Matteo Baldoni, Cristina Baroglio,
and Viviana Mascardi (eds.)

Agent, Web Services, and Ontologies Integrated Methodologies

Int. Workshop MALLOW-AWESOME'007
Durham, September 6th–7th, 2007
Proceedings

MALLOW-AWESOME'007 Home Page:
`http://awesome007.disi.unige.it/`

Preface

Dear attendee, welcome to MALLOW-AWESOME'007!

MALLOW-AWESOME'007 is at its first edition this year, first edition of a hopefully long series. It was born for stimulating discussion among researchers and practitioners working on Agents, Web Services, and Ontologies, in order to help the identification and the definition of Methodologies for integrating them.

The realisation of distributed, open, dynamic, and heterogeneous software systems is, in fact, a challenge that involves many facets, from formal theories to software engineering and practical applications. Scientists in various research areas, such as Semantic Web, Web Services, Agents, Ontologies, are attacking this problem from different perspectives. MALLOW-AWESOME'007 attempts to provide a discussion forum for collecting and comparing such diverse experiences with the aim of fostering cross fertilization.

MALLOW-AWESOME'007 is being held as part of MALLOW'007, the first edition of Multi-Agent Logics, Languages, and Organisations Federated Workshops, in Durham, UK.

This volume contains the ten papers that have been selected by the Programme Committee for presentation at the workshop. In addition to these presentations, Professor Munindar Singh from North Carolina State University, USA, and Professor Julian Padget from University of Bath, UK, will be giving invited talks.

Each paper received at least three reviews in order to supply the authors with a rich feedback. The papers contributions cover hot topics in the fields of agents, web services, and ontologies, including services and ontologies in BDI and goal-oriented agents, interaction processes in service-oriented systems, communication and argumentation among agents, and integrated applications. Selected and expanded papers will be published as a special issue of the Multiagent and Grid Systems International Journal.

We would like to thank all authors for their contributions and the members of the Programme Committee for the excellent work during the reviewing phase.

August 10th, 2007

Matteo Baldoni
Cristina Baroglio
Viviana Mascardi

Workshop Organisers

Matteo Baldoni	Università di Torino, IT
Cristina Baroglio	Università di Torino, IT
Viviana Mascardi	Università di Genova, IT

Programme Committee

Mario Bravetti	Università di Bologna, IT
Antonio Brogi	Università di Pisa, IT
Thomas Eiter	Technische Universitaet Wien, AT
Amal El Fallah Seghrouchni	University of Paris 6, FR
Laura Giordano	Università del Piemonte Orientale, IT
Georg Gottlob	Oxford University, UK
Pilar Herrero	Universidad Politécnica de Madrid, ES
Benjamin Hirsch	TU Berlin, DE
Peter Massuthe	Humboldt University Berlin, DE
Paola Mello	Università di Bologna, IT
Andrea Omicini	Università di Bologna, IT
Viviana Patti	Università di Torino, IT
Adam Pease	Articulate Software, California, US
Marco Pistore	ITC-IRST, Trento, IT
Laurent Prevot	Academia Sinica in Taipei, Taiwan
Paolo Rosso	Universidad Politécnica de Valencia, ES
Munindar Singh	North Carolina State University, US
Leon Sterling	The University of Melbourne, AU
Birna van Riemsdijk	Ludwig Maximilians University, Munich, DE
Chris Walton	Slam Games, UK
Michael Winikoff	RMIT University, AU
Yuhong Yan	National Research Council Institute for Information Technology, Fredericton, Canada
Nobuko Yoshida	Imperial College, London, UK
Gianluigi Zavattaro	Università di Bologna, IT

Additional Reviewers

Raymond Hu	Raman Kazhamiakin
Sara Corfini	Niels Lohmann

Sponsoring Institutions

Matteo Baldoni and Cristina Baroglio have 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 they have also been supported by MIUR PRIN 2005 "Specification and verification of agent interaction protocols" national project.

Viviana Mascardi has partially been funded by the MIUR PRIN 2005 "Specification and verification of agent interaction protocols" national project.

Table of Contents

X

Commitment-Based SOA

Munindar P. Singh
(Joint work with Amit K. Chopra and Nirmit Desai)

Department of Computer Science
North Carolina State University, USA
http://www.csc.ncsu.edu/faculty/mpsingh/

Abstract. The vision of service-oriented computing is centered on business services. By contrast, existing service-oriented architectures are formulated in terms of low-level abstractions that are far removed from business services. This talk describes a new architecture whose components are business services and whose interconnections are modeled in terms of the commitments that support key aspects of service engagements. This talk also shows how this architecture relates to existing SOAs.

Regulation Frameworks, Semantics and Service Oriented Architectures

Julian Padget

Department of Computer SCience
University of Bath, UK
http://www.cs.bath.ac.uk/ jap/

Abstract. The concept of the "institution" as used in the social sciences, management and economics captures the principle of right (and wrong) action and how an observable action in the real world counts as an institutional action, bringing about a change of institutional state. Complementary to these "rules of engagement" is the identification of the right actors - the searcher's problem - and how to describe an actor's attributes effectively - the publisher's problem.
As software development apparently moves towards increasingly open architectures of loosely-coupled software components, we believe institutional models are of increasing relevance as a means to categorize formally the correct and incorrect behaviour of (collections of) software components, that necessarily must operate within multiple regulatory frameworks. Likewise, formal descriptions of institutions and of software components, and the means to reason about them both, appear to have a critical role in supporting the component selection, composition and enactment.
Our work to date has focussed (i) on agent-based systems, developing a formalization of institutions and the interactions between institutions, and (ii) service discovery for semantic web-services, developing a generic matchmaking and brokerage factory framework. The talk aims to draw these strands together and explore how semantic technologies and institutional frameworks might be applied to the construction of service oriented architectures.

Goal-Oriented and Procedural Service Orchestration*

A Formal Comparison

M. Birna van Riemsdijk Martin Wirsing

Ludwig-Maximilians-Universität München, Germany
{riemsdijk, wirsing}@pst.ifi.lmu.de

Abstract. Goals form a declarative description of the desired end result of (part of) an orchestration. A goal-oriented orchestration language is an orchestration language in which these goals are part of the language. The advantage of using goals explicitly in the language is added flexibility in handling failures. In this paper, we investigate how goal-oriented mechanisms for handling failures compare to more standard exception handling mechanisms, by providing a formally defined translation of programs in the goal-oriented orchestration language into programs in the procedural orchestration language, and proving that the procedural orchestration has the same behavior as the goal-oriented orchestration.

1 Introduction

In the field of agent-oriented programming, there is an increasing amount of research on the use of *goals* in agent programming languages (see, e.g., [24, 8, 21, 17, 3, 9, 20]). Goals form a *declarative description* of the desired end result of the execution of (part of) a program. They are thus comparable to postconditions as commonly used in program verification. However, the important difference between goals and postconditions is that goals, in contrast with postconditions, are *part of the program*. A goal-oriented language has language constructs which express the goal that is to be reached by some part of the program.[1]

It is generally argued that one of the advantages of the explicit use of goals in a programming language is *added flexibility* in handling failures [24, 19, Chapter 5]. The idea is essentially that goals are used to monitor the execution of statements, or *plans* in agent terminology. If the execution does not have the desired result, goals are used to *select a different plan*. This mechanism is used recursively, as plans can contain subgoals. The fact that a program and its parts contain explicit representations of the desired result of their execution thus facilitates monitoring their execution and taking appropriate measures by trying alternative courses of action if the execution fails to achieve these results.

* This work has been sponsored by the project SENSORIA, IST-2005-016004.

[1] Goal-oriented programming should not be confused with logic programming. While the latter is in principle purely declarative, goal-oriented programming has both declarative and procedural features.

Goal-oriented programming is targeted at dynamic domains such as agent-based systems in which the programmer does not have full control over all aspects of system behavior, e.g., due to the existence of other agents or environmental aspects outside control of the agent. In such systems, one always needs to take into account that things might "go wrong". In more restricted settings in which one can prove that a program always fulfills some desired post-condition (perhaps assuming some generally valid preconditions), goal-oriented programming is superfluous (apart from possible modeling advantages of using goals). Monitoring the execution for goal achievement does not add anything in that case, as the program was already proven to satisfy the postconditions or goals.

We argue that the domain of service-oriented computing is, like agent-based systems, a domain well-suited for using goal-oriented techniques (see, e.g., [2, 12, 1, 6] for other proposals for combining agent-oriented and service-oriented approaches). In the service-oriented systems domain, services are called on the basis of service descriptions without knowing anything about the internal architecture or workings of the service. One will typically not have or be able to obtain (formal) guarantees that the service behaves as it should. In such a setting, one will thus always need to take into account that a service does not behave as expected or desired. Moreover, in such a context it is more natural than in classical settings to specify alternative plans for reaching a goal. In more classical settings such as database applications there will typically not be alternative ways of reaching a desired result, e.g., in case accessing a database fails. In service-oriented systems, on the other hand, trying alternative ways of reaching a goal is more natural. For example, if booking a ticket with Lufthansa did not succeed, one might try booking a ticket with KLM, or if booking a plane turns out not to be possible as it is too expensive, one might try booking a train.

To investigate how goal-oriented techniques can be applied in the context of service-oriented systems, we have proposed an abstract *goal-oriented orchestration language* [23] (Section 3). A natural question that arises, given that we argue that goal-oriented techniques increase flexibility in handling failures, is how this kind of failure handling compares to more standard exception handling mechanisms. The aim of this paper is to answer this question. Our approach is that we define a procedural orchestration language with an exception handling mechanism inspired by that of WS-BPEL [10] (Section 4). We then show how a program in the goal-oriented orchestration language can be translated into a program in the procedural orchestration language that has *provably* the same behavior (Section 5). We will argue that the kind of abstractions as used in the goal-oriented orchestration language are worth considering as language constructs of an orchestration language, as the programming patterns resulting from the translation do not increase understandability of the code.

It is important to remark that the orchestration language of [23] is not meant to be a full-fledged orchestration language. It is based on propositional logic, and is used to investigate the semantic foundations[2] of goal-oriented orchestration

[2] "Semantic" is here meant in the sense of "semantics of programming languages", not in the sense of "semantic web technology".

languages. The relative simplicity of propositional logic allows us to focus on the essential aspects of such a language. This paper contributes to the investigation of the semantic foundations of goal-oriented orchestration, and hence is also based on the simple language [23]. We are currently investigating how we can replace propositional logic by other logics such as description logic, to make the language more practically useful and to facilitate more extensive experimentation with it. We refer to [5] for the description of a goal-oriented agent programming language and platform based on first-order logic rather than propositional logic, which uses similar goal-oriented techniques as the ones we use in this paper.

Moreover, we remark that this paper addresses the composition of services using orchestration languages. The idea is that the *programmer* specifies which compositions are appropriate, using the constructs of the orchestration language. At run-time, the orchestration is executed as specified. This is in contrast with approaches to service composition based on planning (see, e.g., [14]). In the latter approaches, a composition is generated *automatically* on the basis of service descriptions and a specification of desired behavior. Nevertheless, planning approaches and programming approaches have many commonalities, and can sometimes be combined [16].

2 Example: Car Breakdown

In order to illustrate our approach, we use a very simple car breakdown scenario that is adapted from the automotive case study of the SENSORIA project [25] on service-oriented computing. We have used a variant of this scenario in [23]. In the scenario, the car has a diagnostic system which reports a failure, resulting in the car no longer being drivable. The car is furthermore endowed with orchestration software that should assist the driver in getting the appropriate support by calling, e.g., a service to get road side assistance. We assume there are also services available for calling a taxi, for making garage appointments, for ordering a tow truck, and for getting technical advice over the phone (this service makes sure the driver is phoned by the appropriate technical assistant).

Using the goal-oriented orchestration language, one can specify which plan may be executed for achieving a certain goal, under certain circumstances using so-called *plan selection rules* [19]. Plans essentially consist of service calls (where the goal to be achieved through the service call is passed as a parameter, possibly together with some additional information), subgoals (which are to be achieved by selecting an appropriate plan by means of plan selection rules), and a construct for sequential composition (inspired by the orchestration language Orc [4]) that can be used for passing along the result of service calls to other service calls. Services can be called directly by specifying the service name, or they can be discovered by matching available service descriptions to the goal of a service call, i.e., through *semantic matchmaking*. Goals to be achieved are preceded by an exclamation mark.

In this example, we assume the driver is on his way to work, i.e., he has "being at work" as its top-level goal. If the car is broken, he may either leave the

car behind and call a taxi (if he is in a hurry and near to his office), or try to get the car repaired. There are three alternative plans for getting the car repaired: the driver can repair the car himself with the help of technical support over the phone (if he is a member of this service and the car is repairable on the spot), he can get road side assistance, or have the car towed to a garage (if it is not repairable on the spot). Below, we sketch the corresponding plan selection rules. Plan selection rules have the form $\kappa \mid \beta \Rightarrow \pi$, which intuitively says that the plan π can be used to reach goal κ if β is the case.

$$
\begin{array}{ll}
\text{!atWork} \mid \text{carBroken} \wedge \text{hurry} \wedge \text{nearOffice} & \Rightarrow d(\text{!(taxi} \leq 50 \text{ euro)}) \gg \text{monitor(!atWork)} \\
\text{!atWork} \mid \text{carBroken} & \Rightarrow \text{!carRepaired} \gg \text{monitor(!atWork)} \\
\text{!carRepaired} \mid \text{memberTS} \wedge \text{repOnSpot} & \Rightarrow \text{techSupport(symp, !appTA)} > x > \\
 & \quad \text{notify}(x) \ldots \\
\text{!carRepaired} \mid \mathit{true} & \Rightarrow d(\text{locationCar, !roadSideAss}) \ldots \\
\text{!carRepaired} \mid \neg \text{repOnSpot} & \Rightarrow \text{!appGarage} \gg \text{!appTowTruck} \ldots
\end{array}
$$

We leave out the plan selection rules for the subgoals !appGarage and !appTowTruck for reasons of space. It may be the case that multiple plan selection rules are applicable in a certain situation. For example, if the car is broken and the driver is in a hurry and near the office, either the first or the second rule may be applied. In our abstract formal framework, one applicable rule is selected non-deterministically. However, the framework may be extended with a preference ordering over the rules. If we assume the first rule is selected, it may be the case that discovering a taxi service fails (the "d" stands for "discovery"), e.g., because it was not possible to find a taxi service for less than 50 euro. The plan is then aborted and the goal of being at work has not been achieved, after which the other applicable plan selection rule will be tried, i.e., it will be tried to achieve the subgoal of getting the car repaired. Note that it is thus useful that multiple plan selection rules are applicable in a certain situation, as another plan can then be tried if one fails. The monitoring service monitors whether, e.g., taking a taxi has resulted in the goal of being at work being achieved.

