranchi
Institute of Cognitive Sciences
and Technologies, ISTC
CNR - Roma, Italy
Email: cristiano.castelfranchi@istc.cnr.it

Rino Falcone
Institute of Cognitive Sciences
and Technologies, ISTC
CNR - Roma, Italy
Email: rino.falcone@istc.cnr.it

***Abstract*— Situated agents engaged in open systems continually face with external events requiring adequate services and behavioral responses. In these conditions agents should be able to improve their adaptivity over time, namely 1) to deal with and anticipate relevant changes and critical situations, 2) to temporally define relative priorities between goals varying their importance over time and 3) to use informational feedback to learn from experience and become better at achieving their goals. This work provides an insight to model goal directed agents with these adaptive and anticipatory abilities, based on context awareness and growing experience at achieving their activities. We propose an approach by which affective states are placed as an integrated control mechanism in order to tight different processes and computational modules underlying reasoning.**

I. Introduction

A great variety of goal-directed models of agency, focused at various level on representational, deliberative and action selection mechanisms, have been developed over the last two decades to design adaptive, autonomous and socially interactive agents. We here refer to the goal-directed model of agency, where agents are intended as autonomous, resource bounded entities that attempt to arbitrate between several goals interacting in dynamic, partially observable environments. Typically goal-directed agents are engaged in deliberation to select to which of the concurrent goals devote their resources. Reflecting the original model proposed at the end of 80s [1], traditional deliberative systems process their information reacting in a procedural way: they choose in a repertoire the action to execute according to filtering of conditions (matching rules based on belief formulae, priority hierarchies etc.), whilst the available plan library is handcrafted at design time. An agent can change his environment through a given set of actions and plan operators in order to reach a desired state of affairs from the current state. According to the wide adopted *Belief Desire Intentions* (BDI) model of agency [2], [3], an action is performed when the agent has the intention to achieve a given goal, and the beliefs indicating that the action helps in achieving that goal. Whilst it is theoretically possibly to specify an effective model of behavior in deterministic or probabilistic environments, it is very troublesome to deal in practice with real conditions where the agent has to face with resource limitations (time, computation, memory), partial information (hidden state due to environments constraints, noise, sensor weakness), time-varying goal importance, changing probability distributions and non-stationary, non-probabilistic, non deterministic environments. This negatively reflects upon agent design process and requires the designer to fully understand the dynamics and the complex cases that an agent may face with, and then to implement solutions from scratch for each critical situation may be encountered [4], [5].

The computational model presented here is an attempt to bridge the existing gap between goal directed model of agency and more situated models used in artificial life. Our challenge is to endow deliberative agents with capabilities to operate in *open systems* where dynamism, partial knowledge and unpredictability of future events exact agents to quickly *anticipate* decisions, facing with uncertainty and unexpected events. As a general foundation for adaptiveness in artificial entities [5], there is the need for a proactive adaptation of the internal model over time, in order to exploit informational feedbacks, to learn from experiences and become better at achieving goals. To deal with adaptiveness and anticipation, we refer on a cognitive models of *expectations* as causal precursor of basic *emotions* [6], [7], as far as on recent convergent studies that are pointing out the enhancement of adaptiveness in introducing emotions in reasoning [8], [9], [10], [11].

On the basis of the formal definition of a series of affective states, we provide a description of their functional role, assessing a series of behavioral and mental changes that emotions may induce within agent's internal processes. Our approach is intended at: *1)* Exerting a top-down modulation of emotional reasoning as a result of deliberative process and adaptive responses to relevant events and *2)* Integrating adaptiveness in decision making along with expectations and their causal relation with appraisal/evaluation of events.

The remind of this work is organized as follows: in section 2 we present a model for active and surprise driven perception and belief update, in section 3 we describe an expectation based approach for decisions affected by emotions, in section 4 we present a model for situated reasoning enabling appraisal and coping strategies to unexpected events, section 5 shows as a long term effect of metal states can be integrated in decisions, in section 6 we conclude presenting some related works and

providing a final discussion.

II. From expectations and active perception to surprise

Among all the activities an agent may perform during his tasks, we identify two main typologies:

1. Purposive behavior, supported by practical reasoning, is aimed at achieving terminal goals through the use of practical actions and plans. A goal directed agent should use informational feedback to learn from experience and become better at achieving goals.

2. Situated behavior, supporting coping strategies, is aimed at recruiting resources when some *unexpected* event require services. A situated agent should re-define relative priorities between goals varying their importance over time.

Therefore we here identify two *integrated* levels of reasoning, involving cognitive, slow deliberative processes as well as fast automatic and associative ones. Both levels integrate various mechanisms required to manage expectations, used either to assess alternatives and choices, and to direct cognitive resources towards anticipated events. In more detail, we distinguish between *high level, active expectations* and *background, passive expectations*.

At an high level we deal with *explicit expectations* modulating decisions and thus goal deliberation: we include in the reasoning process a quantitative influence on the terms given by the expected utilities used for arbitrating between alternative courses of actions [9]. As we will show later, these influences can be adjusted on the basis of affective appraisal and experiences and allow agent to learn from experiences.

Besides, in order to enhance agent's adaptiveness, we model situated, *background expectations* to elicit a loss of control of certain deliberative processes and to reconsider the course of agent activities. Typically, these particular kind of reasoning is not part of the specification of an agent in his purposive behavior, rather can be let to *emerge* as a result of the interactions in his environments. In [12], Lazarus indicated this particular process as a reflexive re-assessment of the internal state under context awareness, rather than an explicitly deliberated process.

Each of the aforementioned activities should be supported by an adequate perceptive process. Traditional agent architectures use simplified approaches for perception (i.e. based on hard coded rules for controlling sensor apparatus) thus resulting in monolithic and domain specific mechanisms. On the contrary, many evidences pointed out the need for a more abstract, pro-active and goal-driven model of perception [13], [14]. Unfortunetely, active perception is computationally expensive for resource bounded agents. According to the principle of minimal rationality [4], perception filtering should not overload agent processes with continuous belief revision. This exacts a less demanding strategy of reasoning from precepts. To this end, a lighter peripheral filtering can be used only when *relevant* information comes, even while the agent is not actively searching for it. The agent can thus ignore all incoming inputs which are not relevant with respect to the current task and only consider those information which are relevant [15]. In so doing, we identify two different perceptive strategies that an agent may adopt.

1. Purposive tests relate to *active perception* and used for belief update and control of purposive behavior. They are aimed at signalling discrepancies between what is perceived and what is expected in terms of high level, active expectation upon terminal goals achievement. In this case, the agent actively observes the fulfillment of his purposive actions trying to confirm the validity of related expectations.

2. Situated tests relate to *background perception*. In order to gather information of the near contexts, situated agents need the ability to deal with unexpected events, namely events that are not directly and actively under the focus of attention, but that can strongly influence its course.

Many emotions are in tight relation with perception and expectations. For example, surprise is conceived as an expectation based phenomenon: it has been given in terms of a felt signal which provokes an immediate reaction/response of alert and arousal due to an inconsistency (discrepancy, mismatch, non-assimilation, lack of integration) between an incoming input and prior expectations [16], [17]. Surprise has been related to many effects aimed at solving the inconsistency and at preventing possible dangers. Surprise strongly affects attentive processes [18], while [15] show operational advantages of a expectation-driven perception filtering for belief update. We here refer to a particular form of surprise, due to an experienced *mismatch* between a perceived fact and a scrutinized expectation. A specific kind of mismatch-based surprise can be associated to each type of expectation and to each kind of attentive processes. We here identify two different kind of surprise: the former is based on synchronous mismatches appraised upon action completion (i.e. on goal achievement) the latter when passively expected events occurs asynchronously *during* practical behavior (i.e. action execution)[1]. Bringing perception and expectations at the same level of representation, a computational system can detect and quantitatively evaluate the mismatches [18], [17], [15]. In the next sections we deal with expectations and surprise-related behavior influencing goal directed agents at different levels of reasoning. On an higher level, expectations help to take decisions between alternative courses of actions. As for situated cognition, surprise based on passive expectations can enhance context awareness and elicit responses to recruit operative resources to respond *in advance* to changes.

III. Expectations and decision making

Current BDI oriented implementations provide mechanisms for deliberation (goal selection and intention filtering) but don't share common strategies for decision making. Our model builds on top of a BDI engine an *expectation-driven* decision making process, thus combining deliberative, logical aspects of a BDI model with more quantitative, numerical aspects of

[1]Deeper forms of surprise rely on deducibility [18] plausibility [17] of the incoming percept inferred on the basis of the prior knowledge.

decision theory. We identify slower forms of reasoning with high level cognition, decisions between alternative courses of actions used by agents to arbitrate goal selection. To allow agents to take decisions based on the related expectations we model a long term memory entertaining endogenous anticipatory representations. Each (sub)goal is given along with the representation of its activation formulae (typically first-order belief formulae [19]) and a network of inhibition links (indicating if a given goal has the priority on another goal and under which conditions this priority is applicable [20]). Filtering can be managed through a dynamic arbitration network, providing disambiguation between the precondition rules and the relative dependencies (inhibition links) between the concurrent goals. A deliberation engine reacts to changes in the belief base (i.e. internal events thrown by a belief update) and uses the current internal state to filter out enabling conditions for arbitrating the goal adoption.

As for the decision theoretic paradigm of 'rationality' [21], [22], an artificial agent may act in order to maximize the expected utility, given multiplying utilities (desirability) and probabilities (likelihood). In our model this strategy is delivered at a meta-level reasoning, typically when the agent has to select between alternatives to achieve mutually exclusive sub-goals.

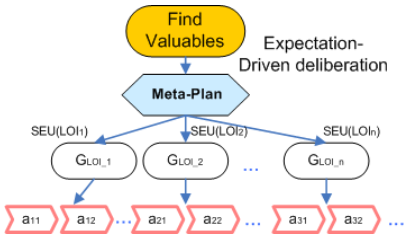


Fig. 1. Given a terminal goal, *Expectation Driven Deliberation* compares Subjective Expected Utilities to choose the most promising course of actions

Imagine an agent being engaged in a foraging task. In normal conditions, the terminal goal is to look for valuables moving to a series of rooms towards some Location of Interest(LOI). Expectation-driven deliberation allows the agent to decide on which LOI to look for considering how the various alternatives are 'promising' (Fig. 1). The scrutinized expectations are built upon two independent quantitative dimensions: *Belief strength*, as a degree of subjective certainty placed in terms of likelihood (the agent is more or less certain about their content) and *Goal value*, a subjective importance strictly dependent on desirability of the goal state and the related motivating forces, but also on context conditions and mental attitudes [6], [7]. Given this, *Subjective Expected Utility* (SEU) can be placed as:

$$SEU(G_i) = \sum_{a_j \epsilon Plan(G_i)} U(O_{G_i}) P(O_{a_j}|a_j) \qquad (1)$$

where G_i is the i^{th} goal to adopt between candidates, O_{G_i} is its related outcome, $U(O_{G_i})$ is the subjective utility of that outcome, a_j the j^{th} action of the plan triggered by G_i and $P(O_{a_j}|a_j)$ is the probability of that outcome, given that the j^{th} action of the plan will have the proper O_{a_j} outcome.

Utilities are coupled to rewards obtained upon goal completion and quantitatively assessed in relation to past experiences. $U(O_{G_i})$ is calculated according to the extent to which an intention (i.e. a given sequence of actions) has fulfilled a subjective desire O_{G_i}[2]. This makes it possible to endow expectations with their *valence*: expectations can be considered *positive* (or *negative*) according to their contribution (or determent) to the ongoing intentions and mental states (e.g. Goals, Beliefs).

Likelihood are subjectively assessed as predictions through a forward model mechanism. In the actual implementation, we are testing different mechanisms for unsupervised learning to determine conditional probabilities of future events, given a sufficiently wide open knowledge base (i.e. EM algorithms for Bayesian networks [23]). Feedback of mismatches between expectations and experienced outcomes are then used to adjust either utilities and predictions. In so doing, even in the same environment, different agents build different subjective models based on their past experiences, thus resulting with different epistemic and motivational states.

A. Emotions modulating high level expectations

Among the consequences of scrutinizing an expectation there is the increment of epistemic activities, aimed at acquiring information from environment *to know whether the expectation can be validated or disconfirmed* [24], [25], [7]. As mentioned, this mechanism is at the basis of any mismatch-based surprise. The idea behind the modulation of expectations with emotions is that an agent can affect the desirability of an outcome by introducing an additional motivation based on an anticipatory feeling[3]: given an active expectation upon a possible reward, agents can appraise their experiences comparing the expected utility and the effective achieved reward. Six cases of mismatch are possible:

1) *Positive increase ($S+$)*: the achieved reward is stronger than the one expected. Can be related to **excitement**.
2) *Negative increase ($S-$)*: the punishment is greater than expected. Can be related to **distress** or **strong disappointment**.
3) *Positive reduction ($\$+$)*: the agent achieve less reward than the one expected. Can be related to **disappointment**.
4) *Negative reduction ($\$-$)*: less punishment than expected. Can be related to **relief**.
5) *No Surprise (NS)*: goal reward matches the expectation and is exactly the one expected.

[2]In more details, for any given goal G_i, the agent stores achieved rewards and calculates $U(O_{G_i})$ by inferring the next value based on an average, or on a linear projection.

[3]In the field of decision theory, a similar solution has been formally proposed by Gmytrasiewicz and Lisetti [9], while Busemeyer et al. [26] formalized how needs change over time under the pressure of external stimulation and internal deprivation

6) *Surprise due to ignorance (IS)*: the reward is not deducible from prior knowledge due to lack of experiences.

Once appraised, agent can use these feelings to give more or less preference to a certain alternative. The agent may introduce an *affective bias* providing an intrinsic anticipatory effect (the experienced surprise enhances the importance of a certain goal, hence the agent believe to obtain more value from its achievement). We define an *Affective Expected Utilities* (AEU) in terms of:

$$AEU(G_i) = \sum_{a_j \epsilon Plan(G_i)} [A_b \times U(O_{G_i})] \times P(O_{a_j}|a_j) \quad (2)$$

where, respect to the SEU given in (1), A_b represents a qualitative and a quantitative appraisal of the experienced mismatch. It introduces an additional, quantitative reinforcement into the deliberation process and further modulates the expected utility in affective terms. The positive increase ($S+$) and the negative reduction ($\$-$) of the monitored signal give a positive indication about the progression of the goal value. Hence, when associated with a specific decision, they present a *positive feeling* towards the related outcome. Contrarily, the negative increase ($S-$) and the positive reduction ($\$+$) cause the agent to experience a *negative feeling* towards that choice, thus inhibiting its value. This is implemented by reinforcing the utility of a choice with an additional factor, in case of a positive feeling, and diminishing it in the case of a negative feeling. A_b is positive for positive feelings and negative for negative ones:

$$A_b(G_i) = \begin{cases} 0.0 & \text{if } E_s(G_i) \text{ is in } \{NS, IS\} \\ (\gamma_+) * E_r(G_i) & \text{if } E_s(G_i) \text{ is a pos. feeling } \in \{S+, \$-\} \\ (\gamma_-) * E_r(G_i) & \text{if } E_s(G_i) \text{ is a neg. feeling } \in \{S-, \$+\} \end{cases}$$

where $E_s(G_i)$ comes from the last appraised mismatch on G_i's reward, $E_r(G_i)$ is the distance between expected reward and sensed reward, γ_+ and γ_- are discount factors (with $\gamma_+ \ll \gamma_-$).

IV. Background expectations and situated reasoning

A central claim of appraisal theory is that emotions are associated with subjective judgments for the significance of external events (e.g. was the event expected in terms of prior beliefs? is the event congruent with adopted goals? is there the power to alter the consequences of the event?). As shown above, background appraisal allows particular contexts and events to be recognized in order to activate background (tacit, passive) expectations. Agent's *situated perception* envisages causal interpretation of situated events by filtering their features into percepts. The events can then be compared with agent goals and endogenously valued as *positive* (indicating that some event establishes the preconditions for achieving goals or create a new opportunity) or *negative* (some event represent a threat or thwart agent current goals). The idea behind the situated control is that clusters of different coping responses can be arranged around how a situation is appraised. Adopting the model of situated reasoning, coping strategies elicited by different kind of *surprise* can be modeled as a *momentary interruption* of deliberative and practical reasoning processes, e.g. diverting attention to past episodes or focusing sensors and effectors to a restricted area.

A. Mental States and affective control

Stored in an associative memory, noticeable events can be exploited to infer local environment features and activate a passive form of expectations. In so doing, an event that is supposed to thwart an active goal is assessed as a potential undesirable (negative) item, hence the agent reconsider his intentions trying to adopt an alternative action to avoid a threatful state. Otherwise, in case of a positive event, the agent can reconsider his intention in order to exploit an opportunity, or to maintain a desirable state. At any instant of time, agent's situated perception filters the world and store surprising events adding items to a Situated Associative Memory (`SAM`). In this case surprise is referred to passive expectations and arises when the agent relieves a mismatch from an unexpected input coming from the the situated perceptual component. For each of these surprising events, the agent stores in the `SAM` a perceptual report. Reports contain descriptions of a defined set of situated properties: they have a symbolic representation including time-stamp, positive or negative valence of the originating event, location where the event has been detected and other specialized fields[4]:

```
evItem  {  valence: enum value="pos/neg"
           time-stamp: class="Time"
           location: class="Location"
           helps: class="Goal"
           thwarts: class="Goal"
        }
```

Once events are translated to their symbolic representation and stored in the `SAM`, they can be manipulated as percepts. Items have a propositional content but a different nature respect to the beliefs[5]. They are designed to provide both *episodic* and *semantic* contents. The `SAM` is episodic and allows the agent to cache a raw description of situated events, thus enabling the reasoning process to exploit local environment features. The percepts are exploited as a 'fast' source of information to adapt the behavior in the near future and anticipate world changes. The presence of a time stamp for each item ensures to relate each percept to a given time and allow the content of the `SAM` to remain ordered at least for a given field[6]. Besides, the memory is semantic and provides a fast belief base to be handled to infer local environments features. The intuition behind the mechanisms is provided by the well known principle of spatial and temporal locality,

[4]In the case of the foraging scenario, also described in [27], we distinguished negative events as harmful collisions, fire threats, and positive events as food objects, valuables and LOI discovering.

[5]Notice that situated percepts may hold to deceitful appearances [13] including false positive or negative items.

[6]Given the items $I1, I2$ and $I3$, by selecting the time stamps we define an ordered set upon SAM: 1) Either $I1 \preccurlyeq I2$ or $I2 \preccurlyeq I1$ (completeness); 2) If $I1 \preccurlyeq I2$, then $I2 \not\preccurlyeq I1$ (consistency); 3) If $I1 \preccurlyeq I2$ and $I2 \preccurlyeq I3$, then $I1 \preccurlyeq I3$ (transitivity).

according to which one may assess that recently cached items of a certain class are likely to be retrieved in the near future. The amount of item locally present in the SAM can be used as an indication to infer passive expectations about the local context[7].

Transition Function and Information Fusion. SAM's content is constantly monitored by an appraisal process in order balance the presence of items and thus decide which is the MS to adopt. Passing from one state another depends on how the events are relieved and appraised in real time. This process can be described through a push down automaton [27], [9]. Generally the agent supervises the buffers (through a background process) by balancing their registered contents: prevalence of negative items leads to passive expectation of undesirable states (i.e. contingencies, risks), hence to cautious attitudes, while positive events lead to positive expectations (i.e. opportunities) and excitation (Fig. 2). In more details, the current state is inferred by the previous state and the perceived input with a transition function $MsTrans : MS \times IN^{\star} \longrightarrow MS$, where MS is the set of definite mental states and $IN^{\star}$ the input events stored in the (possibly empty) SAM. $MsTrans$ realize an *information fusion* within the symbolic items. Notice that the presence of items of different nature may elicit inconsistencies to be resolved (i.e. presence of elements of different meaning as, for instance, interleaved sequences of positive and negative events). To address this problem $MsTrans$ uses a set of rules for combining and aggregating the items of the same type and circumvent the inconsistencies on the basis of the temporal sequencing given by the time stamps. As suggested by [29], the rules used to govern the fusion can be composed of meta-level and domain specific information. For instance, a simple rule of *balancing* may assert to aggregate the items of a given typology, in order to circumvent the set of lower cardinality and to take into account only the information related to the bigger aggregate. By balancing the presence of items of a given type, the appraisal process suitably distinguishes between positive and negative expectations. A similar approach was used in [27], where two buffers are handled to store positive and negative events and the current sate is let to emerge on the basis of the comparison of buffers sizes (Fig. 2). To prevent the agent to switch to an inconsistent state, the transition function is built to take into account a certain grade of *inertia*, thus providing more robustness against occasional events, false positive or negative items (i.e. due to noise or sensor faults etc.).

B. Functional description

On the basis of a principle of *Analogy*, one may asses that an agent can predict with reasonable accuracy what actions and changes to perform in the near future based on his recent experiences and on the appraisal of local events. In so doing, responses and coping strategies given to a given set of related events can be classified an re-used in analogous situations. Hence, library of coping strategies, action alternatives and resource allocation policies can be clustered within a discrete set of frames used as control states. Effects of coping can

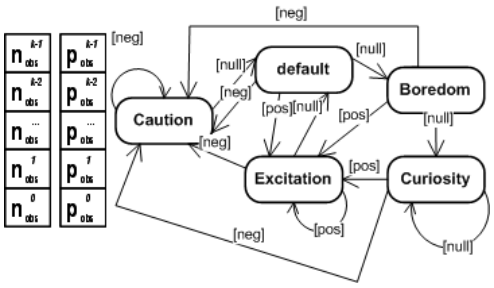


Fig. 2. Controller for Mental States: appraised positive (*p*) and negative (*n*) events are fed to a transition function in order to shift from different mental states.

be modeled in different temporal scale, from immediate and short term reactions, to most persistent long term effects. Given in functional terms, coping strategies includes emotional responses to overturn (in the case of negative emotions) or trigger (in the case of positive ones) control signals to be signalled to the reasoning process. Part of the effect of these signals are conative: mental states, as particular aggregates of control strategies, are modeled to activate particular goals. Besides, MSs are suitable control mechanisms for intention reconsideration. They embed a particular kind of goal activation, bypassing the underlying deliberation processes normally used for practical reasoning. For example, on the short term a MS may attempt to resign the agent to a threat by signalling to the deliberative engine to abandon a goal (thus a related intention) that is becoming inconsistent with the actual belief base or the actual environment state. On the contrary, positive events may elicit goal activation to exploit new opportunities. Furthermore, each MS adopt a *context dependent* configuration of resources (i.e. vision, speed, perception rate, belief update).

Becoming aware of his context, the agent can dynamically adapt his *control frame* in order to reduce performance payoffs and avoid wasting resources for useless activities. Control frames are characterized by the following tuple of dynamic values: $Cf = \langle En, r, Sr, s, G_s \rangle$, *En* indicating the current amount of energy, *r* the range of vision where sensors can retrieve data, *Sr* the situated perception filtering rate, *s* the instant speed and G_s the situated goal to be activated in order to pro-actively respond to the events to cope[8]. Each frame defines the roles that the related MSs play in situated adaptation to contexts and environment dynamism.

Imagine, in the foraging task presented in section III, that the environment presents some threats for agent activities (i.e. the fires, adversary agents etc.). Once the agent has deliberated the best expected location to explore, through the evaluation of

[7]A Similar approach was used by Schank [28], where expectations are generated on the basis of the agent's knowledge encoded in scripts and frames.

[8]We assume that agents spend energy according to a combination of the previous resource costs (e.g. the higher the speed and perception-rate, the higher the spent energy).

MS	Moods	γ_{MS}	Resources		
			r	S_r	S
Default	Exploitation	1.0	.33	.33	.33
Excitement	Reinforcement	1.3	.275	.275	.45
Caution	Prudence	0.5	.45	.45	.10
Boredom	Exploitation	1.0	.33	.33	.33
Curiosity	Exploration	1.0	.45	.10	.45

TABLE I

MENTAL STATES ELICIT THE ADOPTION OF CONTROL FRAMES FOR MOODS, CONFIDENCES AND RESOURCE ALLOCATION POLICIES

the related AEUs, it may happen he registers a close series of harmful (unexpected) events, i.e. fire collisions (Fig. 3.a). This elicit the negative expectation that the agent is approaching to a dangerous area and thus induce him to pass to a **Cautious state** (Fig. 3.b). This negative, background expectation causes the agent to adopt a new control frame, re-allocating his resources to cope harmful circumstances (Tab.I). Cautiousness causes changes both in the long and the short term: firstly it induces arousing by modulating attentive resources (i.e. enhancing *Sr*, looking ahead and augmenting *r* and reducing *s*, see Tab.I). A risk avoidance goal G_s interrupts the ongoing practical action to escape from threats and accordingly the agents arranges activities to better check the situation. On the long term, cautiousness brings to a watchful mood, by reducing the self confidence on beliefs (γ_{MS}), augmenting the control (e.g. enhancing perceptive iterations *Sr*) and/or performing the action in a less risky way(e.g. using safest alternatives in repertoire). Prevalence of positive surprising

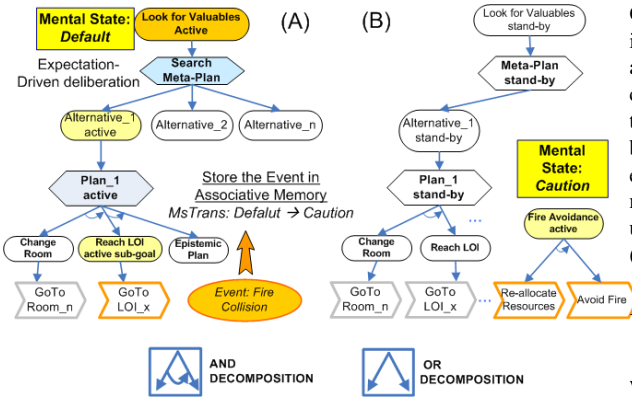


Fig. 3. Intention Reconsideration upon the activation of the Cautiousness Mental State

events induces the agent to shift to **Excitation**, that on the short term is used to arouse the agent, to augment epistemic activities and to search for those 'good' events. A positive surprise (i.e. valuables discovering) may induce the agent to abandon a previous intention and to reformulate his behavior to exploit the new opportunity triggering a new goal G_s. On the long term, excited agent adopt an 'optimistic' mood increasing the confidence (γ_{MS}) of those unexpected, positive events[9]. The lack of surprise progressively empties the SAM and reduces situated perceptive activities. In the long run, it produces a special frame: **Boredom**. Boredom indicates that the environment is almost stationary (no unexpected events are happening) and that the agent can fully exploit his purposive behavior governed by the deliberation driven reasoning. This enhances the subjective confidence in beliefs and in building predictions. Further persistence of boredom leads to **Curiosity**, a control state used to automatically arbitrate from exploitation to exploration activities. The exploration attitude is goal driven: once the agent does not recognize relevant events in his SAM[10], he may infer the low-level expectation that the environment is becoming more static, hence biases his activities towards actions that shows promise to perform a better field coverage and to maintain an updated knowledge. Bypassing the deliberation of practical reasoning, the curious agent pro-actively activates the epistemic G_s of exploring new rooms searching for new facts and events. This has a twofold effect: on the one side it enhances territorial exploration augmenting the chances to discover new LOIs, on the other side it improves knowledge and maintains updated beliefs[11].

V. CONFIDENCE AND MODULATION OF THE PROBABILITY FUNCTION

Effects of MSs can be reconciled with the deliberative level. The intuition behind this integration relies on the fact that each MS endows a certain grade of self-confidence (due to the ongoing mood) that can be related to the belief base. Once we detailed beliefs with a certain strength, one may introduce the self-confidence as a further *discount factor* to affect the likelihood of the predictions. In so doing agents can dynamically adopt a 'more or less confident' capability to build predictions. For example, positive moods provided by excitation can induce the agent to optimistically overestimate the probability of a certain outcome. On the contrary, negative moods like cautiousness may introduce pessimistic under-estimations. On these basis, respect to the one given in (2), the affective expected utility results:

$$AEU'(G_i) = \sum_{a_j \epsilon Plan(G_i)} [A_b \times U(O_{G_i})] \times [\gamma_{MS} \times P(O_{a_j}|a_j)] \quad (3)$$

where γ_{MS} is associated to the ongoing mental state (Tab.I). By associating a given confidence to the subjective capability to make predictions, γ_{MS} introduces a further affective modulation on agent rationality.

[9]Notice that, differently from Excitement emerging on appraisal of an achieved goal, Excitation has been related to a situated positive surprise.

[10]Heuristic thresholds define the k-length time window used for passing from Boredom to Curiosity.

[11]The benefits of interleaving epistemic and practical activities are generally accepted in situated cognition [24]. Different policies can be retrieved in literature to manage exploitation Vs. exploration. Among others, Ahn and Picard [30] proposed an affective signal to abandon exploitation and trigger the process of exploration.

VI. Discussion

Recent computational models are providing simple affective states in terms of their effects on agent's reasoning, behavior and attentive activities [31]. Their functional roles may enable adaptive and situated behaviors and span from reactive methods of control (similar to those employed in primitive biological organisms [32]) to the control of computational resources [33] and the decision making [22], [11]. In our model we propose a quantitative influence of affective states upon the terms of a rational decision. As in appraisal-inspired models, we provide emotions to coordinate the different computational and physical components required to effectively interact in complex environment [34], [35]. Appraisal based systems like Gratch and Marsella's EMA [35], [11], stresses different relations between emotions and cognition, arguing that emotions are a causal precursor of the mechanisms to detect, classify, and adaptively respond to significant changes of environment. Differently from EMA, we adopted a two step approach. First we distinguished between long-term practical reasoning and situated reasoning. The disambiguation of slow, decisional processes from situated ones elicits a clear methodological separation of concerns and may greatly assist the modeler by breaking down the work into two separate and independent activities: while the former is defined referring to the goal overview and clearly involves decisional processes and deliberation of alternative goals, the latter can be defined through control frames, clustering domain dependent strategies, aggregates of heuristics and functional even affective responses used to respond to local events. In the second phase, we reintegrate the two processes by taking into account the correlations and the relative interactions, enlightening how low situated reasoning can be used to inform higher decisional processes. To this end the contribute of MSs is twofold: from the one side they can relieve the deliberative and the attentive processes from the burdens to process weakly relevant information in decision processes, excluding action alternatives that are likely to be less promising or have vanishing likelihood to be achieved. Besides, MSs provide ready to use action selection and resource allocation policies that may relieve agent's need for resource-demanding and meta decision processes. The emergent nature of affective states enables agent to adopt a mental frame while both expectations and emotions are conveyed to inform reasoning for redirecting resources and adopt long term strategies once a disturbing event is detected.

An additionally effect of modeling mental states is for agent's intention reconsideration. Traditional reconsideration strategies indicate an agent to abandon an intention when a related goal is achieved, when a goal become infeasible or when the agent relieve some inconsistencies between the world state and the external conditions necessary for goal achievement. Our model allows basic emotions to elicit an *interruption* on normal cognitive processes when unexpected events require servicing. Once based on expectations of future states, intention reconsideration becomes anticipatory and can be used to coordinate behavior with prediction of future states.

References

[1] M. Bratman, *Intention, Plans and Pratical Reasoning*. Harvard: Harvard University Press, 1987.

[2] A. Rao and M. Georgeff, "BDI agents: From theory to practice," *Proc. of the 1st conf. on MAS (ICMAS95)*, 1995.

[3] M. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. Wooldridge, "The Belief-Desire-Intention Model of Agency," *Proc. of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98)*, vol. 1, p. 1555, 1998.

[4] C. Cherniak, *Minimal rationality*. MIT Press, Cambridge, 1986.

[5] P. Maes, "Modeling adaptive autonomous agents," *Artificial Life, I*, vol. (1&2), no. 9, 1994. [Online]. Available: citeseer.ist.psu.edu/maes94modeling.html

[6] M. Miceli and C. Castelfranchi, "The mind and the future. the (negative) power of expectations," *Theory & Psichology*, vol. 12(3), pp. 335–366, 2002.

[7] C. Castelfranchi, "Mind as an Anticipatory Device: for a theory of Expectations," *1st intern. symposium Brain, Vision, and Artificial Intelligence (BVAI 2005)*, pp. 258–276, 2005.

[8] R. W. Picard, *Affective Computing*. MIT Press, 1997.

[9] P. Gmytrasiewicz and C. Lisetti, "Using decision theory to formalize emotions for multi-agent systems," in *2nd Workshop on Game Theoretic and Decision Theoretic Agents (ICMAS-2000)*, 2000.