In order to achieve the subgoal of getting the car repaired, one might first try to repair the car with help of technical support over the phone (passing the symptoms of the car problem to the service). If contacting the technical support service is successful, the plan continues by passing along the result to a service that notifies the secretary of the driver about this. If calling the technical support service fails, e.g., because it turned out the membership has expired, or the service could not provide satisfactory support, the plan is aborted and another plan for reaching the goal of getting the car repaired can be tried. Our goal-oriented orchestration language tries each alternative plan to achieve a certain (instance of a) subgoal *once*, in order to prevent the orchestration from getting stuck by trying the same plans over and over to reach some subgoal.

A sketch of how this example could be programmed in a procedural orchestration language is provided below (we only show the "carRepaired" part).

$$
\begin{array}{l}
\text{carRepaired}(\mathit{tried}_1, \mathit{tried}_2, \mathit{tried}_3, \mathit{from}) \Rightarrow \textsf{if } \mathit{tried}_1 = \mathit{false} \wedge \text{memberTS} \wedge \text{repOnSpot} \\
\quad \textsf{then } \mathit{tried}_1 := \mathit{true}; x := \text{techSupport(symp, !appTA)}; \\
\qquad \textsf{if } \neg \mathit{ach}(\text{!appTA}) \textsf{ then throw } \text{!carRepaired.}\textsf{planFailedExc else notify}(x) \ldots \textsf{ fi} \\
\quad \textsf{else if } \mathit{tried}_2 = \mathit{false} \textsf{ then } \mathit{tried}_2 := \mathit{true}; d(\text{locationCar, !roadSideAss}); \\
\qquad\quad \textsf{if } \neg \mathit{ach}(\text{!roadSideAss}) \textsf{ then throw } \text{!carRepaired.}\textsf{planFailedExc else } \ldots \textsf{ fi} \\
\qquad \textsf{else if } \mathit{tried}_3 = \mathit{false} \wedge \neg \text{repOnSpot} \\
\qquad\quad \textsf{then } \mathit{tried}_3 := \mathit{true}; \text{appGarage}(\ldots); \text{appTowTruck}(\ldots) \\
\qquad\quad \textsf{else throw } \mathit{from}.\textsf{planFailedExc fi fi fi}
\end{array}
$$

$$
\text{!carRepaired.}\textsf{planFailedExc} \Rightarrow \text{carRepaired}(\mathit{tried}_1, \mathit{tried}_2, \mathit{tried}_3, \mathit{from})
$$

The various plan selection rules for achieving a particular goal (!carRepaired in this case) are combined into one procedure, and subgoals occurring in plans are translated into procedure calls. After each service call, it is checked whether the service call was successful in achieving its goal ($\mathit{ach}(\mathit{goal})$). If not, an exception is thrown, as the plan should be aborted in this case. The exception handler for !carRepaired.planFailedExc as specified above calls the procedure "carRepaired" recursively, so that *another* plan can be tried to achieve the goal. We use the variables tried_i to record which plans have already been tried to reach the goal. If all plans have been tried and/or none are applicable, the exception from.planFailedExc is thrown which is caught lower down in the procedure call stack (in the procedure "atWork" in this case, as the procedure for repairing the car will be called from there, as recorded in the variable from).

We believe the code of this procedural orchestration is less understandable than the goal-oriented version, and we thus argue that goal-oriented abstractions are worth considering as language constructs of an orchestration language.[3] The purpose of the rest of this paper is to analyze the failure handling mechanism of the goal-oriented orchestration language in more detail, and to investigate the relation between the goal-oriented and procedural orchestration language from a foundational perspective by showing how an arbitrary goal-oriented orchestration can be translated into a procedural orchestration that has provably the same behavior.

3 Goal-Oriented Orchestration Language

In this section, we present the syntax and informal semantics of our goal-oriented orchestration language (Section 3.1), and the part of the formal semantics that is relevant for failure handling (Section 3.2). For reasons of space, we cannot provide the full semantics. We refer to [23, 22] for more details and explanation.

[3] Albeit not necessarily to replace procedural programming constructs, but at least in addition to them.

3.1 Syntax and Informal Semantics

Most of the ingredients of the goal-oriented orchestration language have already been introduced informally in Section 2. Here, we provide the full syntax and introduce the formal notation. A program in the goal-oriented orchestration language is called an agent, which is formally a tuple $\langle \sigma_0, \gamma_0, \mathsf{PS}, \mathcal{T} \rangle$. The initial *belief base* σ_0 represents what the agent believes to be the case in the world (comparable with the state of a procedural program), and is a consistent set of propositional formulas [19]. The initial *goal base* γ_0 is the set of top-level goals of the agent. Goals are deleted from the goal base if they are believed to be achieved [19] and are typically denoted by κ. A goal can be either an achievement goal $!p$ (where p is an atom)[4], representing that the agent wants to achieve a situation in which p holds, or a test goal $?p$, representing that the agent wants to know whether p holds. Test goals are to be fulfilled by so-called information providing services, and achievement goals may be fulfilled by world altering services [13]. PS is a set of *plan selection rules*, formally denoted as $\kappa \mid \beta \Rightarrow \pi$, where β is a propositional formula representing a condition on the beliefs that should hold for the rule to be applicable, and π is a plan. The function $\mathcal{T} : (\mathsf{BasicAction} \times \Sigma) \rightarrow \Sigma$ is a partial *belief update function* (where Σ is a set of belief bases) which specifies the belief update resulting from the execution of (internal) actions by the agent. This function is introduced as usual [19] for technical convenience.

The formal definition of the syntax of plans is given below, where x is a variable name.

$$act_\phi ::= x \mid \phi \qquad b ::= a \mid \kappa \mid sn^r(act_\phi, act_\kappa)$$
$$act_\kappa ::= x \mid \kappa \qquad \pi ::= b \mid b >x> \pi$$

Internal actions are typically denoted by a, and κ represents a subgoal. A service call has the form $sn^r(act_\phi, act_\kappa)$, where sn is the name of the service that is to be called (which is d if a service is to be discovered), act_κ represents the goal that is to be achieved through calling the service, and act_ϕ is (or should be instantiated with) a propositional formula representing additional information that forms input to the service. The revision parameter r can be np (non-persistent), meaning that the result of the service call is not stored in the belief base, or p (persistent), meaning that the result is stored in the belief base. The result returned from a basic plan element b is bound to the variable x, which may be used in the remaining plan π. A plan of the form $b \gg \pi$ is used to abbreviate a plan $b >x> \pi$ where x does not occur in π.

The mechanism of applying plan selection rules to goals in the goal base or subgoals in plans is formalized using the notion of a stack. Each element of the stack represents, broadly speaking, the application of a plan selection rule to a particular (sub)goal. The initial stack element is created by applying a plan selection rule to a top-level goal in the goal base, and other stack elements are created every time a subgoal is encountered in the plan of the top element of a stack. A stack element has the form $(\pi, \kappa, \mathsf{PS})$, where κ is the (sub)goal to which

[4] In [23], we used arbitrary propositional formulas for the representation of goals, but for reasons of simplicity we use atoms here.

the plan selection rule has been applied, π is the plan currently being executed in order to achieve κ, and PS is the set of plan selection rules that have not yet been tried in order to achieve κ.

A stack element is popped just after a service call or an action execution if the goal of the stack element is reached, or it is popped if the goal is unreachable, meaning that there are no applicable plan selection rules. In the former case the result of the service call or the part of the belief base that expresses that the subgoal κ has been reached is returned and all occurrences of x in π are substituted with this result. The latter case is explained in Section 3.2.

A configuration of a goal-oriented program has the form $\langle \sigma, \gamma, \mathrm{St}, \mathsf{PS}, \mathcal{T} \rangle$, where St is the stack. The initial configuration of an agent $\langle \sigma_0, \gamma_0, \mathsf{PS}, \mathcal{T} \rangle$ is $\langle \sigma_0, \gamma_0, E, \mathsf{PS}, \mathcal{T} \rangle$, where E denotes an empty stack. In the transition rules, we leave out PS and $\mathcal{T}$ from configurations for reasons of presentation (and these do not change during computation).

3.2 Formal Semantics of Failure Handling

The formal semantics of our goal-oriented orchestration language is defined using a transition system [15]. A transition system for a programming language consists of a set of axioms and transition rules for deriving transitions for this language. A transition is a transformation of one configuration into another and it corresponds to a single computation step. The transition rules specify how to execute the top element of a stack.

In the goal-oriented orchestration language, a *failure* is not only caused by abnormalities in trying to execute some operation, but also by *being unsuccessful in reaching a goal*. In particular, if a service is called and returns some result, the call is only considered to be successful if the goal of the service call is reached through the result that is returned. That is, even if the service returns a "normal" or non-exceptional result, the service call can still be regarded as having failed. Such situations are not unlikely to occur, especially if services are automatically discovered at run-time. It might, e.g., be the case that the service description was not accurate, resulting in an unsatisfactory result. These kinds of failures are typically not considered nor dealt with in more classical programming paradigms, in which a failure or exception is normally caused by the fact that some operation could not be executed properly.

Our goal-oriented orchestration language *handles failures of service calls* by repeatedly trying to find matching services for a service call (in particular if services are to be discovered) until the goal of the service call is reached, or there are no more matching services.[5] If the latter happens, the service call has failed definitively, in which case the plan containing the service call is considered to have failed and the plan is dropped.

The latter case is specified formally in Definition 1 below. The service call construct $sn^r(\phi, \kappa')$ (we assume variables are instantiated when the service is

[5] One might argue that a comprehensive failure handling mechanism should include compensation, but this is without the scope of this paper.

called) is annotated with a set of service descriptions S which represents services that have not yet been called, and the result x_0 of the last service call. In this setting, services are assumed to return a propositional formula that expresses the effect or piece of information resulting from calling a world altering or information providing service, respectively. The predicate $\text{ach}(\kappa, \sigma, x_o)$ holds iff the goal κ is achieved with respect to belief base σ and the service call result x_0. In case κ is an achievement goal, it is achieved if the goal follows from the belief base after it is updated with x_0. In case κ is test goal, it is achieved if the goal or its negation follow from x_0. The idea is that the belief base should not be taken into account when evaluating the achievement of a test goal, as the idea is that a service is called in order to check whether some piece of information is accurate. Then it does not matter whether the agent already believes something about this information. The predicate $\text{match}(sn(\phi, \kappa), \sigma, sd)$ holds iff the service with service description sd matches with the service call $sn(\phi, \kappa)$, given the belief base σ.

Definition 1 *(plan failure)*

$$\frac{\neg\text{ach}(\kappa', \sigma, x_o) \quad \neg\exists sd \in S : \text{match}(sn(\phi, \kappa'), \sigma, sd)}{\langle \sigma, \gamma, (sn^r(\phi, \kappa')[S, x_o] >\!x\!> \pi, \kappa, \mathsf{PS})\rangle \rightarrow \langle \sigma, \gamma, (\epsilon, \kappa, \mathsf{PS})\rangle}$$

As plans are dropped if something goes wrong (if an internal action cannot be executed, the plan is dropped as well), the occurrence of an empty plan in a stack element indicates a failure. It can also be the case that a plan is *completely executed* resulting in an empty plan, without occurrence of a problem with an action execution or service call. However, this also indicates that the plan has failed to reach the goal of the stack element, as the stack element would have been popped immediately if its goal would have been reached after an action execution or service call.

While the handling of failures of service calls is done by trying to call other matching services, the *handling of plan failures* is done by using plan selection rules to select *alternative* plans for reaching a (sub)goal. This is formally specified by the transition rule below. Note that a plan selection rule that is applied is *removed* from the set of available plan selection rules PS. Moreover, note that the fact that we store the subgoal that the agent is trying to reach in the stack elements facilitates the selection of alternative plans to reach this goal. If we would not have such a representation, it would be more difficult to determine what to do if something went wrong.

Definition 2 *(apply rule after plan failure)* Below, $\mathsf{PS}' = \mathsf{PS} \setminus \{\kappa' \mid \beta \Rightarrow \pi\}$.

$$\frac{\kappa \mid \beta \Rightarrow \pi \in \mathsf{PS} \quad \neg\text{ach}(\kappa, \sigma, \top) \quad \sigma \models \beta}{\langle \sigma, \gamma, (\epsilon, \kappa, \mathsf{PS})\rangle \rightarrow \langle \sigma, \gamma, (\pi, \kappa, \mathsf{PS}')\rangle}$$

If the plan of the top stack element is empty and there are *no* plan selection rules applicable to the subgoal κ of this stack element, the subgoal is considered to have failed definitively. Then, the top element of the stack is popped, and the plan $\kappa >\!x\!> \pi$ that contains κ is dropped from the new top element. Consecutively,

the agent can try another plan for reaching the subgoal κ', or, if there are no applicable plan selection rules, the stack element with subgoal κ' is popped as well, etc.

Definition 3 *(subgoal failure)*

$$\frac{\neg\exists(\kappa \mid \beta \Rightarrow \pi) \in \mathsf{PS} : \sigma \models \beta}{\langle \sigma, \gamma, (\epsilon, \kappa, \mathsf{PS}).(\kappa >\!x\!> \pi, \kappa', \mathsf{PS}')\rangle \rightarrow \langle \sigma, \gamma, (\epsilon, \kappa', \mathsf{PS}')\rangle}$$

4 Procedural Orchestration Language

The main ingredients of our procedural orchestration language are standard features of procedural languages, i.e., assignment, test, procedure call, and an exception handling mechanism. The particular instantiations of these features are tailored towards the translation of the goal-oriented orchestration language in the procedural orchestration language. Further, the language includes a construct for service calls, similar to the corresponding one in the goal-oriented orchestration language. The syntax of statements is formally defined below, where e is an exception name, x is a variable name, and act_ϕ, act_κ as in Section 3.1.

$$\begin{array}{lll}
\kappa & ::= & ?p \mid !p \\
v & ::= & \mathit{true} \mid \mathit{false} \mid \phi \mid \kappa \\
t & ::= & \phi? \mid (x = v)? \mid \mathit{ach}(act_\kappa, x)? \mid \mathit{not}\ t \mid t \wedge t' \\
\mathit{act} & ::= & x \mid v \\
\mathit{exp} & ::= & v \mid \kappa(act_1, \ldots, act_n) \mid sn^r(act_\phi, act_\kappa) \mid \mathit{base}(act_\kappa) \\
\mathit{ass} & ::= & x := \mathit{exp} \\
b & ::= & a \mid \mathit{ass} \mid t \mid \mathsf{return}\ \mathit{act} \mid \mathsf{throw}\ e \\
\pi & ::= & b \mid b; \pi \mid \pi + \pi' \mid \texttt{while}\ t\ \texttt{do}\ \pi\ \texttt{od}
\end{array}$$

The language of procedure names is the same as the language of goals of the goal-oriented orchestration language (κ). The (global) state of configurations in this language contains a belief base as also used in the goal-oriented orchestration language. Additionally, procedures may use local variables, typically denoted by x. These local variables may have a value v, which is *true*, *false*, a string denoting a formula ϕ, or a string denoting a procedure name κ. Tests can be global tests on the belief base $\phi?$ (note the difference with test goals $?p$, which can only be fulfilled through service calls), local tests $(x = v)?$ that can be used for testing the value of a variable, or $ach(act_\kappa, x)$, which tests whether the goal act_κ is achieved with respect to the value of the variable x. Expressions are values, procedure calls $\kappa(act_1, \ldots, act_n)$, service calls, or a call to a predefined function $base(act_\kappa)$, which returns a conjunction of formulas from the belief base from which act_κ follows, or *false* if κ does not follow. Intuitively, this represents how κ is achieved. Elementary statements can be actions to change the belief base (as in the goal-oriented orchestration language), assignments to change the value of local variables, tests, returning a variable, and throwing an exception. Composed statements are formed by sequential composition, non-deterministic choice, or a `while` construct.

The exception handling mechanism that we use is inspired by the exception handling mechanism in the service orchestration language WS-BPEL [10]. In WS-BPEL, exception handlers are associated with a scope of a business process. If a fault occurs in a scope and the scope contains a matching handler, the process specified by the handler is executed.[6] If there is no handler, the exception is passed to the enclosing scope. In the context of our procedural language, the scope is formed by procedures, i.e., each procedure call gives rise to a new scope. Therefore, we associate exception handlers to procedures, as defined below. A handler contains the name of the exception that it handles, and a statement that should be executed if the relevant exception is thrown.

Definition 4 *(procedures and exception handlers)* A procedure has the form $\kappa(x_1, \ldots, x_n) \Rightarrow \pi$. Exception handlers, typically denoted by h, have the form $e.\mathsf{Handler} \Rightarrow \pi$, where e is an exception name. A procedure definition is a procedure accompanied with a possibly empty set of exception handlers, denoted by $[\kappa \Rightarrow \pi, H]$, where H is a set of exception handlers.

The semantics is defined by means of a transition system. We use stacks to define the mechanism of calling procedures, analogously to the way this was done for applying plan selection rules. Each stack element (π, θ, H) corresponds to a procedure call, where π is the statement that still needs to be executed, θ is a substitution specifying which values have been assigned to which local variables, and H is the set of exception handlers of the procedure that was called and for which the stack element was created. The set of handlers of a stack element does not change during computation.

A configuration $\langle \sigma, \gamma, \mathrm{St}, \mathsf{P}, \mathcal{T} \rangle$ consists of a belief base σ and goal base γ (together forming the global state), a stack St, a set of procedure definitions P, and a belief update function $\mathcal{T}$. The goal base is simply a set of data elements, i.e., it is a normal data structure that does not have the semantics of its counterpart in the goal-oriented orchestration language. For reasons of space, we do not explain nor define aspects having to do with updating of the goal base in this paper. A program $\langle \sigma_0, \gamma_0, \pi_0, \mathsf{P}, \mathcal{T} \rangle$ has the initial configuration $\langle \sigma_0, \gamma_0, (\pi_0, \emptyset, \emptyset), \mathsf{P}, \mathcal{T} \rangle$. Analogously to the goal-oriented orchestration language, we omit the procedure definitions and the belief update function from configurations in the transition rules below.

We only show the transition rules for exception handling, for reasons of space. The semantics of the other constructs is as one would expect, and for formal details we refer to [22]. The semantics of procedure calls is a simple call-by-value semantics. The first transition rule below expresses that if an exception e is thrown from within a stack element, and the stack element contains a handler $e.\mathsf{Handler} \Rightarrow \pi'$ for this exception, then the statement π' is executed instead of the statement from which the exception was thrown. If the stack element does not contain a handler for e, the exception is passed to the stack element one level lower in the stack.

[6] Additionally, WS-BPEL has a compensation mechanism (see also [11]), which is, however, outside the scope of this paper.

Definition 5 *(throwing exceptions)*

$$\frac{e.\mathsf{Handler} \Rightarrow \pi' \in H}{\langle \sigma, \gamma, (\mathsf{throw}\ e; \pi, \theta, H) \rangle \rightsquigarrow \langle \sigma, \gamma, (\pi', \theta, H) \rangle}$$

$$\frac{\neg \exists h' \in H' : h' \text{ is of the form } e.\mathsf{Handler} \Rightarrow \pi''}{\langle \sigma, \gamma, (\mathsf{throw}\ e; \pi', \theta', H').(\pi, \theta, H) \rangle \rightsquigarrow \langle \sigma, \gamma, (\mathsf{throw}\ e, \theta, H) \rangle}$$

5 Translation and Correctness Result

In this section, we show how the goal-oriented orchestration language can be translated to a procedural orchestration. This translation shows, first of all, *how* goal-oriented orchestration, and in particular its failure handling mechanism, is related to a more standard procedural orchestration language and its exception handling mechanism. Moreover, it shows that the programming patterns resulting from the translation do not increase understandability of the code. As stated in [7] in a more general context, the problem with programming patterns is that "they are an obstacle to an understanding of programs for both human readers and programming-processing programs".[7] We thus argue that the kind of abstractions as used in the goal-oriented orchestration language are worth considering as language constructs of an orchestration language. As our procedural orchestration language and WS-BPEL are comparable in the sense that they have a similar exception handling mechanism, and both are imperative languages without goal-oriented constructs, we conjecture that an implementation of goal-oriented orchestration patterns in WS-BPEL will be similarly involved as in our procedural orchestration language.

In this paper we present the most important parts of the translation, i.e., the translation of plan selection rules and the translation of plans. For the full technical details of the translation, we refer to [22].

Definition 6 *(translating plan selection rules)* Without loss of generality, assume that variables in the goal-oriented orchestration language are not the reserved variables $tried_i$. Let PS be a set of plan selection rules. Let PS_κ be defined as $\{\kappa \mid \beta \Rightarrow \pi \ : \ \kappa \mid \beta \Rightarrow \pi \in \mathsf{PS}\}$ and let $n = |\mathsf{PS}_\kappa|$. We assume an ordering on the elements of PS_κ as follows: $\{\kappa \mid \beta_1 \Rightarrow \pi_1, \ldots, \kappa \mid \beta_n \Rightarrow \pi_n\}$. The translation function t for translating PS_κ into one procedure definition is defined as follows.

$[\kappa(tried_1, \ldots, tried_n, from) \Rightarrow$
$\quad this := \kappa;$
$\quad (+_{1 \le i \le n}((tried_i = false)? \wedge \beta_i?; tried_i := true; u_\kappa(\pi_i); [\alpha_{fail}]\ \mathsf{throw}\ \kappa.\mathsf{planFailedExc}) +$
$\quad (not \bigwedge_{1 \le i \le n}((tried_i = false)? \wedge \beta_i?); [\alpha_{fail}]\ \mathsf{throw}\ from.\mathsf{planFailedExc})),$
$\{\kappa.\mathsf{planFailedExc.Handler} \Rightarrow x_f := \kappa(tried_1, \ldots, tried_n, from); \mathsf{return}\ x_f\}]$

[7] The term "programming patterns" should not be confused with "design patterns". While the former are computational in nature, the latter are concerned with software architecture.

The example in Section 2 already hints at how a translation of a goal-oriented orchestration into a procedural one might be defined. That is, all plan selection rules for a certain goal are translated into one procedure that has this goal as the procedure name. The body of the procedure resulting from the translation of a set of plan selection rules, broadly speaking, consists of a non-deterministic choice between the translated plans of the relevant plan selection rules, guarded by tests on the belief base corresponding with the guards of the plan selection rules.[8] The translation of plans is specified through the function u_κ (Definition 7).

Each situation of failure of the goal-oriented orchestration language as analyzed in detail in Section 3.2, corresponds to the throwing of an exception in the procedural language. That is, we throw a planFailedExc if a plan has been executed completely, as this means that the goal to be achieved by this plan was not reached. Further, a planFailedExc is thrown if all plans have been tried and/or none are applicable (as the belief condition does not hold), corresponding to subgoal failure (Definition 3). The throwing of an exception in case a service call fails is specified in Definition 7.

We annotate each planFailedExc with the name of the procedure in which the exception should be handled. The exception should be handled either in the procedure κ from which it was thrown (in case another plan should be selected for achieving the goal of the procedure), or in the procedure from which κ was first called (as passed to κ through the variable *from*). The latter case represents the failure of a subgoal, and it corresponds to the popping of a stack element in the goal-oriented orchestration language (Definition 3).

We associate with each procedure κ a handler for the exception κ.planFailedExc. This handler specifies that the procedure should be called recursively with the variables $tried_i$ (representing which plans have already been tried) as parameters. This recursive call makes sure that if a plan fails, another plan is tried which has not been tried yet (Definition 2).

Note that the programmer thus needs to program the throwing of exceptions and their handlers explicitly in the procedural orchestration language, while the identification of situations of failure and the consecutive course of action is part of the semantics of the goal-oriented orchestration language. The next definition specifies the function u_κ, which translates plans of the bodies of plan selection rules with head κ into statements of the procedural language. The function is also used to translate the plan of a stack element with subgoal κ.

Definition 7 *(translating plans to statements)* We define a function $u_\kappa(\pi)$ where κ is the head of the plan selection rule of which the body π is translated, or the goal of the stack element containing π. Let $\mathsf{PS}_{\kappa'} = \{\kappa' \mid \beta' \Rightarrow \pi' : \kappa' \mid \beta' \Rightarrow \pi' \in \mathsf{PS}\}$, let $n' = \mid \mathsf{PS}_{\kappa'} \mid$, let $false_{1,\ldots,n'}$ be a vector of length n' of parameters being the value *false*, let $S_\mathcal{O}$ be the set of available service descriptions, and let sd_{sn} be the service description of the service called for service call $sn^r(act_\phi, act_{\kappa'})$.

$$
\begin{array}{ll}
u_\kappa(\kappa' >\!x\!> \pi) & = ((ach(\kappa')?; x := base(\kappa')) + \\
 & \quad (not\ ach(\kappa')?; x := \kappa'(false_{1,\ldots,n'}, \kappa))); u_\kappa(\pi) \\
u_\kappa(a \gg \pi) & = a; ((ach(\kappa)?; x := base(\kappa); \mathsf{return}\ x) + (not\ ach(\kappa)?; u_\kappa(\pi))) \\
u_\kappa(sn^r(act_\phi, act_{\kappa'}) >\!x\!> \pi) & = x := base(act_{\kappa'}); \\
 & \quad ((ach(act_{\kappa'}, x); u_\kappa(\pi)) + (not\ ach(act_{\kappa'}, x)?; S := S_\mathcal{O}; \\
 & \quad \mathtt{while}\ not\ ach(act_{\kappa'}, x)\ \mathtt{do}\ x := sn^r(act_\phi, act_{\kappa'}); \\
 & \quad ((x = nomatch)?; \mathsf{throw}\ \kappa.\mathsf{planFailedExc}) + \\
 & \quad (not(x = nomatch)?; S := S \setminus \{sd_{sn}\})\ \mathtt{od}); \\
 & \quad ((ach(\kappa, x)?; \mathsf{return}\ x) + (not\ ach(\kappa, x)?; u_\kappa(\pi)))
\end{array}
$$

A subgoal $\kappa' >\!x\!> \pi$ is translated into a non-deterministic choice, followed by the translation of π. The non-deterministic choice expresses that if the goal κ' is already reached before calling the procedure κ', x gets a value through the function $base(\kappa')$. If κ' is not yet achieved, the procedure κ' is called, which returns a value (a propositional formula) that expresses how κ' was achieved or an exception in case κ' could not be achieved. The actual parameters for the procedure κ' are a series of *false* values, expressing that no plans have yet been tried to reach κ', and the last parameter is the subgoal κ, which is the goal to be reached through execution of the statement $u_\kappa(\kappa' >\!x\!> \pi)$ (as we are translating plan selection rules with head κ). The translation of an action a expresses that a should be executed, and, depending on whether the goal κ is reached, the orchestration returns or continues with the execution of $u_\kappa(\pi)$. The translation of a service call $sn^r(act_\phi, act_{\kappa'})$ defines that matching services are called until $act_{\kappa'}$ is reached, or there are no more matching services. If the latter is the case, a planFailedExc is thrown (corresponding to Definition 1).

Using the translation functions as defined above, we have defined a function v (see [22] for its definition) for translating agents of the goal-oriented orchestration language into procedural programs in the procedural orchestration language. This function v uses the function t of Definition 6 to translate plan selection rules to procedures. Moreover, an initialization procedure is added, which is called from the initial statement of the resulting procedural program. The purpose of the initialization procedure is to initiate the pursuit of goals of the goal base. Furthermore, the procedure is defined such that the program terminates if the goal base is empty.

We show, broadly speaking, that an agent in the goal-oriented orchestration language has the same behavior as its translation in the procedural orchestration language. We do this by showing that each run of an agent $\mathcal{A}$ has a matching run of agent $v(\mathcal{A})$ and vice versa. A run of $\mathcal{A}$ matches a run of $v(\mathcal{A})$, loosely speaking, if each configuration of the former has a matching configuration in the latter (in the right order). Each transition in a run of $\mathcal{A}$ is matched by a series of transition in a run of $v(\mathcal{A})$, i.e., not each configuration of a run of $v(\mathcal{A})$ has a matching configuration in the corresponding run of $\mathcal{A}$.

The definition of when a procedural configuration matches a goal-oriented configuration is provided by a function z (see [22] for its definition), which trans-

[8] In the example we used if-then-else constructs rather than non-deterministic choice, but in order to make the translation correct, we need non-deterministic choice to match the non-determinism of the goal-oriented orchestration language in selecting plan selection rules.

lates a configuration of the procedural orchestration language into a configuration of the goal-oriented language. The function cannot be defined the other way around, as procedural configurations contain certain implementation details that do not have a counterpart in goal-oriented configurations. The function z translates in particular procedural stacks into goal-oriented stacks by translating statements of stack elements to plans (using the inverse of the function u_κ). The function uses the substitution of stack elements to determine the goal of the resulting goal-oriented stack element and to determine which plan selection rules have not yet been tried to reach the goal.

The correctness of the translation is formulated formally below. We refer to [22] for the proof.