[10] M. Scheutz, "How to determine the utility of emotions," in *Proc. of AAAI Spring Symposium 2004.*, 2004.

[11] J. Gratch and S. Marsella, "A Domain-independent Framework for modelling Emotions," *Journal of Cognitive Systems Research*, vol. 5 (4), pp. 269–306, 2004.

[12] R. Lazarus, *Emotion and Adaptation*. Oxford University Press, 1991.

[13] J. L. Pollock, "Taking perception seriously," in *Proceedings of the 1st International Conference on Autonomous Agents (Agents'97)*, W. L. Johnson and B. Hayes-Roth, Eds. New York: ACM Press, 1997, pp. 526–527.

[14] D. Weyns, E. Steegmans, and T. Holvoet, "A model for Active Perception in situated Multi-Agent Systems," in *Proc. of the 1st European Workshop on MAS*, 2003.

[15] E. Lorini and M. Piunti, "The benefits of surprise in dynamic environments: from theory to practice," in *Proceedings of 2nd conference on affective computing and intelligent interactions (ACII07)*, 2007.

[16] R. Reisenzein, *The message within: The role of subjective experience in social cognition and behavior*. Psychology Press, 2000, ch. The subjective experience of surprise.

[17] E. Lorini and C. Castelfranchi, "The cognitive structure of surprise: looking for basic principles," *Topoi: an International Review of Philosophy*, vol. 26(1), 2007.

[18] A. Ortony and D. Partridge, "Surprisingness and Expectation failure: What is the difference?" in *Proc. of the 10th Inter. Joint Conf. on Artificial Intelligence*, Los Altos, CA, 1987, pp. 106–108.

[19] J. Thangarajah, L. Padgham, and J. Harland, "Representation and reasoning for goals in BDI agents," in *In Proc. of the 25th Australian Computer Science Conf. (ACS2002)*, 2002.

[20] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf, "Goal representation for bdi agent systems," *The Second International Workshop on Programming Multiagent Systems (PROMAS-2004)*, 2004.

[21] L. Savage, *The Foundations of Statistics*. Dover, 1954.

[22] J. Doyle, "Rationality and its roles in reasoning," *Computational Intelligence*, vol. 8 (2), pp. 376–409, 1992.

[23] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM Algorithm." *Journal of the Royal Statistical Society*, vol. 39(1), p. 138, 1977.

[24] D. Kirsha and P. Maglio, "On distinguishing epistemic from pragmatic action," *Cognitive Science: A Multidisciplinary Journal*, vol. 18(4), pp. 513–549, 1994.

[25] E. Lorini and C. Castelfranchi, "The role of epistemic actions in expectations," in *In Proc. of the ABiALS workshop*, 2004.

[26] J. J.R. Busemeyer, J. J.T. Townsend, and J. Stout, "Motivational underpinnings of utility in decision making: Decision field theory analysis of deprivation and satiation," *Emotional Cognition*, 2002.

[27] M. Piunti, C. Castelfranchi, and R. Falcone, "Surprise as shortcut for Anticipation: clustering Mental States in reasoning," in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyberadad, India., 2007.

[28] R. Schank and R. Abelson, *Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures*. Hillsdale, NJ: L. Erlbaum, 1977.
[29] L. Cholvy and A. Hunter, "Information fusion in logic: A brief overview," in *ECSQARU-FAPR*, 1997, pp. 86–95. [Online]. Available: citeseer.ist.psu.edu/cholvy97information.html
[30] H. Ahn and R. W. Picard, "Affective cognitive learning and decision making: The role of emotions," in *18th European Meeting on Cybernetics and Systems Research (EMCSR 2006)*, Vienna, Austria., 2006.
[31] M. Scheutz, "Agents with or without emotions?" in *Proceedings of FLAIRS 02, AAAI Press, 89-94.*, 2002.
[32] M. Scheutz and A. Sloman, "Affect and agent control: Experiments with simple affective states," in *Proceedings of IAT-01*, 2001.
[33] H. A. Simon, "Motivational and emotional controls of cognition," *Psychological Review*, vol. 74, pp. 29–39, 1967.
[34] L. Cosmides and J. Tooby, *Handbook of Emotion (Second Edition)*. Guilford Press, 2000, ch. Evolutionary Psychology and the Emotions, pp. 91–115.
[35] J. Gratch and S. Marsella, *Integrated Models of Cognitive Systems*. Oxford University Press, 2006, ch. The architectural role of emotions in cognitive systems.

Adding Roles to Relationship Patterns

Matteo Baldoni
Dipartimento di Informatica
Università di Torino - Italy.
Email: baldoni@di.unito.it

Guido Boella
Dipartimento di Informatica
Università di Torino - Italy.
Email: guido@di.unito.it

Leendert van der Torre
University of Luxembourg.
Email: leendert@vandertorre.com

***Abstract*—In this paper we study how roles can be added to patterns modelling relationships in Object Oriented programming, and which new relationship patterns can be introduced using roles. Relationships can be introduced in programming languages either by reducing them to attributes of the objects which participate in the relationship, or by modelling the relationship itself as a class whose instances have the participants of the relationships among their attributes. However, even if roles have been recognized as an essential component of relationships, also in modelling languages like UML, they have not been introduced in Object Oriented programming when it is necessary to model relationships. Introducing roles allows to add attributes and behaviors to the participants in the relationship, rather than to the relationship itself, and to distinguish natural types as classes participating in the relationships from the roles the participants acquire in the relationships. In this paper we show how the role model proposed in powerJava can be used to endow relationships with roles, both in the relationship as attribute and in the relationship object pattern. Finally, since these patterns have different advantages and limitations, we propose a third pattern based on roles which benefits from the advantages of the two previous patterns when modelling relationships.**

I. Introduction

The need of introducing the notion of relationship as a first class citizen in Object Oriented (OO) programming, in the same way as this notion is used in OO modelling, has been argued by several authors, at least since Rumbaugh [1]. For example, one would like to be able to model the following scenario: a student can be related to a university by an enrollment relationship, he can attend a course, and give exams. Moreover, a course can be a basic course in one curriculum and an advanced one in another. Similarly, other relationships link professors to students and courses, students to tutors, *etc*. Another example is the case of a contract net protocol, where two objects participate in a negotiation relationship, and inside this they can perform negotiation moves.

Relationships are also known as collaborations or associations, like they are called in UML, to distinguish them from specialized relationships like aggregation, relating an object to its parts, and inheritance, relating a class to a superclass.

Rumbaugh [1] claims that relationships are complementary to, and as important as, objects themselves. Thus, they should not only be present in modelling languages, like ER or UML, but they also should be available in programming languages, either as primitives, or, at least, represented by means of suitable patterns.

Two main alternatives have been proposed by Noble [2] for modelling relationships by means of patterns:

- The relationship as attribute pattern: the relationship is modelled by means of an attribute of the objects which participate in the relationship. For example, the `Attend` relationship between a `Student` and a `Course` can be modelled by means an attribute `attended` of the `Student` and of an attribute `attendee` of the `Course`.
- The relationship object pattern: the relationship is modelled as a third object linked to the participants. A class `Attend` must be created and its instances related to each pair of objects in the relationship. This solution underlies programming languages introducing primitives for relationships, e.g., Bierman and Wren [3].

These two solutions have different pros and cons, as Noble [2] discusses. But they both fail to capture an important modelling and practical issue. If we consider the kind of examples used in the works about the modelling of relationships, we notice that relationships are also essentially associated with another concept: students are related to tutors or professors [3], [4], basic courses and advanced courses [4], customers buy from sellers [5], employees are employed by employers, underwriters interact with reinsurers [2], *etc*. From the knowledge representation point of view, as noticed by ontologists like Guarino and Welty [6], these concepts are not natural kinds like person or organization. Rather, they all are *roles* involved in a relationship.

Roles have different properties than natural kinds, and, thus, are difficult to model with classes: roles can be played by objects of different classes, they are dynamically acquired, they depend on other entities - the relationship they belong to and their players. Moreover, when an object of some natural type plays a certain role in a relationship, it acquires new properties and behaviors. For example, a student in a course has a tutor, he can give the exam and get a mark for the exam, another property which exists only as far as he is a student of that course.

Thus, roles cannot simply be modelled as subclasses or superclasses of natural types by means of dynamic reclassification [7]: a student is not simply a subtype of person nor viceversa.

As Steimann [7] argues, there is an intrinsic role of roles as intermediaries between relationships and the objects that engage in them. Thus, in this paper, we focus on the following

research questions: How to introduce roles in the relationship as attribute pattern and in the relationship object pattern? Which other patterns are possible for modelling relationships when roles are introduced in Object Orientation? As subquestions: How to distinguish natural types from roles when designing a program? Which are the pros and cons of the two patterns when roles are added? How to overcome the limitations of the existing patterns?

In this paper we do not propose a new primitive of relationship in programming languages, but we introduce roles in patterns for relationships, and as methodology we use our model of roles in OO programming languages, an extension which adds roles to the Java programming language, called powerJava, described in [8], [9], [10], [11], for which a precompiler has been built.

The language powerJava introduces roles as a way to structure the interaction of an object with other objects calling their methods. Roles express the possibilities of interaction offered by the object to other ones, i.e., the methods they can call. First, these possibilities change according to the class of the callers of the methods. Second, a role maintains the state of the interaction with a certain individual caller. As roles have a state and a behavior, they share some properties with classes. However, roles can be dynamically acquired and released by an object playing them. Moreover, they can be played by different types of classes. This is why roles in powerJava can be useful in modelling relationships, where the behavior of an object changes when it enters a relationship, until it subsequently abandons it.

In Section II we discuss why and how relationships are introduced in OO programming. In Section III we discuss the link between relationships and roles. In Section IV we summarize our model of roles in powerJava and in Section V we use it to introduce roles in the relationship as attribute and relationship object patterns. In Section VI, we describe a new pattern combining the previous ones. Conclusions end the paper.

II. Introducing relations in OO

To understand the importance of relations in programming consider the efforts done to model relationships in defining patterns for them [2], [5], [12] or in extending existing languages like Java [3].

There is not yet a standard language with the relationship primitive, notwithstanding some interesting proposals like [3]. Hence, in this paper, to discuss the role of roles in relationships, we will focus on patterns for modelling relationships. The most important patterns for modelling relationships are the relationship as attribute pattern and the relationship object pattern [2]. We will not consider here other solutions like the collection object, mutual friends and active value patterns.

We will describe these two alternatives with reference to a university domain. Consider a student who can attend different kind of courses: basic ones and advanced ones. The same course can be a basic one in the curriculum of a senior student and an advanced one for junior student. A student can give the exam of the basic course he is attending and it is possible to send a message to the student of the course. Finally, a course is associated with a tutor if it is taken as a basic course; the tutor, which is not present in advanced courses, can be different for every student attending it.

The relationship as attribute pattern is described in Figure 1: the relationship between a student and a basic course he attends is modelled by means of an attribute `attends` of the instances of class `Student` which participate in the relationship. The type of the attribute is a set of `BasicCourse`. Symmetrically, the `Student` appears in the attribute `attendees` of the class `BasicCourse` of type set of `Student`. The `BasicCourse` is also related with other courses by a relationship representing the prerequisites.

This solution, however, does not allow to add a state and behavior to the elements related by the relationship. For example, it is not possible to specify a different tutor for each `Student` of the `BasicCourse`. Moreover, the `enrol` method is arbitrarily implemented in `BasicCourse` rather than in `Student`.

The relationship object pattern is instead described in Figure 2: the relationship `AttendBasicCourse` is modelled by a class whose instances link each `Student` to the `BasicCourse` he attends. The second solution solves some of the issues discussed in this section, in particular, it facilitates the cohesion of the program, by factoring in the class `AttendBasicCourse` all the relevant information. In particular, the class can contain the properties and the operations which the participants are endowed with when they enter the relationship. For example, a `Student` can take the exam of the `BasicCourse` and get a mark if he is successful. Note that the mark is a property belonging to the relationship. Moreover, the `Student` can be associated with a tutor in a `BasicCourse`.

Also this solution can be modelled in UML, which specifies information proper of an association via an association class, where the properties and behaviors of the relationship are represented. An association class has exactly one instance for each set of objects linked through the association and a lifetime delimited by the existence of the association. If a link is dissolved, the association class instance is destroyed. Due to the association, certain information exists that is specific to the association. In UML a dashed line is used to specify an association class.

But the relationship object solution shares with the relationship as attribute some limitations. First, we would like to model the university scenario introducing natural types like `Person` and `Course` rather than the `Student` and `BasicCourse` classes only. The reason for such modelling choice is that a `Person` is not always a `Student`, and he can play also other roles at the same time as he is a `Student`. Moreover, a `Person` is a `Student`, and, thus, he can give exams or receive communications concerning the course, only if he is related by the `AttendBasicCourse` relationship with a `Course` which he follows as a `BasicCourse`. He has different marks in different exams, and even different

```
class Student {
  String name;
  int number;
  HashSet<BasicCourse> attends; }

class BasicCourse {
  String code;
  String title;
  HashSet<Student> attendees;
  HashSet<BasicCourse> prerequisites;
  void enrol(Student s) {
    attendees.add(s);
    s.attends.add(this); } }
```

Fig. 1. The relationship as attribute pattern

```
class Student {
  String name;
  int number; }

class BasicCourse {
  String code;
  String title; }

class AttendBasicCourse {
  BasicCourse attended;
  Student attendee;
  Person tutor;
  int mark;
  AttendBasicCourse(Student s, Course c) {
    attended = c;
    attendee = s; }
  int giveExam(String work){mark = ...}
  void communicate(String text){...} }
```

Fig. 2. The relationship object pattern

students can have different tutors for the same course. Analogously a `Course` has a tutor only if it plays the role of `BasicCourse`.

Second, the relationship as attribute allows to add new properties and behaviors. However, it does not allow to satisfy completely the requirement that properties and behaviors are associated to the participants: this pattern does not distinguish which properties belong to the `Student` and which ones to the `BasicCourse`. This problem is more evident in the case of behaviors, since all the methods are invoked on the relationship object of class `AttendBasicCourse` rather than on the two related objects `Student` and `BasicCourse`. This is not only a modelling problem. It is not possible to have a method with the same name which should be called on either participant, `Student` and `BasicCourse`, with a different meaning. Thus, polymorphism is limited, for example, when the method should be specified as part of an interface implemented by both classes participanting in the relationship.

As noticed by Steimann [13], some of these problems cannot be solved by using subclassing: playing a role is not equivalent to subclassing (a `Person` becomes a `Student`), since a role can be played by instances of different classes. Consider the case of a role customer which can be played either by a person or by an organization.

Finally, these patterns do not consider a further dimension: the complexity of encapsulation when relations are considered. This problem has been highlighted by Noble and Grundy [5]: “Extra relationship objects existing ‘outside’ their participating objects may also be seen as breaking the participating object’s encapsulation [1]. The first point to note here is that many relationships occur between objects which are themselves parts of another aggregate object: that is, the relationship and the participating objects may all be encapsulated by another object. The second point here is that if encapsulation is broken by the relationship, this is because the encapsulated objects need to be accessed by the relationship object in order to implement the semantics of the relationship. Without the explicit relationship object, the analysis relationship would have to be implemented in another way, by being built in to the participating objects. If the relationship requires access to the ‘inside’ of an object breaking its encapsulation, these objects would therefore need to break each other’s encapsulation anyway. In short, using an explicit relationship object cannot worsen breaches of encapsulation. The root of the problem is not the relationship *object* (i.e., how the relationship is implemented), but the existence of the relationship as part of the problem domain.

In some circumstances, relationship objects may actually increase encapsulation, as the implementation of the relationship itself becomes encapsulated against the participating objects when it is moved in to a separate relationship object.”

In the next section we will explain how roles and relationships are related and how to overcome these problems.

III. Roles and relationships

Relations are deeply connected with roles. This is accepted in several areas: from modelling languages like UML and ER to knowledge representation discussed in ontologies and multiagent systems.

The link between roles and relationships is explicit in modelling languages like UML in the context of collaborations: a classifier role is a classifier like a class or interface, but “since the only requirement on conforming instances is that they must offer operations according to the classifier role, [...] they may be instances of any classifier meeting this requirement” [14]. In other words: a classifier role allows any object to fill its place in a collaboration no matter what class it is an instance of, if only this object conforms to what is required by the role. Classification by a classifier role is multiple since it does not depend on the (static) class of the instance classified, and dynamic (or transient) in the sense above: it takes place only when an instance assumes a role in a collaboration [15].

As noticed by Steimann [13], roles in UML are quite similar to the concept of interface, so that he proposes to unify the two concepts. Instead, there is more in roles than in interfaces. Steimann himself is aware of this fact: “another problem is that defining roles as interfaces does not cover everything one might expect from the role concept. For instance, in certain

situations it might be desirable that an object has a separate state for each role it plays, even for different occurrences in the same role. A person has a different salary and office phone number per job, but implementing the Employee interface only entails the existence of one state upon which behaviour depends. In these cases, modelling roles as adjunct instances would seem more appropriate."

To do this, Steimann [7] proposes to model roles as classifiers related to relationships, but such that these classifiers are not allowed to have instances. In Java terminology, roles should be modelled as abstract classes, where some behavior is specified, but not all the behavior, since some methods are left to be implemented in the class extending them. These abstract classes representing roles should be then extended by other classes. However, given that in Java multiple inheritance is not allowed, this solution is not viable, and roles can be identified to interfaces only.

Roles as defined in programming languages [11], [16], [17], instead, are different from interfaces, even if they share some properties with interfaces, like the fact of being partial specifications of behavior, thus allowing objects of different classes entering the same role in a relationship. In particular, roles have a state, add new operations to their players, and depend on a context [11], [17].

Also Whitehurst [18] argues that behavior depends on roles: "the behavior of an object can change depending on the role it plays. When an association is formed between two instances, the behavior of the associated instances is altered in some way. A real world example is a person who becomes a parent. The person has a parental association with a young person (a child) and the behavior of the person is changed due to this association".

Pearce and Noble [12] notice that relationships have similarities with roles. Objects in relationships have different properties and behavior: "behavioural aspects have not been considered. That is, the possibility that objects may behave differently when participating in a relationship from when they are not. Consider again the student-course example [...]. In practice, a course will have many more attributes, such as a curriculum, than we have shown. Such attributes will change over time in line with changes to the course. A useful constraint would be to prevent any changes when students are attending the course it would be unfair if the curriculum changed just before the exam! Thus, Course objects behave differently (i.e., they don't accept changes) when they are participating in a relationship from when they are not (i.e., they do accept changes)."

Thus, roles and relationships are strictly related.

In UML, it is possible to specify information and behavior specific to an association via an association class, but not with roles, which are partial description of behavior which do not add anything to their players.

In conclusion, it seems that besides the relationship objects it is necessary to introduce further objects representing the roles as adjunct instances of new classes.

IV. ROLES IN POWERJAVA

Baldoni *et al.* [8], [9], [10], [11] introduce roles as affordances in powerJava, an extension of the object oriented programming language Java. powerJava is translated into Java by means of a precompiler, whose details are described in [11].

We only summarize here the powerJava language.

Java is extended with:

1) A construct defining the role with its name, the requirements and the signatures of the methods offered to the objects by playing the role, called powers.
2) The implementation of a role as a class, inside an object, and according to the definition of its powers.
3) A construct for playing a role and invoking the operations offered to the role.

We illustrate powerJava by means of an example. Let us suppose to have a printer which supplies two different ways of accessing it: one as a normal user, and the other as a superuser. Normal users have the power to print their jobs and the number of printable pages is limited to a given maximum. Superusers have the power to print any number of pages and can query for the total number of prints done so far. To be a user one must have an account, which is printed on the pages. The role specifications are the following:

```
role User playedby Accounted {
    int print(Job job);
    int getPrintedPages(); }

role SuperUser playedby Accounted {
    int print(Job job);
    int getTotalPages(); }
```

Requirements must be implemented by the objects which act as players.

```
interface Accounted
   { Login getLogin(); }

class Person implements Accounted {
    Login login;  // ...
    Login getLogin() {return login;} }
```

Instead, roles are implemented in the class which offers the role. To implement roles inside it we revise the notion of Java *inner class*, by introducing the new keyword `definerole` instead of `class` followed by the name of the role definition that the class is implementing (see the class `Printer` in Figure 3).

As a Java inner class, the methods of a role implementation, called powers, have access to the private fields and methods of the outer class (in the above example the private method `print` of `Printer` used both in role `User` and in role `SuperUser`) and of the other roles defined in the outer class. This possibility does not disrupt the encapsulation principle since all roles of a class are defined by the same programmer who defines the class itself. In other words, an object that has assumed a given role, by means of the role's methods, has access and can change the state of the object the role belongs to and of the sibling roles.

All the constructors of roles have an implicit first parameter to which it must be passed as value the player of the role: to construct a role we need both the object the role belongs to (the object the construct `new` is invoked on) and the player of the role (the first implicit parameter). This parameter has as its type the requirements of the role and it is assigned to the keyword `that`. A role instance is created by means of the construct `new` starting from the object offering the role and by specifying the name of the inner class implementing the role which we want to instantiate. This is like it is done in Java for inner class instance creation. Differently than other objects, role instances do not exist by themselves and are always associated to their players and to the object the role belongs to.

The following instructions create a printer object `laser1` and two person objects, `chris` and `sergio`. `chris` is a normal user while `sergio` is a superuser. Indeed, instructions four and five define the roles of these two objects w.r.t. the created printer.

```
Printer laser1 = new Printer();

//players are created as Person
Person chris = new Person();
Person sergio = new Person();

//roles are created
laser1.new User(chris);
laser1.new SuperUser(sergio);
```

An object has different (or additional) properties when it plays a certain role, and it can perform new activities, the powers, as specified by the role definition. Moreover, a role represents a specific state which is different from the player's one, which can evolve with time by invoking methods on the roles. The relationship between the object and the role must be transparent to the programmer: it is the object which has to maintain a reference to its roles.

The behavior of a role instance depends on the player instance of the role, so in the method implementation the player instance can be retrieved via a new reserved keyword: `that`, which is used only in the role implementation. In Figure 3, `that.getLogin()` is a parameter of the method `print`.

Methods can be invoked from the players, given that the player is seen in its role. To do this, we introduce the new construct, called *role cast*. Role casting views an object as having a different state and different behaviors when playing different roles. Role casting allows to make transparent to the programmer the association of a role and an object instance: the programmer invokes a method of a role on the object playing it casted into the role; the language transforms this method invocation in a message sent to the delegated role instance, which is hidden in its player.

In the example the two users invoke the method `print` on `laser1`. They can do this because they have been empowered of printing by playing their roles. The act of printing is carried on by the private method `print`. Nevertheless, the two roles

```
class Printer {
  private int totalPrintedPages = 0;
  private void print(Job job, Login login) {
    totalPrintedPages += job.getNumberPages();
    // performs printing
  }

  definerole User {
    int counter = 0;
    public int print(Job job) {
      if (counter > MAX_PAGES_USER)
         throws new IllegalPrintException();
      counter += job.getNumberPages();
      Printer.this.print(job, that.getLogin());
      return counter;}
    public int getPrintedPages()
      { return counter; }
  }

  definerole SuperUser {
    public int print(Job job) {
      Printer.this.print(job, that.getLogin());
      return Printer.this.totalPrintedPages;}
    public int getTotalpages()
     { return Printer.this.totalPrintedPages; }
  }
}
```

Fig. 3. The `Printer` class and its roles

of `User` and `SuperUser` offer two different ways to interact with the `Printer`: `User` counts the printed pages and allows a user to print a job if the number of pages printed so far is less than a given maximum; `SuperUser` does not have such a limitation. Moreover, `SuperUser` is empowered also of viewing the total number of printed pages. Notice that the page counter is maintained in the role state and persists through different calls to methods performed by a same sender/player towards the same receiver as long as it plays the role.

```
((laser1.User) chris).print(job1);
((laser1.SuperUser) sergio).print(job2);
((laser1.User) chris).print(job3);
System.out.println("The printer printed" +
  ((laser1.SuperUser) sergio).getTotalPages());
```

Since an object can play multiple roles, the same method will have a different behavior depending on the role which the object is playing when it is invoked. It is sufficient to specify which is the role of a given object, we are referring to. In the example `chris` can become also `SuperUser` of `laser1`, besides being a normal `user`

```
laser1.new SuperUser(chris);
((laser1.SuperUser) chris).print(job4);
((laser1.User)chris).print(job5);
```

In this case two different sessions will be kept: one for `chris` as normal `User` and the other for `chris` as `SuperUser`. Only when it prints its jobs as a normal `User` the page counter is incremented.

```
role Student playedby Person {
  int giveExam(String work); }

role BasicCourse playedby Course {
  void communicate(String text); }

class Person{
  String name;
  private Queue messages;
  private HashSet<BasicCourse> attended;

  definerole BasicCourse {
    Person tutor;
    void communicate (String text){
      Person.this.messages.add(text);}
    BasicCourse(Person t){
      tutor=t;
      Person.this.attended.add(this); }
  }
}

class Course {
  String code;
  String title;
  private HashTable registry =
     new HashTable();
  private HashSet<Student> attendees;
  private int evaluate(String x){...}

  definerole Student {
    int number;
    int mark;
    int giveExam(String work){
      mark = Course.this.evaluate(work);
      registry.set(that.hashCode(), mark);
      return mark;
      }
    Student (){
      Person.this.attended.add(this); }
  }
}
```

Fig. 4. Relationship-role as attribute pattern in powerJava

V. Relationships with roles using powerJava

In this section we describe how new patterns for modelling relationships with roles can be defined, in analogy with both the relationship as attribute and the relationship object pattern. We will use the example of Section II to present them.

First of all, using powerJava we can model instances of natural types like `Person` and `Course` which become, respectively, `Student` and `BasicCourse` when they enter the relationship. This is possible because `Student` and `BasicCourse` are roles represented in powerJava by instances associated with the players of the roles, which include the state and behaviors acquired by the players of the roles in the relationship.

In the relationship-role as attribute pattern, a relationship is not reduced only to two symmetric attributes `basicCourses` and `attendees`. The relationship is modelled also by means of a pair of roles. The `Person` plays the role of `Student` with respect to the `Course` and the `Course` plays the role of `BasicCourse` with respect to the `Person` (see Figures 4, 5 and 6 where the UML representation is illustrated[1]).

```
public static void main (String[] args){
  Course c = new Course();
  Person p = new Person();
  //create a role Student for p in c
  Student s = c.new Student(p);
  BasicCourse b = p.new BasicCourse(c,tutor);
  //p as a Student of Course c gives the exam
  ((c.Student)p).giveExam(work);
  //a message text is sent
  ((p.BasicCourse)c).communicate(text); }
```

Fig. 5. Using the relationship-role as attribute in powerJava

Thus, the attribute `attendees` of type `Student` in `Course` is not replaced by one of type `Person`. Rather, `Student` is defined as role and `Person` is a class which can play the role (see the role definition connecting a role to the classes playing it). The role `Student` is associated with players of type `Person` in the role definition, which specifies that a `Student` can give an exam (`giveExam`). Analogously, the role `BasicCourse` is associated with players of type `Course` in the role definition, which specifies that a `Course` can communicate with the attendee.

The role `Student` is implemented locally in the class `Course` and, viceversa, the role `BasicCourse` is defined locally in the class `Person`. Note that this is not contradictory, since roles describe the way an object offers interaction to another one: a `Student` represents how a `Person` can interact with a `Course`, and, thus, the role is defined inside the class `Course`. Moreover the behavior associated with the role `Student`, i.e., giving exams, modifies the state of the class including the role (it access the `registry` variable) or calls its private methods (`evaluate`), thus violating the standard encapsulation. Analogously, the `communicate` method of `BasicCourse`, modifies the state of the `Person` hosting the role by adding a message to the queue. These methods, in powerJava terminology, exploit the full potentiality of powers of violating the standard encapsulation of objects.

To associate a `Person` and a `Course` in the relationship, the role instances must be created starting from the objects offering the role, e.g.: `c.new Student(p)` (see the `main` in Figure 5).

When the player of a role must invoke a power it must be first role casted to the role. For example, to invoke the method `giveExam` of `Student`, the `Person` must first become a `Student`. To do that, however, also the object offering the role must be specified, since the `Person` can play the role `Student` in different instances of `Course`; in this case the `Course c`: `((c.Student)p).giveExam(...)`.

The alternative relationship-role object pattern introduces an `AttendBasicCourse` class modelling the relationship between `Person` and `Course`. However, the

[1]The arrow starting from a crossed circle, in UML, represents the fact that the source class can be accessed by the arrow target class.

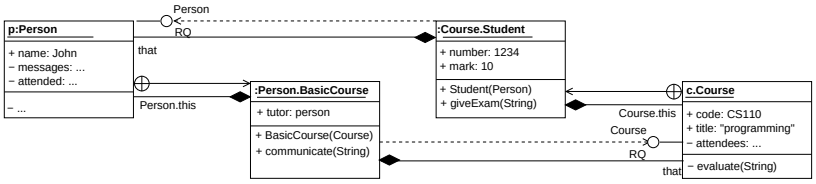


Fig. 6. The UML representation of the relationship-role as attribute pattern example

`AttendBasicCourse` class is not linked to a `Person` and a `Course`. Rather, the `Person` plays the role `Student` in the class `AttendBasicCourse` and the `Course` the role `BasicCourse` (see Figures 7 and 8). Like in the previous solution the roles are modelled as classes implemented, in this pattern, in the class `AttendBasicCourse` whose instances contain the properties and behaviors added when instances of `Person` and `Course`, respectively, participate in the relationship. Additionally, properties and behaviors which are associated to the relationship itself, like entering in the relationship and constraints on the participants can be added to the relationship class.

To relate a `Person` and a `Course` in a relationship, an instance of `AttendBasicCourse` must be created, together with an instance of `Student` played by the `Person` and of `BasicCourse` played by the `Course`. To invoke a power of `Student`, a `Person` must be role casted to the role `Student` starting from an instance of the class `AttendBasicCourse`.

With respect to the previous pattern, it is possible to notice that the roles can interact with each other: the role `Student` invokes in the method `giveExam` the private method `evaluate` of the `BasicCourse` role. However, the roles cannot anymore access the private state and methods of the player of the class. For this reason, it is necessary to add a public `getMessage` method in `Person`, and to define `evaluate` in the role rather than in the `Course` class.

The two patterns have different pros and cons; the following list integrates Noble [2]'s discussions on them.

Advantages of the Relationship-role as attribute pattern:

- It allows simple one-to-one relationships: it does not require a further class and its instance to represent the relationship between two objects.
- It allows to introduce a state and operations to the objects entering the relationship, which was not possible without roles in the relationship as attribute pattern.
- It allows the integration of the role and the element offering it by means of powers.
- It allows to show which roles can be offered by a class, and, thus, in which relationships they can participate, since they are all defined in the class.

Disadvantages of the Relationship-role as attribute pattern:

- It requires that the roles are already implemented offline inside the classes which participate in the relationship.
- It does not assure coherence of the pair of roles like student-course, buyer-seller, bidder-proponent, since they are defined separately in two different classes.
- The role cast to allow a player to invoke a power of its role requires to know the identity of the other participant in the relationship.
- It does not allow to distinguish which is the role played in the other object participating in the relationship (e.g., a `Student` in the `attendees` set of a `Course` can follow the `Course` as a `BasicCourse` or an `AdvancedCourse`).

Advantages of the Relationship-role object pattern:

- It allows to introduce a state and operations of the relationship besides the state and operations added to the objects entering the relationship.
- It allows to list all instances of the relationship and centralize operations like entering the relationship and to check constraints on the relationship.
- It enforces to create both role instances at the same time, since they are linked to the same relation instance, thus avoiding the risk of inconsistencies.
- It allows the integration of the role with the relationship and with the other role, since the powers of a role can access both. In this way it is possible to deal with coordination issues [8].
- To make a role cast it is necessary only to know the relationship instance, thus, the other participant can change without notice.
- It does not require that the classes of players already implement the role classes. To play a role it is sufficient to satisfy the requirements.