Theorem 1 Let $\mathcal{A}$ be a program in the goal-oriented orchestration language with initial configuration c_0 and $v(\mathcal{A})$ the translation of $\mathcal{A}$. Then it holds for any run $c_0 \rightarrow c_1 \rightarrow \ldots$ that there exist indices $0 = p_0 < p_1 < \ldots$ and configurations $d_0, d_1, \ldots$ such that $d_0 \rightsquigarrow d_1 \rightsquigarrow \ldots$ is a run in the procedural orchestration language, d_0 is the initial configuration of $v(\mathcal{A})$, and for all p_i with $i \geq 0$ it holds that $z(d_{p_i}) = c_i$.

Let P be a program in the procedural orchestration language with initial configuration d_0 such that there is some program $\mathcal{A}$ of the goal-oriented orchestration language with $v(\mathcal{A}) = P$. Then it holds for any run $d_0 \rightsquigarrow d_1 \rightsquigarrow \ldots$ that there exist indices $0 = p_0 < p_1 < \ldots$ and configurations $c_0, c_1, \ldots$, such that $c_0 \rightarrow c_1 \rightarrow \ldots$ is a run in the goal-oriented orchestration language, c_0 is the initial configuration of $\mathcal{A}$, and for all p_i with $i \geq 0$, it holds that $z(d_{p_i}) = c_i$.

6 Conclusion

In this paper, we have shown how the goal-oriented orchestration language of [23] can be correctly translated to a procedural orchestration language. As we have argued that the failure handling mechanism of the goal-oriented orchestration language is one of its main advantages, it is important to investigate whether a similar mechanism cannot be implemented just as easily in a more traditional language. As we have shown, however, the translation is non-trivial and the programming patterns resulting from the translation do not increase understandability of the code. We thus argue that the kind of abstractions as used in the goal-oriented orchestration language are worth considering as language constructs of an orchestration language.

We are currently working on the extension of the goal-oriented orchestration language towards more practically usable versions, e.g., by making use of description logic instead of propositional logic. This will allow us to experiment with the language in order to further investigate the usefulness of such a language in the domain of service orchestration. The usefulness of goal-oriented abstractions will not only have to be investigated on the level of orchestration languages, but also on the modeling level. One possible direction for future research is to investigate whether the KAOS goal-oriented requirements engineering methodology [18] can be adapted to fit the goal-oriented orchestration language.

References

1. M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Interaction protocols and capabilities: A preliminary report. In *Principles and Practice of Semantic Web Reasoning, 4th International Workshop (PPSWR'06)*, pages 63–77, 2006.
2. L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta. CooWS: Adaptive BDI agents meet service-oriented programming. In *Proceedings of the IADIS International Conference WWW/Internet 2005*, volume 2, pages 205–209. IADIS Press, 2005.
3. L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal representation for BDI agent systems. In *Programming multiagent systems, second international workshop (ProMAS'04)*, volume 3346 of *LNAI*, pages 44–65. Springer, Berlin, 2005.
4. W. R. Cook and J. Misra. Computation orchestration: A basis for wide-area computing, 2007. To appear in the Journal on Software and System Modeling.
5. M. Dastani, M. B. van Riemsdijk, and J.-J. Ch. Meyer. Programming multi-agent systems in 3APL. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, Berlin, 2005.
6. I. Dickinson and M. Wooldridge. Agents are not (just) web services: considering BDI agents and web services. In *Proceedings of the 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005)*, Utrecht, The Netherlands, 2005.
7. M. Felleisen. On the expressive power of programming languages. In N. Jones, editor, *ESOP '90 3rd European Symposium on Programming, Copenhagen, Denmark*, volume 432, pages 134–151. Springer-Verlag, New York, N.Y., 1990.
8. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming with declarative goals. In *Intelligent Agents VI - Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL'2000)*, Lecture Notes in AI. Springer, Berlin, 2001.
9. J. F. Hübner, R. H. Bordini, and M. Wooldridge. Declarative goal patterns for AgentSpeak. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*, 2006.
10. M. Juric, P. Sarang, and B. Mathew. *Business Process Execution Language for Web Services 2nd Edition*. Packt Publishing, 2006.
11. R. Lucchi and M. Mazzara. A pi-calculus based semantics for WS-BPEL, 2006. To appear in Journal of Logic and Algebraic Programming (JLAP), Elsevier press.
12. V. Mascardi and G. Casella. Intelligent agents that reason about web services: a logic programming approach. In *Proceedings of the ICLP'06 Workshop Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS2006)*, pages 55–70, 2006.
13. S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.
14. M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *Proceedings of the fifth international conference on automated planning and scheduling (ICAPS'05)*, pages 2–11, 2005.
15. G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.
16. S. Sardina, L. P. de Silva, and L. Padgham. Hierarchical planning in BDI agent programming languages: A formal approach. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS'06)*, pages 1001–1008, Hakodate, Japan, 2006. ACM Press.

17. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 2003.
18. A. van Lamsweerde and E. Letier. From object orientation to goal orientation: a paradigm shift for requirements engineering. In *Radical Innovations of Software and Systems Engineering in the Future: 9th International Workshop (RISSEF'02)*, volume 2941 of *LNCS*, pages 325–340, London, UK, 2004. Springer-Verlag.
19. M. B. van Riemsdijk. *Cognitive Agent Programming: A Semantic Approach*. PhD thesis, 2006.
20. M. B. van Riemsdijk, M. Dastani, J.-J. Ch. Meyer, and F. S. de Boer. Goal-oriented modularity in agent programming. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems (AAMAS'06)*, pages 1271–1278, Hakodate, 2006.
21. M. B. van Riemsdijk, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in Dribble: from beliefs to goals using plans. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems (AAMAS'03)*, pages 393–400, Melbourne, 2003.
22. M. B. van Riemsdijk and M. Wirsing. Goal-oriented and procedural service orchestration: A formal comparison, 2007. `http://www.pst.ifi.lmu.de/~riemsdijk/goalproc.pdf`.
23. M. B. van Riemsdijk and M. Wirsing. Using goals for flexible service orchestration: A first step. In J. Huang, R. Kowalczyk, Z. Maamar, D. Martin, I. Mueller, S. Stoutenburg, and K. Sycara, editors, *Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE'07)*, volume 4504 of *LNCS*, pages 31–48, 2007.
24. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proceedings of the eighth international conference on principles of knowledge respresentation and reasoning (KR2002)*, Toulouse, 2002.
25. M. Wirsing, A. Clark, S. Gilmore, M. Hölzl, A. Knapp, N. Koch, and A. Schroeder. Semantic-based development of service-oriented systems. In *Formal Techniques for Networked and Distributed Systems (FORTE'06)*, volume 4229 of *LNCS*, pages 24–45. Springer-Verlag, 2006.

Argonaut: Integrating Jason and Jena for context aware computing based on OWL ontologies

Douglas Michaelsen da Silva[1], Renata Vieira[1]

[1] Universidade do Vale do Rio dos Sinos
Av. Unisinos, 950 - CEP 93.022-000 São Leopoldo - RS - Brasil
michaelsen@gmail.com, renatav@unisinos.br

Abstract. In this paper, we present the integration of the agent-oriented programming framework Jason and the semantic web framework Jena to support ontology-based context aware computing. These technologies together allow for the development of context aware multi-agent systems base on ontologies that describe context.

Keywords: Semantic Web, Ontologies, Agents, Context aware computing.

1 Introduction

The Semantic Web project and the related development of applications that make use of knowledge resources are attracting much of current research interest. The Semantic Web proposed technologies are also proving adequate as a basis for other important areas of Computer Science such as Ubiquitous Computing [3]. These new computing technologies are changing the way users interact with applications. These changes are due to the fact that users, devices and applications are given mobility. In this scenario, context awareness is a relevant requirement for applications. The computational representation of context is thus a growing field of research and technology development. Semantic Web technologies currently available, such as description logic based ontologies and intelligent agents are promising solutions for context representation and manipulation. These technologies allow pro-active contextual help and guidance for mobile users and applications.

Although applications for the Semantic Web are already being proposed, often based on agents paradigm, most of such efforts does not consider proper agent-oriented programming languages. Besides, web ontology languages and agent oriented programming languages have both been developed independently from each other. On the other hand, the integration of such agent oriented programming languages, such as AgentSpeak [1], with automatic reasoning over ontologies can have a major impact on the development of agents and multi-agent systems that can operate in a SemanticWeb context. In fact, the theoretical aspects of such integration have been already proposed [5]. However, the practical integration of such technologies for developing real applications is still a challenge.

In this work, as a first approach to explore the integration of these technologies in a practical way, for the development of typical mobile computing applications, we show an AgentSpeak/Jason[1] prototype in which BDI agents access an OWL ontology through the Jena framework[2]. The prototype implements agents that help users to find out about locally situated services.

The paper is organized as follows: Section 2 presents an ontology that describes contextual information; Section 3 presents an overview of the agent programming language AgentSpeak; Section 4 presents the integration of Jason and Jena for a context aware application prototype; Section 5 concludes the paper.

2 Argonaut Ontology

Among the key components of the Semantic Web are *domain ontologies* [7]. They are the proposed model for knowledge resources, underlying specific web languages. Ontologies are therefore the component responsible for the specification of the domain knowledge. As they can be expressed logically, they allow for reasoning in the specified domain. Indeed, several ontologies are being proposed for the development of a large variety of applications [2, 3].

OWL is a language developed for representing ontology information on the semantic web. OWL is based on descriptions logics which are appropriate for ontological reasoning. OWL can be used to describe concepts and their relationships as well as specific properties and restrictions through logical axioms. According to different underlying logics, there are three versions of OWL: OWL Full, OWL DL and OWL Lite.

OWL ontologies have been developed for ubiquitous and pervasive applications; the SOUPA ontology is an example [2]. It was designed to support mobile applications, its vocabulary is derived from other existing ontologies, some examples are: FOAF, an ontology for personal relationship information, people and their basic data such as address, phone number, e-mail, etc; DAML-Time, an ontology for common knowledge about time and temporal events; Spatial ontologies (such as OpenCYC and RCC) for spatial concepts and reasoning about localization.

Since ontologies are to be shared and reused, we developed a small ontology by adapting some of the main concepts and relations of SOUPA. We created instances corresponding to a university environment. The main concepts adapted were *Person*, *Geographic Space* and their subclasses. New concepts were created to accommodate the application we had in mind. The new concepts are service and distance. We used Protégé [4] and its OWL plugin to build the ontology that is the basis of our application. The classes *Person* and *Service* describe, respectively, the users of the application and the services provided in different regions of the campus. The relation "at" (for is situated at), shown in Figure 1, holds for persons and services with geographical spaces.

[1] http://jason.sourceforge.net/
[2] http://jena.sourceforge.net/

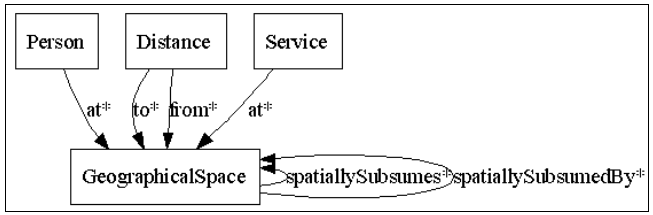


Fig. 1. Relationships among concepts.

The concept *GeographicalSpace* can be specialized as *GeographicalRegion*, *FixedStructure* (used for buildings) and *SpaceInAFixedStructure* (used for rooms). A geographical space may be spatially subsumed by another one. For example, a fixed structure can be spatially subsumed by a geographical region and can spatially subsume a space in fixed structure. For example, buildings can be situated in a campus and they can contain rooms. The concept *Distance* relates to geographical spaces through the relations "from" and "to". All instances of this concept represent a distance between two spaces. In a specific situation we could have the user Maria, who is an instance of *Person*. She is located at LabOne which in turn is a *SpaceInAFixedStructure*. The instance BuildA spatially subsumes the instance LabOne and is spatially subsumed by instance CentreX. In this situation agents that perceive the presence of Maria in LabOne can infer that Maria is at BuildA in CentreX.

3 AgentSpeak

As ontologies, agents are also considered a fundamental component of the semantic web. Agents are responsible for helping the users in their service requests. They can make use of the available knowledge; autonomously interact with other agents, so as to act on the user's interest. Of course, on the view of the Semantic Web agents can only achieve these requirements by sharing domain ontologies.

Here we consider the agent oriented programming language AgentSpeak. Jason is the interpreter for AgentSpeak, which is available Open Source under GNU LGPL at http://jason.sourceforge.net. It implements the operational semantics of AgentSpeak given in [8]. AgentSpeak provides an elegant abstract framework for programming agents. An AgentSpeak agent (or program) is defined by a set of beliefs, which is a set of ground (first-order) atomic formulas, and a set of plans which form its plan library. AgentSpeak distinguishes two types of goals: achievement and test goals. Achievement goals are formed by an atomic formula prefixed with the '!' operator, while test goals are prefixed with the '?' operator. An achievement goal states that the agent wants to achieve a state of the world (and it will look for a stated plan in his plan library for that). A test goal states that the agent wants to test whether the associated atomic formula is (or can be unified with) its beliefs. Being a reactive

planning system, the events it reacts to are related either to changes in its beliefs due to perception of the environment, or to changes in the agent's goals that originate from the execution of plans triggered by previous events. A triggering event can trigger the execution of a particular plan. Plans are written by the programmer so that they are triggered by the addition ('+') or deletion ('-') of beliefs or goals (the "mental attitudes" of AgentSpeak agents). These elements are exemplified in Figure 2.

Consider a scenario where a student or academic visitor is walking around the university campus. The student may be notified about locally available services, or scheduled invited talks, according to the user's preferences. In the example (Figure 2) we show some AgentSpeak plans for this scenario.

```
+lecture(A,V,T) : interested_in(U,A)
  !inform(U,A,V,T)

+!inform(U,A,V,T) : ¬busy(U,T)
  show(U,A,V,T)…
```

Fig. 2. Examples of AgentSpeak plans.

The first plan tells us that, when a lecture A is announced at venue V and time T (so that, from the perception of the context, a belief lecture(A,V,T) is *added* to the belief base of the agent), then if a user U is interested in A, it will have the new goal of inform interested users for that lecture. The second plan tells us that whenever this agent adopts the goal of informing users about lectures, if it is the case that the user is not busy at T, according to his agenda, then it can proceed to execute that plan consisting of performing the basic action show(U,A,V,T) (assuming that it is an atomic action that the agent can perform). This brief introduction will help to guide the reading of the next section, in which we present our prototype. More details about AgentSpeak can be found in [1].