Disadvantages of the Relationship-role object pattern:

- It requires a further class and its instance.
- It does not allow the integration of roles with the objects offering them (e.g., `Student` is defined separately of the class `Course`, which, as a consequence, cannot be accessed). Thus, private variables of classes offering the role cannot be accessed anymore (see `registry` of `Course` in Figure 4), otherwise an object is required to offer additional public methods to access them (see `getMessage` in Figure 7), which endangers encapsulation.
- The roles cannot be tailored anymore with the class offering the role. E.g., the method `evaluate` cannot be anymore modelled as a private method of `Course`: different courses cannot evaluate an exam in different manners.

```
role Student playedby Person {
  int giveExam(String work); }

role BasicCourse playedby Course {
  void communicate(String text); }

class AttendBasicCourse{
  Student attendee;
  BasicCourse attended;
  static Hashset<AttendBasicCourse> all;

  definerole Student {
    int mark;
    int number;
    int giveExam(String work){
      mark = AttendBasicCourse.this.attended.evaluate(work); }
  }

  definerole BasicCourse {
    String program;
    Person tutor;
    private int evaluate(String work){...}
    void communicate(String text){
     //invoke the requirement of the Person playing the role
     AttendBasicCourse.this.attendee.that.getMessage(text);}
   }

   AttendBasicCourse(Person p, Course c, String p, Person t){
     attendee = this.new Student(p);
     attended = this.new BasicCourse(c);
     AttendBasicCourse.all.add(this);
   }
  void communicate(String text){
    foreach (AttendBasicCourse x: all)
      x.attended.communicate(text);}
  }

class Person{
  String name;
  private Queue messages;
  void getMessage(String text) {
    messages.add(text) };
}

class Course {
  String code;
  String title;
}

class University{
  public static void main (String[] args){
    Person p = new Person();
    Course c = new Course();
    a = new AttendBasicCourse(p,c,program,tutor);
    //p as a Student  gives the exam
    ((a.Student)p).giveExam(work);
    //c is used to send a message
    ((a.BasicCourse)c).communicate(text);}
```

Fig. 7. Relationship-role object pattern

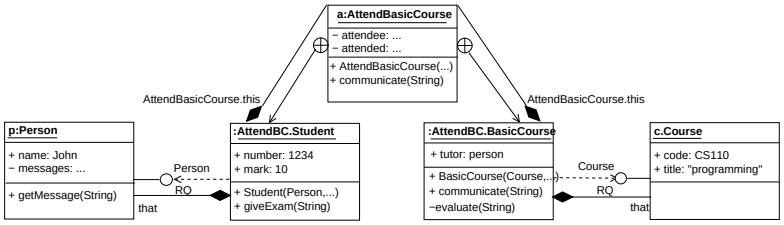


Fig. 8. The UML representation of the relationship-role object pattern example

In summary, we can define an informal program transformation, which is common to both patterns, to add roles to relationship patterns using powerJava:

1) Identify the natural types of the objects playing the roles (e.g., `Person` for `Student`, or `Person` and `Organization` for `Customer`).
2) Change the type of the classes which participate in the relationship from the name of the role to the name of the natural kind playing the role (now there can be more than one class playing the role); e.g., the class `Student` becomes `Person`.
3) Add a role definition relating the role to the natural types which can play the roles, or to an interface implemented by these natural types, and insert in the role definition the signature of the powers (e.g., `communicate`, `giveExam`).
4) Identify the two links to the participants in the relations, either in the classes representing the participants (e.g., `attendees` of type `Student` in `Course`), or in the class representing the relationship (for example `attendee` of type `Student` and `attended` of type `BasicCourse` in `AttendBasicCourse`).
5) In the same class the link belongs to, add a role class implementing the role definition with the same name as the type of the link (e.g., `Student` in the `BasicCourse` class which is now called `Course`, or `Student` and `BasicCourse` in `AttendBasicCourse`). Add to this role class the attributes and the implementation, according to the role definition, of the powers.
6) In the code which relates the two participant instances to the relationship, instead of adding the players to the links, first create two roles instances played by the respective playes, and, second, add these instances to the links modelling the relationship (either in the class of the players, e.g., `Person` or in the class modelling the relationship object, e.g., `AttendBasicCourse`).
7) When a method added by the relationship must be invoked, first, make a role cast from the object playing the role to the role it plays.

VI. A NEW RELATIONSHIP PATTERN WITH ROLES

The two relationship patterns added with roles still leave some unsolved problems. On the one hand, the relationship-role as attribute pattern allows a strict coupling between the role and the class offering it. For example, the role `Student` can access the class `Course` when the `Person` playing the role gives an exam: the registry of the courses can be added with a new entry with the mark of that exam.

On the other hand, the relationship-role object pattern allows a better coordination of the two roles. All roles of a relationship share the same namespace, and, thus, can access each other and the relation too. In this way, it is possible to define the interaction between the roles separately from the classes of possible players, and to guarantee that that the interaction among the players will be performed in the desired way. In contrast, the roles are separated from the class offering them in the relationship-role as attribute patterns, and, thus, roles cannot access the classes offering the roles. They only share the relationship namespace.

So the two patterns offer a tradeoff between the coupling of the role together with the class offering it (`Student` of a `Course`, `Employee` of an `Organization`, *etc.*), and the coordination among the roles. The ideal situation should allow both aspects to be dealt with. These problems are the result of the complexities concerning encapsulation arising when relatioships are taken seriously, as noted by Noble and Grundy [5] and reported in Section II.

A solution is possible in powerJava by exploiting an often disregarded feature of Java. The idea is as follows, and it is illustrated and in Figure 9 as an UML diagram. First, as in the relationship-role object pattern, a class for creating relationship objects is created, containing the roles (e.g., `Student` and `BasicCourse` in `AttendBasicCourse`), see Figure 10 . The interaction between the roles is defined at this level since the powers of each role can access the state of the other roles and of the relationship.

Differently from the relationship-role pattern, these roles must be defined as abstract and so cannot be instantiated. Moreover, the methods containing the details about how these methods describing interaction work can be left unfinished and declared as abstract.

Second, the same roles can be defined according to the relationship-role as attribute pattern in the classes offering them (and, thus, they can be developed separately), see Fig-

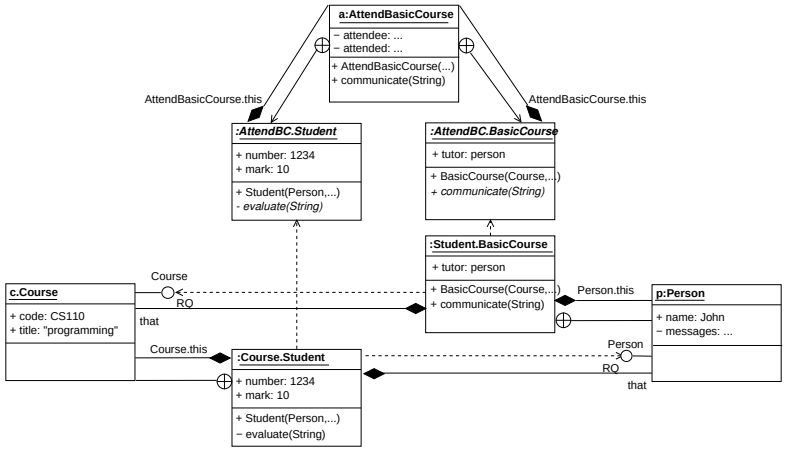


Fig. 9. The UML representation of the new relationship-role pattern

ure 11. However, these roles, rather than being implemented from scratch (e.g., `Student` and `BasicCourse`), they extend the abstract roles of the relationship object class, filling the gaps left by abstract methods in the abstract roles. The extension is necessary to customize the roles to their context. Methods which are declared as final in the abstract roles cannot be overwritten, since they represent the interaction among roles in the scope of the relationship. Further methods can be declared, but they are not visible from outside since both the abstract role and the concrete one have the signature of the role declaration.

Note that the abstract roles are not extended by the classes participating in the relationship (e.g., `Course` and `Person`), but by roles offered by (i.e., implemented into) these classes. Otherwise, the classes participating in the relationship could not extend further classes, thus limiting the code reuse possibilities.

The advantage of these solutions is that roles can share both the namespace of the relationship object class and the one of the class offering the roles, as we required above. This is possible since extending a role implementation is the same as extending an inner class in Java: roles are compiled into inner classes. When a class extends an inner class in Java, it maintains the property that the methods defined in the inner class it is extending continue to have access to the outer class instance containing the inner class. If the inner class is extended by another inner class, the resulting inner class belongs to the namespaces of both outer classes. Moreover, the inner class instance has a reference to both outer class instances so to be able to access their states. The possible ambiguities of identifiers accessible in the two outer classes and in the superclass are resolved by using the name of the outer class as a prefix of the identifier (e.g., `Course.this.registry`).

This feature of Java, albeit esoteric, has a precise semantics, as discussed by [19]. We exploit this mechanism for extending roles in relationship object classes, thus giving a new, more usable, meaning to this construct and hiding the complexities to the programmer.

Basing on this idea we propose here an extension of powerJava, which allows to define abstract roles inside relationship object classes, and to let standard roles extend them. The resulting roles will belong both to the namespace of the class offering them and to the relationship object class. Moreover, the resulting roles will inherit the methods of the abstract roles.

Note that the abstract roles cannot be instantiated, so that the are used only to implement both the methods which define the interaction among the roles, and the methods which are requested to be contextualized. The former will be final methods which are inherited, but which cannot be overwritten in the extending role: they will access the state and methods of the outer class and of the sibling roles. The latter will be abstract protected methods, which are used in the final ones, and which must be implemented in the extending class to tailor the interaction between the abstract role and the class offering the role. If these methods are declared as protected they are not visible outside the package. These methods have access to the class offering the extending roles.

Besides adding the property `abstract` to roles, three other additions are necessary in powerJava.

First of all, the methods of the abstract role can make reference to the outer class of the extending role. This is realized by means of a reserved variable `outer`, which is of type `Object` since it is not possible to know in advance which classes will offer the extended role. This variable is visible only inside abstract roles.

Second, to create a role instance it is necessary to have at disposal also the relationship object offering the abstract roles, and the two roles must be created at the same time. For

```
role Student playedby Person complements BasicCourse {
  int giveExam(String work); }

role BasicCourse playedby Course complements Student {
  void communicate(String text); }

class AttendBasicCourse{
  Student attendee;
  BasicCourse attended;

  abstract class Student {
    int mark;
    int number;
    //method modelling interaction
    final int giveExam(String work){
      return mark = evaluate(work);}
    //method to be implemented
    abstract protected int evaluate(String work);
  }

  abstract class BasicCourse {
    String program;
    Person tutor;
    //method to be implemented
    abstract void communicate(String text);
  }

  AttendBasicCourse(Person p, Course c, String pr, Person t){
    attendee = c.new Student(p,this);
    attended = p.new BasicCourse(c,this,t);
  }
}
```

Fig. 10. The new relationship-role pattern

example:

```
AttendBasicCourse(Person p, Course c){
  ...
  c.new Student(p,this);
  p.new BasicCourse(c,this);
}
```

Where `Student` and `BasicCourse` are the name of the concrete roles implemented in `p` and `c` and it is the same as the abstract roles defined in the relation.

The types of the arguments `Person` and `Course` are the requirements of the roles `Student` and `BasicCourse`.

Moreover, the first and the second argument of the constructor are added by default: the first one represents the player of the role, while the second one, present only in roles extending abstract roles, is the reference to the relationship object. This is necessary since the inner class instance represented by the role has two links to the two outer classes it belongs to. This reference is used to invoke the constructor of the abstract role, as required by Java inner classes, for example, the constructor of the role `Course.Student` is the following one.

```
Student(Person p, AttendBasicCourse a){
  a.super();
  ... }
```

However, these complexities are hidden by powerJava which adds the necessary parameters and code during precompilation.

The entities related by the relationship must preexist to it:

```
Person p = new Person();
Course c = new Course();
AttendBasicCource r =
  new AttendBasicCourse(p,c);
((c.Student)p).giveExam(w);
```

Note that the role cast `((r.Student)p)` is equivalent to `((c.Student)p)`.

Third, in the extension of powerJava abstract roles only come in pairs (e.g., `Student` and `BasicCourse`). Thus, the definition of a role must be extended to specify not only that the possible players comply with the requirements, but also which role must be offered in turn to play a role. E.g., a class `Person` to play the role `Student` has to offer in turn the role `BasicCourse`. For this reason, the role definition is extended with the keyword `complements` specifying the other role of the relation. E.g.,

```
role Student playedby Person
  complements BasicCourse {
    int giveExam(String work); }
```

Thus, a class can play a role not only if it implements the requirements in the role definition, but also if it offers the role specified as complementary.

Finally, we add an additional constraint to powerJava: if a role implementation extends another role, it must have the

```
class Course {
  String code;
  String title;
  private HashSet<Student> attendees;
  private HashTable registry = new HashTable();

  class Student extends AttendBasicCourse.Student {
    Student(){
      Course.this.attendee = this;
    }
    //abstract method implementation
    protected int evaluate(String work){
      mark = ...
      Course.this.registry.set(that.hashCode(), mark);
      return mark;
    }
  }
}

class Person {
  String name;
  private Queue messages;
  private HashSet<BasicCourse> attended; //courses followed as BasicCourse

  class BasicCourse extends AttendBasicCourse.BasicCourse{
    BasicCourse(Person t){
      tutor=t;
      Person.this.attended=this;
    }
    //abstract method implementation
    void communicate (String text) {
      Person.this.messages.add(text);
    }
  }
}
```

Fig. 11. The new relationship-role pattern

same name. Thus, the abstract and concrete role have the same requirements. Moreover, it is possible to extend only abstract roles, while general inheritance among roles is not discussed here.

In Figures 10 and 11, we report the example used in the previous sections using the new pattern. Note in particular, that the class `Person` does not have anymore a `getMessage` method, like in the example of Figure 7, since the role `BasicCourse` of `Person` has access directly to the private queue of messages of a person. Moreover, the `registry` of exams in a `Course` can be updated when giving an exam, as in the relationship as attribute solution, since the role `Student` has access to the class offering it. Finally, the method `evaluate` is defined inside the role implementation `Student` of the class `Course`, so that it can be tailored to different kind of courses.

This example, however, does not show how the interaction among roles can be separated from the classes of the players and gathered inside the relationship object class. For this reason we add also another example.

Consider a relationship like a negotiation protocol `CNP` (Contract Net Protocol). It relates two objects: first, an object of type `Manager` which plays the role of `Initiator` and, as `Initiator`, makes calls for proposals `cfp`; second, an object of type `Worker`, who is able to execute a task, can play the role of `Participant` and, as such, to make `proposal`s in return to the `cfp`. Note that as in the relationship-role as attributes pattern, the `Initiator` role is offered by `Worker` to allow the `Manager` to call the method `cfp` to interact with the `Worker`. Viceversa, the role `Participant` is offered by the `Manager` to the `Worker` to respond with a `proposal` to the `Manager`. However, differently than in the relationship-role as attributes pattern all the interaction among the roles happens inside the abstract roles defined in the class `CNP`. In this way, objects entering a negotiation are guaranteed that the role offered by the other participant is coherent with the one offered by their own. The only function of the role `Manager.Participant` and `Worker.Initiator` is to tailor the behavior of the abstract roles to the classes offering their extensions.

VII. Conclusion

In this paper we discuss why roles need to be introduced when relationships are modelled in OO programs: it is possible to distinguish between the natural type of objects populating the program and the state and behaviors they acquire when

```
role Initiator playedby InitiatorReq complements Participant {
   void cfp(Task task);
   void rejectProposal(Proposal proposal);
   void acceptProposal(Proposal proposal);
   }

role Participant playedby ParticipantReq complements Initiator {
  void propose(Proposal proposal);
  void refuse();
  void inform(String result);
  void failure();
  }

public class CNP {
  final static int STATE_1 = 1;
  final static int STATE_2 = 2; //...
  int state = STATE_1;

  abstract definerole Initiator {
    public final void cfp(Task task) throws IllegalPerformativeException {
      if (state != STATE_1) throw new IllegalPerformativeException ();
      state = STATE_2;
      if (evaluateTask(task))
        ((that.Participant)outer).propose(getProposal(task));
      else
        ((that.Participant)outer).refuse();
    }
    public final void rejectProposal(Proposal proposal) throws ... {
      if (state != STATE_3) throw new IllegalPerformativeException();
      state = STATE_4;
    }
    public final void acceptProposal(Proposal proposal) throws ... {
      if (state != STATE_3) throw new IllegalPerformativeException();
      if (performTask(proposal, task))
        ((that.Participant) outer).inform(performTask(proposal,task));
      else ((that.Participant)) outer).failure(error);
      state = STATE_5;
    }//methods to be implemented
    abstract protected boolean evaluateTask(Task task);
    abstract protected String performTask(Proposal proposal, Task task);
    }

  abstract definerole Participant {
    public final void propose(Proposal proposal) throws ... {
      if (state != STATE_2) throw new IllegalPerformativeException();
      state = STATE_3;
      if (evaluateProposal(proposal))
        ((that.Initiator)outer).acceptProposal(proposal);
      else
        ((that.Initiator)outer).rejectProposal(proposal);
      }
    public final void inform(String s) throws ... {
      if (state != STATE_2) throw new IllegalPerformativeException();
      state = STATE_3;
    }
    public final void refuse() throws ... {
      if (state != STATE_2) throw new IllegalPerformativeException();
      state = STATE_6;
    }
    public final void failure() throws ... {
      if (state != STATE_2) throw new IllegalPerformativeException();
      state = STATE_7;
    }
    abstract protected boolean evaluateProposal(Proposal proposal);
  }
}
```

Fig. 12. The CNP example

```
class Manager implements InitiatorReq{

  definerole Participant extends CNP.Participant {
    protected boolean evaluateTask(Task task){...}
  }
}

class Worker implements ParticipantReq {

  definerole Initiator extends CNP.Initiator{
    protected String performTask(Proposal proposal, Task task){...}
    protected boolean evaluateProposal(Proposal proposal){...}
   }
}

public static void main (String[] args){
  Worker w = new Worker();
  Manager m = new Manager();
  CNP c = new CNP(m,w);
  try{((w.Initiator)m).cfp(...);}
  catch (IllegalPerformativeException e){}; }
}
```

Fig. 13. The CNP example

they participate in a relationship. The state and behaviors which are dynamically acquired are modelled by roles.

Using the language powerJava, a role endowed version of Java, we show how to introduce roles in the two major patterns for modelling relationships: the relationship as attribute pattern and the relationship object pattern. We discuss the pros and cons of both patterns when roles are introduced. In particular, we show that the relationship as attribute pattern extended with roles enables to model the extension of behavior of the objects entering a relationship, without the introduction of a further class modelling the relationship.

The two resulting patterns differ also for the fact that the former emphasise the coupling of the role with the class offering it (e.g., `Student` and `Course`), while the latter emphasise the coupling of the roles with the relationship class and with each other.

Finally we propose a new pattern where both couplings can be considered at the same time: first abstract roles are defined in the relationship object class, which specify the interaction, and then the roles are defined in the classes offering them. This pattern solves the encapsulation problems raised when relationship are introduced in OO.

Future work includes studying how to introduce roles for relationship patterns developed for aspect programming, like the one proposed by Pearce and Noble [12].

REFERENCES

[1] J. Rumbaugh, "Relations as semantic constructs in an object-oriented language." in *Procs. of OOPSLA*, 1987, pp. 466–481.

[2] J. Noble, "Basic relationship patterns," in *Pattern Languages of Program Design 4*. Addison-Wesley, 2000.

[3] G. Bierman and A. Wren, "First-class relationships in an object-oriented language." in *Procs. of ECOOP*, 2005, pp. 262–286.

[4] A. Albano, R. Bergamini, G. Ghelli, and R. Orsini, "An object data model with roles," in *Procs. of Very Large DataBases (VLDB'93)*, 1993, pp. 39–51.

[5] J. Noble and J. Grundy, "Explicit relationships in object-oriented development," in *Procs. of TOOLS 18*, 1995.

[6] N. Guarino and C. Welty, "Evaluating ontological decisions with ontoclean," *Communications of ACM*, vol. 45(2), pp. 61–65, 2002.

[7] F. Steimann, "On the representation of roles in object-oriented and conceptual modelling," *Data and Knowledge Engineering*, vol. 35, pp. 83–848, 2000.

[8] M. Baldoni, G. Boella, and L. van der Torre, "Roles as a coordination construct: Introducing powerJava," *Electronic Notes in Theoretical Computer Science*, vol. 150, no. 1, pp. 9–29, 2006.

[9] ——, "Modelling the interaction between objects: Roles as affordances," in *Procs. of Knowledge Science, Engineering and Management, KSEM'06*, ser. LNCS, vol. 4092. Berlin: Springer, 2006, pp. 42–54.

[10] ——, "Interaction among objects via roles: sessions and affordances in powerjava," in *Procs. of PPPJ '06*. New York (NY): ACM, 2006, pp. 188–193.

[11] ——, "Interaction between objects in powerJava," *Journal of Object Technology*, vol. 6, no. 2, pp. 7–12, 2007.

[12] D. Pearce and J. Noble, "Relationship aspects." in *Procs. of AOSD*, 2006, pp. 75–86.

[13] F. Steimann, "A radical revision of UML's role concept," in *Procs. of UML2000*, 2000, pp. 194–209.

[14] OMG, *OMG Unified Modeling Language Specification, Version 1.3*, 1999.

[15] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Addison-Wesley, 1999.

[16] B. Kristensen and K. Osterbye, "Roles: conceptual abstraction theory and practical language issues," *Theory and Practice of Object Systems*, vol. 2, no. 3, pp. 143–160, 1996.

[17] S. Herrmann, "Object teams: Improving modularity for crosscutting collaborations," in *Procs. of Net.ObjectDays*, 2002.

[18] A. Whitehurst, "Association frameworks in simulation reuse," in *Procs. of OOS*, 1998.

[19] M. Smith and S. Drossopoulou, "Inner classes visit aliasing," in *ECOOP 2003 Workshop on Formal Techniques for Java-like Programming, 2003*, 2003.

XML-based Trust Management in MAS

Agostino Poggi and Michele Tomaiuolo, *University of Parma*

***Abstract*—This work deals with trust management in open and decentralized agent-based environments. Analysis and solutions are geared towards peer to peer networks, intended not just as a technology, but above all as a web of trust relationships, where parties interoperate directly, without reliance on any centralized directory or authority.**

***Index Terms*— Cooperative systems, Security, Public key cryptography, Resource management**

I. Introduction

WHILE a number of architectures and systems are being proposed to deal with the problem of service composition, some issues remain open. In particular, multi-agent systems allow the dynamical and intelligent composition of services by means of delegation of goals and duties among partners. But these delegations can never come into effect, if they're not associated with a corresponding delegation of privileges, needed to access some resources and complete delegated tasks, or achieve desired goals.

This work deals with trust management in open and decentralized agent-based environments. Analysis and solutions are geared towards peer to peer networks, intended not just as a technology, but above all as a web of trust relationships, where parties interoperate directly, without reliance on any centralized directory or authority.

In particular, this paper will show how it is possible to join multiple XML-based certificates in a delegation chain, expressing a degree of trust between two agents. This kind of delegation can allow secure collaboration also among agents who don't have direct acquaintance.

Securing access to the resources made available by the peers is a requirement to make peer to peer networks a more widespread paradigm of cooperation among loosely coupled software agents. The secure management of trust relationships, the ability to precisely control the flow of delegated permissions to trusted entities, are a fundamental requirement to allow the composition of the more disparate services provided on the network.

A. Poggi is with University of Parma, Dipartimento di Ingegneria dell'Informazione, Viale G. Usberti 43100 Parma, Italy (poggi@ce.unipr.it).

M. Tomaiuolo is with University of Parma, Dipartimento di Ingegneria dell'Informazione, Viale G. Usberti 43100 Parma, Italy (tomamic@ce.unipr.it).

II. Rethinking PKI

Traditionally, the problem of identity management was considered equivalent to PKI, and in some sense it is. However, in practice, all efforts to deploy a X.509 [1] infrastructure have all fallen below expectations. Professionals share with users a widespread bad taste about PKI. PKI is expensive and hard to manage, even harder to use for the average human, and implementations lack the broad interoperability the standards promised. [8]

According to a number of proposals (e.g. PolicyMaker [2], KeyNote [3], Simple Distributed Security Infrastructure [4], Simple Public Key Infrastructure [5]) the very foundation of digital certificates needs to be re-thought, trying to make them really useful in application scenarios. The main rationale is that what computer applications need is not to get the real-life identity of keyholders, but to make decisions about them as users. Often these decisions are about whether to grant access to a protected resource or not.

In available PKI systems, these decisions should be taken on the basis of a keyholder's name. However, a keyholder's name does not make much sense to a computer application, other than use it as an index for a database. For this purpose, the only important thing is the name being unique, and being associated with the needed information. Given this reasoning, it is extremely unlikely that the given name by which we identify people could work on the Internet, as it will not be unique.

Moreover, since the explosion of the Internet, contact with person became often only digital, without ever encountering partners personally. In this cases, which are more and more common, there is no body of knowledge to associate with the name. Trying to build an on-line, global database of facts and people is obviously unfeasible, since it will face privacy problems, as well as business unwillingness to disclosure sensible data about their employees and their contacts.

A. Authorization Certificates

In fact, security cannot be founded just on identity, or given names, but more appropriately on principals and authorization. In general, a principal is any entity that can be taken accountable for its own actions in the system, and in particular, principals could be simply thought as entities associated with public keys. The SPKI documentation goes even further, dealing with principals as they "are" public keys. This means that each principal must have its own public key, through which it can be identified, and each public key can be granted rights to access system resources.

The key concept in a trust management system, in fact, is authorization, and more precisely distributed authorization. Each entity in the system has the responsibility to protect its own resources, and it is the ultimate source of trust, being able to refuse or accept any request to access the resource.

On the other end, each entity can access some resources without being listed in a comprehensive Access Control List (ACL).

In fact, relying on local authorities and delegation, ACLs can be relegated to a marginal role, while a central role is played by authorization certificates. A basic authorization certificate defines a straight mapping: *authorization -> key*.

The complete structure of a certificate can be defined as a 5-tuple:

1. *issuer*: the public key (or an hash of it) representing the principal who signs the certificate;
2. *subject*: the public key (or, again, an hash, or a named key) representing the principal for whom the delegation is intended to; other types of subjects are allowed, but they can be always resolved to a public key; for example, a threshold subject can be used to indicate that k of n certificate chains must be resolved to a single subject (i.e. to a public key) to make the authorization valid;
3. *delegation*: a flag to allow or block further delegations;
4. *authorization*: an s-expression which is used to represent the actual permissions granted by the issuer to the subject through the certificate;
5. *validity*: the time window during which the certificate is valid and the delegation holds.

Thus, through an authorization certificate, a manager of some resources can delegate a set of access rights to a trusted entity. This newly empowered principal can, on its side, issue other certificates, granting a subset of its access rights to other entities. When finally requesting access to a resource, the whole certificate chain must be presented. Precise algorithms are presented in the SPKI proposal to combine certificates in a chain and to solve them to an authorization decision.

It can be easily noted that, in the whole process of delegation, identities and given names never appear. Keyholder names are certainly important, and careful identification is obviously a necessary condition before delegation can be granted. Otherwise principals (i.e. public keys) cannot be associated with the humans ultimately responsible for their actions.

But the interesting thing is that this association is never used in the authorization process, as in fact it is not necessary. The result is a radical simplification of the whole security infrastructure. Also, the whole system is much more flexible, allowing arbitrary delegation of permissions and anonymous communications (in the sense that user's identity is never communicated through the network). Above all, trust chains are made part of the system, being its real core, and they can be easily traced following the chains of authorization certificates issued by the involved principals.

B. Name Certificates

The Simple Digital Security Infrastructure (SDSI) [4], which eventually became part of the SPKI [5] proposal, showed that local names could not only be used on a local scale, but also in a global, Internet-wide, environment. In fact local names, defined by a principal, can be guaranteed to be unique and valid in its namespace, only. However local names can be made global, if they are prefixed with the public key (i.e. the principal) defining them.

A convention of SDSI is to give names defined in a certificate a default namespace, being the issuer of the certificate itself. Otherwise local names have always to be prefixed with a public key which disambiguates them. When used in this way, names become Fully Qualified SDSI Names. Compound names can be built by joining local names in a sequence. So, for example, PK1's Joe's Bill can be resolved to the principal named Bill by the principal named Joe by the principal (holding the public key) PK1.

Another type of SPKI certificates is defined for associating names with their intended meaning: *name -> subject*. A SPKI Name Certificate doesn't carry an authorization field but it carries a name. It is 4-tuple:

1. *issuer*: the public key (or an hash of it) representing the principal who signs the certificate;
2. *name*: a byte string;
3. *subject*: the intended meaning of the name; it can be a public key or another name;
4. *validity*: the time window during which the certificate is valid and the delegation holds.

There's no limitation to the number of keys which can be made valid meanings for a name. So in the end, a SPKI name certificate defines a named groups of principals. Some authors [9] interpret these named groups of principals as distributed roles.

C. SAML Overview

The main scope of this work is an integrated environment, where multi-agent systems, as well as systems built on different models and technologies, can interoperate both providing and accessing services. For this purpose, it is also important to use security models which can enable a corresponding interoperability with regard to the management and delegation of privileges, allowing trusted partners to access protected resources even when their particular application is founded on different models and technologies.

The Security Assertion Markup Language [6], being standardized by OASIS, is an open, XML-based format to convey security information associated with a principal. While SAML allows to exploit digital signature and PKI technologies, its specifications are not about the deployment of some PKI, but about their use in a federated environment along with other technologies. The Liberty Alliance, for example, concentrates its work on SSO, to allow the use of

services from different providers without repeating the login operations at every site.

The approach of SAML is radically different from X.509, above all as its specifications start from realistic use cases, which deal with problems that traditional, X.509 based, PKI was never able to solve. The lack of attention to real world cases is probably one of the worst characteristics of X.509. SAML and federated identity, instead, deal with the problem of system security following a bottom-up software engineering approach, taking into account already existing infrastructures.

Instead of defining, and imposing, a top down model, SAML and federated security credentials enable already deployed systems to grow and join others, on the basis of precise and limited agreements. This way, the experience gained in the implementation and deployment of security infrastructures is not lost. On the contrary, it is the basis for the new generation of integrated security systems.

Moreover, SAML is based on XML, and so it easily integrates with Web-services and other XML based applications. It can leverage existing standards and protocols, like XML Digital Signature, XML Encryption, SOAP, WSDL and WS-Security.

SAML itself deals with three different kinds of assertions:

- authentication assertions;
- attribute assertions;
- authorization decision assertions.

Authorization decision assertions are a somehow "frozen" feature in current specifications, suggesting a better solution is to rely on other available standards for security policies, like XACML. A profile to integrate XACML authorization decisions into a SAML assertion has been standardized with SAML 2.0.

The three types of assertions are issued, in principle, by three different authorities. The Policy Enforcement Point (PEP) represent the component of the system which takes care of analyzing provided assertions, and generate authorization decisions, about whether to grant or to deny access to a protected resource.

The generic structure of a SAML assertion makes evident it is very similar to what is usually called a "digital certificate". Like in every other certificate, an issuer attests some properties about a subject, digitally signing the document to prove its authenticity and to avoid tampering. Conditions can be added to limit the validity of the certificate. As usual, a time window can be defined. Moreover, it can e limited to a particular audience or to a one-time use. Conditions can also be put on the use of the certificate by proxies who want to sign more assertions on its basis.

D. SAML from the "Trust Management" Perspective

Being designed to allow interoperability among very different security systems, SAML offers a variety of schemes to format security assertions. In particular, there are a number of possible ways to represent a subject, which also allow to keep away X.500 directories and DN names.

One interesting possibility is to use a SubjectConfirmation object to represent a subject directly by its public key, which resembles the basic concepts of SPKI, where, at the end, principals "are" always public keys.

Thinking about the use of SAML as a representation of SPKI authorization certificates, it would be important to have access rights, or permissions, associated with the subject. Simple authorization decisions could be encoded directly in SAML assertions till version 1.1. In the latest specifications, these assertions are considered "frozen", even if not yet deprecated. However, the very same specifications suggest alternative schemes, first of all integrating an XACML policy into a SAML assertion. The precise way to accomplish this is described in a separate profile, which will be briefly discussed in the following pages.

But, apart from direct delegation of permissions, SPKI-like trust management frameworks can also be used to implement distributed RBAC access control systems, as discussed in [LI2]. For this purpose, local names are particularly important, as they allow each principal to manage its own name space, which, on the other hand, is also one of the foundations of "federated identity" and SAML.

In fact, while SAML allows the use of X.509 distinguished names, it also support a number of other heterogeneous naming schemes. In this sense, its reliance on XML for assertion encoding is not irrelevant, as it provide intrinsic extendibility through schemas and namespaces.