4. The Argonaut prototype

Argonaut is a multi-agent system that integrates Jason and Jena in order to allow agents to interact on the basis of contextual knowledge represented in an OWL ontology. Jason provides the means for the specification of the environment in which the agents actuate. In the specified environment, agents perceive the user and user requests, they communicate with each other to provide the user information relative to their distance to required services. Through some defined internal actions the agents consult an OWL ontology that contains contextual information. The interaction with the OWL ontology is done through the Jena framework.

Our scenario is such that the user is located somewhere in the university campus and he intends to know about the available services nearby. He also intends to know the distance from his location to some required service (library, food court, computers lab, shops, etc). The mobile device is perceived by a server and the local interface agent communicates with him offering the locally available services. When the local interface agent perceives the arrival of the user, it greets him and then shows him the

locally available services. The user selects one service, the interface agent then asks about the location of the selected service to the location agent. The location agent gets the service and the user location and queries the distance between the user and the service, returning it to the interface agent. The interface agent shows to the user his distance from the required service.

In our prototype the presence of the user is simulated, the service selection is done through the user interface and they are external events that occur in the environment. The interface agent perceives the user selection and communicates with the location agent, which is responsible for consulting the context ontology. Events and actions are implemented in Jason. External events are the user arrival, which is perceived by the environment, and the selection of one available service, made by the user. Internal actions do not modify the environment, in the Argonaut they are responsible for consulting the OWL ontology through the Jena framework. Queries to the ontology return contextual information which is added in the agents' belief base.

The system overview [6] is illustrated in Figure 3. Events are represented by stars and actions by arrows. The two actions named with *jena* corresponds to internal actions of the agents for querying the contextual OWL ontology. The result for such queries is contextualized information that is stored in the agents' belief bases.

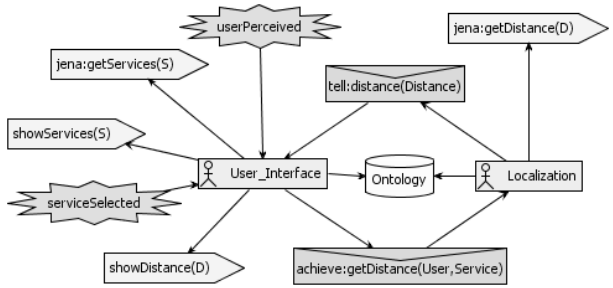


Fig. 3. Argonaut system overview.

The agents are named *localization* and *user_interface*, there is a local interface for each University Institute. Each Institutue, or region of the University, would delegate an agent to serve as its local service interface. In this way, each locally perceived user is tied to a local user interface agent. The prototype follows a MVC (Model View Controller) architectural pattern. The model-view-controller separates data access and business logic from data presentation and user interaction, by introducing an intermediate component: the controller. The environment where the agents actuate corresponds to the Controller module, which is also an observer to the other modules. It is necessary because each event that happens in the View or in the Model need to be handled by the Jason environment. Therefore, the Model module implements the consults to the ontology, which are made through RDQL/Jena, and the View module is responsible for the interface of the prototype, implementing some functionality to handle graphical configurations.

These two modules implement the observable class that notifies the environment when something happens. In this case, when there are relevant plans to deal with events they are activated. An example is when a service is selected by the user, the environment is notified, the agent perceives this change and the corresponding plan is activated, as explained below.

In Jason, the environment is responsible for providing events (perceptions) to the agents, in our prototype these events are the presence of the user and the selection for a service made by the user through the graphical interface. The available services are described in the ontology and they are presented to the user by the interface agent.

Next we describe the plans of the *user_interface* agent (Figure 4). A belief about the presence of the user is added to the belief base of the agent. This triggers the plan for the arrival of a new user, a plan to show the available services. The second plan is triggered when the user selects a service. First, an action to exhibit the selected service is executed. A test goal identifies who is the user perceived, then an achieve message is sent to the localization agent, so that an appropriate plan of the localization agent is triggered.

```
+userWasPerceived(User):
       <- ont.GetServices(Centre,S);
       showServices(S).

+selected(Service)  :  true
       <- .print(Service);
       ?userWasPerceived(User);
       .send(localization, achieve,
       getDistance(User, Service)).

+distance(D) : true <- showDistance(D).
```

Fig. 4. User_interface agent.

The localization agent (Figure 5) executes the plan with two internal actions: an action that consults the ontology and queries the distance between the user and the agent; and a *.send* action to return the resulting distance to the agent that sent the achieve message. This *tell* message will cause a belief addition in the belief base of the user_interface agent, triggering the plan for showing the retrieved distance to the user.

```
+!getDistance(User, Service)[source(SAg)] : true
       <-
       ont.GetDistance(User, Service, Distance);
       .send(SAg, tell, distance(Distance)).
```

Fig. 5. Localization agent.

In our prototype, the locally available services and distance between locations are described in the OWL ontology. Ontology queries are done with Jena, an open source

Java framework for building Semantic Web applications. It has an API that aims to provide a consistent programming interface to the semantic web application developer. It provides an environment for querying ontologies and includes a rule-based inference engine. Consults using RDQL[3] (RDF *Data Query Language*) are done using this model. RDQL consists in a graph of triples. Each triple contains variables which are instantiated with the corresponding required values.

The localization agent executes the action *GetDistance* that returns the distance between the user and the requested service. For example, Maria is at LabOne, which is subsumed by BuildA. The coffe service is located at LabTwo which is subsumed by BuildC. These identifications are necessary because, in our model, the distances are defined between instances of *FixedStructure* (buildings). The localization agent's returns the distance which is added to the belief base of the user_interface agent. Then the user_interface agent shows the distance to the user.

5 Conclusions

In this paper we have shown a practical prototype that integrates BDI agents with the Semantic Web framework Jena. This integration was proposed to deal with the dynamic nature of mobile computing applications. Agents in the environment in which the user is located communicate with the user personal agent to inform about locally situated services. Contact with the user can be triggered by matching the user profile and context description. Ontologies are well suited for providing such profile and context descriptions. They represent the required knowledge in a structured and organized way, allowing inference for integrating user's goals with the context features. Also, this knowledge can be both queried and modified by agents.

In [5] AgentSpeak with underlying ontological reasoning was first proposed and formalized. That extension was shown to have the following effects: (i) more expressive queries to the agent belief base; (ii) refined belief update, new beliefs can only be added if the resulting belief base is consistent with the concept description; (iii) more flexible plan search based on the subsumption relation between concepts; and (iv) knowledge sharing by using web ontology languages such as OWL. In that paper, it was shown how extending an agent programming language with the descriptive and reasoning power of description logics can have a significant impact on the way agent-oriented programming works in general and in particular for the development of Semantic Web applications using the agent-based paradigm. However, the practical development of that previous proposal would require changes in the current AgentSpeak framework (Jason).

The Jason/Jena integration proposed and illustrated here is, of course, a much simpler approach from what was proposed in that work. However, one advantage of our proposal is that it allows the use of OWL ontologies without any modification or redefinition in the currently available frameworks. The prototype has shown the main components of such an architecture, which can be exploited for more elaborated applications in future work. For example, we haven't fully explored inference and

[3] http://www.w3.org/Submission/RDQL/

reasoning that is provided by the Jena framework. For now we have only used RDQL to query an OWL knowledge basis. In fact, with this first prototype we have only accounted for the improvements referred to in (i) and (iv) above. We believe that points (ii) and (iii) could as well be pursued through a deeper integration of the technologies that we have adopted here.

Our prototype implements a fairly simple application, which serves mainly to the purpose of illustrate the potentiality that these technologies bring about when they are put together, being a first practical approach integrating OWL, Jason and Jena. For now, when executing the Argonaut, the presence of a user is simulated. Ideally, that is, in a real implementation, this perception would happen through sensors located at different locations. Also, in a more sophisticated implementation, the agents could be distributed over a network. The View module could be customizable based on information retrieved from the ontology, in a contextualized way, considering the user profile and device characteristics. For this, an ontology extension would be required. Other similar interesting applications could be explored for the presented prototype, such as, for instance, cars communicating with available services in the road.

References

1. Bordini, R., Hubner, J. and Wooldridge, M.: Programming AgentSpeak Agents with Jason. John Wiley & Sons. 288p (2007).
2. Chen, H., Chen, H., Perich, F., Finin, T., Joshi, A: SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications, In Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous 2004), Boston, MA, August (2004).
3. Chen, H., Chen, H., Perich, F., Chakraborty, D., Finin, T., Joshi, A.: Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 19(5):69–79, November/December 2004.
4. Horridge, M., H. Knublauch, A. Rector, R. Stevens, C. Wroe.: A Practical Guide To Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools, Technical Report, Ed. 1.0, The University Of Manchester (2004).
5. Moreira, A., Vieira, R., Bordini, R., Hubner, J.: Agent-Oriented Programming with Underlying Ontological Reasoning. In: Declarative Agent Languages and Technologies III: Third International Workshop, Utrecht, The Netherlands, Selected and Revised Papers. Vol. 3904, pp. 155--170. Springer, Berlin (2006)
6. Padgham, L. and Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. LNCS, vol. 2585, pp. 174--185. Springer (2003)
7. Staab, S. and Studer, R. (eds.): Handbook on Ontologies. International Handbooks on Information Systems. Springer-Verlag, Berlin–Heidelberg (2004)
8. Vieira, R., Moreira, A., Bordini, R. and Wooldridge, M.: On the Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language. Journal of Artificial Intelligence Research, Vol 29, p. 221-267 (2007)

Agent Societies and Service Choreographies: a Declarative Approach to Specification and Verification

Federico Chesani[1], Paola Mello[1], Marco Montali[1], and Sergio Storari[2]

[1] DEIS, University of Bologna - Viale Risorgimento 2 - 40136 Bologna, Italy
{fchesani|pmello|mmontali}@deis.unibo.it
[2] ENDIF, University of Ferrara - Via Saragat 1 - 44100 Ferrara, Italy
strsrg@unife.it

Abstract. The need for specifying choreographies when developing service oriented systems recently arose as an important issue. Although declarativeness has been identified as a key feature, several proposed approaches model choreographies by focusing on procedural aspects, e.g. by specifying control and message flows of the interacting services. A similar issue has been addressed in Multi-Agent Systems (MAS), where declarative approaches based on social semantics have been used to capture the nature of agents interaction without over-constraining their behavior.
In this paper we show how DecSerFlow can be mapped to $\mathcal{S}$CIFF in an automatic and complete way. DecSerFlow is a graphical language capable to model in an intuitive and declarative fashion service flows, whereas $\mathcal{S}$CIFF is a framework based on abductive logic programming originally developed for dealing with social interactions in MAS. By means of a running example, we show how the conjunct use of both approaches could be fruitfully exploited to declaratively specify and verify service choreographies.

1 Introduction

The service oriented paradigm and the related technologies for implementing and interconnecting basic services are reaching a good level of maturity and a widespread adoption. Nevertheless, modeling service interaction from a global viewpoint, i.e. representing service choreographies, is still an open challenge [1]. Indeed, the need for specifying choreographies when developing service oriented systems recently arose as an important issue.

As pointed out in [1, 2], the current major proposals for modeling service interaction, such as WS-BPEL [3] and WS-CDL [4], miss to tackle some key concepts. As a consequence of the adoption of a "global view" (which inherently crosses organizational boundaries and should be consequently independent from the perspective of single participants), declarativeness becomes a fundamental requirement. Each organization perceives a choreography as a public contract which provides the rules of engagement for making all the interacting parties

correctly collaborate, without stating how such a collaboration is concretely carried out; in our view, this latter information should be kept private in the entities' definition/implementation, and not directly addressed at the choreography level.

The main problem is that, although declarativeness has been identified as a key feature, several proposed approaches model choreographies by focusing on procedural aspects, e.g. by specifying the control and message flow of the interacting services. This often causes the modeler to miss the real focus of the choreography, leading to over-constrain the choreography under study and to consequently loose some acceptable interactions.

To overcome these limits, van der Aalst and Pesic have proposed DecSerFlow [5], a truly declarative graphical language for the specification of service flows. DecSerFlow adopts a more general and high-level view of services specification, by defining them through a set of policies or business rules. It does not give a complete and procedural specification of services, but concentrates on what is the (minimal) set of constraints to be fulfilled in order to successfully accomplish the interaction. Beyond its appealing graphical representation, DecSerFlow concepts have an underlying semantics in terms of Linear Temporal Logic (LTL).

The issue about what information should be captured or left out by the global view of interaction has been (and is still) matter of discussion also in the MAS research community, and in both settings we find similar efforts and proposed solutions. Therefore, it is not surprising that multi-agent and service-oriented systems share many similarities [6] (see Table 1).

	MAS	SOA
interacting agents	autonomous heterogeneous agents	autonomous heterougeneous services
communication	communicative acts	messages
local view of interaction	(external) agents policies	behavioral interfaces
global view of interaction	global interaction protocols	choreographies

Table 1. Some similarities between multi-agent and service-oriented systems

When dealing with the problem of modeling global interaction protocols within a MAS, we mainly find two complementary approaches, as in the case of choreographies: approaches with aim to exactly specify how the interaction protocol should be executed by the interacting agents (such as for example AUML [7]), and approaches which consider MAS as open societies and model interaction protocols as a way to declaratively constrain the possible interactions. Social approaches abstract away from the nature of interacting entities, supporting heterogeneity, and adopt an open perspective, i.e. let participants autonomously behave as they want, where not explicitly forbidden. Furthermore, their aim is not only to support the specification task, but also to define a precise semantics of interaction, enabling the possibility to perform verification tasks. Many prominent works center around the concept of commitment in social agencies, to represent the state of affairs during the social interaction. For example, in

[8] the semantics of communicative acts is defined by means of transitions on a finite state automaton which describes the concept of commitment; in [9], the authors adopts a variant of Event Calculus to commitment-based protocols, where commitments evolve in relation to events and fluents and the semantics of messages is given in terms of predicates on such events and fluents (to describe how messages affect commitments). In the last years, Singh et al. have applied the concept of commitment-based protocols in the context of the Service Oriented Architecture and Business Process Management, by addressing the problem of business process adaptability [10] and of protocols composition [11]. The idea of taking social semantics from the MAS world and applying it to the specification of service choreographies has been adopted also in [12], although the focus is more on the procedural aspects, rather than on the declarative ones.

Within the SOCS EU Project [3] we have developed a language, called $\mathcal{S}$CIFF, for specifying global interactions protocols in open agent societies, giving its declarative semantics in terms of Abductive Logic Programming (ALP) [13]. Furthermore, we have equipped the $\mathcal{S}$CIFF language with a corresponding proof procedure, capable to verify at run-time (or a posteriori, by analizing a log of the interaction) whether interacting agents behave in a conformant manner w.r.t. the modeled interaction protocol. Protocols are specified only by considering the external observable behavior of interacting entities (i.e. the different observable events which occurred during the interaction), and by the concept of expectation about desired events and interactions; occurred events and positive/negative expectations are linked by means of forward rules called Social Integrity Constraints.