Assigning a local name to a public key, or to a set of public keys, is as simple as defining a role, as in SAML names, and roles, are not considered globally unique by design. And also assigning a named principal to a local name, or to a role, is perfectly possible.

E. XACML Overview

The eXtensible Access Control Markup Language (XACML) [7] is a language for specifying role or attribute based access control policies. It is standardized by the OASIS group and, at the time of this writing, its latest release is 2.0.

A high level model of the XACML language is shown in the following picture. Its main components are:

- *Rule* – the basic element of each policies;
- *Policy* – A set of rules, together with the algorithms to combine them, the intended target and some conditions;
- *Policy set* – A set of policies, together with the algorithms to combine them, the intended target and some obligations.

In particular, each XACML rule is specified through its:

- *target* – indicating the resources, the subjects, the actions and the environment to which the rule applies;
- *effect* – can be Allow or Deny;
- *condition* – can further refine the applicability of the rule.

F. XACML from the "Trust Management" Perspective

As described in the previous sections, trust management is

based on public keys as a mean to identify principals, and on authorization certificates to allow delegation of access rights among principals. SAML defines some rudimentary structures to convey authorization decisions in assertions. However, these structure are not able convey all the information that can be represented using the XACML language. On the other hand, XACML lacks means to protect requests and responses of its Policy Enforcement Points (PEP). It is clear, and so it appeared to both the SAML and the XACML working groups, that the two languages were in many senses complementary, and thus a SAML profile of XACML was defined. It effectively makes the two languages work together in a seamless way.

From the "trust management" perspective, the conjunction of SAML and XACML, in particular the inclusion of XACML authorization decisions into SAML assertions, provides a rich environment for the delegation of access rights. From this point of view, the fact that logic foundations of the XACML language exist is very important, as they provide XACML with a clear semantic. The problem is to find algorithms through which the combination of permissions granted in a chain of certificates could be computed in a deterministic way, as it is already possible in SPKI.

In fact, even if the semantic of a XACML policy is logically sound, nevertheless subtle problems can appear when different policies are linked in a chain of delegation assertions. One major problem is about monotonicity of authorization assertions, which cannot be guaranteed in the general case.

Using XACML authorization decisions as SAML assertions, it is possible to assert that access to a particular resource is denied, instead of allowed. Though being a perfectly legal and meaningful concept, the denial of a permission (a "negative permission") is is not desirable in decentralized environments. In this case, a service provider can never allow access, as it cannot be sure to possess all issued statements. On the other hand, the non-monotonicity of the system can also lead to attacks, as issued assertions can be prevented to reach the provider, this way leading it to take wrong authorization decisions.

Therefore, it is necessary to define a specific profile of SAML and XACML which could enable the secure delegation of permissions in decentralized environments. One of the first requirements is to make "negative permissions" illegal.

III. Implementation

The first step to implement the architecture described in the previous sections consisted in evaluating available software tools which can manipulate SAML and XACML structures. Unfortunately, probably due to the relative of relevant standards (especially for their latest versions), the software park is not particularly vast.

With regards to SAML, the choice fall on the OpenSAML library. In fact, while still being in a development phase, it is the only one supporting all functionalities of SAML 2.0 and, above all, allowing to define new classes with relative simplicity. Extensibility is in fact particularly important, in our case, to realize a "glue" level between SAML and XACML, embodied by the *XACMLPolicyStatement* element.

About XACML, instead, the choice of Sun's XACML Implementation was obliged, in practice, as it's the only valid open source tool to deal with the language. Anyway, a little explaining is necessary. In fact, to handle the version 2.0 of XACML, the CVS version of the library must be used. Such version, anyway, presents a small defect which, at this moment, has not been corrected yet. In fact its APIs miss a method to obtain the list of targets of policies. The author of the library, contacted about this, confirmed the missing and suggested us to implement the method (which we did) waiting for the definitive release of the 2.0 version of the library.

Then, it was decided to give a standard structure to our library, realizing its API like a Java security provider. The Java Cryptographic Architecture (JCA) foresees in fact the possibility to realize packages, called security provider, which provide JDK with a concrete implementation of a subset of Java cryptograohic functionalities. For developers wanting to use the library, the main advantage of this choice is the availability of a set of API with a well known and collaudated structure. Moreover, this will allow the use of certificates and paths which will be realized with normal Java API, without duplicating their functionalities. In fact, in principle any component (also external ones), operating on a Java certificate, will be able to operate on SPKI certificate of the new library, too.

To realize an extension of the Cryptographic Architecture (JCE), first of all it was necessary to extend Java basic data types, which in our case are represented by certificates and paths; then engine classes had to be realized, which specify algorithms to be implemented. Finally, a master class for the provider had to be implemented, which is necessary to register new classes and allow them to be used by Java.

A. Certificates

To represent certificates, Java cryptographic APIs define an abstact class: *Certificate.* Within it, all basic methods to manage public key certificates can be found.. Extending this class, an abstract class, *SPKICertificate*, has been realized, containing common methods of name certificates and authorization certificates.

In particular, the *SPKINameCertificate* class extends *SPKICertificate* and describes a name certificate. Within it, all methods need to set the certificate subject are present, in all three possible forms: public key, namespace qualified local name and hash of a public key. Different get methods exist, corresponding to each type of subject to obtain. Moreover, the *getPublicKey()* method, defined by Java API, also returns the subect's public key.

Two methods, *getStatedName()* and *setStatedName()*, allow instead to set and read the SPKI name field, i.e. the name associated to the certificate subject by the certificate creator. In both methods just the local part of the name is needed, as the issuer's namespece is taken as a default.

```
1.public void sign( KeyPair keys, SignatureParameters params )
2.{
3.  // Create the SignatureBuilder and build the signature
4.  Signature sign = (Signature) buildSAMLObject( Signature.DEFAULT_ELEMENT_NAME );
5.
6.  // Set up signature parameters
7.  sign.setSigningKey( keys.getPrivate() );
8.  sign.setSignatureAlgorithm( params.getAlgorithm() );
9.  sign.setCanonicalizationAlgorithm( params.getCanonicalization() );
10.
11.  // Add the public key to the signature
12.  KeyInfo keyInfo = (KeyInfo) buildSAMLObject( KeyInfo.DEFAULT_ELEMENT_NAME );
13.
14.  keyInfo.setPublicKey( keys.getPublic() );
15.  sign.setKeyInfo( keyInfo );
16.
17.  // Link the signature and the assertion togheter
18.  SAMLObjectContentReference contentReference = new SAMLObjectContentReference( assertion );
19.  sign.getContentReferences().add( contentReference );
20.  assertion.setSignature( sign );
21.
22.  // Delete del old marshalled assertion
23.  marshalledAssertion = null;
24.}
```

Fig. 1. A code snippet regarding certificate signature.

The *SPKIAuthorizationCertificate* class also extends *SPKICertificate*, and allows to create and manage authorization certificates.

Its most important method is *addPolicy()*, which allows to add a security policy to the certificate (in fact, according to SAML specifications, it's possible to add more security policies to a single certificate). Such policy has to be represented by a class implementing the simple *AuthorizationPolicy* interface, whose only needed functionality is to convert the policy to a Policy object, as defined by Sun's XACML Implementation. The *getXACMLPolicies()* method allows to obtain the security policies contained in the certificate, represented as Sun's XACML classes.

The *allowDelegation()* method, instead, allows to indicate an explicit delegation, as defined in the SPKI proposal. The opposite method, *denyDelegation()*, is useful only in the case you want to remove the delegation attribute from a certificate, where it's present. In fact, in the case *allowDelegation()* is not explicitly invoked, the delegation is not active.

Finally, the *getPublicKey()* method is also present, because it is required by the Certificate abstract class; however, as an explicit key is not normally present in this type of certificate, to be associated with the subject, the method always returns null.

B. Certificate Path Validation

An algorithm to evaluate the correctness of a certificate chain is described in the original SPKI proposal. To this aim, Java APIs define the *CertPathValidator* class. By means of it, through the implementation of its *validate()* method, the validity of a chain can be controlled.

Similarly to what happens for the creation of certification paths, the validation operation requires, as parameters, the *CertPath* that is meant to be verified and an implementation of the *CertPathParameters* interface, describing all parameters needed by the validation algorithm. The result of the operation will be eventually represented by an implementation of the *CertPathValidatorResult* interface.

Thus, a subclass of *CertPathValidator* had to be developed, implementing the SPKI validation algorithm. Parameters of the validation process are represented as *ValidatorParameters* objects, containing the list of keys trusted by the principal operating the verification, and possibly additional parameters.

The result of the process, in the case of SPKI certificates, is represented by a set of roles associated with the public key which terminates the chain. The information is included into a *KeyRoles* object.

The other operation to be offered by the library is that of validating a request to access a local resource. The request itself is represented by an instance of the *AuthorizationRequest* interface. Users of the library can provide different implementations of the interface, according to their needs.

It must be noted, however, that the authorization request must be formulated in compliance with the structure of security policies. Thus, the implementations of *AuthorizationPolicy* and *AuthorizationRequest* must use the same types and the same identifiers. For this reason, the

SimpleAuthorizationRequest has been implemented, as a request corresponding to security policies in use by the library.

Apart from the request, the algorithm with the list of authorization certificates to use and the list of trusted keys needed during the certificate verification process must be provided. Finally, in the case some additional conditions exist, it could be necessary to specify additional parameters for the verification process.

The validation happens through the creation of a Policy Decision Point (PDP). The Sun's XACML library provide the methods for creating such a decision block. However, to be able to obtain all needed policies, to validate the request, the PDP class of XACML uses various finder modules allowing to retrieve information. It was thus necessary to develop a finder module, called *AuthzPolicyFinderModule*, which is in charge of retrieving policies from authorization certificates provided as parameters.

During the process of creation of a PDP it is possible to insert additional finder modules. Such modules can be specified in the phase of construction of the *AuthorizationEvaluator* object and allow to extend the object's capabilities to search for information. Moreover, this way it is possible to provide the validation module with a series of local policies which are not stored within SPKI authorization certificates.

The final result of the operation is a list of *AuthorizationResponse* objects, one for each resource which was asked to be accessed. Each instance contains in its structure an identifier of the resource which it refers to, a decision value and a status code.

C. RAIS

As a first test, the library has been integrated into RAIS, a distributed system developed at University of Parma. RAIS (Remote Assistant for Information Sharing) is a peer-to-peer and multi-agent system composed of different agent platforms connected through the internet. Each agent platform acts as a "peer" of the system and is based on three agents: a personal assistant, an information finder and a directory facilitator; moreover, another agent, called personal proxy assistant, allows a user to remotely access her/his agent platform.

The RAIS system has been designed and implemented taking advantage of agent, peer-to-peer, information retrieval and security management technologies and, in particular, of three main software components: JADE, JXTA and Google Desktop Search.

Using the security library described here, a RAIS user can not only provide the permission to access his own files, but can also assign the permission to upload a new version of one or more existing files. In this case the PA informs his/her user about the updated files the first time he/she logs in. This functionality can be useful for the members of a workgroup involved in common projects or activities. Basic versioning capabilities are planned to be added to the Distribute Desktop Search system in the near future.

IV. CONCLUSION

Federated identities and security assertions are a novel technique to loosely couple already existing security systems, without requiring to design and deploy a whole new one, which could hardly fit the extremely heterogeneous variety of goals and requirements the different applications have.

Particular attention deserve SAML and XACML, for their wide applicability, their intrinsic extensibility, and their XML grounding, which allows them to easily fit into the existing web-based applications, as well as into new systems based on web or grid services. While being proven to have sound grounding in logical models, anyway they can be used in a distributed environment only under some restrictions. Otherwise, the combination of different assertions and policies could lead to unexpected results, or, even worse, expose the system to attacks.

REFERENCES

[1] Housley, R. et al. Internet X.509 PKI Certificate and CRL Profile. IETF RFC 3280, April 2002. http://www.ietf.org/rfc/rfc3280.txt.

[2] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management". In Proc. of the 17th Symposium on Security and Privacy. 1996, pp. 164-173, IEEE Computer Society Press.

[3] M. Blaze, J. Feigenbaum, J. Ioannidis and A. Keromytis. 1999. "The KeyNote Trust-Management System Version 2." RFC 2704.

[4] Rivest, R.L., Lampson, B. SDSI - A Simple Distributed Security Infrastructure. http://people.csail.mit.edu/rivest/sdsi11.html.

[5] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T. SPKI certificate theory. IETF RFC 2693, September 1999.

[6] OASIS Security Services (SAML) TC. http:// www.oasis-open.org/committees/security/.

[7] OASIS eXtensible Access Control Markup Language (XACML) TC. http://www.oasis-open.org/committees/xacml/.

[8] Lewis, J. "Reinventing PKI: Federated Identity and the Path to Practical Public Key Security". 1 March 2003. Available from: http://www.burtongroup.com/.

[9] Li, N. Local names in SPKI/SDSI. In Proc. 13th IEEE Computer Security Foundations Workshop, pages 2--15. IEEE Press, 2000.

[10] Li, N., Grosof, B. A practically implementable and tractable delegation logic. Proc. 2000 IEEE Symposium on Security and Privacy (Oakland CA, May 2000), 29-44.

Preserving player's goals: a choreography-driven matchmaking approach

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

***Abstract*—An agent interaction protocol, a service choreography, can quite naturally be interpreted as an alliance of parties, which cooperate to achieve a goal. On the other hand, each participant entered the alliance moved by goals of its own, which it would like to fulfill by playing one of the roles. The achievement of the shared and of the specific goals depend both on the interaction schema, that is captured by the choreography, and on the participant's capabilities, where by this word we mean the skills of the participant, the actions that it can execute. We show in this paper that the choice of which capabilities to use cannot rely totally on local criteria, as instead it is commonly done by the approaches to matchmaking, but it must take into account the choreography/protocol. This happens whenever the match is not exact, e.g. when plugin match is used. We also describe an extended plugin match that takes into account also the constraints given by the choreography for performing the capability selection.**

I. Introduction

Web services have a platform-independent nature, that endeavors enterprises to develop new business processes by combining existing services, retrieved over the web. Web service composition is still much of a costly and manual process, which is made more and more difficult by the growing width of the space to search. Hence, the need of methods for reducing the search space and for making compositions in an automatic way. This direction has been suggested in [1], where a UML specification of a business process was used to abstract the description of a composition away from the specification of the actually composed services. This abstract specification defined a *model*, used for driving the retrieval and the composition task. The idea of capturing the overall schema of interaction of a set of entities is exploited also in other areas, like multi-agent systems. In this context, the role of the abstract specification is played by the so-called "interaction protocol" [2], while actions take the place of services..

Kolp, Giorgini, and Mylopoulos have investigated [3] the possibility of using real world organizational structures as a metaphor for defining MAS architectures. The derived architectures are evaluated w.r.t. a set of quality attributes, amongst which predictability and adaptability. One of the studied human organizational structures is "strategic alliance". A strategic alliance links specific facets of a group of organizations and is defined with the purpose of achieving an overall, shared goal. The organizations within the alliance, however, remain independent and have control over the assigned tasks. Each actor, however, has to reconcile and adjust its own views with the policies of the organization. This is particularly true in the case of "co-optation", a special kind of strategic alliance, in which the partners become part of a newly founded organization, hence having, on the one hand, a commitment on pursuing the alliance goals and interests, and, on the other hand, their own and specific goals to reach.

Among the many quality attributes that the authors define, the following are particularly interesting w.r.t. our work.

- *Coordinability*: agents are not really useful if they cannot coordinate. Coordination is used to distribute expertise, information, etc. among agents, which depend on one another. *Cooperativity* is a form of coordination. Cooperation is achieved either communicative or non-communicative.
- *Modularity*: it increases the efficiency of task execution, results in higher flexibility and reduces the communication overhead, although, on the other hand it constrains inter-module communication.
- *Predictability*: agents have many degrees of freedom in the way they undertake action, in their domains. The capability of predicting the behaviour of the individuals is important when we need to aggregate such individuals in an organization.

It is quite natural, then, in the case of multi-agent systems and of (web) services, to interpret an interaction protocol, or a choreography, as the specification of an alliance (in particular, a joint venture or a co-optation), because they specify a coordination pattern based on communication. Roles can be considered as modules that capture an activity within the schema, constraining the interaction of the partners. The fact that a partner takes a role in a choreography guarantees that the partner will behave as expected (predictability of the behavior). The choreography/protocol can be seen as an alliance of independent partners. This alliance is aimed at pursuing a goal, that is subscribed by all the participants, but each partner has also *its own* goals, that motivate its taking part to the alliance. The achievement of the shared goal and of the specific agent's goal not only depends upon the choreography-given schema of interaction but also on the skills that each

agent has, i.e. by each agent's specific *capabilities*. Indeed, every agent has control over the ways for accomplishing the assigned tasks. Thus, before an agent subscribes the alliance, there is a need to check if its capabilities match with those that are requested by a role, i.e. if its capabilities allow it to achieve its goal in the context of the given choreography. It is implicit that the choreography (the protocol) specify in some way the necessary capabilities. In [4], we have shown that there is the need of enriching the choreography specification by introducing the concept of *capability requirement*. A capability requirement expresses an operation that a peer should be able to perform at some specific point of the choreography.

(Web) services share many facets with multi-agent systems [5]. The introduction of the concept of "choreography" (and of languages like WS-CDL [6]) has opened new perspectives on the way an abstract specification of a system of services should be described. Choreographies can be used to build policies that some peer will execute. In order for a policy to be "playable" by a peer, the peer must have the requested capabilities. For instance, it must have some means for producing or retrieving (e.g. by contacting a third service) a piece of information to send. This approach can be extended by considering different kinds of actions (e.g. communicative actions) or a different granularity (e.g. agents, services, or other software components). More in details, a role specification in a choreography can be used to produce a *policy skeleton*, which is to be completed by *substituting* capabilities to capability requirements, so to make it executable. The substitution can be defined by applying a *matching process* between the abstract specification given by capability requirements and the available capabilities. In general, it is unlikely to have capabilities that perfectly match the requirements; the retrieval process will identify capabilities which *slightly differ* from the specification. If one wants to use them anyway, rather than writing new software, it is necessary to verify that the policy obtained after the substitution still allows the achievement of the goal, which is not granted anymore [4].

The task of retrieving capabilities that match given requirements is analogous to the task of service discovery. We can, then, think to use the same techniques, e.g. [7], [8], [9], [10]. In particular, in this work we focus on the matches proposed by Zaremski and Wing in their seminal work [8], where various kinds of relaxed match are proposed. We show that none of the matches (but the so-called exact pre/post match) guarantee that a synthesized policy, in which capabilities have been selected according to them, will still allow to reach the goal of interest. The reason is that they take into account only the "local" information given by the capability requirement and do not consider constraints posed by the choreography ("global" constraints). We also show how to integrate the *plugin* match in the context given by a choreography in such a way that the goal is preserved by the substitution.

The article is organized as follows. In Section II we recall the matches introduced in [8], and explain their relations with a choreography and with the goals. Section III introduces a simple representation for services and choreographies, that is based on a declarative language. Section IV shows that the local matches alone do not guarantee the preservation of the goal, and it also shows how to integrate the plugin match so to produce substitutions that preserve the goal. We will introduce the notion of *conservative substitution*. Conclusions and related works end the paper.

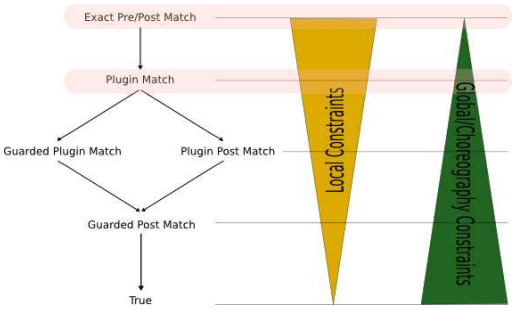


Fig. 1. The lattice of the different local matches: on top the strongest. Our claim is that the local and global constraints are related; the stronger the local match, the weaker the global constraints.

II. Capability match: local vs. global properties

We suppose, in the line of previous work [4], that choreographies are enriched with additional descriptions of those actions, that peers must be able to perform for playing roles (*capability requirements*). Capability requirements are used to select the specific capabilities that are necessary to build an executable policy. As mentioned in the introduction, this selection can be done by applying matching techniques that are analogous to those used for service discovery. Zaremski and Wing [8] propose a formal specification to describe the behavior of software components. Each software component has precondition S_{pre} and postcondition S_{post} written as predicates in first-order logic. Requirements are coherently specified as having precondition R_{pre} and postcondition R_{post}. Five kinds of relaxed match between R and S are defined:

- EM (*Exact Pre/Post Match*): $R_{pre} \Leftrightarrow S_{pre} \wedge R_{post} \Leftrightarrow S_{post}$
- PIM (*Plugin Match*): $R_{pre} \Rightarrow S_{pre} \wedge S_{post} \Rightarrow R_{post}$
- POM (*Plugin Post Match*): $S_{post} \Rightarrow R_{post}$
- GPIM (*Guarded Plugin Match*): $R_{pre} \Rightarrow S_{pre} \wedge ((S_{pre} \wedge S_{post}) \Rightarrow R_{post})$
- GPOM (*Guarded Post Match*): $((S_{pre} \wedge S_{post}) \Rightarrow R_{post})$

Exact pre/post match states the equivalence of R and S. *Plugin match* is weaker: S must only be behaviorally equivalent to R when plugged-in to replace R. *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. The different matches can be organized according to a lattice [8], that we have reported in Fig. 1.

In our application domain, R will be a capability requirement, while S will be a capability. Capability requirements

are contextualized in some choreography. Capabilities are specific software components and depend on the player of a choreography role: they are matched against requirements in the process of checking if a player can play a certain role, by selecting –at the same time– its right capabilities. In general, since the final aim is software reuse, it will be quite difficult to retrieve an exact match for a capability requirement. More likely (and more interesting) is the case when one of the other four degrees of match hold.

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, a capability) independently of the context of usage (in our work, the service choreography). In other words, the software specification must respect some constraints. Relaxing the exact match means relaxing these "local constraints" (see Fig. 1). On the other hand, a choreography defines the *global execution context*, in which capability requirements are immersed. Intuitively, the selection of a capability (for replacing a capability requirement) should *preserve* those properties of the choreography that motivated its choice, in particular, the *goal* for which it was chosen. In the case of the *exact match*, the whole verification is done *locally*. Due to the fact that it is a kind of equivalence, matching exactly a requirement is a sufficient condition to preserve the goal. As we will see in Section IV, the other kinds of match do not give this guarantee. It becomes, therefore, necessary to *add some constraints* by using the available source of global information: the choreography.

Our claim (see Fig. 1) is that the more relaxed is the local match, the stronger should be the compensation supplied at the global level. The extreme is given by the bottom of the lattice: the match that returns always *true*. In this case, the choice of the capabilities could be performed, for instance, by randomly choosing capabilities and by substituting the to the requirements while simulating the execution of the policy. When the goal is not verified by the current choice, a *backtracking* mechanism allows the revision. The whole process *relies on* the choreography. Checking global constraints can be expensive but it is possible to reduce the costs by limiting the attention to those capability requirements which belong to the execution traces, which actually allow to achieve the goal.

III. Reasoning about capabilities

For what concerns the representation of choreographies and specific peers, in order to abstract from the specific language (e.g. WS-CDL, WSDL) and from the details of the implementation, we adopt a *declarative* representation and focus on the study of the properties of interest.

Each choreography is made of a set of *interacting roles*. It can be described as a set of subjective views of the interaction that is encoded, each corresponding to one of the roles. We call the implementation of each role a *policy*. We will represent both *roles* and policies by means of the *declarative language* DyLOG [11], by interpreting interactions among services, capabilities and capability requirements as *actions*, and by using *reasoning about actions* for making predictions about the effects of role and policies executions.

DyLOG has been developed as a language for programming agents and is based on a logical theory for reasoning about actions and change in a modal logic programming setting. DyLOG is equipped with a communication kit for dealing with interactions, and has already been used for customizing Web service composition [12]. An agent's behavior is described in a non-deterministic way by giving the set of actions that it can perform. Each action can have preconditions to its application and cause some effects. Given this view of actions, we can think to the problem of reasoning as the act of building or of traversing a sequence of transitions between *states*. A state is a set of *fluents*, i.e., properties whose truth value can change over time. Such properties encode the information that flows during the execution of the agent actions. In DyLOG we do not assume that the value of each fluent in a state is known: it is possible to represent unknown fluents and to reason about the execution of actions on incomplete states. We introduced an epistemic operator $\mathcal{B}_i$, to represent the beliefs that an entity i has about the world: $\mathcal{B}_i f$ means that the fluent f is believed to be true by the entity i, $\mathcal{B}_i \neg f$ means that the fluent f is believed to be false. A fluent f is undefined, $u_i(f)$, when both $\neg\mathcal{B}_i f$ and $\neg\mathcal{B}_i \neg f$ hold. Thus each fluent in a state can have one of the three values: *true*, *false* or *unknown*.

In a DyLOG description of a service role (or policy) the interactions between the service and its interlocutor(s) can be defined in terms of communicative actions performed by the service (*speech acts*) and *get-message actions*. A *speech act* is an atomic action of form *performative(sender, receiver, content)*, where *performative* is the kind of speech act (e.g. inform), *sender* and *receiver* are the name of the interacting peers, while *content* is a *fluent literal* representing the piece of information that is passed by its execution. The set of all performatives is supposed to be *shared by the two parties*. *Get-message* actions allow to represent the reception of information and to reason about the outcome of the speech acts performed by the interlocutor. The range of possible incoming speech acts is supposed to be finite: the interlocutor is supposed to use a performative out of a finite and predefinite set to produce its answer within a choreographed interaction. *Capability requirements*/*capabilities* in a service role/policy are represented as (possibly communicative) atomic actions.

Complex behaviors can be specified in DyLOG by means of *procedures*, Prolog-like clauses built upon the other kind of actions mentioned. We represent the behavior of both *roles* and *policies* by DyLOG procedures [1]. Intuitively, a *role* is a procedure that combines speech acts, get-message acts, *capability requirements* and procedure calls, and a *policy* is a procedure combining speech acts, get-message acts, *capabilities* and procedure calls.

A *role* in a choreography can, therefore, be specified as a quadruple of the form $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$, where:

[1]Since our focus is to study the preservation of global properties, we will assume that the sets of terms used for representing speech acts and capabilities are the same in the choreography and in the peer description.

1) $\mathcal{S}_\mathcal{A}$ is a set of *speech acts*, represented as [2]:

$$\mathsf{performative}(sender, receiver, l) \quad \textbf{causes} \ \{E_1, \ldots, E_n\}$$
$$\mathsf{performative}(sender, receiver, l) \quad \textbf{possible if} \ \{P_1, \ldots, P_t\}$$

where E_i, and P_j are respectively: the fluents that are obtained as effect of the speech act, and the precondition to the execution of the performative.

2) $\mathcal{G}_\mathcal{A}$ is a set of *get-message actions*, they are represented as: $\mathsf{receive\_act}(receiver, sender, [l_1, \ldots, l_n])$ **receives** $\mathcal{I}$, where $\mathcal{I}$ is a set of alternative speech act, that can be received by the executor of receive_act; each speech act in $\mathcal{I}$ has an element in $[l_1, \ldots, l_n]$ as content.

3) $\mathcal{CR}$ is a set of capability requirements, they are modeled as atomic actions and are represented as:

$$c \ \textbf{causes} \ \{E_1, \ldots, E_m\}$$
$$c \ \textbf{possible if} \ \{P_1, \ldots, P_t\}$$

where c is the name of the required capability and the semantics of the clauses is the same as above. We will use the functions $\mathsf{Effs}(c) = \{E_1, \ldots, E_m\}$ and $\mathsf{Precs}(c) = \{P_1, \ldots, P_t\}$ to return the effects and the preconditions of c. The same functions apply also to speech acts.

4) $\mathcal{P}$ encodes the behavior for the role; it is represented as a collection of clauses of the kind p_0 **is** $p_1, \ldots, p_n$ $(n \geq 0)$, where p_0 is the name of the procedure and p_i, $i = 1, \ldots, n$, is either an atomic action, a *get-message* action, a test action, or a procedure name (i.e. 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.

Policies are defined in a way that is analogous to role descriptions. Let $\mathcal{C}$ be the set of capabilities of a peer, then, a *policy* is quadruple $P_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P} \rangle$, where $\mathcal{S}_\mathcal{A}$, $\mathcal{G}_\mathcal{A}$, and $\mathcal{P}$ are defined as above.

Example 1: As an example, let us introduce a choreography (enriched with capability requirements) that rules a simple room reservation protocol with two roles: the *buyer* wants to book a room at the hotel managed by the *seller*. Figure 2 depicts the interaction between the two roles: first the buyer sends to the seller the date for the room reservation; then, the seller must have the capability of performing a *reserveRoom* action, and inform the buyer about the room price. The buyer checks the price, by performing an *evaluatePrice* action. Then, it informs the seller about the results of this evaluation: it can either decide to refuse the offer and conclude the interaction or it can inform the seller about the desired payment mode (*cash* or *credit card*). At this point, the seller must have the capability

[2]In DyLOG the semantics of speech acts is inspired to the standard semantics of FIPA Communicative Acts [13]. Therefore speech acts are characterized in terms of (a) feasibility preconditions denoting the ability of the speaker to perform the act and (b) the desired and rational *perlocutionary effects* of the utterance. See [11] for more details.

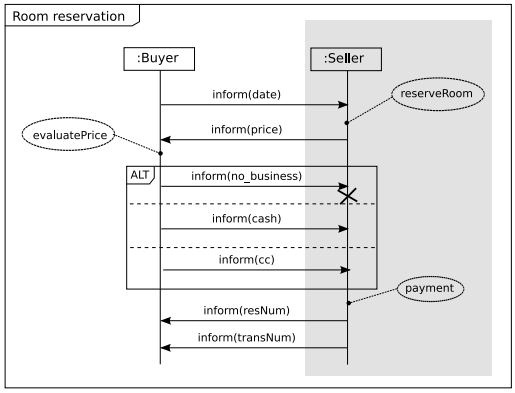


Fig. 2. The Room Reservation Protocol, represented by means of UML sequence diagrams, and enriched with capability requirements (oval elements).

of performing the *payment* action, and finalize the business transaction. Finally it notifies the buyer the reservation and transaction numbers.

Let us focus on the *seller* role description [3] $R_{seller} = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$, where $\mathcal{P} = \{\mathsf{booking}, \mathsf{finalize\_reservation}\}$, $\mathcal{S}_\mathcal{A} = \{\mathsf{inform}(s, b, price), \mathsf{inform}(s, b, resNum), \mathsf{inform}(s, b, transNum)\}$, $\mathcal{G}_\mathcal{A} = \{\mathsf{receive\_date}(s, b, date), \mathsf{receive\_evaluation}(s, b, [no\_business, cash, cc])\}$, $\mathcal{CR} = \{\mathsf{reserve\_room_{CR}}, \mathsf{payment_{CR}}\}$. The procedures in $\mathcal{P}$ are described by the following clauses:

$\mathsf{booking}$ **is** $\mathsf{receive\_date}(s, b, date)$,
  $\mathsf{reserve\_room_{CR}}, \mathsf{inform}(s, b, price)$,
  $\mathsf{receive\_evaluation}(s, b, [no\_business, cash, cc])$,
  $\mathsf{finalize\_reservation}$
$\mathsf{finalize\_reservation}$ **is** $\mathcal{B} no\_business?$
$\mathsf{finalize\_reservation}$ **is** $\mathsf{payment_{CR}}, \mathsf{inform}(s, b, resNum)$,
  $\mathsf{inform}(s, b, transNum)$

The get_message actions in $\mathcal{G}_\mathcal{A}$ are described by:

$\mathsf{receive\_date}(s, b, date)$ **receives** $[\mathsf{inform}(b, s, date)]$
$\mathsf{receive\_evaluation}(s, b, [no\_business, cash, cc])$
  receives $[\mathsf{inform}(b, s, no\_business)$ **or** $\mathsf{inform}(b, s, cash)$ **or** $\mathsf{inform}(b, s, cc)]$

The capability requirements in $\mathcal{CR}$:

$\mathsf{reserve\_room_{CR}}$ **causes** $\{\mathcal{B} price\}$
$\mathsf{reserve\_room_{CR}}$ **possible if** $\{\mathcal{B} date\}$
$\mathsf{payment_{CR}}$ **causes** $\{\mathcal{B} transNum, \mathcal{B} resNum\}$
$\mathsf{payment_{CR}}$ **possible if** $\{\mathcal{B} PcashSupported, \mathcal{B} PccSupported\}$

Finally, the semantics of the $\mathsf{inform}(sender, receiver, l)$ actions in $\mathcal{S}_\mathcal{A}$ and $\mathcal{G}_\mathcal{A}$ is given by the rules (for more details see [12]):

$\mathsf{inform}(s, b, l)$ **possible if** $\{\mathcal{B}_s l\}$
$\mathsf{inform}(s, b, l)$ **causes** $\{\}$
$\mathsf{inform}(b, s, l)$ **possible if** $\{\}$

[3]In the following examples all the beliefs refer to the seller mental state, thus, for sake of readability we will omit to index the modal operator $\mathcal{B}$.

$\mathsf{inform}(b, s, l)$ **causes** $\{\mathcal{B}_s l\}$

Intuitively, the first two clauses state that I (the seller) can execute an inform act only if I believe l; the execution of the action will modify the interlocutor's mental state, while do not have any effects on my mental state. The last two clauses describe what happen in my mental state when I am the receiver of the information. In this case, since I am not the actor, the action of informing is considered *always* executable; moreover I will adopt l as my own belief. ■

In DyLOG, it is possible to perform a form of reasoning known as *temporal projection*, by means of *existential* queries of the form: Fs **after** p, where p is a policy name and Fs is a conjunction of fluents. Checking if a formula of this kind holds corresponds to answering the query "Is there an execution trace of p that leads to a state in which Fs is true?". By execution trace we mean a sequence of atomic actions, i.e. speech acts and capabilities (capability requirements). When the answer is positive, such sequence is a plan to bring about Fs. This plan can be *conditional* because whenever a *get-message* action is involved none of the possible answers from the interlocutor can be excluded. In other words, we will have a different execution branch for every option.

Let us consider a role description $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$. We can apply temporal projection to $\mathcal{P}$ to find 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 DyLOG query: Fs **after** p, where Fs is the set of fluents that we want to be true after the execution of p. Given a state S_0, containing all the fluents that we know as being true in the beginning, we will denote the fact that G is successful in R_d by:

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

The execution of the above query returns as a side-effect an *execution trace* σ of p. The execution trace σ can either be *linear*, i.e. a terminating sequence $a_1, \ldots, a_n$ of atomic actions, or it can be *conditional*, when the procedure contains get-message actions. Intuitively, by this mechanism it is possible to verify, by reasoning about the choreography, if the role allows for an execution after which a condition of interest holds.

Example 2: In the context of the Example 1, let us consider the goal:

$$G = \{\mathcal{B}transNum, \mathcal{B}resNum\} \ \textbf{after} \ \mathsf{booking}$$

where the initial state S_0 contains the fluents $\{\mathcal{B}PcashSupported, \ \mathcal{B}PccSupported\}$, while all the other fluents are unknown. There are two possible execution traces that lead to a state where G holds, hereafter we report one of them:

$\sigma = \mathsf{inform}(b, s, date); \mathsf{reserve\_room}_{\mathsf{CR}};$
$\mathsf{inform}(s, b, price); \mathsf{inform}(b, s, cc); \mathsf{payment}_{\mathsf{CR}};$
$\mathsf{inform}(s, b, resNum); \mathsf{inform}(s, b, transNum).$

■

A *policy* can be built from a *role description* by substituting capability requirements with a set of capabilities of a peer that should play the role. If we denote by $\mathcal{C}$ the capabilities of the peer, by $\mathcal{CR}$ the capability requirements, and by θ the substitution $[\mathcal{C}/\mathcal{CR}]$, the policy built from the role description $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$ will be $P_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta \rangle$. Given a policy description $P_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta \rangle$, a goal $G = Fs$ **after** p, and an initial state S_0, we can verify if G is successful in P_d by:

$$(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta \rangle, S_0) \vdash G$$

Intuitively, this allows to verify, by reasoning about the peer description, if the policy allows for an execution that brings about the condition of interest.

IV. Choreography-driven match

When the matching process is applied for selecting a capability that is part of a role specification, the desire is that the selected capability preserves the properties of the specification. Generally, the matchmaking process will result in a set of alternative θ_i because each capability requirement has a set of matching capabilities. The selected θ not only must satisfy the matching rules but it must also be *conservative*, i.e. it must guarantee that those *goals*, that can be achieved by reasoning on the *role specification*, will be achieved also after the *substitution*. Then, the following implication must hold:

Definition 1 (Conservative substitution): Let $\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$ be a role description, S_0 the initial state, and G the goal of interest. Suppose that the following relation holds:

$\exists \sigma, \theta = [\mathcal{C}/\mathcal{CR}_\sigma], \ \mathcal{CR}_\sigma \subseteq \mathcal{CR} \ s.t.$
$(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle, S_0) \vdash G$ w.a. $\sigma \Rightarrow$
$(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta \rangle, S_0) \vdash G$ w.a. $\sigma\theta$

where σ is an execution trace which makes the goal true when reasoning at the level of the choreography, and θ is a substitution $\mathcal{CR}_\sigma \rightarrow \mathcal{C}$, where $\mathcal{CR}_\sigma \subseteq \mathcal{CR}$, $\mathcal{CR}_\sigma = \{cr \in \mathcal{CR} \mid cr \text{ occurs in } \sigma\}$. In this case, the substitution θ is conservative.

Notice that we are interested in a substitution θ that involves only the capability requirements contained in the execution trace σ, which is, therefore, used to select the requirements to be matched. The substitution θ is obtained by applying one of the matching rules, described in Section II, that we here rephrase as follows (c represents a single capability and cr a single capability requirement):

- EM (*Exact Pre/Post Match*): $\mathsf{Precs}(cr) = \mathsf{Precs}(c) \wedge \mathsf{Effs}(cr) = \mathsf{Effs}(c)$
- PIM (*Plugin Match*): $\mathsf{Precs}(cr) \supseteq \mathsf{Precs}(c) \wedge \mathsf{Effs}(c) \supseteq \mathsf{Effs}(cr)$
- POM (*Plugin Post Match*): $\mathsf{Effs}(c) \supseteq \mathsf{Effs}(cr)$
- GPIM (*Guarded Plugin Match*): $\mathsf{Precs}(cr) \supseteq \mathsf{Precs}(c) \wedge ((\mathsf{Precs}(c) \cup \mathsf{Effs}(c)) \supseteq \mathsf{Effs}(cr))$
- GPOM (*Guarded Post Match*): $((\mathsf{Precs}(c) \cup \mathsf{Effs}(c)) \supseteq \mathsf{Effs}(cr))$

For short, we will respectively denote by θ_{EM}, θ_{PIM}, θ_{POM}, θ_{GPIM}, θ_{GPOM}, the substitutions obtained by applying the five degrees of match. For simplicity we will call a substitution obtained by applying the plugin match a PIM substitution, the one obtained by applying Exact Pre/Post match an EM

substitution, and so on for the other kinds. It is immediate to see that any substitution, obtained by applying the *exact pre/post match*, satisfies Definition 1. In other words, the local constraints are sufficient to guarantee the property (see Fig. 1). However this is not true for the other kinds of match.

Theorem 1: The class of PIM, POM, GPIM and GPOM substitutions are not conservative.

Proof: The proof is given by a counterexample.

Let us consider a role description $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$, where $\mathcal{P} = \{p \ \textbf{is} \ cr_1, \ a\}$, $\mathcal{S}_\mathcal{A} = \{a\}$, $\mathcal{G}_\mathcal{A}$ is empty, and the capability requirement cr_1 in $\mathcal{CR}$ and the speech act a in $\mathcal{S}_\mathcal{A}$ are described by [4]:

cr_1 **causes** $\{\mathcal{B}l_1\}$ $\quad$ a **causes** $\{\mathcal{B}l_2\}$
cr_1 **possible if** $true$ $\quad$ a **possible if** $\{\mathcal{B}l_1, \mathcal{B}l_3\}$

Assuming as goal $G = \mathcal{B}l_2$ **after** p, where the initial state contains $\mathcal{B}l_3$ while all the other fluents are unknown, the reasoning process will generate the execution trace $\sigma = cr_1; a$ for achieving G. If we consider the set of capabilities $\mathcal{C} = \{c_1\}$:

c_1 **causes** $\{\mathcal{B}l_1, \mathcal{B}\neg l_3\}$
c_1 **possible if** $true$

By applying the substitution $\theta = \{[c_1/cr_1]\}$ we obtain the new policy $\mathcal{P}\theta = \{p \ \textbf{is} \ c_1, \ a\}$. However, by using this policy, the query $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta_{PIM} \rangle, S_0) \vdash G$ does not succeed: in fact, the additional effect $\mathcal{B}\neg l_3$ of the capability c_1 inhibits the executability of the speech act a. On the other hand, it is easy to check that θ is an instance of all the kinds of substitutions that we have listed, i.e. it is a PIM substitution as well as a POM substitution, a GPIM and a GPOM substitution. ■

This example witnesses that working at the level of the local constraints is not sufficient. Our claim is that, in general, in order for a substitution to be conservative, it must take into account not only the *local* aspects but also the *overall structure*, encoded by the choreography. The locality of the matches used in the matchmaking phase, indeed, seriously limits the possibility of re-using software (services) by selecting and composing it in an automatic way.

Let us now focus on the *plug-in match* (PIM), which is one of the most used and which immediately follows the exact match in the lattice (therefore it is the strongest of the flexible matches). We show that, by introducing appropriate constraints at the level of the choreography, it is possible to guarantee the selection of conservative substitutions. To this aim, we take into account the *dependencies* between actions, which produce as effects fluents, that are used as preconditions by subsequent action. Intuitively, the idea is to verify that the "causal chain" which allows the execution of the sequence of actions, is not broken by the differences between capabilities and capability requirements, as instead happens in the example. 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 of the role specification, which allows the achievement of the goal. We will use this trace for defining the additional properties for the match.

Let us start by introducing the notions that define dependencies between actions and dependency sets for fluents. Consider a role description $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$ and suppose that, given the initial state S_0, the goal $G = Fs$ **after** p succeeds, thus obtaining as answer the successful sequence of actions $\sigma = a_1; a_2; \ \ldots; a_n$, which is an execution trace of p.[5] We denote by $\overline{\sigma}$ the sequence of actions $a_0; a_1; a_2; \ldots; a_n; a_{n+1}$, where a_0 and a_{n+1} are two *fictitious* actions 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 $\mathsf{Effs}(a_0) = S_0$, and that a_{n+1} has no effect but $\mathsf{Precs}(a_{n+1}) = Fs$.

Consider two indexes i and j, such that $j < i$, $i, j = 0, \ldots, n+1$. We say that *in* $\overline{\sigma}$ the action 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 $\mathcal{B}l \in \mathsf{Effs}(a_j)$, $\mathcal{B}l \in \mathsf{Precs}(a_i)$, and there is not a k, $j < k < i$, such that $\mathcal{B}l \in \mathsf{Effs}(a_k)$. Given a fluent $\mathcal{B}l$ and a sequence of actions 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 $[c/c_r]$ be a specific substitution of a capability requirement with a capability, that is contained in θ_{PIM}, we say that a fluent $\mathcal{B}l \in \mathsf{Effs}(c) - \mathsf{Effs}(c_r)$ (i.e. an additional effect of the capability c w.r.t. the effects of the capability requirement cr) is an *uninfluential fluent* w.r.t. the sequence $\sigma\theta_{PIM}$ iff for all pairs $(j, i) \in \mathsf{Deps}(\mathcal{B}\neg l, \sigma)$, identifying by k the position of c_r in σ, we have that $k < j$ or $i \leq k$, Intuitively, this means that the fluent will not break any dependency between the actions 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 $\sigma\theta_{PIM}$ have the same length and are identical as sequences of actions but for the fact that in the latter capabilities substitute capability requirements. For this reason, we can reduce to reasoning on σ for what concerns the action positions. A substitution θ_{PIM} is called *uninfluential* iff for any substitution $[c/c_r]$ in θ_{PIM}, all beliefs in $\mathsf{Effs}(c) - \mathsf{Effs}(c_r)$ are uninfluential fluents w.r.t. σ. Now we are in position to prove that a substitution which exploits the *plugin match* and which is also *uninfluential*, is conservative.

Theorem 2: Let G be a goal and let $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$ a role description. If $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle, S_0) \vdash G$ w.a. σ and there is an uninfluential substitution $\theta_{PIM} = [\mathcal{C}/\mathcal{CR}_\sigma]$, $\mathcal{CR}_\sigma \subseteq \mathcal{CR}$ then $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta_{PIM} \rangle, S_0) \vdash G$ w.a. $\sigma\theta_{PIM}$.

Proof: The proof is by absurd and it uses the proof theory introduced in [11]. Let us assume that $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle, S_0) \vdash G$ w.a. σ but $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta_{PIM} \rangle, S_0) \nvdash G$ w.a. $\sigma\theta_{PIM}$. Since, by hypothesis, for any substitution $[c/c_r]$ in θ_{PIM}, $\mathsf{Effs}(c) \subseteq \mathsf{Effs}(cr)$ holds, there exists a fluent F such that $a_0, a_1, \ldots, a_{i-1} \vdash F$ but $(a_0, a_1, \ldots, \ a_{i-1})\theta_{PIM} \nvdash F$, where $\sigma = a_0, a_1, \ldots, a_{i-1}, a_i, \ldots, a_n$ and $F \in \mathsf{Precs}(a_i)$. Now, since $a_0, a_1, \ \ldots, a_{i-1} \vdash F$, there exists $j \leq i - 1$,

[4] In the following, for the sake of readability, we will omit the indexing of the modal operator $\mathcal{B}$ when it is clear that the beliefs belong to the same role.

[5] In this work we focus on linear plans. Conditional plans can be tackled by considering each path separately.

such that $a_0, a_1, \ldots, a_j \vdash F$ and $F \in \mathsf{Effs}(a_j)$ but $(a_0, a_1, \ldots, a_j)\theta_{PIM} \nvdash F$, that is $F \notin \mathsf{Effs}(a_j\theta_{PIM})$. This is absurd due to the hypothesis that θ_{PIM} is an uninfluential substitution. ■

Example 3: Let us refer to the running example introduced in Section III and let us consider the set of capabilities $\mathcal{C} = \{\mathsf{reserve\_room_{C1}}, \mathsf{reserve\_room_{C2}}, \mathsf{payment_C}\}$:

$\mathsf{reserve\_room_{C1}}$ **causes** $\{\mathcal{B}\neg PccSupported, \mathcal{B}price\}$
$\mathsf{reserve\_room_{C1}}$ **possible if** $\{\mathcal{B}date\}$
$\mathsf{reserve\_room_{C2}}$ **causes** $\{\mathcal{B}freeDinner, \mathcal{B}price\}$
$\mathsf{reserve\_room_{C2}}$ **possible if** $\{\mathcal{B}date\}$
$\mathsf{payment_C}$ **causes** $\{\mathcal{B}transNum, \mathcal{B}resNum\}$
$\mathsf{payment_C}$ **possible if** $\{\mathcal{B}PcashSupported, \mathcal{B}PccSupported\}$

By choosing the *plugin match* as matching rule, there are two possible substitutions called θ'_{PIM} and θ''_{PIM} respectively:

$\theta'_{PIM} = \{[\mathsf{reserve\_room_{C1}}/\mathsf{reserve\_room_{CR}}], [\mathsf{payment_C}/\ \mathsf{payment_{CR}}]\}$,
$\theta''_{PIM} = \{[\mathsf{reserve\_room_{C2}}/\mathsf{reserve\_room_{CR}}], [\mathsf{payment_C}/\ \mathsf{payment_{CR}}]\}$.

While $\mathsf{payment_C}$ exactly matches $\mathsf{payment_{CR}}$, $\mathsf{reserve\_room_{C1}}$ and $\mathsf{reserve\_room_{C2}}$ slightly differ from the requirement. By applying the substitution θ'_{PIM} we obtain the set of policies $\mathcal{P}\theta'_{PIM}$:

booking **is** $\mathsf{receive\_date}(s, b, date), \mathsf{reserve\_room_{C1}}, \mathsf{inform}(s, b, price), \mathsf{receive\_evaluation}(s, b, [no\_business, cash, cc]), \mathsf{finalize\_reservation}$
$\mathsf{finalize\_reservation}$ **is** $\mathcal{B}no\_business?$
$\mathsf{finalize\_reservation}$ **is** $\mathsf{payment_C}, \mathsf{inform}(s, b, resNum), \mathsf{inform}(s, b, transNum)$

Differently than in Example 2, by using the resulting policies, the query $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta'_{PIM}\rangle, S_0) \vdash G$ does not succeed: in fact, the additional effect $\mathcal{B}\neg PccSupported$ of the capability $\mathsf{reserve\_room_{C1}}$ inhibits the executability of the capability $\mathsf{payment_C}$. On the other hand, we observe that the application of the other substitution θ''_{PIM}, provides the agent with a set of policies ($\mathcal{P}\theta''_{PIM}$) that allows to satisfy the query $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta'_{PIM}\rangle, S_0) \vdash G$. Thus, θ''_{PIM} represents an *uninfluential substitution.* ■

The verification that a substitution is uninfluential involves the derivation σ, and it is based on checking whether the chains of dependencies between actions for the various fluents are not interrupted by some opposite fluent. Obviously, if the domain is such that no fluent, once asserted, can be negated, any θ_{PIM} will be conservative. This can be verified statically on the choreography and the set of capabilities, by checking that every fluent (that appears as effect of some action) is always positive or negative, including the initial state and the goal in the verification. Indeed, the application domains in which actions produce *knowledge* are of this kind. One example is given by e-learning applications where the capabilities supply knowledge elements that are either supplied or used as prerequisites.

V. Conclusions and related works

In this work we have studied the relation between the matchmaking and the achievement of a goal in an interaction ruled by a choreography. We have proved that local matches (but the exact match) do not preserve the goal when capabilities are substituted to capability requirements. It is necessary to introduce a verification that involves the choreography definition. We argue that the more relaxed are the local matches, the stricter must be the the global verification. As an example, we have presented the integrated approach in the case for the plugin match.

In the agent framework, the adoption of an interaction policy has been proposed in CooBDI and Coo-AgentSpeak [14], [15]. These works extend the BDI (*Belief, Desire, Intention*) model in such a way that agents are enabled to exchange plans. This mechanism is activated when the agent cannot find a plan, for pursuing a goal of interest, by just exploiting its own capabilities. The ideas behind the CooBDI theory have been implemented by means of web services technologies, leading [16] to the development of CooWS agents. Another recent work is the one by [17]. Here, in the setting of the DALI language, agents can cooperate by exchanging sets of rule that can either define a procedure, or constitute a module for coping with some situation, or be just a segment of a knowledge base. Agents have reasoning techniques that enable them to evaluate how useful the new knowledge is. Nevertheless, these techniques cannot be directly imported in the context of service-oriented computing. The reason is that, while in agent systems it is not a problem to discover during the interaction that an agent does not own all the necessary actions, in service composition it is necessary that all the actors are known before the interaction takes place.

In [18] (inspired by JACK [19] and extended in [20]), the term "capability" is used for identifying the "ability to react rationally towards achieving a particular goal" in the BDI framework. An agent has the capability to achieve a goal if its plan library contains a plan for reaching the goal. Therefore, an agent's goals and intentions are constrained to be compatible with its capabilities.

For what concerns (web) services and matchmaking, it is not easy to be exhaustive. The matches proposed in [8] have inspired most of the semantic matches for web service discovery. Amongst them, Paolucci et al. [9] propose four degrees of match (exact, plugin, subsumes, and fail) that are computed on the ontological relations of the outputs of an advertisement for a service and a query.

WSMO (Web Service Modeling Ontology) [10] 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 [21] an approach that is based on [8] and on [22], which, in turn, is based upon [9]. More recently, a WSMO matchmaker has been proposed in [23], which combines several aspects: type matching, relation matching, constraint matching, parameter matching, intentional matching. Last but not least, in [7] a

multi-level evaluation model is proposed, for deciding whether two services are composable. This is done through four levels of control (quality, dynamic semantics, static semantics, and syntax). Dynamic semantics is the name given to the matches of [8]. None of these approaches relates the matching with the possible context of application of the sought services, even WSMO which, as a framework, includes the possibility of composing orchestrations of services. On the other hand, so far we have not yet tackled the integration of ontological reasoning in our work. This is surely an interesting extension that we will face soon, given that all these proposals as well as ours have the same kernel, and we expect similar results.

The idea of synthesizing a policy from an abstract specification (a choreography) is also stated in [24], where it is observed that services are often conceived so as to be delivered individually, while there is a growing need of reusing this software, either by composing services or by tailoring a composition to some specific client. In [25] a tool for service (activity in the paper) coordination and evaluation is introduced, based on the MetaFrame open tool coordination environment. Differently than in our approach, there is no specification of a choreography as we have used here but the desired behavior is given in terms of global constraints. Temporal logic is used to express both the constraints and the goal to achieve, enabling the automatic synthesis of a composition of activities.

Finally, works like [26], [27] propose approaches for goal-driven service composition based on planning. However, this task is accomplished without reference to any choreography. In particular, in [26] the composition phase and the semantic reasoning phase (carried on on inputs and outputs) are separated and the latter is performed on a local basis only.

ACKNOWLEDGMENT

This research has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net), and it has also been supported by MIUR PRIN 2005 “Specification and verification of agent interaction protocols” national project. Claudio Schifanella is partially supported by the fellowship program “Fondazione CRT - Progetto Lagrange” (cf. http://www.progettolagrange.it).

REFERENCES

[1] B. Örriens, J. Yang, and M. Papazoglou, “Model driven service composition,” in *ICSOC 2003*, 2003.

[2] M. P. Huget and J. Koning, “Interaction Protocol Engineering,” in *Communication in Multiagent Systems*, ser. LNAI 2650. Springer, 2003, pp. 179–193.

[3] M. Kolp, P. Giorgini, and J. Mylopoulos, “Multi-agent architectures as organizational structures,” *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 1, 2006.

[4] 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*, 2007, to appear.

[5] M. Singh and M. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley and sons, Ltd., 2005.

[6] WS-CDL, “http://www.w3.org/tr/ws-cdl-10/.”

[7] B. Medjahed and A. Bouguettaya, “A multilevel composability model for semantic web services,” *IEEE Trans. on KDE*, vol. 17, no. 7, pp. 954–968, 2005.

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

[9] 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.

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

[11] 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

[12] 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.

[13] F. for Intelligent Physical Agents, “FIPA communicative act library specification, 2002.” [Online]. Available: http://www.fipa.org/repository/aclspecs.html

[14] D. Ancona and V. Mascardi, “Coo-BDI: Extending the BDI Model with Cooperativity,” in *Proc. of the 1st Declarative Agent Languages and Technologies Workshop (DALT’03), Revised Selected and Invited Papers*, J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, Eds. Springer, 2004, pp. 109–134, lNAI 2990.

[15] D. Ancona, V. Mascardi, J. F. Hübner, and R. H. Bordini, “Coo-AgentSpeak: Cooperation in AgentSpeak through Plan Exchange,” in *Proc. of AAMAS 2004*. ACM press, 2004, pp. 698–705.

[16] L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta, “CooWS: Adaptive BDI agents meet service-oriented computing,” in *Proceedings of the Int. Conference on WWW/Internet*, 2005, pp. 205–209.

[17] A. T. S. Costantini, “Learning by knowledge exchange in logical agents,” in *Proc. of WOA 2005: Dagli oggetti agli agenti, simulazione e analisi formale di sistemi complessi*, F. Corradini, F. De Paoli, E. Merelli, and A. Omicini, Eds. Camerino, Italy: Pitagora Editrice Bologna, november 2005.

[18] L. Padgham and P. Lambrix, “Agent capabilities: Extending BDI theory,” in *AAAI/IAAI*, 2000, pp. 68–73. [Online]. Available: citeseer.ist.psu.edu/625805.html

[19] P. Busetta, N. Howden, R. Ronquist, and A. Hodgson, “Structuring bdi agents in functional clusters,” in *Proc. of the 6th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL99)*, 1999.

[20] V. Padmanabhan, G. Governatori, and A. Sattar, “Actions made explicit in BDI,” in *Advances in Artificial Intelligence*, ser. LNCS, no. 2256. Springer, 2001, pp. 390–401.

[21] 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.

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

[23] F. Kaufer and M. Klusch, “WSMO-MX: A logic programming based hybrid service matchmaker,” in *Proc. of ECOWS’06*. IEEE Computer Society, 2006, pp. 161–170.

[24] F. Casati and M. Chien, “Dynamic and adaptive composition of e-services,” *Information Systems*, vol. 26, pp. 143–163, 2001.

[25] B. Steffen, T. Margaria, and V. Braun, “The electronic tool integration platform: Concepts and design.” *STTT*, vol. 1, no. 1-2, pp. 9–30, 1997.

[26] M. Pistore, L. Spalazzi, and P. Traverso, “A minimalist approach to semantic annotations for web processes compositions.” in *ESWC*, 2006, pp. 620–634.

[27] 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.

simpA-WS: A Simple Agent-Oriented Programming Model & Technology for Developing SOA & Web Services

Alessandro Ricci, Enrico Denti

***Abstract*— Service-Oriented Architecture (SOA) is more and more recognised by the industry as the reference blueprint for building inter-operable, distributed enterprise applications based on open standards such as Web Services (WS). In the current state-of-the-art, the programming models for engineering SOA systems proposed by the leading industries are essentially *component-based* – typically, rooted in object-oriented abstractions and technologies. On the side, such a choice benefits from the well-know advantages of component-based software engineering and from the maturity of the available technologies; on the other, however, the abstraction level provided is inadequate to model some fundamental SOA aspects – such as autonomy, control-uncoupling, data-driven interaction, activities – as first-class concepts. Such features can be modelled quite naturally by adopting an *agent-oriented* perspective.**

In this paper we describe simpA-WS, a Java-based framework for developing SOA/WS applications which adopts an agent-oriented programming model based on the general-purpose *Agents and Artifacts* meta-model (A&A). simpA-WS makes it possible to conceive, design and program services (and applications using services) as workspaces where ensemble of pro-active, activity-oriented entities (agents) work together by exploiting different kinds of passive function-oriented entities (artifacts) used as resources, along with tools to support their business activities. Accordingly, we first present the simpA-WS framework and the related simpA language – an extension of Java aimed at capturing the A&A metaphors as first-class entities; we then show how agents and artifacts can be programmed in simpA and how SOA/WS applications can be programmed in simpA-WS; a simple running example is discussed for concreteness.

I. Introduction

Nowadays Web Services (WS) represent the reference standard technologies for setting up distributed systems that need to support interoperable machine-to-machine interaction between heterogeneous applications distributed over a network [17]. In that context, Service-Oriented Architecture (SOA) appears to be more and more the reference software architecture promoted by leading industries—IBM, Microsoft, Sun, IONA, Bea, to cite few ones—as a blueprint for organising, designing and building distributed enterprise applications based WS open set standards [2], [4].

Generally speaking, SOA can be defined as an open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, inter-operable, discoverable, and potentially reusable services [4]. From a software architecture perspective, SOA defines how to use loosely coupled software services for supporting the requirements of the business and software users, making resources available on a network as independent services that can be accessed without knowing their implementation platform. From an information systems perspective, SOA enables the creation of applications by combining loosely-coupled, inter-operable services. Despite the specific perspective, service-oriented architectures based on Web Services are well going to be adopted by the industry as the reference choice for inter-operable, distributed systems.

A key issue here is the *programming model* to be adopted for SOA applications [7] – that is, the model defining the concepts and abstractions made available to developers. From this viewpoint, SOA *per-se* is not committed to any specific programming model: however, the ones currently promoted by leading software vendors are essentially *component-based* [16], so as to rely on mature and widespread technologies. Yet, in this paper we argue that such a choice is unable to handle some essential requirements of SOA systems—such as autonomy, control-uncoupling and data / message-driven interactions—at a suitable abstraction level. For this reason, there is a need for a programming model based on *agent-oriented abstractions*, which makes it possible to deal with such requirements in an effective and more natural way.

Agents and Multi-Agent Systems (MAS) have already been recognised as suitable approaches for engineering complex, intelligent service-oriented applications, aimed at integrating research outcomes from different contexts such as Semantic Web and Artificial Intelligence [10]. Here, however, we explicitly focus on designing and programming issues, discussing how agents and MAS could provide effective building blocks for the design and development of SOA applications.

The remainder of the paper is organised as follows. In Section II we focus on the requirements that any programming model for SOA application programming should satisfy, and briefly review from this viewpoint the main programming models currently promoted by the industry. Then, in Section III we introduce an agent-oriented programming model for SOA/WS based on agents and artifacts. In Section V we present simpA-WS, as a simple Java-based middleware for supporting such a programming model; conclusions are drawn in Section VI.

II. Background: SOA Programming Models

In order to evaluate the effectiveness of a programming model for SOA, it is first necessary to outline the main properties that a SOA system should exhibit according to the reference literature (see for instance [2], [4]).

Encapsulation is the basic property for achieving service independency from the context: services encapsulate their logic, whose size and scope can vary, and can possibly encompass the logic provided by other services; in other words, one or more services can be composed into a collective service.

Autonomy is strongly related to encapsulation, since services must clearly have control over the logic they encapsulate. As

A. Ricci is with the DEIS Department, Università di Bologna, Cesena.
E. Denti is with the DEIS Department, Università di Bologna.

a consequence of such an autonomy, inter-service relationships should minimise dependencies – in particular, control dependencies – retaining only the *awareness* of each other: this is what we mean by *loose coupling*. Such an awareness is achieved through the use of *service descriptions*, which are exploited by users to understand how to use and interact with the service. Communication is another fundamental dimension in SOA, since services must exchange information in order to interact and accomplish their task.

Autonomy, encapsulation and loose coupling properties clearly condition the *interaction model* that can be used to enable communication both between the service user and service providers, and among services. In principle, any interaction model capable of preserving loosely coupled relationship can be adopted: *messaging* is the reference communication framework typically considered for this purpose. Conversely, interaction models based on Remote Procedure Call (RPC) or method invocation are inadequate, since they involve a *control coupling* between the interacting parts. This is indeed quite a critical aspect: most of the frameworks currently proposed as killer technologies for the rapid prototyping of Web Service applications adopt a pure OO-style in defining and interacting with Web Services, mapping – for instance – service invocations onto method invocations. A clear example of this trend is the programming model adopted by the Java API for XML Web Services (JAX-WS) [8], which defines a Web Service by simply annotating the corresponding Java class class with the @WebService annotation, and its methods – which implementing the Web Service operations – with the @WebMethod annotation. An analogous support can be found in the Web Service Extension (WSE) provided by the Microsoft .NET platform.

Our view is that the above critical aspect is mainly due to a fundamental mismatch between the SOA and object-orientated paradigm – with the object-oriented paradigm often adopted to engineer distributed (and concurrent) systems – rather than to weaknesses in today's technologies. In fact, although it is possible to build such kinds of systems on top of available OO platforms exploiting middleware such as CORBA, RMI or alike, the *abstraction level* provided is inadequate for application design and implementation, in that OO lacks abstractions to deal with loose-coupled communication, concurrency, and distribution.

Consequently, new programming models are needed for implementing SOA systems, which preserve the basic properties required from Web Services. For this purpose, some proposals have been pushed by leading industries in the state-of-the-art: Service Component Architecture (SCA) [5], for instance, is promoted by independent software vendors such as IBM, SAP, IONA, Oracle, BEA, TIBCO — to cite some. Analogous initiatives are the Windows Communication Foundation (previously called Indigo), promoted by Microsoft, and the Java Business Integration (JBI), promoted by the Java Community process [9]. All such approaches adopt a *component-based* programming model: components implement the business logic, offer their capabilities to other components, and consume functions offered by other components through suitable Service-Oriented interfaces (Figure 1 shows

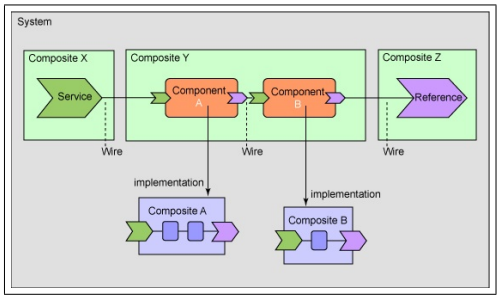


Fig. 1. An abstract representation of the Service Component Architecture, reported in [5].

abstract representation of the Service Component Architecture, taken from [5]) using a minimum of middleware APIs. Components are linked together according to some *wiring model*, which is meant to support different kinds of interaction models and features, including synchronous and asynchronous invocation, transactional behaviour of components invocation, and so on. Service implementation and service composition are uncoupled from both the details of the infrastructure and of the access methods used for service invocation: these typically include Web services, Messaging systems and CORBA IIOP.