We believe that the conjunct use of declarative approaches coming from the Service Oriented Computing (SOC) and Multi Agent Systems (MAS) research areas could be fruitfully exploited to specify and verify service choreographies. To this aim, in this paper we show how DecSerFlow can be mapped to $\mathcal{S}$CIFF in an automatic and complete way, making the two proposals benefit from each other. We motivate the importance of adopting a declarative approach for modeling choreographies and show the feasibility of our approach by considering a simple but interesting running example.

The paper is organized as follows: sections 2 and 3 respectively introduce the running example and describe some issues which arise when modeling a choreography. Section 4 briefly introduce the DecSerFlow language, showing how the running example could be successfully modeled by using it; then, section 5 presents the $\mathcal{S}$CIFF framework and how DecSerFlow can be expressed in terms of $\mathcal{S}$CIFF Integrity Constraints. Discussion and Conclusions follow.

2 A running example

Let us consider a choreography that envisages three different roles: a *customer* which interacts with a *seller* to place an order of a set of items, and a *warehouse*

[3] SOcieties of heterogeneous ComputeeS, IST-2001-32530 (home page http://lia.deis.unibo.it/research/SOCS/).

which could participate to the interaction by communicating to the seller if it is able (or not) to ship the ordered items.

Each execution of the choreography (a *choreography instance*) is identified by the concept of *order*. The customer makes up an order by choosing one or more items from the seller list. During the order building phase (i.e. before committing an order), it is always possible to cancel the order; in this case, the user cannot choose other items within the same instance anymore, and the choreography terminates (a canceled order cannot be committed). After having committed an order, the customer expects a positive or negative answer from the seller. In case of a positive answer, a payment phase will be performed: the customer will pay for the order and, finally, the seller will deliver a single corresponding receipt.

The seller could freely decide whether to confirm or refuse customer's order, but sometimes it has also to consider the opinion of the warehouse about the shipment:

- the seller can confirm the order only if the warehouse has previously confirmed the shipment;
- if the warehouse states that it is unable to execute the shipment, then the seller should refuse (or have refused) the order.

3 What is the focus of a choreography?

By looking at the choreography description of the previous section, we notice that it is inherently declarative. It does not fix the control flow of the involved services, nor how they should exchange messages in order to accomplish the choreographic strategic goal. Rather, it focuses on a more abstract level, trying to capture the essential of the interaction by adopting a global and open perspective, not driven by implementation needs. This is the reason why we find, inside the description, different kinds of constraints, as for example:

- time-ordered relationships among activities ("*after* having committed an order, the customer expects a positive or negative answer");
- cardinality constraints ("the seller will deliver a *single* corresponding receipt");
- negative relationships, to express also what is forbidden during the choreography execution ("the user *cannot* choose other items [...] anymore")
- non-deterministic/opaque choices as well as non-oriented relationships among activities (e.g., the seller can refuse independently from the warehouse answer).

It is worth noting that negative information, as far as we are concerned, is not addressed by current proposals: they adopt a procedural-oriented control flow approach making the implicit assumption that all that is not explicitly modeled is forbidden. As pointed out in [5], the impossibility of expressing negative relationships forces the modeler to explicitly enumerate all the allowed possibilities, introducing ambiguous decision points. This often leads to over-constrain the model, forbidding possible executions which actually correctly realize the intended choreography (see [14] for a discussion).

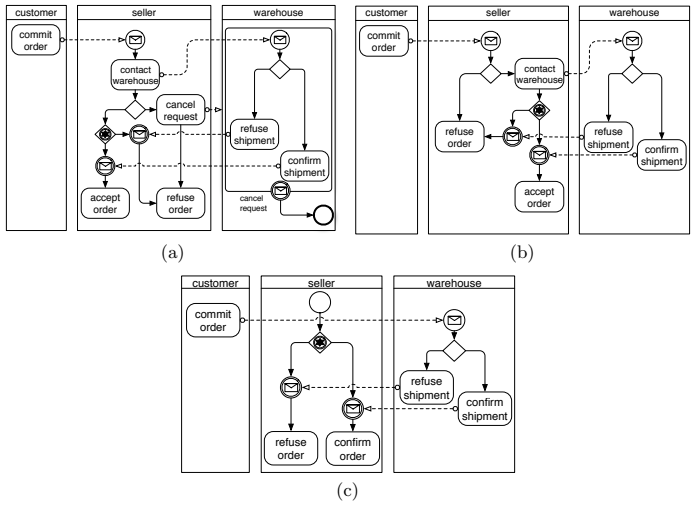


Fig. 1. Three different possible realizations of the acceptance phase in BPMN.

3.1 Avoiding over-specifications

Avoiding over-specifications is a key issue when modeling choreographies. Instead of strictly specify one of the possible behaviors which is able to respect the choreography, the aim of the modeler should be the identification of the minimal set of constraints that correctly regulate the interaction, achieving a trade-off between the specification of what is forbidden/expected and what is allowed.

An interesting example which clearly shows such issue is the order acceptance phase described in Section 2. The aim of this phase is to identify when a committed order should be accepted or rejected by the seller, taking into account (in some cases) the warehouse too. At a choreographic level, the coupling between seller and warehouse and between customer and warehouse is reduced at a minimum. First of all, when and how the warehouse is contacted is not specified; furthermore, there could be different choreography executions in which the warehouse is not contacted at all. An execution in which the seller autonomously decides to reject the order, without asking warehouse's opinion, is clearly accepted by the choreography; the case in which the warehouse refuses the shipment without observing the commited order (because e.g. it is overloaded) is implicitly envisaged too.

The over-specification problem arises if we try to model the acceptance phase by using one of the current proposed languages for choreographies. Figure 1

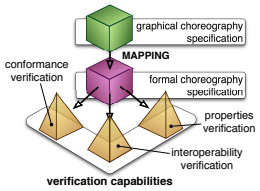


Fig. 2. A general framework for the specification and verification of choreographies

shows three different over-specified possible realizations of the acceptance phase by adopting BPMN [15] collaborative models.

Diagram 1(a) shows a choreography where, after having received order's commitment, the seller contacts the warehouse in order to know if it can ship the order or not. Then, if the seller evaluates that, due to a private policy, it is in any case unable to confirm the order, it will send a message to the warehouse in order to stop the processing of its decision; otherwise, the seller will confirm or refuse the order by considering warehouse's answer. In diagram 1(b), instead, we find that the seller firstly evaluates its internal policies, and contacts the warehouse only if the choreography prescribe to do so (i.e. only if it would accept the order; in this case, receiving an answer from the warehouse is a mandatory requirement). Finally, diagram 1(c) shows a different message flow from customer's side, and envisages a seller who does not apply any private choice, but simply forwards what has been decided by the warehouse.

The three diagrams shows that approaching the choreography modeling task by adopting a typical control+message flow perspective leads to pointlessly complicate the model, loosing some acceptable interactions. We think that such a perspective should be matter of a second phase, in which the choreographic model is grounded on a set of service behavioral interfaces, to be developed from scratch or selected from an already existing repository.

3.2 Towards a framework encompassing semantics and verification capabilities

Besides being able to really capture the different concepts involved in a choreography, possibly in a user-friendly way, a modeling language should be supported by an underlying formal (possibly declarative) semantics, hence making possible different kind of verifications. Figure 2 shows the schema of a general choreography specification and verification framework.

The framework is mainly composed by three different parts: *(i)* a (graphical) high-level modeling language, capable to specify choreographies; *(ii)* an underlying formal language, equipped with different verification capabilities; and *(iii)* a mapping between the two specification languages, in order to automatically obtain the formal description from the graphical one.

W.r.t. the verification issue, we cite three fundamental ones:

- properties verification, to ensure that a choreography meets some general (such as livelock and deadlock freedom) or specific (i.e. domain dependent) properties;
- conformance verification, to verify (at run-time or a posteriori, by analyzing a message log) whether a set of services executing the choreography behaves as prescribed by the model;
- interoperability verification [16], to check if a concrete service behavioral interface is capable to play a given role within the choreography.

It is worth noting that such three verification issues are the same as the ones introduced by Guerin and Pitt in the context of open MAS [17]: *(i)* verify protocol properties, *(ii)* verify compliance by observation, and *(iii)* verify that an agent will always comply.

We propose to ground the general framework shown in Figure 2 by adopting DecSerFlow as a graphical specification language, and to exploit $\mathcal{S}$CIFF as its underlying formalism. To demonstrate the feasibility of our approach, we show how our running example could be successfully expressed in DecSerFlow, and then provide the mapping of the different DecSerFlow concepts to $\mathcal{S}$CIFF Integrity Constraints. In [18] we already introduced the use of $\mathcal{S}$CIFF for specifying choreographies and performing the conformance verification task, leaving out the high-level specification language and the corresponding mapping; this work could be considered as a first step to fill this gap.

4 Choreography modeling in DecSerFlow

In [5], van der Aalst and Pesic propose DecSerFlow, a declarative language for modeling service flows. Besides declarativeness, its advantages rely on its appealing graphical appearance, its extensibility and its formal semantics given by means of Linear Temporal Logic (LTL).

As described in [5], modeling service specifications in DecSerFlow starts by identifying the different involved activities (i.e. atomic logical unit of work), and then to identify constraints on their execution, a lá policies/business rules. Constraints are given as templates, i.e. as relationships between two (or more) whatsoever activities: typically, the terms *source* and *target* activities indicate activities linked by a relationship, where the execution of the source activity "activates" the relation and impose some constraint on the target activity. The meaning of each constraint template is expressed as an LTL formula, hence the name "formulas" to indicate DecSerFlow relationships.

DecSerFlow core relationships are grouped into three families:

- *existence formulas*, unary relationships used to constrain the cardinality of activities;
- *relation formulas*, which define (positive) relationships and dependencies between two (or more) activities;

source		**template name**	**target**	**description** (from the example)
cancel order	C_1	negation response	choose item	in case of cancelation, the user cannot choose other items [...] anymore
	C_2	responded absence	commit order	a canceled order cannot be commited
commit order	C_3	response	refuse or confirm order	after having committed an order, the customer expects a positive or negative answer from the seller
	C_4	precedence	confirm shipment	the seller could confirm the order only if the warehouse has previously confirmed the shipment
confirm order	C_5	response	payment	in the former situation [positive answer], a payment phase will be performed
refuse shipment	C_6	responded existence	refuse order	if the warehouse [...] is unable to execute the shipment, then the seller should refuse (or have refused) the order
payment	C_7	response	receipt delivery	the customer will pay for the order and, then, the seller should deliver a single corresponding receipt
receipt delivery	C_8	cardinality 0..1		the seller will deliver a single corresponding receipt

Table 2. Mapping the statements of the running example to DecSerFlow constraints

– *negation formulas*, the negated version of relation formulas.

In order to present the DecSerFlow notation and how it could be effectively used to model service choreographies, we show how our running example could be expressed as a DecSerFlow diagram. In our example, we will use only a limited number of DecSerFlow relations, such as *responded existence* (if A is performed, then also B must be performed, either before or after A) and *response* (if A is performed, then B must be performed after). For a complete description of the DecSerFlow language and its underlying LTL formalization, the interested reader is referred to [5].

4.1 Modeling the running example

Table 2 shows how the different statements of our running example could be translated to DecSerFlow activities and constraints in an intuitive and straightforward way.

For example, to specify that only a single receipt should be delivered by the seller, we may use the DecSerFlow *absence*(1) existence formula. The *absence*(N) formula indeed states that the involved activity cannot be executed more than N times, i.e. constrains its cardinality between 0 and N. A *responded existence* relation is used to model the relationship between the refusal of shipment and order: it states that if the shipment is refused by the warehouse, the refuse order activity should be executed too, either before or after it. DecSerFlow's *response* relation imposes a forward temporal order on the responded existence formula; for example, constraint C_3 states that after having executed the order commitment, then a positive or negative answer from the seller is expected to be performed afterwards (when more target activities are involved, they are considered in a disjunctive manner). Obviously, a *precedence* formula is provided too, (e.g. C_4).

DecSerFlow defines also more complex relationships, which are not part of our running example. An example is the *chain response* formula, which allows the user to model the typical strict sequence relationships of business processes: it states that whenever the source happens, then the target should be performed immediately after it.

For each positive relationships, DecSerFlow defines a corresponding negative version. Basically, negative relations forbids the execution of the target activity under certain conditions. E.g., the *responded absence* relationship (which is actually the negation of the *responded existence* one) states that if the source activity is executed, then the target activity is forbidden. Such a negative relationship is used e.g. to model the impossibility to commit an order if it is canceled by the customer (constraint C_2). It is worth noting that, as pointed out in [5], some negative relations are equivalent; e.g., stating that B is responded absence of A is equivalent to specify that A and B should not coexist in the same execution instance.

4.2 Completing the DecSerFlow model

By deeply analyzing the running example, we could complete the DecSerFlow diagram shown in Table 2 with other useful inferred constraints, in order to really model all the intented concepts of the description; the result is shown in Table 3, while in Figure 3 the whole set of constraints is shown using the DecSerFlow graphical notation (see also Tables 4 and 5 for the correspondence between the DecSerFlow graphical symbols and their meaning).

C_{15} and C_{16} deal with the core concept of the choreography, which is actually the commitment of one order. Since such an order could be canceled, we attach an *absence*(1) constraint to the order commitment activity (to express that at most one order can be committed), and bind the cancelation and the commitment with a *mutual substitution* DecSerFlow relation, which states that at least one of the two bounded activities has to be executed (i.e. an order should be committed or canceled).

5 Mapping DecSerFlow to the $\mathcal{S}$CIFF framework

The $\mathcal{S}$CIFF [13] language was originally introduced for the specification of global interaction protocols in open agent societies. As we have already pointed out, it does not make any assumption about participants internals, but instead focuses

source		**type**	**target**	**description** (from the example)
refuse order	C_9	precedence	commit order	An answer from the seller is valid only if it is performed after order commitment
confirm order	C_{10}	precedence	commit order	
payment	C_5	precedence	confirm order	A valid payment should be preceded by the confirmation of the order
deliver receipt	C_7	precedence	payment	The receipt should be delivered only if the order has been paid
target		**type**	**target**	**description** (from the example)
confirm order	C_{11}	not co-existence	refuse order	Possible answers are mutually exclusive
confirm shipment	C_{12}	not co-existence	refuse shipment	
commit order	C_{13}	precedence	choose item	an order is made up by at least one chosen item
cancel order	C_{14}	precedence	choose item	
commit order	C_{15}	cardinality 0..1		the choreography centres around the concept of a single order, which could possibly be canceled
commit order	C_{16}	mutual substitution	cancel order	

Table 3. Inferred DecSerFlow constraints to complete the running example

on the observable and relevant events which occur within the society at runtime. To let the user decides which are the relevant events inside the considered domain, the $\mathcal{S}$CIFF language completely abstracts from the problem of deciding "what is an event".