As it can be expected, such an approach inherits on the one side the well-known strong points of the component-oriented paradigm in terms of dynamic configurability, reusability, etc., but also its weakness in dealing with processes and activities, concurrency, autonomy, distribution, decentralisation and encapsulation of control – to cite some. Neither object-oriented, nor component-based programming models provide first-class abstractions to explicitly model and manage the above issues: in particular, both objects and components are passive entities encapsulating their state and behaviour, but not the control of such a behaviour, which is typically hidden in some part of the component's "container" – whatever this may be. As a consequence, even if components are meant in principle to encapsulate the business-level logic, they fail to encapsulate some key aspects of such a logic – such as, for instance, the execution and control of (possibly concurrent, possibly interacting) business activities and processes. To overcome these limitations, in the next section we introduce a programming model based on *agent-oriented* abstractions, aimed at capturing the above aspects in a full-fledged way. ***troppo forte?***

III. An Agent-Oriented Programming Model for SOA and Web Services

Interestingly, the word *agent* appears both in the abstract description of the Web Service reference architecture provided by W3C [17] (sketched in Figure 2), and – more generally – in the high level characterisation of SOA [4]. There, an agent is used to represent:

- the *service requestor*, which encapsulates the business logic on how to *use* services: from an interaction point of view, this results in sending and receiving messages in compliance with the service interface specification;

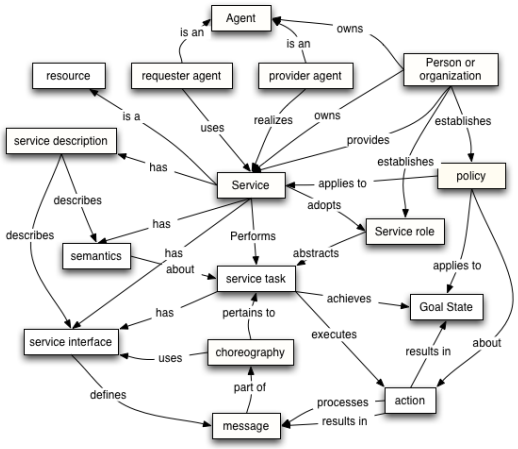


Fig. 2. Service Model of Web Services, according to W3C

- the *service provider*, which encapsulates the business logic of the service: this processes the requestor messages, executes the related activities and interacts with the requestor via the message exchange protocol specified in the service description.

So, some notion of agent already appears in the standards as a key part of the picture, representing the entities that perform some activity or achieve some goal, thus shaping the business logic either on the user's or on the service's side. However, such abstraction level disappears when moving from the abstract characterisation down to the design and development levels, as discussed in the previous section. Our proposal is to *keep that abstraction level alive* throughout the engineering process, exploiting agents and MAS as the basic bricks of a programming model explicitly tailored to the definition of services and of applications using such services.

The fundamental outcome of this approach is to reduce the gap between the business-level description and the models and architectures used at the system implementation level.

In fact, despite the differences between the existing agent-oriented methodologies, models and architectures, the agent-oriented paradigm *in se* provides precisely the high-level concepts — activity, goal, task, message-driven interaction, ...— that are needed from a programming model in order to map the metaphors used at the business description level. In the next Section we introduce an agent-oriented programming model called SA&A, based on a the A&A conceptual model.

A. The A&A Conceptual Model

A wide range of agent programming models, architectures and platforms can be found in literature (see [6] for a brief survey of the programming languages and platforms). For historical reasons, most of them are AI-oriented, thus with a characterisation of the agent and MAS abstractions focussed on AI concepts, aimed at building systems exhibiting a somewhat intelligent behaviour. The *Agents and Artifacts* conceptual model (A&A henceforth) [12], instead, was defined with a software engineering perspective in mind: as such, it highlights the features needed for an effective design and development of complex software systems.

Grown from inter-disciplinary studies involving Activity Theory and Distributed Cognition [11], A&A adopts *agents* and *artifacts* as high-level abstractions to design and build distributed, concurrent software systems. These metaphors are taken from human cooperative working environments, where "systems" are composed by individual autonomous entities (humans) who pro-actively carry on some kind of work (activities) by interacting and cooperating. A fundamental aspect of such cooperative systems is the context—i.e. the *environment*—that makes it possible for such activities to take place. Humans cooperative environments are full of suitable artifacts and tools, that humans produce, consume and use to support their work. Following Activity Theory, the term *artifact* is used here to identify both the resources and objects constructed during the activities, as well as whatever instrument built or exploited by humans to support their activities.

In A&A these metaphors are brought into the software engineering process, modelling complex software systems in terms of *workspaces* where ensembles of pro-active entities—the agents—work together by producing, consuming, sharing and cooperatively using different kinds of artifacts, analogously to the human case.

B. Agents and Artifacts

Agents represent entities with a (pro-)active behaviour, designed by engineers so as to perform some kind of useful work, cooperatively and concurrently to the work of the other agents. The agent abstraction is well suited for encapsulating the execution and control of the business activities and processes that are part of the business logic. Artifacts, in turn, represent passive entities that populate the agents' working environment: they are designed by engineers as resources and tools to be used by agents for their (individual or collective) work.

So, on the agent side, A&A promotes an *activity-oriented* model, where the agents' pro-active behaviour is modelled in terms of activities whose execution and control is fully encapsulated inside the agent; on the other, agents manipulate, produce, exploit, update artifacts which constitute the tools needed for their work.

Activities are expressed in terms of *actions*, that is atomic step determining some kind of change either in the agent state (internal actions) or in the environment (external actions or simply actions). *Sensing*—representing here the action of perceiving —is the basic mechanisms that enables an agent to get information from its environment.

Artifacts are used by agents as source or target of their work, and greatly vary nature and function — including, for instance, the tools for enabling agent communication and coordination such as blackboards, message boxes, and calendars, which are typical *coordination artifacts* [13]. Instead, shared knowledge bases or artifacts representing or wrapping I/O devices are typical examples of *resource artifacts*. Each artifact is explicitly designed by MAS engineers to encapsulate some kind of

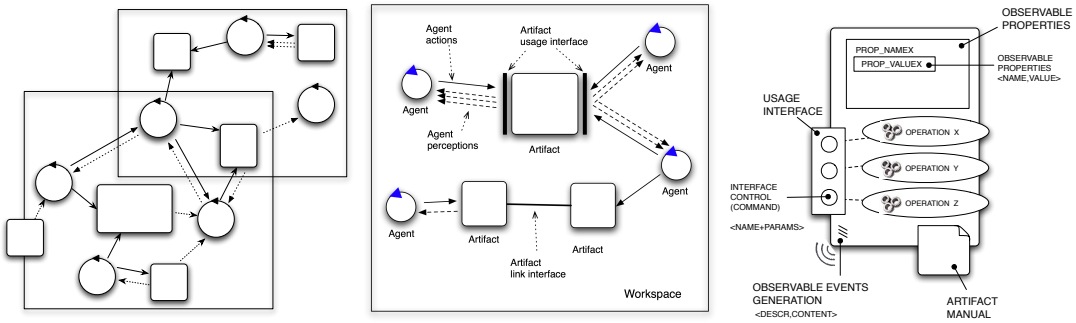


Fig. 3. *(Left)* An abstract representation of an application according to the A&A programming model, as a collection of agents (circles) sharing and using artifacts (squares), grouped in workspaces. *(Center)* An abstract representation of an agent, as an entity executing actions and getting perceptions from the environment where it is logically situated. *(Right)* An abstract representation of an artifact, with its usage interface and observable properties in evidence.

function, here synonym of "intended purpose"; any function is structured into a set of *operations*. In order to be used by agents, each artifact exposes a *usage interface*, which defines the set of controls on which the agents can act upon so as to trigger and control the execution of operations. Such execution can result in the generation by the artifact of *observable events*, that can be perceived by the agents which are using the artifact. Usage interface controls have a name and possibly parameters, which must be specified by agents when using the artifact.

Summing up, the interaction between agents and artifacts is based on the notions of *use* and *observation*, and strictly mimics the way in which humans use their artifacts. As a simple example, a coffee machine is an artifact whose usage interface provides controls to make coffee and select the sugar level, and whose state and behaviour are observable by the generation of events exposed through a display.

Artifacts can also be composed together by means of *link interfaces*, which make it possible to create complex artifacts as dynamic compositions of existing simpler artifacts.

Although a detailed description of A&A is outside the scope of this paper (interested readers are referred to [12]), our aim here is to identify some essential properties that make it an interesting reference for a SOA programming model. First, the agent abstraction explicitly enables and captures the *encapsulation* of control, along with a notion of *autonomy* as depicted by SOA requirements. Moreover, the interaction model adopted for agents and artifacts interaction is strongly uncoupled and data-oriented (vs. control oriented), thus providing for *uncoupled and data-driven interaction*: in fact, there are no flows of control from an agent to an artifact or other agents, as it happens instead in the case of Remote Procedure Calls (RPC) or classical object-oriented method invocation. Finally, *concurrency* can be naturally modelled both in the form of concurrent activities carried on by an individual agent, and as separate works carried on independently by distinct agents (*seamless concurrency support*).

C. A SOA/WS Programming Model Based on A&A

In this section we introduce a basic programming model for SOA based on A&A abstractions, referenced in the following as SA&A. Both services and service-user applications in SA&A are uniformly modelled as a workspace where an ensemble of agents work together, interacting both via direct communication and by producing, consuming, sharing and cooperatively using a dynamic set of artifacts. Agents encapsulate the responsibility of the execution and control of the business activities that characterise the SOA specific scenario, while artifacts encapsulate the business resources and tools needed by agents to operate in the application domain. Figure 4 represents an abstract picture of a (web) service designed upon the SA&A programming model: agents act as service providers processing incoming service messages, but some of them also act as clients of other services, as an example of service composition.

Two kinds of artifacts are used in almost any service-oriented application: ws-service-panel and ws-service-interface. Both are used as *interfaces* or *media* enabling the communication with the service clients or with external Web services, based on open standards. In particular, the former is used by agents implementing the business logic to retrieve and be aware of the requests and messages sent to the (web) service: so, for each service, one instance of such an artifact encapsulates the functionalities related to a specific WSDL and WS-Policy service description. In the simplest model, the usage interface of this artifact provides just controls to manage messages and requests—for instance, a control to retrieve the messages to be processed, another to send response messages, one further to check the number of pending messages, etc. In more complex models, however, this artifact could encapsulate the management of some quality-of-service aspects (such as security, reliability, etc.), as defined by WS-* specification. The latter one, instead, is used by agents to interact with an existing service. So, an instance of the ws-service-interface is usually first instantiated referring to a specific WSDL and possibly WS-Policy description, and then used (by one or more agents) to interact with a specific Web Service. In the simpler case, its usage interface should provide controls just to invoke services and observe possible response messages. However, other functionalities could be encapsulated here for the management of QoS aspects (described in the WS-Policy

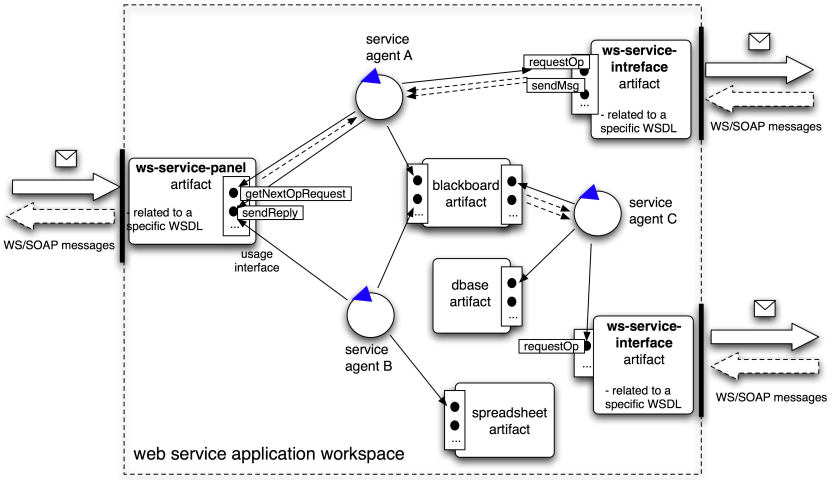


Fig. 4. An abstract representation of a (web) service architecture according to the SA&A programming model, composed by agents and artifacts building blocks. Artifact usage interface is represented as a panel with some controls inside. Some of them are labelled with a name which is equal to the operation that the control is meant to trigger. The ws-service-panel and ws-service-interface artifacts in the figure are used respectively to collect request messages and to interact with existing (web) services.

specification).

In the abstract representation in Figure 4, other kinds of general purpose artifacts are represented, such as a shared knowledge base, a blackboard, a spreadsheet. Specific kind of artifacts could be instantiated dynamically, or disposed of, according to the evolution of the service provision. Two remarks are worth before closing this section. First, the picture refers to the service side of a SOA: the client-side would be similar, yet with no the need for ws-service-panel artifacts[1] Moreover, only the basic aspects of a service have been presented, since our aim is to give the reader the "taste" of the shift from state-of-the-art, component-based to agent-oriented approaches, rather than developing a full-fledged application scenario.

IV. Programming Agents and Artifacts in simpA

simpA is an open-source extension[2] of the Java platform aimed at assuming the A&A abstractions as the basic high-level building blocks to program concurrent applications [15]. This approach contrasts with most of the current approaches, which often model concurrency aspects by "adapting" object-oriented abstractions (classes, objects, methods)—e.g. [3]. Rather, we introduce the new A&A abstractions, and exploit real object-orientation to model any basic low-level data structure used to program agents and artifacts, as well as any information exchanged through interaction. This approach leaves concurrency and high-level organisation aspects orthogonal to the object-oriented abstraction layer, leading, in principle, to a more coherent programming framework.

Currently, the simpA extension is realised as a library, exploiting Java annotations to define the new programming constructs: consequently, a simpA program can be compiled and executed using the standard Java compiler and virtual machine, with no need for specific extensions of the Java framework (preprocessors, compilers, class loaders, or JVM patches). Hence, the newest constructs take the form of annotated classes and methods—which, however, are clearly separated from their non-annotated, underlying object-oriented versions used at the implementation level. The choice of using the library & annotations solution to implement a language and a platform extension has the advantage to maximise the reuse of a widely adopted platform like Java: at the same time, it has some relevant drawbacks, due to the lack of agents and artifacts as first-class abstractions both in the language and in the virtual machine. Accordingly, part of our ongoing work is devoted to the definition and the prototype implementation of a new full-fledged language and platform called simpAL, rooted on agents and artifacts as real first-class entities.

In the remainder of the section we first describe how to define the structure of an agent (Subsection IV-A) and of an artifact (Subsection IV-B), then present the API supporting the agent-artifact interaction (Subsection IV-C) and the overall shape of a simpA application (Subsection IV-D).

A. Defining Agents

Since one of our main objectives was to minimise the number of classes to be defined by users for introducing new agents and artifacts, we adopted a very simple, one-to-one mapping—just one class per agent or artifact template—so as

[1]Of course, agents would encapsulate the business activities of the client side of the application.

[2]The simpA technology is available for download at the simpA web site, http://www.alice.unibo.it/simpa

to make things as agile as possible. Accordingly, a new agent template[3] is defined by extending the `alice.simpa.Agent` base class provided in the simpA API: the class name is equal to the agent template's name. At runtime, new instances of this agent type can then be spawn when needed. The execution of an agent consists in executing the activities specified in its template, starting from the main one.

In the following, we stress the four key aspects of agent templates' definition: the *memo-space* as a way to provide long-term memory (Subsection IV-A.1), the definition of *atomic vs. structured activities* (Subsection IV-A.2), the *coordination* of such (sub-)activities (Subsection IV-A.3), and the definition of *cyclic behaviours* (Subsection IV-A.4).

1) Agents' long-term memory: the memo space: Agent long-term memory is realised as an associative store called *memo-space*, where the agent can dynamically attach, associatively read and retrieve chunks of information called *memos*. A memo is a tuple, characterised by a label and an ordered set of arguments, possibly bound to data objects. If some arguments are left unbound, the memo is partially specified. A *memo-space* is just a dynamic set of memos: each memo is identified by its label and argument list.

Each agent is provided of internal actions—available in the implementation as protected methods—to atomically and associatively access and manipulate the memo space. In particular, `memo(`*`Label`*`,`*`Arg0`*`,`*`Arg1`*`,...)` is used to create a new memo with a specific label and arguments: these can be null or bound to specific data objects. Conversely, `readMemo(`*`Label`*`,`*`Arg0`*`,`*`Arg1`*`,...):Memo` and `removeMemo(`*`Label`*`,`*`Arg0`*`,`*`Arg1`*`, ...):Memo` respectively read and remove a memo that matches both the label and the given arguments: these can be either concrete values or variables—in the latter case, represented as instances of the `MemoVar` class. In the special but frequent case that, due to the designer's own choice and convention, the label alone is enough to uniquely identify the memo type—that is, the same label in not used twice with a different argument list to denote different memo types—and that a single tuple of a given type is present in the memo space at a time, two linguistic shortcuts are provided: `getMemo(`*`Label`*`):Memo` and `delMemo(`*`Label`*`):Memo` respectively get and remove a memo with a given label, chosen non-deterministically among the existing ones.

By default, the `boot_args(`*`Arg0`*`,`*`Arg1`*`,...)` memo is available in each agent's memo space at the agent's boot time, and contains the parameters optionally specified when the agent has been instantiated.

It is worth remarking that the memo-space is the only data structure adopted for supporting the agent's long-term memory: the instance fields of agent classes are not used.

2) Atomic and structured activities: Agent activities can be either *atomic*—i.e. not composed of sub-activities—or *structured*, composed by some kinds of sub-activities. Atomic activities are implemented as methods with the `@ACTIVITY` annotation, with no input parameters and with `void` return type. The body of a the method specifies the computational behaviour of the agent corresponding to the accomplishment of the activity. Method local variables are used to encode data-structures representing the short-term memory related to the specific activity. By default, the main activity of an agent is called `main`, and must be defined by every agent template. Here is a naïve example of agent template:

[3]The term "template" is used here as a higher-level synonym of "class", intended as the entity describing the structure and behaviour of all the template instances.

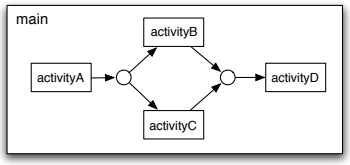


Fig. 5. A representation of an agent's main structured activity composed of two parallel sub-activities activityB and activityC to be executed after activityA; activityD is executed after the completion of both activityB and activityC.

```
public class MyAgent extends Agent {
  @ACTIVITY void main(){
    log("Hello, world!");
  }
}
```

In this case, the agent behaviour simply logs the "Hello, world" message onto standard output and then terminates.

Structured activities are (hierarchically) composed of sub-activities. The notion of *agenda* is introduced to specify the set of the potential sub-activities composing the activity, referenced as *todo* in the agenda. Each *todo* specifies the name of the sub-activity to be executed, and optionally a pre-condition. When a structured activity is executed, all the *todo*s in the agenda are executed as soon as their pre-conditions hold: no pre-condition means that the *todo* can be executed immediately. So, multiple sub-activities can be executed concurrently in the context of the same (super) activity.

A structured activity is implemented by a method annotated with an `@ACTIVITY_WITH_AGENDA` annotation, which contains the *todo* descriptions as a list of `@TODO` annotations. Each `@TODO` specifies the name of the sub-activity to be executed, as well as a `pre` property for the optionally precondition, expressed as a boolean expression of Prolog predicates, possibly combined through the classical *and*, *or* and *not* connectors (represented by the `,`, `;`, and `!` symbols, respectively). Predicates can be predefined or user-defined—actually, any valid Prolog expression (clause body) can be specified. Essentially, these predicates make it possible to specify conditions on the current state of the activity agenda, in particular on *(i)* the state of the sub-activities (*todo*s)—whether they have completed / aborted / started, and on *(ii)* the *memos* that could have been attached to the agenda. Preconditions can depend only on the local (inner) agent's state, not on the agent-environment state.

Now let us see a simple example of an agent with a structured activity, whose agenda is composed by four *todo*s: `activityA`, `activityB`, `activityC`, and `activityD`

(see Figure 5). `activityA` is meant to be executed as soon as the main activity starts, `activityB` and `activityC` are executed in parallel when `activityA` completes, while `activityD` starts when both `activityB` and `activityC` have been completed.

```
public class MyAgent extends Agent {

  @ACTIVITY_WITH_AGENDA({
    @TODO("activityA"),
    @TODO("activityB", pre="completed(activityA)"),
    @TODO("activityC", pre="completed(activityA)"),
    @TODO("activityD",
       pre="completed(activityB),completed(activityC)")
  }) void main(){}

  @ACTIVITY void activityA(){
    memo("x",1);          // attach a new memo x(1)
  }

  @ACTIVITY void activityB(){
    int v = getMemo("x").intValue(0); // retrieve 1st arg
    memo("y", v+1, v-1);  // attach a new memo y(2,0)
  }

  @ACTIVITY void activityC(){
    memo("z", getMemo("x").intValue(0)*5); // attach z(5)
  }

  @ACTIVITY void activityD(){
    MemoVar y0 = new MemoVar();
    MemoVar y1 = new MemoVar();
    readMemo("y",y0,y1);  // read memo arguments
    int z = getMemo("z").intValue(0);          // z =  5
    int w = z*(y0.intValue() + y1.intValue());  // w = 10
    log("the result is: "+w);           // should log 10
  }
}
```

In this example, the agent attaches and retrieves some memos in the memo-space to share data among its (sub-)activities and store the result of its work. In particular, in `activityA` the agent stores a memo `x(1)`, then in `activityB` and `activityC` reads the memo labelled with `x` and uses its content to create the two new memos `y(2,0)` and `z(5)`; finally, in `activityD`, it reads both memos `y` and `z` and uses them to compute the desired result. The `Memo` class provides methods for accessing the memo content: for instance, `intValue(`*`i`*`)` retrieves the *i*-th argument as an integer value.

It is worth noting that local method variables are exploited as a kind of *short-term memory*, in contrast with the memo-space exploited as a long-term memory.

3) Coordinating sub-activities: Memos can be used both to contain data objects elaborated by activities, and to support the coordination of sub-activities. This is possible by exploiting the `memo` predicate in the specification of the pre-conditions so as to test the presence of a specific memo in the memo space—and possibly to associatively retrieve its argument values, if needed. Below is a variant of the previous example, where the pre-conditions for the execution of sub-activities are no longer expressed as conditions on the completion of other activities, but are based on the availability of the information that each sub-activity needs in order to be executed:

```
public class MyAgent extends Agent {
  @ACTIVITY_WITH_AGENDA({
    @TODO("activityA"),
    @TODO("activityB", pre="memo(x(_))"),
    @TODO("activityC", pre="memo(x(_))"),
    @TODO("activityD", pre="memo(y(_)),memo(z(_))")
  }) void main(){}
  ...
}
```

Accordingly, `activityB` is triggered as soon as a memo matching the template `x(_)`[4] is found in the memo space, and the same for the other activities.

4) Cyclic behaviour: In order to define a cyclic behaviour, a *todo* can be specified to be *persistent*: then, once it has been completely executed, it is automatically re-inserted in the agenda, so that it is eventually executed again. In the following example, the agent's main activity consists of repeatedly acquiring a new task to do and serving it concurrently with the other running tasks.

```
public class MyAgent extends Agent {

  @ACTIVITY_WITH_AGENDA({
    @TODO("getNewTaskTodo", persistent=true),
    @TODO("doTask", pre="memo(new_task_todo)",
                    persistent=true)
  }) void main(){}

  @ACTIVITY void getNewTaskTodo(){
    // <wait for a task todo>
    memo("new_task_todo");
  }

  @ACTIVITY void doTask(){
    removeMemo("new_task_todo");
    // <do task>
  }
}
```

B. Defining Artifacts

Analogously to agents, artifacts are mapped onto a single class, too. An artifact template is described by a single class extending the `alice.simpa.Artifact` base class. Again, the elements defining an artifact—its inner and observable state and the operations defining its computational behaviour—are mapped onto suitably annotated class elements. The instance fields of the class are used to encode the inner state of the artifacts, while suitably annotated methods are used to implement artifacts operations.

In particular, for each operation (control) listed in the usage interface, a method with no return parameter and annotated with the `@OPERATION` annotation must be defined: the method name and arguments must coincide with the name and arguments of the operation to be triggered. Any method annotated with `@OPERATION` represents the first computational step executed when the homonymous operation is triggered. Moreover, since any useful artifact has to be somehow *observable*, the `signal` primitive is used to generate *events* that can be observed by the agent using the artifact.

As a simple example, the following code shows the definition of a `Count` artifact functioning as a simple counter, whose usage interface defines just one operation (`inc`) for incrementing the counter value:

```
public class Count extends Artifact {
  int count;

  public Count(){ count = 0; }

  @OPERATION void inc(){
     count++;
     signal("new_count_value", count);
  }
}
```

[4]Following the Prolog syntax, the underscore means *any value*. Analogously, symbols starting with an uppercase letter represent variables.

An observable event is characterised by a label describing the kind of the event and possibly an object representing the event data. In the previous example, for instance, a `new_count_value` event is generated each time the counter is updated.

Some events are automatically generated for any operation execution: in particular, `op_execution_completed` and `op_execution_failed` are generated when an operation completes with a successful or a failure result, respectively.

Besides observable events, an artifact can define a number of *observable properties*—that is, labelled inner state variables whose change is made observable to agents which are *focussing* the artifact (this aspect is discussed more in detail in Subsection **??**). Observable properties are expressed as instance fields annotated with the `@OBSPROPERTY` annotation: a basic set of primitives is available to manipulate the property values. As an example, let us consider a variant of the `Count` artifact, which defines the `count` observable property:

```
public class Count extends Artifact {
  OBSPROPERTY int count;

  public Count(){ count = 0; }

  @OPERATION void inc(){
     updateProperty("count",count++);
  }
}
```

Now, each time the operation `inc` is executed, the property value is updated by the `updateProperty` primitive, which causes the generation of an observable event of type `property_updated(count)`: the event data carry the new property value.

In the following, we stress more in detail three key aspects of artifact definition: the definition of structured operations (Subsection IV-B.1), temporal guards (Subsection IV-B.2), and linkability (Subsection IV-B.3); other artifact features are reported in Subsections IV-B.3, IV-B.4 and IV-B.5.

1) Structured operations: In previous examples, artifact operations were always atomic—i.e., made of a single step. However, structured operations, composed of multiple (atomic) steps, can also be implemented: to this end, each operation step has to be encoded by a method annotated with `@OPSTEP` annotation, which is triggered by the `nextStep` primitive. This primitive specifies the name of the step to be triggered along with its parameters, as a kind of continuation.

In addition, each operation or operation step can be provided with a *guard*, that is, a condition that must hold for actually executing the triggered operation or operation step. Guards are implemented as boolean methods annotated with the `@GUARD` annotation, whose arguments must exactly match those of the guarded operation or operation step: as soon as the guard is evaluated to true, the step is executed. For the sake of concreteness, let us consider the following example:

```
public class MyArtifact extends Artifact {
  int m;

  @OPERATION void op1(){
    m = 1;
    nextStep("opStepB");
  }

  @OPSTEP(guard="canExecOpStepB") void opStepB(){
    log("op1 completed.");
  }

  @GUARD boolean canExecOpStepB(){ return m == 5; }

  @OPERATION void op2(){ m++; }
}
```

Here the operation `op2` is atomic, while the operation `op1` is composed of two steps: the first coincides with the operation itself[5] (and initialises `m` to 1), while the second, `opStepB`, is explicitly labelled and encoded by the homonymous method (which writes a message to the log). Of course, the definition order of these methods is not significant—the above writing order is just a matter of readability.

The `opStepB` step is triggered by the first step in `op1` through the explicit invocation of the `nextStep` primitive: once triggered, the step is executed only when (and as soon as) its guard, `canExecOpStepB`, evaluates to true. This guard conditions the step execution to the value of the internal artifact variable `m`, which must be equal to 5 for `opStepB` to be actually executed. In turn, `m` is incremented by the operation `op2`: so, `opStepB` is executed only after agents have invoked `op2` four times, raising `m` to 5. This completes the execution of operation `op1`.

As anticipated above, a guard can also be applied to an operation, with the obvious meaning that the guard must be true for (the first step of) the operation to be executed:

```
public class MyArtifact extends Artifact {
  int m;
  public MyArtifact(){ m=1; }
  @OPERATION(guard = "canExecOp1") void op1(){ m++; }
  @GUARD boolean canExecOp1(){ return m < 10; }
}
```

It is worth noting that multiple instances of the same operation can be triggered for execution, typically by distinct agents: in that case, a strict ordering semantics applies on their execution, based on to the time when the operation has been triggered (besides the evaluation of the guard).

Summing up, structured operations make the implementation of long-term operations encapsulated, flexible and effective, allowing for multiple structured operations to be executed concurrently by interleaving the guarded execution of their steps, while enforcing the mutual exclusion on the access to the artifact state. For further details, we forward the interested reader to the simpA manual available at [1].

2) Temporal guards: Besides guards based on the artifact state, simple *temporal guards* are also supported: their evaluation becomes true after a given time is elapsed since they have been triggered. To define a temporal guard, a `tguard` property must be specified inside the `@OPSTEP` annotation instead of `guard`: the property should then be assigned a (long) positive value, indicating the guard duration in milliseconds. Let us consider the following example:

```
public class Clock extends Artifact {
  OBSPROPERTY int nticks;
  boolean stopped;

  public Clock(){ nticks = 0; stopped = false; }
```

[5]This choice makes it easier to express single-step operations, which are the most frequent case.

```
  @OPERATION void start(){
    stopped = false;
    nextStep("tick");
  }

  @OPSTEP(tguard=1000) void tick(){
    if (!stopped){
      updateProperty("nticks",nticks+1);
      nextStep("tick");
    }
  }

  @OPERATION void stop(){ stopped = true; }
}
```

Once started via the `start` interface command, this artifact generates a tick event approximatively every second, which increments the clock value. When some agent issues the `stop` command, the counting finally terminates.

3) Linkability: Besides the usage interface, artifacts can be provided with a *link interface*, that is, a set of operations to be invoked (*linked*) by other artifacts (not by agents). This feature makes it possible to create complex artifacts by dynamically composing simpler artifacts, mimicking the way in which human artifacts are linked together. From the concurrency model viewpoint, linked operations have the same behaviour of operations triggered by the usage interface: the only difference is that the observable events generated by a linked operation are made observable not to the artifact linking the operation (which would not make sense in the simpA model), but to the agent originating the execution chain.

4) Observable states: The observable behaviour of an artifact can be partitioned in states, each equipped with a different usage interface and observable property set.

5) Artifact manual: Finally, each artifact should be equipped with a document containing a formal machine-readable, semantic-based description of the artifact functionality and usage instructions (usually called operating instructions). Such a description is meant to provide a semantic layer making it possible to envision, in principle, scenarios where agents could be able to select and use artifacts that are added dynamically to their working environment, without a pre-programmed knowledge about their functionality and use. Again, the interested reader is forwarded to the simpA web site for up-to-date documentation on the work in progress.

C. Agent API for Interacting with Artifacts

The API for enabling agents to interact with artifacts concerns two main categories: use and observation (Subsection IV-C.1) on the one side, instantiation and discovery (Subsection IV-C.2), on the other.

1) Artifact use and observation: Artifact *use* is the basic form of interaction between agents and artifacts. In fact, also artifact instantiation and artifact discovery are realised by using proper artifacts—a *factory* and a *registry* artifacts—which are supposed to be available in any working environment.

Following the A&A model, the use of an artifact by an agent involves two basic aspects: *(i)* executing operations on the artifact, and *(ii)* perceiving the observable events possibly generated by the artifact through agent sensors.

Agents execute operations on artifacts by exploiting the interface controls (or commands) provided by the artifact usage interface. The `use` basic action is provided for this purpose, and specifies the identifier of the target artifact, the operation to be triggered and optionally the identifier of the sensor used to collect observable events generated by the artifact. When the action execution succeeds, the value returned by `use` is the operation's unique identifier. If, instead, the action execution fails—for instance, because the interface control specified is not part of artifact usage interface—an exception is raised.

Sensors are represented by specific classes, which extend the basic abstract class `alice.simpa.AbstractSensor`. A concrete default implementation is provided, called `alice.simpa.DefaultSensor`. Default sensors provide a simple FIFO policy in managing observable events collected from the environment. Sensors are dynamically created as normal objects—specifying a logic name—and are then dynamically linked to / unlinked from agents' bodies. An agent can link / unlink any number of sensors, possibly of different kind, according to its own strategy for sensing and observing the environment, by means of specific primitives (`linkSensor`, `unlinkSensor`, and `linkDefaultSensor`).

The `sense` primitive makes it possible to retrieve the events collected by a sensor: it waits until a matching event appears — that is, until an event collected by a given sensor matches an optional pattern (for data-driven sensing) — or until an optional timeout is reached. Pattern matching is based on regular-expression patterns, matched over the event type (a string). In the case of a successful execution, the event is removed from the sensor and a perception related to that event—represented by an object instance of the class `Perception`—is returned. If, instead, no perception is sensed for specified time, a `NoPerceptionAvailableException` is raised. The following code shows the `CountUser` agent creating and using a `Count` artifact, then locating and using the `myArchive` artifact (instantiation and discovery will be described later).

```
public class CountUser extends Agent {
  @ACTIVITY void main() {

    SensorId sid = linkDefaultSensor();
    ArtifactId countId = makeArtifact("myCount","Count");

    use(countId,new Op("inc"));
    use(countId,new Op("inc"),sid);

    try {
      Perception p = sense(sid,"count_value",1000);
      long value =  (Long) p.getContent();

      ArtifactId dbaseId = lookupArtifact("myArchive");
      use(dbaseId, new Op("write",new DBRecord(value));

    } catch (NoPerceptionException ex){
      log("No count_value perception from the count");
    }
  }
}
```

The agent activity accounts for: *(i)* creating a `Count` artifact as described in previous section; *(ii)* using the artifact, executing twice the `inc` operation provided by `Count` usage interface and observing the `count_event` generated by the artifact (carrying the count value) only the second time that

the operation is executed; *(iii)* locating and using a `DBase` artifact called `myArchive`, performing the `write` operation to record the value perceived by `myCount`.

The class `ArtifactId` is exploited to represent artifacts' unique identifiers. The first time that an agent executes `inc`, it is not interested in observing the events generated by the operation execution, so no sensor is specified. The `sense` primitive is used to perceive only the events matching `count_value`, and with a timeout of one second. The `Perception` class provides a `getContent` method to get the content of the perception (event).

It is worth remarking here the similarities but also the deep differences between the notion of sensor, on the one side, and the *future* construct / pattern used in concurrent programming for handling asynchronous calls, on the other. In fact, a sensor can be used to collect possibly-multiple observable events generated as a consequence of possibly-multiple `use` actions, on possibly-distinct artifacts. Futures, instead, are typically used to get asynchronously the single result of the execution of a single call.

Finally, a support for *continuous observation* is provided. If an agent is interested in observing every event generated by an artifacts—including those generated as a result of the interaction with other agents—two primitives can be used: `focus` and `unfocus`. The former starts observing the artifact, and therefore specifies the sensor to be used to collect the events and optionally the reg-ex filter defining the events to be observed; the latter obviously stops the observation process.

2) Artifact instantiation and discovery: In the above example two further actions, `makeArtifact` and `lookupArtifact`, are used to instantiate and lookup artifacts. As briefly mentioned in the previous Subsection, both these auxiliary actions are realised on top of a bunch of `use` and `sense` actions executed on two pre-defined artifacts available in each simpA application—the `factory` and the `registry` artifacts, respectively.

In particular, instantiation is actually handled by the factory's `makeArtifact` operation, which takes the logical name of the new artifact, its template (full class name or `Class` type) and possibly the parameters needed for its creation (which coincide with the constructor parameters of the artifact template class). In case of success, a `make_succeeded` event is generated, and the artifact identifier is provided as the event content. The factory also provides an analogous operation, `makeAgent`, to instantiate & spawn agents.

In quite the same way, artifact discovery is handled by the registry's `lookupArtifact` operation, whose argument is the name of the artifact to be located: upon success, the artifact identifier is returned as the content of a `lookup_succeeded` event.

D. A simpA *Application*

The core of a simpA application is a simple main class where a simpA working environment is created and an initial sets of agents are booted, possibly with a starting set of artifacts.

In the following example, the `my-app` workspace is created, initially composed of an instance of the `DBase` artifact (not reported) and a `CountUser` agent (which in turn will create a `Count` artifact):

```
public class MyApp {
 public static void main(String[] arg) throws Exception{
   ISimpaWorkspace env = Simpa.createWorkspace("my-app");
   env.createArtifact("myArchive","DBase");
   env.spawnAgent("a-user","CountUser");
 }
}
```

Any non-naïve application, however, can be expected to demand the creation of multiple agents working concurrently in a workspace populated by multiple artifacts of different kinds.

V. Programming SOA/WS Applications on Top of simpA: The simpA-WS Framework

simpA-WS is a framework on top of simpA which makes it possible to build Web Service applications as simpA workspaces with agents and artifacts, following the model described in Subsection III-C. Besides simpA, simpA-WS exploits Apache AXIS2 open-source technology[6] as an enabling low-level Java-based web service technology effective for managing (SOAP) message exchange, for an effective management of XML data, etc. simpA-WS is a fully Java-based open-source technology, and can be downloaded at the simpA-WS web site[7].

A. simpA-WS *Programming Interface*

The simpA-WS framework provides a uniform model to conceive both Web services and applications interacting with Web Services as simpA workspaces with agents and artifacts encapsulating the business logic: such pre-defined artifacts can be exploited as part of the service-oriented infrastructure. There are two basic kinds of artifacts:

- `ServiceInterface`—an implementation of the ws-service-interface artifact, meant to be instantiated and used by agents to interact with a specific Web Service, to send messages for executing operations and to get the replies sent back by the service. Multiple `ServiceInterface` artifacts can be instantiated and used in a given workspace to interact with multiple Web services.
- `ServicePanel`—an implementation of the ws-service-panel artifact, meant to be used by agents in a service application to manage the incoming requests and messages from the Web Service. One `ServicePanel` must be created and used for each (web) service to implement. Multiple `ServicePanel` artifacts can be instantiated and used in the same workspace to implement multiple services within the same simpA-WS application.

So, in simpA-WS the interaction with and among Web service applications is totally conceived in terms of message exchanges; operations represent, from this point of view, the context in which message exchange protocols or MEPs take place,

[6]Apache AXIS2 web site is available at: http://ws.apache.org/axis2/
[7]http://www.alice.unibo.it/simpa-ws

as described in the service documents (WSDL, choreography, etc.). Even if this is quite obvious by considering the reference SOA/WS model, most of the existing platforms supporting SOA/WS hides the message level to the programmer, providing API where the execution of an operation typically accounts for invoking a method in stub objects or components providing an interface that mirrors the service interface. Conversely, in simpA-WS interacting with a service accounts (from an agent viewpoint) for using a `ServiceInterface` artifact to send messages to the service according to the protocol characterising by the operation to be executed.

The message level is adopted here also at the service side, while in non-agent oriented platforms the message processing is typically not realised by the components that encapsulate the service's business logic. Conversely, in simpA-WS the message level is brought to the business logic, so that one or multiple agents—encapsulating the business logic activities—exploit a `ServicePanel` artifact to retrieve and process messages, possibly sending one or multiple replies during their activities (which can be long-term).

Summing up, a Web service application in simpA-WS (both on the service and on the user's side) is programmed as a workspace where one or multiple agents create and use one or multiple `ServiceInterface` artifacts to interact, even concurrently, with services; at the same time, they share and use other kinds of artifacts that represent useful resources and tools needed to support their (cooperative) activities. On the service side, one or multiple `ServicePanel` artifacts—the latter case concerns multiple services implemented in the same Web service application—are created and used by agents to process the requests, possibly in parallel.

In the following we describe the `ServiceInterface` and `ServicePanel` artifacts more in detail, and provide a simple running example.

B. Interacting with existing Web Services

`ServiceInterface` defines a simple usage interface for executing operations on a Web Service, providing a direct support for exchanging messages realising any possible MEP, from simple ones with at-most one message input and one message output to more articulated ones—as defined by WSDL 2.0—possibly including multiple input and output messages in the context of the same operation.

A `ServiceInterface` artifact must be instantiated by specifying—as artifact configuration parameters—the URI of the WSDL containing the service description, the name of the specific service and port type (interface) "pointed" by the artifact, and a local endpoint name, which represents the endpoint to which the artifact is bound to receive messages (such as replies). Thus, the artifact usage interface provides two basic operations:

- `sendMsg`, to send a message to the service, in the context of an operation. Abstract description:

  ```
  sendMsg(opName:String, {,msgName:String,}
          msgContent:OMElement {,relatedToMsgID:String})
  ```

  where `msgName` identifies the message to be sent (according to the WSDL), `opName` is the name of the operation, `msgContent` is the content of the message (a XML piece of data, according to the XML schema described in the WSDL), and optionally `relatedToMsgID` is the identifier of the message to which this message is related to. `msgName` is optional: if it is not specified, its value is determined by accessing the WSDL. The execution of the message generates an observable event `msg_sent(msgID:String)` if it succeeded, containing the identifier of the message sent, or an event `msg_send_failed` if it fails.
- `getReply`, to get the reply to a message previously sent during an operation. Abstract description:

  ```
  getReply(msgID:String})
  ```

  where `msgID` is the identifier of the message to which the reply message must be related. When an output message related to `msgID` is received by the artifact, the operation generates an observable event `msg_reply(msg:MsgInfo)`.

Besides these two basic operations, other auxiliary operations are provided to directly support basic MEPs; for instance:

- `requestOp`, which implements the basic request-response (in-out) MEP by sending a request message and generating an event with the response message as soon as it arrives. Abstract signature:

  ```
  requestOp(opName:String, msgContent:OMElement)
  ```

  where `opName` is the name of the operation and `msgContent` is the content of the message (a XML piece of data, according to the XML schema described in the WSDL). It's worth noting that in this case the message name is automatically retrieved from the description of the operation in the WSDL. The main observable events generated by the operation are the following: `request_sent(msgID:String)` if the request message is sent successfully (`request_failed` otherwise), and `msg_reply(msg:MsgInfo)` as soon as the reply is received.

C. Processing requests and messages for a Web Service

A `ServicePanel` artifact is used to manage and control the messages which arrive to a service, providing basic functionalities to retrieve them and to send the related replies. A `ServicePanel` artifact is instantiated by specifying—as artifact configuration parameter—the URI of the WSDL containing the description of the service. The artifact usage interface provides two basic operations:

- `getNextRequestMsg`, used to retrieve incoming messages representing new operation requests to be served. Abstract signature:

  ```
  getNextRequestMsg(filter:MsgFilter)
  ```

  where `filter` can be specified to select the request messages to which the agent is interested to (for instance, related to a specific operation). The operation generates an observable event `new_op_request(msg:MsgInfo)` as soon as an message matching the filter is received by the artifact, containing information about the message arrived.

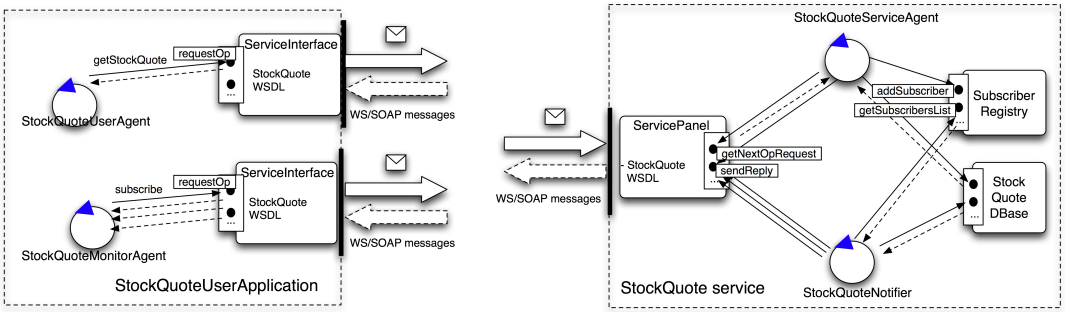


Fig. 6. An abstract representation of the stock quote user and service applications, with in evidence the agents and artifacts involved.

- `getMsgRelatedTo`, used to retrieve incoming messages related to previously sent messages. Abstract signature:

  ```
  getMsgRelatedTo(msgID:String)
  ```

  where `msgID` is the identifier of the message to which the reply is related. The operation generates an observable event `new_msg(msg:MsgInfo)` as soon as an message matching the filter is received by the artifact, containing information about the message arrived.
- `sendReply`, used to send message replies. Abstract signature:

  ```
  sendReply(toMsg:MsgInfo,
            msgContent:OMElement)
  ```

  where `toMsg` contains the information about the message to which the reply is related and `msgContent` is the content of the reply message (a XML piece of data, according to the XML schema described in the WSDL).

D. A simple example: Stock Quote with Agents and Artifacts

To give a concrete taste of simpA-WS, in the following we report the sketch of the implementation of a stock quote service, which is typically found among the basic examples of Web Service technologies: here we consider a slightly more articulated version, with a support not only for in-only and in-out (request-response) MEPs, but also out-only.

1) Stock Quote Service: The stock quote service is characterised by three basic operations (a sketch of the WSDL is reported in the appendix): `GetLastTradePrice`, an in-out (request-response) operation useful to get the current value of a stock given its name, `Subscribe`, an in-only (fire-and-forget) operation to subscribe the service in order to receive periodically the quote of a specified stock, and `NotifyTradePrice`, an out-only operation executed by the service to notify the value of a stock to a previously subscribed user. Figure 6 shows the architecture of a possible implementation of the service using simpA-WS: Table I, Table II, Table III and Table IV show the implementation of the service side, while Table V, Table VI and Table VII show the implementation of a simple application interacting with the service.

The service is composed by two types of agents: *(i)* `StockQuoteServiceAgent`, responsible for creating the service panel artifact and using it to process the incoming `GetLastTradePrice` and `Subscribe` requests; *(ii)* a `StockQuoteNotifierAgent`, responsible for periodically notifying its subscribers of the stock quotes. Besides these agents, two kinds of artifacts are exploited (other than `ServicePanel`): a `StockQuoteDBase` artifact, which functions as a store containing the stock quote values, and a `SubscribersRegistry`, which is used to keep track of the list of the service subscribers.

`StockQuoteServiceAgent` uses `ServicePanel` to process the requests as soon as they are collected: then, in the case of `GetLastTradePrice` requests, the agent accesses the database artifact to get the current value of the stock quote, and sends a reply with such information; instead, in the case of `Subscribe` requests, the agent registers the new subscribers by executing the `addSuscriber` operation on the `SubscribersRegistry` artifact. `StockQuoteNotifierAgent` periodically retrieves the subscribers' list by executing a `getSubscribers` operation on the `SubscribersRegistry` artifact: for each subscriber it then sends a `NotifyTradePrice` message with the updated value of the stock quote, retrieved from the database artifact. Table I and Table II report a sketch of the source code of the agents, Table III the skeleton of the artifacts, and Table IV the main of the service application (called `StockQuoteService`), which actually creates the initial set of agents and artifacts composing the application (apart the `ServicePanel` artifact, which is created by the `StockQuoteServiceAgent` agent).

Despite its simplicity, this example should give a quite concrete idea about how complex services—characterised, for instance, by complex message-exchange protocols, possibly including both pro-active and reactive parts, short and long-running activities, etc.—can be structured in terms of agents and artifacts.

2) Stock Quote User Application: Table V, Table VI and Table VII show an example of a simpA-WS user application exploiting Web services, in particular the stock quote service described by the WSDL reported in the Appendix. The application is com-

```
 public class StockQuoteServiceAgent
extends Agent {
  @ACTIVITY_WITH_AGENDA( {
    @TODO(activity="preparing"),
    @TODO(activity = "processingMsg",
                    pre="completed(preparing)", persistent = true),
  }) void main() {}

  @ACTIVITY void preparing() throws ActivityFailure {
    makeArtifact("StockQuoteService","alice.simpaws.ServiceInterface",
                           new ArtifactConfig("http://localhost:8080/axis2/wsdl/StockQuoteService.wsdl"));
  }

  @ACTIVITY void processingMsg() throws ActivityFailure {
    ArtifactId panel = lookupArtifact("StockQuoteService");
    SensorId sid = linkDefaultSensor();

    use(panel, new Op("getNextOpRequest"), sid);
    Perception p = sense(sid, "new_op_request", 300000);
    MsgInfo msg = (MsgInfo) p.getContent(0);
    String name = msg.getOperationName();

    if (name.equals("GetLastTradePrice")) {
      OMElement replyMsg = getStockValue(msg.getMsgContent());
      use(panel, new Op("sendReply", msg, replyMsg));
    } else if (name.equals("Subscribe")) {
      String stockName = getStockName(msg.getMsgContent());
      ArtifactId reg = lookupArtifact("subscribers-registry");
      use(reg,new Op("addSubscriber",msg,stockName));
    }
    unlinkSensor(sid);
  }

  private OMElement getStockValue(OMElement msg) {...}
  private String getStockName(OMElement elem){...}
}
```

TABLE I

STOCK QUOTE AGENT ON THE SERVICE SIDE

```
 public class StockQuoteNotifier
extends Agent {

  @ACTIVITY_WITH_AGENDA({
    @TODO(activity = "notifyingStockQuotes", persistent = true),
  }) void main() {}

  @ACTIVITY void notifyingStockQuotes() throws ActivityFailure {
    ArtifactId reg = lookupArtifact("subscribers-registry");
    SensorId sid = linkDefaultSensor();

    use(reg, new Op("getSubscribersList"), sid);
    Perception p = sense(sid, "subcribers_list");

    List<SubscriberInfo> list = (List<SubscriberInfo>) p.getContent();
    if (list.size()>0){
      ArtifactId panel = lookupArtifact("StockQuoteService");
      for (SubscriberInfo sinfo: list){
        OMElement replyMsg = getStockValue(sinfo.getStock());
        use(panel, new Op("sendReply", sinfo.getMsgSource(), replyMsg));
      }
    }
    unlinkSensor(sid);
    suspendActivityFor(2000);
   }

  private OMElement getStockValue(String stock) {...}
}
```

TABLE II

NOTIFIER AGENT ON THE SERVICE-SIDE

```
 public class SubscriberInfo {
  MsgContext msg;
  String quote;

  public SubscriberInfo(MsgContext msg, String quote){
    this.msg = msg;
    this.quote = quote;
  }
  public MsgContext getMsgSource(){ return msg; }
  public String getStock(){ return quote; }
}

public class SubscriberRegistry extends Artifact {
  private ArrayList<SubscriberInfo> list;

  public SubscriberRegistry(){
    list = new ArrayList<SubscriberInfo>();
  }

  @OPERATION void addSubscriber(MsgContext ctx, String quote){
    list.add(new SubscriberInfo(ctx,quote));
  }

  @OPERATION void getSubscribersList(){
    signal("subcribers_list", list.clone());
  }
}

public class StockQuoteDBase extends Artifact { ... }
```

TABLE III

ARTIFACTS USED ON THE SERVICE SIDE

```
 public class StockQuoteService extends
WSApplication {
  public void setup(){
    try {
      createArtifact("stock-quote-dbase",StockQuoteDBase.class);
      createArtifact("subscribers-registry",SubscriberRegistry.class);
      spawnAgent("stock-notifier-agent",StockQuoteNotifier.class);
      spawnAgent("stock-quote-agent",StockQuoteServiceAgent.class);
    } catch (Exception ex){
      ex.printStackTrace();
    }
  }
}
```

TABLE IV

ENTRY POINT (MAIN) OF THE WEB SERVICE APPLICATION (SERVICE-SIDE)

```
 public class StockQuoteUserAgent
extends WSAgent {

  @ACTIVITY  void main() throws ActivityFailure {
    SensorId sid = linkDefaultSensor();
    ArtifactId ServiceInterface =
            makeServiceInterface("interface-1",
               "http://localhost:8080/axis2/wsdl/StockQuoteService.wsdl",
               "StockQuoteUserApplication");

    use(ServiceInterface,new Op("requestOp", "getStockQuote", makeGetStockQuoteMsg("ACME")),sid);
    Perception res = sense(sid, "msg_reply", 300000);
    OMElement replyMsg = (OMElement) res.getContent();

    log("Result: " + replyMsg.toString());
  }

  private  OMElement makeGetStockQuoteMsg(String stockName) {...}
  private void log(String st){...}
}
```

TABLE V

STOCK QUOTE USER AGENT ON THE CLIENT SIDE

```
 public class StockQuoteMonitorAgent
extends WSAgent {

  @ACTIVITY_WITH_AGENDA({
    @TODO(activity="subscribing"),
    @TODO(activity="collectingQuotes",
          pre="completed(subscribing)", persistent=true)
  }) void main(){}

  @ACTIVITY void subscribing() throws ActivityFailure {
    SensorId sid = linkDefaultSensor();
    ArtifactId ServiceInterface =
            makeServiceInterface("interface-2",
               "http://localhost:8080/axis2/wsdl/StockQuoteService.wsdll",
               "StockQuoteUserApplication");

    OMElement subscribeMsg = makeRegisterMsg("ACME"));
    use(ServiceInterface,new Op("sendMsg","Subscribe",subscribeMsg),sid);
    Perception p = sense(sid, "msg_sent|msg_send_failed", 5000);
    if (p.getLabel().equals("msg_sent")){
      memo("subscribe_message_id",(String)p.getContent());
      unlinkSensor(sid);
    } else {
      throw new ActivityFailure();
    }
  }

  @ACTIVITY void collectingQuotes() throws ActivityFailure {
    SensorId sid = linkDefaultSensor();
    ArtifactId ServiceInterface = lookupArtifact("interface-2");
    String msgId = (String)getMemo("subscribe_message_id").getArg(0);

    use(ServiceInterface, new Op("getReply", msgId), sid);
    Perception res = sense(sid, "msg_reply", 300000);
    OMElement replyMsg = (OMElement) res.getContent();

    log("New quote value: "+replyMsg.toString());
    unlinkSensor(sid);
  }

  private  OMElement makeRegisterMsg(String stockName) {...}
  protected void log(String st){...}
}
```

TABLE VI

STOCK QUOTE MONITOR AGENT ON THE CLIENT-SIDE

```
 public class StockQuoteUserApplication
extends WSApplication {
  public void setup(){
    try {
      spawnAgent("stock-quote-user-agent",StockQuoteUserAgent.class);
      spawnAgent("stock-quote-monitor-agent",StockQuoteMonitorAgent.class);
    } catch (Exception ex){
      ex.printStackTrace();
    }
  }
}
```

TABLE VII

ENTRY POINT (MAIN) OF THE WEB SERVICE APPLICATION (CLIENT-SIDE)

posed by two agents, `StockQuoteUserAgent` and `StockQuoteMonitorAgent`. The former is characterised by a simple main activity, which creates a proxy artifact for interacting with the stock quote service—in particular, to request the `GetLastTradePrice` operation (specifying `ACME` as stock quote name), and get the reply, which is then logged to the standard output. The latter has a slightly more complex behaviour: first, it creates a service interface artifact to interact with the stock quote service (alternatively, a single service interface artifact could be shared and used by the two agents), and subscribes (via the `Subscribe` operation) to receive a periodic notification. The agent then starts its cyclic `collectingQuotes` activity, receiving and logging all the notifications sent back by the service. A sketch of the source code of the agents is reported in Table V and Table VI, while Table VII shows the main of the application (called `StockQuoteUserApplication`), which simply spawns an instance of both agents.

VI. CONCLUSIONS

In this paper we introduced an agent-oriented programming model for developing SOA/WS applications, based on the A&A meta-model, and simpA-WS, a framework built on top of the Java platform for concretely programming service and user applications in terms of agents and artifacts. This approach contrasts with the choices adopted by leading software

vendors in the state-of-the-art, which is typically based on a component-oriented programming model. This paper extends a previous work [14], with the description of the simpA-WS framework.

Actually, as widely reported in literature [10], Service-Oriented Computing and Web Services are considered among the most promising and important application contexts for agents and MAS. However, the focus of existing agent and MAS research approaches is typically on exploiting AI-related features and techniques for supporting the dynamic discovery and flexible composition and orchestration of services. In this paper, instead, we focussed on programming and software engineering issues, highlighting the value of agent-oriented abstractions as basic building blocks for designing and programming services and applications using services.

Indeed the programming model introduced and the related technologies—such as simpA-WS—are still in their infancy and further work is needed along several directions. In the paper we have just considered the very basic issues concerning SOA and Web Services, without dealing with other important advanced topics such as quality of service (security, reliability, trust,...), service composition and coordination (orchestration, choreography, transactions), along with the related WS-* specifications (WS-Security, WS-Trust, WS-Coordination, etc). Future work will then be devoted to frame such issues in the SA&A programming model, and to stress the approach with real-world case studies and applications, besides the simple toy examples included in the current distribution. In particular, as a concrete case-study we will consider the sample application provided in the WS-I web site[8], trying to compare our approach with those of the leading software vendors (IBM, Microsoft, Sun, SAP—to mention some) available on such site.

REFERENCES

[1] The aliCE Research Group. simpA official web site. http://www.alice.unibo.it/projects/simpa.

[2] Sriram Anand, Srinivas Padmanabhuni, and Jai Ganesh. Perspectives on service oriented architectures. In *Proceedings of the 2005 IEEE International Conference on Service Computing*, volume 2. IEEE, 2005.

[3] Nick Benton, Luca Cardelli, and Cedric Fournet. Modern concurrency abstractions for C#. *ACM Trans. Program. Lang. Syst.*, 26(5):769–804, 2004.

[4] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[5] IBM et al. Service component architecture. http://www-128.ibm.com/developerworks/library/specification/ws-sca/, 2006.

[6] Rafael Bordini et. al. A survey of programming languages and platforms for multi-agent systems. In *Informatica 30*, pages 33–44, 2006.

[7] Donald F. Ferguson and Marcia L. Stockon. Service-oriented architecture: Programming model and product architecture. *IBM Systems Journal*, 44(4):753–780, 2005.

[8] Sun Microsystems. The java API for XML web services (JAX-WS 2.0). http://java.sun.com/webservices/jaxws/.

[9] Sun Microsystems. Service oriented business integration. http://java.sun.com/integration/.

[10] Michael N. Huhns, Munindar P. Singh, Mark Burstein, Keith Decker, Edmund Durfee, Tim Finin, Les Gasser, Hrishikesh Goradia, Nick Jennings, Kiran Lakkaraju, Hideyuki Nakashima, H.Van Dyke Parunak, Jeffrey S. Rosenschein, Alicia Ruvinsky, Gita Sukthankar, Samarth Swarup, Katia Sycara, Milind Tambe, Tom Wagner, and Laura Zavala. Research directions for service-oriented multiagent systems. *IEEE Internet Computing*, 9(6):69–70, November 2005.

[11] Bonnie A. Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996.

[12] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. *Agens Faber*: Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 150(3):21–36, 29 May 2006.

[13] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, and Luca Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 286–293, New York, USA, 19–23 July 2004. ACM.

[14] Alessandro Ricci, Claudio Buda, and Nicola Zaghini. An agent-oriented programming model for soa & web services. In *5th IEEE International Conference on Industrial Informatics (INDIN'07). Special Session on Agents Theory and Practice for Industry (ATPI)*, Vienna, Austria, July 2007.

[15] Alessandro Ricci and Mirko Viroli. simpA: An agent-oriented approach for prototyping concurrent applications on top of java. In *Proceedings of the ACM International Conference on Principle and Practice of Programming in Java (PPPJ'07)*, Lisboa, Portugal, September 2007. ACM.

[16] Clemens Szyperski. *Component Software*. Addison-Wesley Professional, November 2002.

[17] W3C WS Working Group. Web Services Architecture. http://www.w3.org/TR/ws-arch/.

[8] http://www.ws-i.org/

APPENDIX

This appendix reports a sketch of the WSDL code of the stock quote service discussed in the text.

```
<?xml version="1.0"?>
<definitions name="StockQuote"
             targetNamespace="http://example.com/stockquote.wsdl"
             xmlns:tns="http://example.com/stockquote.wsdl"
             xmlns:xsd1="http://example.com/stockquote.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
         <complexType>
           <all>
             <element name="price" type="float"/>
           </all>
         </complexType>
      </element>
      ...
    </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>
  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>
  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>
  <message name="SubscribeMsg"> ... </message>
  <message name="NotifyTradePriceMsg"> ... </message>

  <portType name="StockQuotePortType">

    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>

    <operation name="Subscribe">
      <input name="Subscribe" message="tns:SubscribeMsg"/>
    </operation>

    <operation name="NotifyTradePrice">
      <output name="NotifyTradePrice" message="tns:NotifyTradePriceMsg"/>
    </operation>

  </portType>

  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
      <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    ...
  </binding>

  <service name="StockQuoteService">
    <documentation>Stock quote service example </documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>

</definitions>
```

An Agent-Based Service Oriented Architecture

Agostino Poggi, Michele Tomaiuolo, Paola Turci

Abstract — **Industry is more and more interested in executing business functions that span multiple applications. This demands high-levels of interoperability and a more flexible and adaptive business process management. The trend is to have systems assembled from a loosely coupled collection of Web services, which are ubiquitous and organically integrated. This technical area appears to be a natural environment in which the agent technology can be exploited with significant advantages.**

In the present paper, we propose a framework with the aim of supporting an agent-based SOA. The peculiar characteristic and strength of our research work is the integration of the agent technology with other strategic technologies, that is Web services, workflow, rule engine and semantic Web.

Index Terms — **Multi-agent systems, service oriented architecture, workflow, ontology, rule engine, trust management.**

I. Introduction

Most of the technology and market research companies, which provides their clients with advice about technology's impact on business and consumers, agree on the fact that the adoption of a SOA paradigm is strategic and should be part of the most forward-looking software projects. Nevertheless the paradigm shift is still quite challenging.

Agent technology is more and more considered one of the most interesting technologies to successfully support SOA. In fact, besides being an ideal mechanism for implementing complex systems, agent technology is well-suited to applications that are communication-centric, based on distributed computational and information systems, and requiring autonomous components readily adaptable to changes.

Considering their peculiar features, the central role that agents should play in a SOA scenario is to efficiently support distributed computing and to allow the dynamically composition of Web services. To be successful, it is crucial to appropriately engineer and integrate agent technology with other technologies that have found and will find a purpose within enterprise computing: workflows, rule engines, the semantic Web and Web services.

The vision which is making its way into the research community is to encapsulate the organization's functionalities within appropriate interfaces and advertise them as one or more Web services, which could be integrated, when brought into play, in workflows. This innovative idea brings with it new outstanding opportunities but also new great issues, related mainly to the ability of automatically discovering and composing Web services. An answer to these problems could come from the semantic Web technology.

Recently, we have seen an explosion of interest in ontologies as artefacts to represent human knowledge and as a critical component in several applications; among these the e-business applications. Moreover the "marriage" between agents and ontologies seems to be the kind of technology that can significantly change the face of enterprise software.

On the one hand, ontologies should accomplish the task of giving a precious support to solve two tricky problems: how to efficiently discover Web services and how to make possible the interoperability of heterogeneous Web services. In order to facilitate the resolution of such a structural and semantic heterogeneity, Web services, which play the role of workflow components, will have their interfaces semantically described by ontological concepts.

On the other hand, ontologies enable agents to communicate in a semantic way, exchanging messages which convey information according to explicit domain ontologies.

In this scenario agents represent the backbone of the system and the "glue" that could hold these pieces together and make them perform properly.

Assuming to adopt an agent-based approach, a typical agent-based SOA scenario would be characterized mainly by three actors: service providers, business process manager and users, playing roles which would be allocated to different concrete agents. The system architecture would likely be organized in communities constituted by different kinds of agents: service providers, personal assistants and middle agents (e.g. service brokers, user profile managers, workflow managers, etc). In order to achieve their goals (semantic matching, service contracting and so on) these autonomous agents should be able to perform their tasks in cooperation or competition with other agents and to interoperate with external entities (e.g., legacy software systems). Moreover they should show reasoning capabilities and should have a support for dynamic behaviour modification based on business rules. Finally they should be able to build workflows, compose the external Web services and monitor their execution. The entire process should be supported by a distributed trust management.

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

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

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

Clearly the researchers are well aware that such a scenario is quite ambitious and the outlined objectives difficult to achieve in a short period. Indeed there are several overlooked technical issues and the existing technology presents significant limitations. Nevertheless the realizations of prototype systems centred on the underlying infrastructure can be of great help in order to raise awareness of these issues and to delineate possible solutions.