$\mathcal{S}$CIFF adopts an explicit notion of time, and models the occurrence of an event Ev at a certain time T as $\mathbf{H}(Ev, T)$, where Ev is a logic programming term and T is an integer, representing the discrete time point at which the event happened (the bold **H** stand for "Happened"). The set of all the events that have happened during a protocol execution constitutes its interaction log.

Beside the explicit representation of what has already happened, $\mathcal{S}$CIFF introduces the concept of "what" is expected to happen, and "when". The notion of expectation plays a key role when defining interaction protocols, choreographies, and more in general any dynamically evolving process: it is quite natural, in fact, to think of such processes in terms of rules of the form "if A happened, then B should be expected to happen, under certain conditions". In agreement with DecSerFlow, $\mathcal{S}$CIFF pays particular attention to the openness of interaction: interacting peers are not completely constrained, but they enjoy some freedom. This means that the prohibition of a certain event should be explicitly expressed in the model and this is the reason why $\mathcal{S}$CIFF supports also the concept of neg-

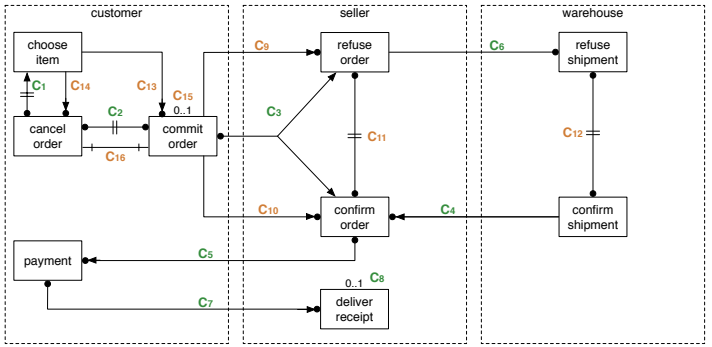


Fig. 3. DecSerFlow model of the running example

ative expectations (i.e. of what is expected not to happen). Positive expectations about events come with form $\mathbf{E}(Ev, T)$, where Ev and T could be variables, or they could be grounded to a particular (partially specified) term or value respectively. Constraints (a là Constraint Logic Programming), like $T > 10$, as well as Prolog predicates can be specified over the variables; attaching the example constraint on the above expectation means that the expectation is about an event to happen at a time greater than 10. Conversely, negative expectations about events come with form $\mathbf{EN}(Ev, T)$; just to give an intuition, variables used inside negative expectations are universally quantified: writing $\mathbf{EN}(Ev, T) \wedge T > 10$ means that Ev is forbidden at any time which is greater than 10.

Social Integrity Constraints are forward rules used to link happened events and expectations in order to define the declarative rules which regulate the course of interaction, i.e. model the interaction protocol. They come as rules of the form $body \rightarrow head$, where $body$ can contain (a conjunction of) happened events and expectations, and $head$ can contain (a disjunction of conjunctions of) positive and negative expectations. For example, to model that "if a customer sends the payment to the seller, then the seller should answer delivering the corresponding receipt, within 24 hours" we could use the following Integrity Constraint:

$$\mathbf{H}(pay(Customer, Seller, Item), T_p) \rightarrow \mathbf{E}(deliver(Seller, Customer, receipt(Order, Id)), T_d) \wedge T_d > T_p \wedge T_d < T_p + 24.$$

$\mathcal{S}$CIFF accepts also a (Prolog) knowledge base, where the user can define all the pieces of knowledge which are independent from the interaction. Defined predicates could be used inside Integrity Constraints, reconciling forward, abductive reasoning with backward, goal-oriented reasoning. Finally, note that interaction is considered to be goal oriented: the same interaction protocol could be seamlessly used for achieving different goals, which can be expressed by means of Prolog predicates and expectations.

The $\mathcal{S}$CIFF semantics is based on Abductive Logic Programming: an interaction specification (i.e. the set of rules regulating the allowed possible interactions) is mapped to an Abductive Logic Program, where Integrity Constraints define the interaction protocols, and positive/negative expectations are considered as abducibles. The operational counterpart of the language, namely the $\mathcal{S}$CIFF proof procedure, is indeed able to verify conformance of a set of interacting entities w.r.t. the considered protocol by hypothesizing positive (resp. negative) expectations and checking whether a matching happened event actually exists (resp. does not exist). For a detailed description of the $\mathcal{S}$CIFF language, as well as its declarative semantics and the corresponding proof procedure, the interested reader is referred to [13].

5.1 Expressing DecSerFlow concepts as Integrity Constraints

Let us now consider again our running example, in order to explain how the different DecSerFlow concepts could be mapped to $\mathcal{S}$CIFF Integrity Constraints.

Roughly speaking, each DecSerFlow constraint is mapped to a set of $\mathcal{S}$CIFF Integrity Constraints. The body of the Integrity Constraint which maps a relation or negation formula is constituted by the happened event which corresponds to the formula's source (each DecSerFlow relation is triggered when its source activity is performed). Depending on the nature of the relation, the head is instead is determined by (a disjunction of) positive or negative expectations.

For example, to specify that a generic activity A is subject to an $absence(N)$ cardinality constraint, $\mathcal{S}$CIFF uses an Integrity Constraint which states that if N different executions of A are performed, then the $N+1$-th is forbidden. Since $\mathcal{S}$CIFF adopts an explicit notion of time, differences between executions are modeled as differences between the involved execution times; hence, the $absence(N)$ on activity A can be specified as follows[4]:

$$\bigwedge_{i=1}^{N}\Big(\mathbf{H}(A,T_i)\wedge T_i > T_{i-1}\Big) \rightarrow \mathbf{EN}(A,T)\wedge T > T_N.$$

Furthermore, thanks to the explicit notion of time, another interesting feature of the mapping is that the "response" and "precedence" version of each formula are formalized in the same way, but by imposing opposite constraints on the involved times. Table 4 explicitly points out such similarities by showing how the responded existence, response and precedence constraints, as well as their negated version, can be mapped to $\mathcal{S}$CIFF.

Some DecSerFlow formulas are translated to $\mathcal{S}$CIFF in a slight different way. In particular, their mapping do not have a triggering part but simply generates a set of expectations (see Table 5). Therefore, they define, in some sense, the initial goal of the choreography, since the corresponding expectations are generated independently from the interaction.

Table 6 represents the complete mapping of the DecSerFlow model shown in Figure 3. For the sake of simplicity, we have left out the information about

[4] We suppose that $T_0 = 0$ and that at a given time only one activity can happen.

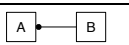
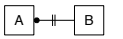
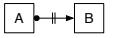
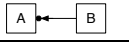
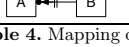
DecSerFlow formula	Meaning	$\mathcal{S}$CIFF Integrity Constraint
	if A is executed, then B should be executed too	$\mathbf{H}(A,T_A) \rightarrow \mathbf{E}(B,T_B)$
	if A is executed, then B cannot be executed	$\mathbf{H}(A,T_A) \rightarrow \mathbf{EN}(B,T_B)$
	if A is executed, then B should be executed after it	$\mathbf{H}(A,T_A) \rightarrow \mathbf{E}(B,T_B) \quad \wedge T_B > T_A$
	if A is executed, then B cannot be executed after it	$\mathbf{H}(A,T_A) \rightarrow \mathbf{EN}(B,T_B) \wedge T_B > T_A$
	if A is executed, then B should be executed before it	$\mathbf{H}(A,T_A) \rightarrow \mathbf{E}(B,T_B) \quad \wedge T_B < T_A$
	if A is executed, then B cannot be executed before it	$\mathbf{H}(A,T_A) \rightarrow \mathbf{EN}(B,T_B) \wedge T_B < T_A$

Table 4. Mapping of the simple DecSerFlow relation and negation formulas in $\mathcal{S}$CIFF

DecSerFlow formula	Meaning	$\mathcal{S}$CIFF Integrity Constraint
	A is forbidden	$\rightarrow \mathbf{EN}(A,T_A)$
	A has to be executed at least N times	$\rightarrow \bigwedge_{i=1}^{N}\Big(\mathbf{E}(A,T_i)\wedge T_i > T_{i-1}$
	A or B should be executed	$\rightarrow \mathbf{E}(A,T_A)\vee\mathbf{E}(B,T_A)$

Table 5. Mapping of "goal-oriented" DecSerFlow formulas

activities originators (i.e. about the role responsible for an activity); such an information could be seamlessly added to the $\mathcal{S}$CIFF formalization, but it is not envisaged in the current version of DecSerFlow.

As already pointed out, DecSerFlow defines other constraints, missing in our running example. Anyway, they are mapped to $\mathcal{S}$CIFF Integrity Constraints too (see [19] for a complete description of such a mapping). For example, the following rule maps the *chain response* between A and B:

$$\mathbf{H}(A,T_A) \rightarrow \mathbf{E}(B,T_B)\wedge T_B > T_A \wedge \mathbf{EN}(X,T_X)\wedge T_X > T_A \wedge T_X < T_B.$$

The translation tries to intuitively capture the notion of *next state*, which is directly expressed in LTL as a temporal modality (by using the operator $\circ$). It relies on the fact that if B should belong to the next state of A, then between the two execution times no other activity should be performed. For a description of the complete translation of core DecSerFlow concepts to $\mathcal{S}$CIFF, see [19].

6 Discussion and Conclusions

In this work we have proposed a conjunct use of declarative approaches coming from the SOC and MAS research areas, to the aim of specifying and verifying service choreographies.

C_1	$\mathbf{H}(cancel\_order, T_c) \rightarrow \mathbf{EN}(choose\_item, T_i) \wedge T_i > T_c.$
C_2	$\mathbf{H}(cancel\_order, T_c) \rightarrow \mathbf{EN}(commit\_order, T_o).$
	$\mathbf{H}(commit\_order, T_o) \rightarrow \mathbf{EN}(cancel\_order, T_c).$
C_3	$\mathbf{H}(commit\_order, T_o) \rightarrow \mathbf{E}(confirm\_order, T_c) \wedge T_c > T_o$
	$\vee\ \mathbf{E}(refuse\_order, T_r) \wedge T_r > T_o.$
C_4	$\mathbf{H}(confirm\_order, T_o) \rightarrow \mathbf{E}(confirm\_shipment, T_s) \wedge T_s < T_o.$
C_5	$\mathbf{H}(confirm\_order, T_c) \rightarrow \mathbf{E}(payment, T_p) \wedge T_p > T_c.$
	$\mathbf{H}(payment, T_p) \rightarrow \mathbf{E}(confirm\_order, T_c) \wedge T_c < T_p.$
C_6	$\mathbf{H}(refuse\_shipment, T_s) \rightarrow \mathbf{E}(refuse\_order, T_o).$
C_7	$\mathbf{H}(payment, T_p) \rightarrow \mathbf{E}(deliver\_receipt, T_d) \wedge T_d > T_p.$
	$\mathbf{H}(deliver\_receipt, T_d) \rightarrow \mathbf{E}(payment, T_p) \wedge T_p < T_d.$
C_8	$\mathbf{H}(deliver\_receipt, T_{d1}) \rightarrow \mathbf{EN}(deliver\_receipt, T_{d2}) \wedge T_{d2} > T_{d1}.$
C_9	$\mathbf{H}(refuse\_order, T_r) \rightarrow \mathbf{E}(commit\_order, T_o) \wedge T_o < T_r.$
C_{10}	$\mathbf{H}(confirm\_order, T_c) \rightarrow \mathbf{E}(commit\_order, T_o) \wedge T_o < T_c.$
C_{11}	$\mathbf{H}(refuse\_order, T_r) \rightarrow \mathbf{EN}(confirm\_order, T_c).$
	$\mathbf{H}(confirm\_order, T_c) \rightarrow \mathbf{EN}(refuse\_order, T_r).$
C_{12}	$\mathbf{H}(refuse\_shipment, T_r) \rightarrow \mathbf{EN}(confirm\_shipment, T_c).$
	$\mathbf{H}(confirm\_shipment, T_c) \rightarrow \mathbf{EN}(refuse\_shipment, T_r).$
C_{13}	$\mathbf{H}(commit\_order, T_c) \rightarrow \mathbf{E}(choose\_item, T_i) \wedge T_i < T_c.$
C_{14}	$\mathbf{H}(cancel\_order, T_c) \rightarrow \mathbf{E}(choose\_item, T_i) \wedge T_i < T_c.$
C_{15}	$\mathbf{H}(commit\_order, T_{c1}) \rightarrow \mathbf{EN}(commit\_order, T_{c2}) \wedge T_{c2} > T_{c1}.$
C_{16}	$\rightarrow \mathbf{E}(commit\_order, T_o)$
	$\vee\ \mathbf{E}(cancel\_order, T_c).$

Table 6. Mapping of the DecSerFlow running example to $\mathcal{S}$CIFF

In particular, we have chosen DecSerFlow as the modeling language and $\mathcal{S}$CIFF as its underlying formalization. To make DecSerFlow benefit of $\mathcal{S}$CIFF in an automatic way, we have shown how the different DecSerFlow concepts can be mapped to $\mathcal{S}$CIFF Integrity Constraints and applied our methodology on a running example. The advantage of such a translation is twofold: on one hand, it is possible to specify $\mathcal{S}$CIFF rules by using an intuitive, extensible and user-friendly graphical language; on the other hand, a DecSerFlow model may be grounded not only on LTL but also on the $\mathcal{S}$CIFF abductive framework, acquiring some new advantages and features, such as:

- Expressivity of the language. The $\mathcal{S}$CIFF language is capable to model rich constraints and conditions on data and execution times involved in the interaction; we are currently studying how DecSerFlow could be extended to graphically represent such constraints.
- Verification capabilities of the $\mathcal{S}$CIFF framework. As described in [13, 18], by translating DecSerFlow to a $\mathcal{S}$CIFF specification we could automatically use it to perform the conformance verification task. Furthermore, $\mathcal{S}$CIFF has been extended to deal also with the verification of properties [20] and interoperability [21]; we intend to study how such extended proofs could be applied to DecSerFlow models, aiming at covering all the building parts of the general framework schema shown in figure 2.
- Possibility to mine DecSerFlow models from execution traces. Since $\mathcal{S}$CIFF belongs to the logic programming setting, it is possible to apply all the reasoning techniques developed inside such a setting on it. In particular, in [22] we have shown how an Inductive Logic Programming algorithm can be adapted to mine $\mathcal{S}$CIFF rules from event logs; thanks to the one-to-one mapping of DecSerFlow concepts to $\mathcal{S}$CIFF, it is then possible to automatically obtain a corresponding DecSerFlow description of the mined model.