Bearing in mind what said above, the aim of the present paper, which is an evolution of our previous paper [23], is to introduce a framework, under development at the University of Parma, for the realization of an agent-based SOA.

In the next section we discuss the related work in the fields of the emergent and more established technologies which we aim at integrating with agent technology. Section 3 describes the framework aiming at being the basis for the realization of successful and innovative agent-based SOA. In this section we focus mainly on its architecture, the BPEL engine, the ontological support, the integration with a rule engine and our proposal for a distributed trust management. The paper ends by drawing some conclusions around the results of the work done, and by outlining some considerations regarding our future research directions.

II. Related Work

There is evidence from several research studies [1],[24] that agents represent one of the most suitable technologies which can be used to meet the performance needs for innovative business applications. In particular the current interest in using agents for developing e-business applications, business process management and enterprise integration is rising mostly because different works have shown how agent technology can be leveraged if used together with technologies exploited in the Internet, that is, semantic Web, Web services and workflows [5],[7],[12],[19],[22].

Semantic Web technologies appear to be the right means to provide the semantic integration between data and processes across systems that can be owned by different enterprises [6]. This technology is not completely mature yet; some major activities related to the definition of languages for expressing the semantics of the Web are still in progress [16],[8]. Nevertheless different works have shown how the powerful synergism between agents and semantic Web could be very promising [19],[29] and some efforts have been made in order to define ontology models and develop tools suitable for agents aiming at being truly semantic aware agents. The research community contributions have been mainly devoted to cope with three different issues:

- The formal definition of a standard language for expressing semantics on the web which has led to the Web Ontology Language,
- The development of integral software infrastructures, for writing semantic web applications, offering a variety of tools to engineer ontologies.
- The development of ontological supports specifically thought for multiagent systems.

The second and third points are strictly connected to the first one since OWL is considered the reference language; therefore the work carried out, starting from OWL, has developed tools more suitable for different contexts.

As far as the second point is concerned, an interesting approach is characterized by the definition of a meta-model that closely reflects the OWL syntax and semantics. This is the case of the modelling APIs of Jena, which is the most famous and widely used tool in the sphere of the semantic web (and recently also in the context of multiagent systems).

Considering the third point, the focus is on the specific needs of multiagent systems, and the objective is to provide a communication support enabling agent to perform the proper semantic checks on a given content expression. A significant example of the efforts made in this direction is represented by the ontological support of JADE, designed to represent, using Java objects, a taxonomy of concepts. Such semantically aware agents should then be able to discover, invoke, compose and monitor those Web resources that provide services. In order to make agents able to use a service, they need a computer interpretable description of the service itself and furthermore to know the means by which it is accessible. To that purpose, a community of researchers is developing an ontology of services, called OWL-S, with the aim of providing a semantic orientation to the description of Web services.

To enable software systems for innovative business applications, security issues have to be carefully analysed and sound solutions have to be deployed. A number of different solutions for the problems of authentication and authorization in open systems have been proposed in the scientific literature, and some standards have emerged through the years. Most of them are based on some kind of PKI and signed certificates issued by a Certification Authority. In particular, this is the case of X.509, which is the best known and adopted standard for authentication and authorization. However, its weaknesses have been clearly demonstrated in a number of works [15], above all related to its effort to create a global directory of unique names. Relying on an external entity as root of all certifications represents an additional, not directly controllable, point of failure for the whole system.

In contrast, different approaches have been proposed, based on local names. Both SDSI and PetName Markup Language allow local names to be used in a global scale by prefixing them with the public key of the principal defining them, in the form of (key, name) couples. This way, name conflicts are solved thanks to the uniqueness of the public keys. In [32], authors show that local names and YURLs are more robust than global names to phishing attacks, arguing the root for these attacks lie in the global namespace itself. Moreover, in [21], authors shows that local names and a subset of the SDSI/SPKI standard [9] can be used to implement a distributed RBAC infrastructure, in which local names are interpreted as distributed roles, whose name is localized to their defining principal (key).

Local names and delegation certificates are the key to build systems adhering *trust management principles* [18]. These systems are completely distributed as they avoid any centralized authority. This way they can easily scale to large peer-to-peer networks, where each node is in charge of protecting its own resources and to show proper credentials when accessing resources of other nodes.

In recent years a lot of research work has been undertaken ranging from the use of workflows in distributed systems to the use of agent technology for the management of workflows [7][24][19]. The most important contribution of our work is firstly the use of agents as a support of all the activities involved in the development and execution of a business process, i.e. the workflow generation, the distribution of workflow tasks, the control of their execution and finally the re-allocation of tasks in case of failure of some service components. Secondly, the integration of the agent technology with those technologies we consider crucial for accomplishing strategic business objectives.

To conclude just a few remarks on JADE since it is considered the reference implementation of the FIPA specifications and one of the most used and promising agent development framework. The present release of JADE tries to provide agent developers with a support integrating almost all these technologies, even if in our opinion only partially. As a matter of fact, JADE agents can exploit an ontological model of the application domain to improve their interactions, are able to interact with external Web services [12] and finally different works have shown how the integration of a JADE agent with the Jess rule engine is feasible. But this simply represents a first step towards an effective support of the SOA paradigm.

III. TOWARDS A SERVICE ORIENTED ARCHITECTURE

To overcome the limits of the present release of JADE, we have realized an agent based framework called MASE (Multi-Agent Service Environment), which is the evolution of our previous framework GAIN [22], that allows dynamically composing Web services. Its architecture is based on a society of agents, mostly composed of two kinds of agents: component managers and workflow managers.

Each component manager is associated to one or more Web services and is responsible for the interaction with them. Through the use of the WSIG JADE add-on [12], the component managers are able to invoke a Web service, converting ACL messages into WSDL descriptions and vice versa. Moreover, a component manager allows a flexible provision of services defining "on the fly" the features of the services (price, timing, etc.) through a set of business rules managed by a rule engine and modifiable by the operators of the service provider through a Web interface.

Workflow managers have the goal of supporting users in the process of building the workflows, composing external Web services and monitoring their execution. To accomplish this complex activity the workflow managers provide the users with two alternative automatic procedures:

Predefined workflow; the workflow is extracted from a repository of standard and common templates, e.g. templates used in previous computations. In this case the duty of the workflow manager is to support the user in the selection of the most appropriate Web services for the execution of the different workflow tasks. The workflow manager is able to select a matching service thanks to the exploitation of a shared ontology that gives a common knowledge background to all the agents in the system.

Dynamic workflow; the workflow manager, according to the user's requirements, creates a new workflow, composing the atomic services available in the system. This is done by applying a planner (we have realized it by extending the SGP planner [28]) that works on the operators extracted from the OWL-S descriptions of the Web services, provided by component managers. After the composition of the final workflow, the workflow manager is able to update it and possibly replace those Web services that are failed or no more available or cannot satisfy the execution time constraints.

Moreover, MASE offers to the users the possibility of manually building workflows. In this case, a personal assistant (i.e. an agent, associated with each user active in the system, responsible for the interaction between the user and the other parts of the system) helps its user presenting her/him the tasks (Web services) that can be composed and possibly informing her/him when the realized workflow does not satisfy the composition rules, coming from the related OWL-S descriptions. When a complete workflow is realized, the user can ask its personal assistant to delegate the workflow execution to a workflow manager. The enactment is clearly a problematic phase. When a workflow is going to be executed, a Web service could be no more available due to the expiration of a timeout, a failure of a resource or other unpredictable problems. In this case the workflow manager helps the user finding a new solution, creating a new contract phase with all the component managers that are able to satisfy the task and suggesting to the user the replacement of the failed service with the new one.

So far we have given a concise description of the system architecture and the responsibilities of the major system components, intentionally leaving out the treatment of the issues connected to the tools needed by the agents, in order to carry out their activities. In the following subsections, we will go into details, illustrating our proposals and the implemented tools.

A. The BPEL Engine

The WS-BPEL specification defines an XML-based language for the formal description of a business process based on Web services orchestration. It is an open standard recently approved as an OASIS standard.

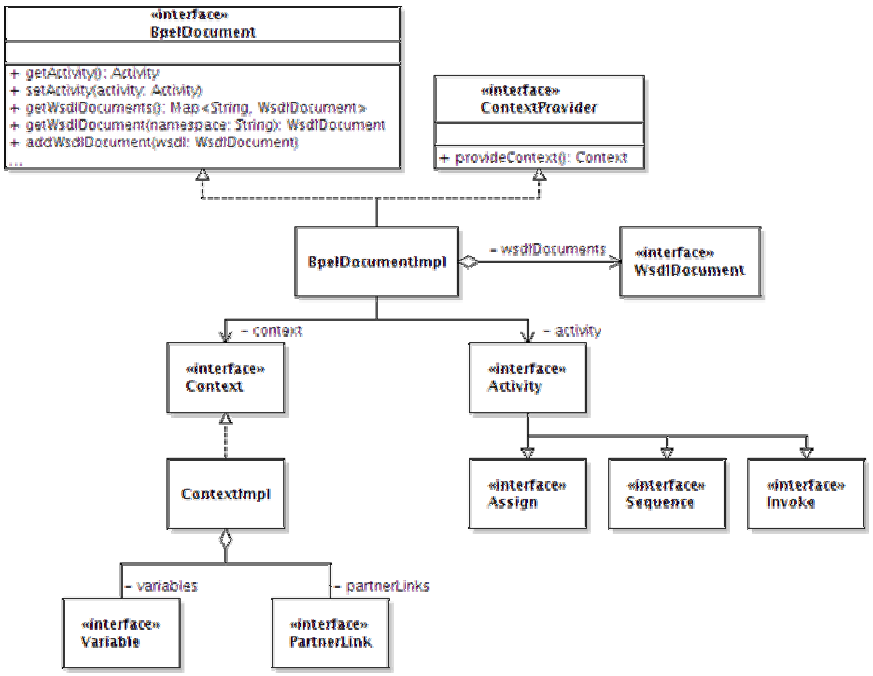


Figure 1 - Excerpt of the internal model representing the BpelDocumentImpl class and its relationship with the correlated classes

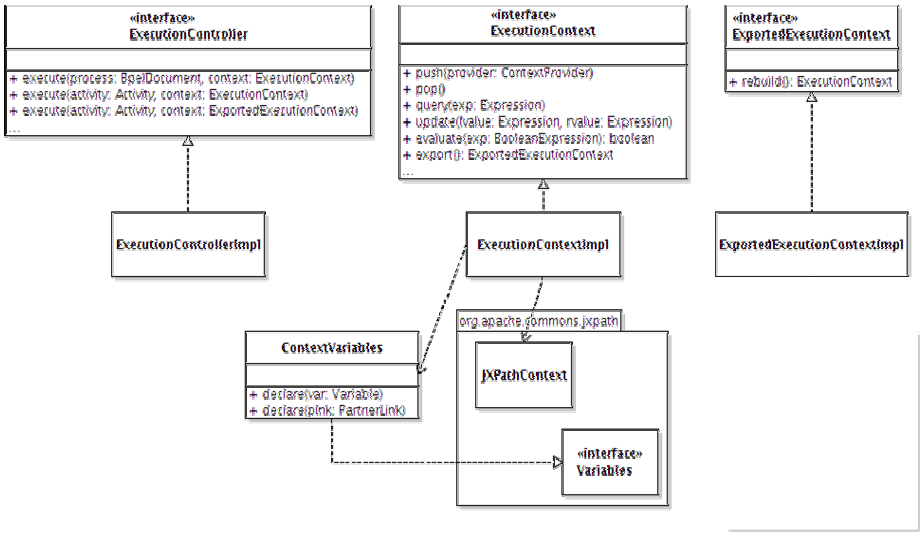


Figure 2 - Excerpt of the internal model representing the ControlFlowActivity class and its relationship with the correlated classes

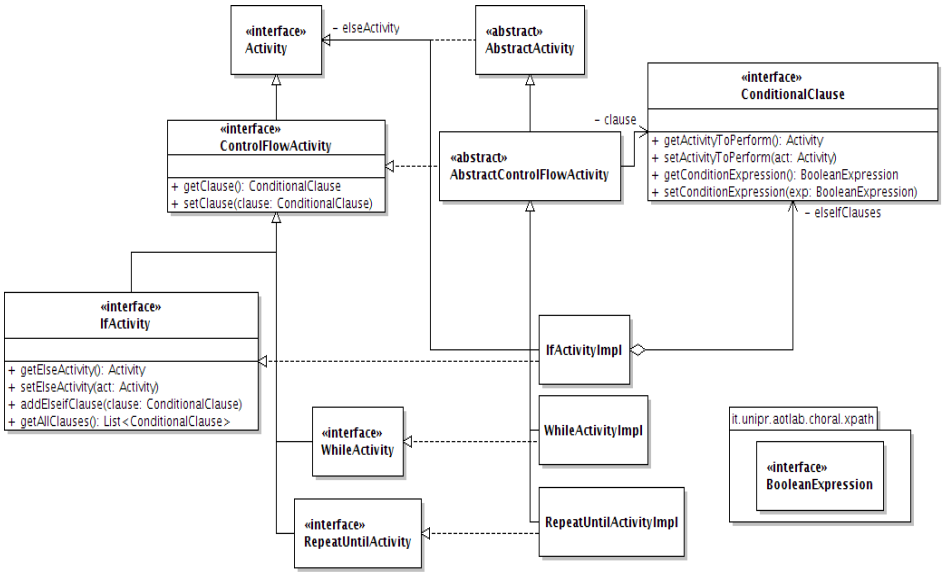


Figure 3 - The major classes of the BPEL engine

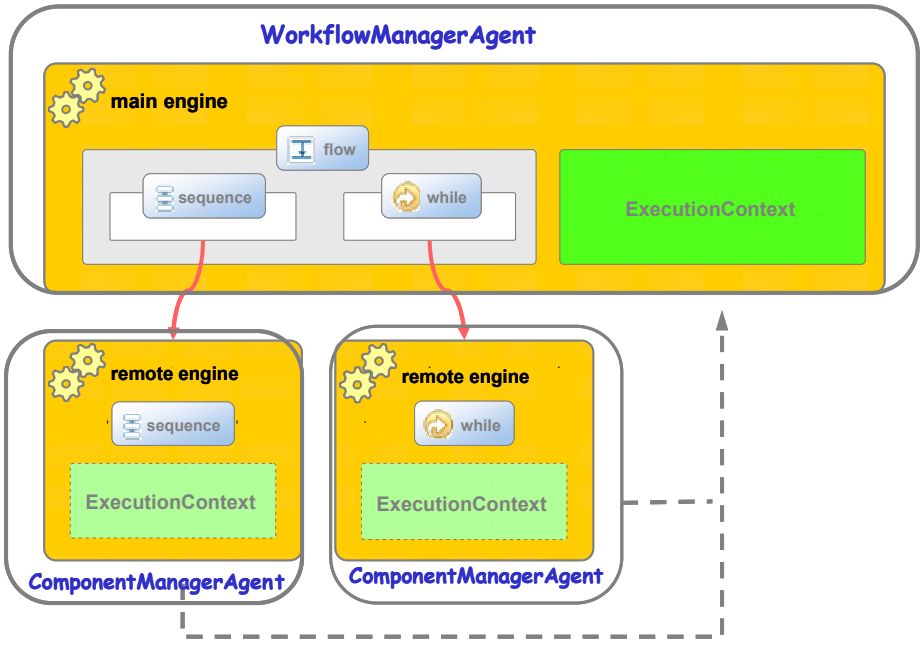


Figure 4 - An example of distributed execution

A WS-BPEL workflow is a structured XML document composed of three main parts: (i) the definition of the process' attributes, (ii) the definition of the execution context and (iii) the activities to be executed. Due to industry's increased interest on business process management and the wide acceptance of WS-BPEL as the language to use in the workflow definition, several vendors are producing software tools for workflow design, specification and enactment. The main drawbacks of these tools are that they enact the workflow in a centralized manner and furthermore they are not able to dynamically exploit new Web services in case of unpredictable event.

In the attempt to give an answer to such problems, we have realized a framework for the distributed execution of a BPEL process and the dynamically composition of Web services. The BPEL process execution is constituted of three phases: (i) interpretation of the BPEL document, (ii) creation of an internal process model, aiming at describing in a consistent way the business process characteristics and at the same time to make easy and efficient the execution of the business process itself, (iii) preparation of the execution context and distributed execution, possibly providing for the exploitation of new Web services.

In the first phase of the execution process we have utilized XMLBeans (a framework, part of the Apache XML project.), which has the advantage of fully supporting XML Schema and XML Infoset. As far as a BPEL process is concerned there are two schemas that have to be provided to XMLBeans in order to parser the BPEL document: WS-BPEL schema and WSDL schema (we have referred to WS-BPEL 2.0 and WSDL 1.1).

Regarding the second phase, in order to give an idea of how the internal model is structured, in Figure 1 and Figure 2 two excerpts of the model are shown. In particular, they represent respectively the model core class, i.e. the BPELDocumentImpl class, and a representative BPEL structured activity, i.e. the AbstractControlFlowActivity class, together with their major correlated classes.

The engine is responsible for efficiently executing a BPEL process, instance of the model. The classes, playing a key role in its implementation, are shown in Figure 3.

The main engine, part of the workflow manager agent, is responsible for initiating and coordinating the entire execution process. It creates the execution context, an instance of the ExecutionContextImpl class, which will represent the reference context during the execution process. Next, it will identify those parts of the workflow (e.g. scope activities, sub-activities of the flow activities, and so on), that if executed remotely will positively affect the performance of the system, and will delegate their execution to specific component manager agents. Figure 4 shows an example of distributed execution.

From what said above, it emerges clearly that we have chosen to have stateless engines and thus to share the execution context. As a consequence the remote engines have to send messages to the main engine in order to update consistently the reference execution context. We have found out that this choice has several advantages, primarily it makes easier to handle the possibly dependency between activities.

B. Ontology Support

The idea which mostly inspired the design of the JADE content language and ontological support was to define an ontology independent abstract model of the content language that could be subsequently bound to any domain ontology representation expressed using an object-oriented data model. This ontological support has been conceived when the Semantic Web was on its very early stage of research and development and OWL was not already established as a standard. Consequently its expressive power is clearly limited with respect to OWL and basically allows expressing taxonomy of concepts, predicate and actions and therefore it is not able to represent completely the different application domains where JADE agent may be used.

In order to provide a JADE agent with an adequate expressive power (i.e., equivalent to the one offered by OWL DL), it is necessary either to replace or to integrate the JADE ontological support. In the attempt to find a suitable solution to this problem one has to choose among the proposals described above and others, each characterized by different domain knowledge modelling techniques and answering different needs. The majority of the research work in this field is thought for the semantic Web. But while in the vision of the semantic Web the increasing interest in ontologies is driven by the large volumes of information available and by the need of automating many information retrieval activities, in the agent context the focal point is slightly different and it is mainly on communicative acts - communications which implies actions. Agents would use ontologies to perform the proper semantic checks on a given content expression, and therefore ontologies should include concepts (objects of the domain of the discourse) but also predicates (assertions on properties of concepts) and actions (that agents can perform in the domain). Moreover a peculiar characteristic of the agent community is the heterogeneity of resources available and the roles played by different agents of a system. This leads us to choose different approaches in different contexts. Our solution was to realize a compound tool, called OWLBeans [5], that allows the use of ontologies described by using OWL DL. These ontologies can be used by agents for performing their tasks in cooperation with other agents, for interoperating with external entities (e.g., legacy software systems) and for performing a semantic matching of Web services, described by using OWL-S and having inputs and outputs associated with concepts belonging to a domain ontology.

OWLBeans is based on a two-level approach with the aim of coping with both the issues of managing complex ontologies and of providing ontology management support to lightweight agents, which seldom need to deal with the whole complexity of a OWL DL ontology. Therefore, lightweight agents maintain the simple JADE ontology support whereas one or more dedicated agents, acting as ontology servers, are able to

use and manage complete OWL DL ontologies and provide the service to the agents that need it.

The main functionality of OWLBeans is to extract JADE ontologies from OWL DL ontologies realizing a set of ontologies usable by JADE agents, with the obvious shortcoming that not all the information maintained in the original OWL ontologies are taken into account. Therefore, for all those systems that need a complete support for OWL DL ontologies, OWLBeans offers a set of ontology server agents implemented as JADE agents, providing a common knowledge base and reasoning facilities. These ontology servers use the Jena toolkit to load, maintain and reasoning about OWL ontologies. The other agents of the system do not need to know anything about the Jena toolkit given that these ontology servers provide them with a set of simple actions for querying and manipulating the ontologies. Furthermore, ontology servers take into account proper authorization mechanisms. In particular, the underlying trust management support (discussed in the following subsection) has been leveraged to implement a certificate-based access control. Only authenticated and authorized principals will be granted access to managed ontologies. A delegation mechanism allows the creation of communities of trusted entities, which can share a common ontology, centrally managed by the ontology server.

Finally, despite the fact that the JADE ontological support is quite simple, it could still be complex for some devices with limited resources such as smart phones. This is the reason why we have decided to improve OWLBeans adding a further feature which allows agents to import taxonomies and classifications from OWL ontologies, in the form of a hierarchy of Java classes with the purpose of providing very simple artefacts to access structured information. Given its modular architecture, based on an intermediate ontology model, OWLBeans also provides further functionalities, e.g., saving a JADE ontology into an OWL file, or generating a package of JavaBeans from the description provided by a JADE ontology.

C. Production Rule Management

JADE provides the integration with the JESS rule engine, which is probably the most known Java rule engine implementing the Rete algorithm [11]. This integration is realized through a so-called JessBehaviour that allows the encapsulation of a JESS rule engine inside a JADE agent and has the duty of storing and retrieving information in/from the rule engine. The main limits of this solution are: i) the rule engine is completely hidden to the other agents of the system and there is not any support for the cooperation among different rule-based agents (i.e., agents encapsulating a JESS rule engine) and ii) JESS is a commercial software and so we have additional costs if we plan to realize commercial applications by using JADE together with JESS.

In order to cope with the limits of the current JADE support for rule engines, we realized a software library, called D4J (Drools4JADE) [4], that integrates JADE agents with the Drools rule engine. Drools is a well known, freeware implementation of the so-called Rete-OO algorithm. Apart of its open-source availability, one of the main advantages of Drools is exactly the fact that it is not just a literal implementation of the Rete algorithm, but rather an adaptation for the object-oriented world. This greatly eases the burden of integrating the rule engine and the application rules with the existing external objects. In Drools, asserted facts are simple Java objects, that can be modified through their public methods and properties. Where Jess requires hundreds of lines of code, for example to simply access an ACL message mapped into a Java object, Drools rules can obtain the same result in a dozen of easy-reading code lines.

D4J guarantees both the advantages of full rule-based agents, i.e., agent whose behaviour and/or knowledge is expressed by means of rules [17], and the advantages of rule-enhanced agents, i.e., agents whose behaviour is not normally expressed by means of rules, but that use a rule engine as additional component to perform specific reasoning, learning or knowledge acquisition tasks [14]. In facts, in D4J, the Drools rule-engine is integrated into an agent as a JADE behaviour, but it also provides an API for interacting with it through ACL messages allowing both remote storing and retrieval of knowledge and the cooperation among different rule-based agents. Moreover, this API allows rules mobility, i.e., a rule-based agent can move a rule to another rule-based agent.

Given their nature, business rules often refer to domain specific concepts and, especially when dealing with data on the semantic Web, these concepts are part of a domain ontology. To better support this scenario, rule-enhanced JADE agents should be augmented with a tool for the automatic transformation of concepts, relations and individuals of an OWL ontology to java classes, properties, and instances. The D4J framework can be integrated with OWLBeans, which enables the extraction of JavaBeans from an OWL ontology. The JavaBeans can then be directly asserted as facts into the working memory of Drools.

D. Distributed Trust Management

Out of the box e-business applications are not certainly possible if security problems are not analyzed and addressed. Our framework supports the implementation and deployment of secure systems, adhering trust management principles. For this purpose, local names, interpreted as distributed roles, and delegation certificates are made available, to build peer-to-peer networks of trusted entities.

The accurate release of authorizations is often the most critical point of security systems. Ideally, systems should respect the principle of *least privilege*, but this often contrasts with other requirements, as easiness of understanding, scalability and manageability. In this respect, the RBAC model has proven to be a good abstraction to manage large and complex systems, up to corporate and virtual-organizations environments.

Following the RBAC model, each resource manager of our system (i.e. each node in the peer-to-peer network) has to deal

with three main concepts: principals (i.e. authenticable entities which act as users of resources and services), permissions (i.e. rights to access resources or use services) and roles.

A many to many relationship binds principals and the roles they are assigned to. In the same way, a many to many relationship binds permissions and the roles they are granted to, thus creating a level of indirection between a principal and his access rights. This also leads to a better separation of duties (between the assignment of principals to roles and the definition of role permissions), to implement privilege inheritance schemes among superior and subordinate roles and to permit temporary delegations of some of the assigned roles towards other principals. The fundamental principle here is that each node is in charge of defining its own roles, and of assigning principals to them.

Dynamic delegation of access rights is made possible through the use of delegation certificates, whose structure is based on the theory of [9]. But we have avoided s-expressions, preferring XML to them, as it provides a better ground to exploit and integrate standard technologies. In particular, in recent times SAML [27] have emerged as a language to express properties of authenticable principals, encoded in the generic form of signed security assertions. SAML assertions can easily bind public keys to local names, and certify the links between two different local namespaces, as it happens in SDSI/SPKI certificates.

Delegation is particularly important to deal with the activation of intermediate agents, acting between the human user and the concrete service providers, i.e. personal agents, workflow managers and agents providing composite services. In this case, privileges must be forwarded in the form of delegation certificates from the user toward each agent in the chain, up to the final service provider, which will check them for consistency with local security policies. These policies are stored and managed locally as XACML documents [31].

To cope with the security problems coming from the remote utilization of the rule engine and from the mobility of rules, also D4J exploits this security layer to enforce security policies at two different levels: proper authorization is necessary to modify the working memory and the rule set of an agent; moreover, each rule is associated with a specific protection domain, limiting the resources made accessible when it is scheduled for execution.

IV. Conclusion

In this paper, we have presented an agent-based framework for SOA that integrates agent technology with other technologies that have found, and will find, a purpose within enterprise computing: web services, workflows, ontologies and rule engines.

Up to now, we have not experimented the system with real users and real services, but we have tested and evaluated the system functionalities implementing some “artificial” services and involving a group of students, acting either as service provider operators or as customers. Some of the information used by the service providers, implemented just for the experimentation of the system, comes from the Web site of some real service providers (e.g., flight companies, hotel brokers, etc.). The results, though still preliminary, are quite encouraging

We are well aware that the current multi-agent solutions need to be improved since the technologies used are still not completely mature. However a lot of researchers and software developers are really interested in giving a significant contribution in this direction, driven by the motivation of providing a strengthening of the related standards and new methodologies, algorithms and implementations to realize real flexible, adaptive SOA [1],[10].

Our future activities will be oriented towards the aforementioned goal. In particular, we will continue working on the JADE software environment in order to both improve the integration of the JADE agents with the most interesting knowledge and internet-oriented technologies and realize real adaptive agents that will be the basis of next and future business applications. At present, we are working in three main directions: i) to finalize the implementation of a full OWL DL support through a home-made framework supplying ontology management and reasoning functionalities, with the main purpose of reducing the amount of computational resources and time required (compared to the Jena engine), ii) to enhance the distribution algorithm in order to achieve a more efficient execution of a workflow and iii) to finalize the implementation of a framework for the dynamic composition of semantic Web services.

References

[1] AgentLink III. Agent Technology Roadmap. Available from http://www.agentlink.org/roadmap/index.html

[2] Akkermans, H. Intelligent E-Business - From Technology to Value. IEEE Intelligent Systems, 16(4):8-10, 2001.

[3] Bechhofer, R. Volz, and P. Lord. Cooking the semantic web with the OWL API. In Proc. Int Semantic Web Conference, pp. 659-675, Sanibel Island, FL, 2003.

[4] Beneventi, A., Poggi, A., Tomaiuolo, M., & Turci, P. Integrating Rule and Agent-Based Programming to Realize Complex Systems. WSEAS Trans. on Information Science and Applications, 1(1):422-427, 2004.

[5] Bergenti, F., Poggi, A., Tomaiuolo, M., Turci, P. An Ontology Support for Semantic Aware Agents. In Proc. Seventh International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2005 @ AAMAS), Utrecht, The Netherlands, 2005.

[6] Berners-Lee, T., Hendler, J., Lassila O. The Semantic Web - A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. 284(5):34-43, 2001.

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

[8] de Bruijn, J., Polleres, A., Lara, R., Fensel, D. OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web. In Proc. of the 14th Int. World Wide Web Conference (WWW2005), pp. 623-632, Chiba, Japan, 2005.

[9] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T. SPKI Certificate Theory. RFC 2693, 1999.

[10] Fensel, D., Bussler, C. The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications 1(2): 113-137, 2002.

[11] Forgy, C. L. (1982). Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem, Artificial Intelligence 19(1), pp. 17-37.

[12] Greenwood, D., Callisti, M. Engineering Web Service-Agent Integration. In IEEE Conference of Systems, Man and Cybernetics, 2004. Available from http://www.whitestein.com/resources/papers/ieeesmc04.pdf

[13] Gibbins, N., Harris, S., Shadbolt, N. Agent-based semantic web services. In Proc of the 12th International World Wide Web Conference (WWW2003), Budapest, Hungary, 2003.

[14] Gutknecht, O., Ferber, J., Michel, F. Integrating tools and infrastructures for generic multi-agent systems. In Proc. of the 5th International Conference on Autonomous Agents. Montreal, Canada, 2001.

[15] Gutmann, P. (2000). X.509 Style Guide. Available from http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt

[16] Horrocks, I. and Patel-Schneider, P. F. A proposal for an OWL rules language. In Proc. of the Thirteenth International World Wide Web Conference (WWW 2004), pp. 723-731, 2004.

[17] Hindriks, K.V., de Boer, F.S., van der Hoek, & W., Meyer, J.C. Control Structures of Rule-Based Agent Languages. In Lecture Notes In Computer Science, Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages, Vol 1555, pp. 381-396, 1998, London, UK: Springer-Verlag.

[18] Khare, R., Rifkin, A. Weaving a Web of trust, World Wide Web Journal Special Issue on Security, 2(3):77–112, 1997.

[19] R. Kishore, H. Zhang & R. Ramesh, Enterprise integration using the agent paradigm : foundations of multi-agent-based integrative business information systems, Decision Support Systems, 42 (2006) (1), pp. 48–78.

[20] Labrou, Y. Agents and ontologies for e-business. Knowledge Engineering Review, 17(1):81-85, 2002.

[21] Li, N., Mitchell, J.M.. RT: A Role-based Trust-management Framework. In Proc of the Third DARPA Information Survivability Conference and Exposition (DISCEX III), pp. 201-212, 2003. Washington, D.C.

[22] Negri, A., Poggi, A., Tomaiuolo, M., Turci, P,. Dynamic Grid Tasks Composition and Distribution through Agents,. Concurrency and Computation: Practice and Experience, 2005.

[23] Negri A., A. Poggi, M. Tomaiuolo, P. Turci, Agents for e-Business Applications, In Proc. AAMAS 2006, Hakodate, Japan 2006

[24] Papazoglou, M.P.. Agent-oriented technology in support of e-business. Communication of ACM, 44(4):71-77, 2001.

[25] Papazoglou, M.P. The World of e-Business: Web-Services, Workflows, and Business Transactions In Lecture Notes In Computer Science, CAiSE '02/ WES '02: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web, Vol 2512, pp. 153-173, 2002. London, UK. Springer-Verlag

[26] Poggi, A., Rimassa, G., Tomaiuolo, M. Multi-user and security support for multi-agent systems. In Proc. of WOA 2001, pp. 13-18, 2001. Modena, Italy: Pitagora.

[27] SAML - Security Assertion Markup Language. Available from http://xml.coverpages.org/saml.html

[28] Sensory Graph Planner software and documentation. Available from http://www.cs.washington.edu/ai/sgp.html.

[29] Silva, N., Rocha, J., Cardoso, J. E-Business Interoperability Through Ontology Semantic Mapping. In Proc. of the Processes and Foundations for Virtual Organizations, pp. 315-322, 2003. Lugano, Switzerland.

[30] Weikum G. Special Issue on Infrastructure for Advanced E-services, IEEE Data Engineering, 24(1), 2001.

[31] XACML - Extensible Access Control Markup Language. Available from http://xml.coverpages.org/xacml.html

[32] YURL - Decentralized Identification. Available from http://www.waterken.com/dev/YURL.

Indice degli Autori