Finally, as future work we envisage a deep comparison between $\mathcal{S}$CIFF and LTL, to better understand their strength, weaknesses and relationships and to exploit the possibility to have two different mappings of DecSerFlow.

References

1. Barros, A., Dumas, M., Oaks, P.: A critical overview of the web services choreography description language (WS-CDL). BPTrends (2005)
2. van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., Russell, N., Verbeek, H.M.W., Wohed, P.: Life after BPEL? In Bravetti, M., Kloul, L., Zavattaro, G., eds.: International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005, Proceedings. Volume 3670 of Lecture Notes in Computer Science., Springer (2005) 35–50
3. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services version 1.1. (2003)
4. W3C: Web services choreography description language version 1.0
5. der Aalst, W.M.P.V., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM'06. Volume 4184 of LNCS., Springer (2006)
6. Baldoni, M., Baroglio, C., Martelli, A., Patti, V., Schifanella, C.: Verifying the conformance of web services to global interaction protocols: A first step. In Bravetti, M., Kloul, L., Zavattaro, G., eds.: International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005, Proceedings. Volume 3670 of Lecture Notes in Computer Science., Springer (2005) 257–271
7. Bauer, B., M´uller, J.P., Odell, J.: Agent uml: a formalism for specifying multiagent software systems. In: First international workshop, AOSE 2000 on Agent-oriented software engineering, Springer-Verlag (2001) 91–103
8. Fornara, N., Colombetti, M.: Operational specification of a commitment-based agent communication language, Bologna, Italy (July 15–19 2002) 535–542

9. Yolum, P., Singh, M.: Flexible protocol specification and execution: applying event calculus planning using commitments. 527–534
10. Desai, N., Chopra, A.K., Singh, M.P.: Business process adaptations via protocols. In: 2006 IEEE International Conference on Services Computing (SCC 2006), 18-22 September 2006, Chicago, Illinois, USA, IEEE Computer Society (2006) 103–110
11. Mallya, A.U., Desai, N., Chopra, A.K., Singh, M.P.: Owl-p: Owl for protocol and processes. In Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M., eds.: 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands, ACM (2005) 139–140
12. Walton, C.: Protocols for web service invocation. Proceedings of the AAAI Fall Symposium on Agents and the Semantic Web (ASW05), Arlington, Virginia, USA. (November 2005)
13. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. TOCL (2007) Accepted for publication.
14. Chopra, A.K., Singh, M.P.: Producing compliant interactions: Conformance, coverage, and interoperability. In Baldoni, M., Endriss, U., eds.: DALT. Volume 4327 of Lecture Notes in Computer Science., Springer (2006) 1–15
15. White, S.A.: Business process modeling notation specification. Technical report, OMG (2006)
16. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: A priori conformance verification for guaranteeing interoperability in open environments. In: Service-Oriented Computing - ICSOC 2006, 4th International Conference, Chicago, IL, USA, December 4-7, 2006, Proceedings. Volume 4294 of Lecture Notes in Computer Science., Springer (2006) 339–351
17. Guerin, F., Pitt, J.: Proving properties of open agent systems, Bologna, Italy (July 15–19 2002) 557–558
18. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., Storari, S., Torroni, P.: Computational logic for run-time verification of web services choreographies: Exploiting the *socs-si* tool. In Bravetti, M., Núñez, M., Zavattaro, G., eds.: Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006, Proceedings. Volume 4184 of Lecture Notes in Computer Science., Springer (2006) 58–72
19. Chesani, F., Mello, P., Montali, M., Storari, S.: Towards a decserflow declarative semantics based on computational logic. Technical Report DEIS-LIA-07-002, DEIS, Bologna, Italy (2007)
20. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Security protocols verification in Abductive Logic Programming: a case study. In Dikenelli, O., Gleizes, M.P., Ricci, A., eds.: Proceedings of ESAW'05, Kuşadasi, Aydin, Turkey, October 26-28, 2005. Volume 3963. (2006) 106–124
21. Alberti, M., Gavanelli, M., Lamma, E., Chesani, F., Mello, P., Montali, M.: An abductive framework for a-priori verification of web services. In Bossi, A., Maher, M.J., eds.: Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 10-12, 2006, Venice, Italy, ACM (2006) 39–50
22. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: Proceedings of the 5th International Conference on Business Process Management (BPM 2007), LNCS (2007) To appear

Mapping BPMN to Agents: An Analysis

Holger Endert, Tobias Küster,
Benjamin Hirsch, and Sahin Albayrak

DAI-Labor, Technische Universität Berlin
Faculty of Electrical Engineering
and Computer Science
{holger.endert|tobias.kuester|
benjamin.hirsch|sahin.albayrak}@dai-labor.de

Abstract. In industry the development of software applications is usually a complex and demanding task, and the design and the technical realisation is often spread among different roles, which leads to a time consuming and error-prone exchange of knowledge. In order to ensure the correct translation from business idea to implementation it is crucial to allow for the correct and complete exchange of information between these roles.
In this paper, we describe an automated mapping from business process diagrams to agent concepts that simplify the transfer of knowledge between the roles involved in the software development process. Our approach benefits from building upon an intuitive visual specification language on the one hand, and from using a powerful and flexible execution platform on the other.

1 Motivation

Multi-agent systems (MAS) arguably provide an answer to the creation of complex distributed applications which are manageable and adapt to the state of the environment. However, even though agent research has been ongoing for more than a decade now, it still is a mainly academic subject. The rapid uptake of technologies such as web-services, which aim squarely at the same problem space as multi-agent systems [9], however suggests that a corresponding demand is existent in industry. We believe that one important reason for the slow adoption of agents in industry is the disconnect that exists between business- and multi-agent oriented software development.

It appears that while introducing mentalistic notions to model agents is a very intuitive approach, business users tend to think in terms of processes, and business entities – and they are the ones that make overall design decisions! Therefore, we approach the issue of designing multi-agent frameworks from the vantage point of the business user, and provide a mapping from the Business Process Modeling Notation (BPMN [12]), a graphical language used to represent business processes, to agent concepts.

Choosing the BPMN as the source language for our mapping, and thus as the language for designing distributed business applications, our approach can

offer certain advantages, such as providing a simple and intuitive graphical notation. Although its basic concepts are simple, further specification options make the language sufficiently expressive. The BPMN is also suitable for defining processes of different levels of abstraction. As the focus lies on the process model of an application, an extension will be required in order to integrate the support for structured data types. It seems reasonable to rely on existing specification languages for that purpose. Since the BPMN has emerged from a standardisation effort for business processes by the OMG, it will most likely play a role in software specification in the near future, which additionally encourages our approach.

Using multi-agent systems as target model provides the capabilities of a powerful execution environment and an intuitive abstraction model. For instance, having an explicit representation of a process participant, i.e. an agent, makes the application more accurate w.r.t. its structure. In contrast to that, a mapping to BPEL (e.g. as presented in [12] or [19]) would discard this structural information completely. MAS offer some further properties, for instance their intuitive design through mentalistic notions, scalability and flexibility, which help to deploy the resulting applications. Because of the variety of existing multi-agent frameworks, we have to define the requirements for the target model as generically as possible. Therefore we decided to map into BDI-type MAS, because these seem to be most suitable to capture the process model of a business process diagram.

To summarise, our main contribution of this paper is to provide one step towards the connection of business process design and the design of multi-agent systems by means of an automated mapping. Our focus lies on the identification of BDI-related concepts that can represent specific elements of the BPMN, rather than specifying all details of the control flow, which is already covered by several other authors (see for example [13] or [19]). The key feature of this approach is to facilitate the correct and fast transformation of the original concepts into code that can be used directly or with adaptations within the resulting business application.

The rest of this paper is structured as follows. In Section 2 and 3 we will provide a short introduction to BPMN and agents and state a formal description for both. In Section 4 we will describe the actual mapping from BPMN to agents. Then we will present related work in Section 5, and finally we will conclude and give rise to future work in Section 6.

2 BPMN

The *Business Process Modeling Notation* (BPMN), which is maintained by the Object Management Group, is a graphical notation for describing various kinds of processes. The main notational elements in BPMN are *FlowObjects*, that are contained in *Pools* and connected via *Sequence-* and *MessageFlows*. They subdivide in *Events*, atomic and composite *Activities* and *Gateways* for forking and joining. SequenceFlows describe the sequence in which the several FlowObjects have to be completed, while MessageFlows describe the exchange of messages

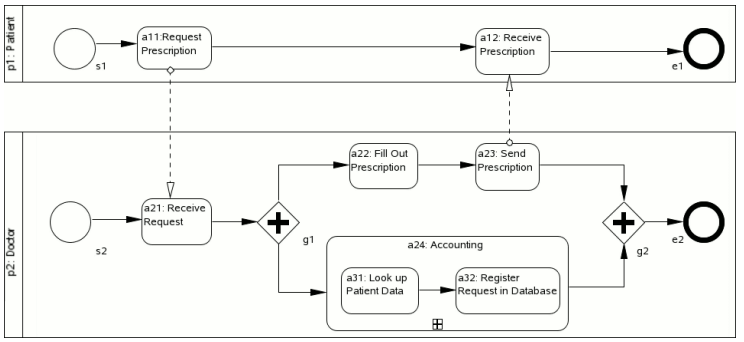


Fig. 1. BPMN example diagram.

between Pools. Thus, BPMN combines the definition of local workflows and the interaction between them.

An example diagram is shown in Figure 1. As can be seen, the meaning of the diagram can be understood intuitively and without any specific knowledge about the semantics of BPMN. This strength is at the same time a weakness in that BPMN is more of a graphical notation than a formal one, which is required for automatic interpretation. Although the specification includes some semantics and even defines a mapping to WS-BPEL [12, Chapter 11], much of the semantics is especially tailored for this mapping and the focus of the specification is clearly the visual representation.

2.1 Graph-Based BPMN Representation

For the purpose of our mapping, we can simplify the BPMN by discarding all layout information. What is left is a graph-like structure with several types of vertices and edges that should satisfy certain properties in order to be a correct business process diagram (BPD). Therefore, it is straightforward to define a BPD in terms of a graph and graph properties[1]. This representation has the additional advantage of allowing further analyses that help to implement correct diagrams. In general, a BPD is defined as the following graph structure:

Definition 1. *(BPD-Graph) - Let* $BPD = (O, F, src, tar)$ *be a graph with*

- O *— the set of nodes (objects) in the BPD-Graph.*
- F *— the set of edges (message and sequence flows) in the BPD-Graph.*
- $src, tar : F \rightarrow O$ *two functions, which identify the source and target objects of each edge.*

[1] More specifically, *typed attributed graphs* are required in order to capture the attributes of the nodes that are necessary for the mapping.

In order to distinguish the different nodes and edges in a BPD-Graph, some additional notations are required. Therefore, let O be partitioned into the disjoint subsets O^E, O^A, O^G, O^P, where

- O^E — the set of event-nodes, which can be further partitioned into the disjoint subsets O^E_S, O^E_E, O^E_I, i.e. start-, end- and intermediate events.
- O^A — the set of activity-nodes, which can be further partitioned into the disjoint subsets O^A_{At}, O^A_{Sub}, i.e. the atomic activity nodes and the subprocess nodes.
- O^G — the set of gateway-nodes, which can be further partitioned into the disjoint subsets O^G_S and O^G_M, i.e. the splitting and the merging gateway nodes. These can again be partitioned into the subsets for exclusive (XOR), inclusive (OR) and parallel (AND) split and merge gateways ($O^G_{S,X}, O^G_{S,O}, O^G_{S,A}, O^G_{M,X}$, $O^G_{M,O}$ and $O^G_{M,A}$).
- O^P — the set of pool-nodes.

Elements of the BPD have attributes, which are also relevant for the mapping, because they contain the data and the parameters used within the process. In slight abuse of notion, we refer to attributes of an element by using the common dot-syntax. For example, if p is a pool, the term $p.name$ refers the *name*-attribute of this element.

Note that not every BPD graph, as presented here, refers to a valid diagram according to the specification. Hence, a BPD is said to be *correct* if it satisfies a set of additional properties. Actually, these properties claim even more than correctness, namely that the diagram is *normalised*, i.e. it conforms to a canonical representation. BPDs can be normalised through a graph transformation, such that our approach is not generally limited to a subset of diagrams. Normalised BPDs simplify the mapping because there are less cases to consider. Both the normalisation and correctness is analysed in [10], and hence not part of this paper.

In the next section, we will present a formal description of the agent concepts, which we are going to use in the mapping.

3 Agents

Throughout the literature, there is no single and universally accepted definition of the term agent (for a number of possible definitions see e.g. [11]). Nevertheless, certain properties are required in order to translate a complete BPD graph. In our opinion, agents that follow the BDI paradigm [6] are best suited to capture all the functionality that is expressed with BPMN. Hence, we propose to use agents that are capable of performing BDI-related reasoning on goals, plans, intentions and beliefs. Subsequently, we define an agent as a tuple containing plans, goals, intentions, beliefs and an identifier:

Definition 2. *(Agent) - An agent is a tuple Λ = (id, Π, Γ, I, Θ) with*

- *id — A unique string that allows to identify an agent.*

- *Π — A set of plans,that the agents knows.*
- *Γ — A set of goals an agent is trying to achieve.*
- *I — A set of intentions (i.e. selected plans).*
- *Θ — A set of beliefs an agent knows.*

Plans define an agents possible behaviour. A plan may be selected for execution by the agent, which results in an intention. Goals refer to usual achievement goals, and lead to new intentions in order to fulfil them. Beliefs is the set of facts that an agent assumes to be true. These building-blocks are subsequently specified in more depth.

3.1 Plans

Plans define the actions of an agent that may be carried out in order to achieve a certain goal. A plan must have a defined signature and an executable (or interpretable) script that is capable of performing BDI-related operations, such as manipulating the beliefs or goals of an agent. Additionally, it must allow to organise these elements using control flow, e.g. for conditional or parallel branching. Often plans consist also of preconditions and effects, but since we do not require them in this work, we omitted them for simplicity reasons.

Definition 3. *(Plan) - A plan is a tuple π = (Name, In, Out, X), where the elements are defined as follows:*

- *Name — The name that identifies the plan.*
- *In — The list of input variables of the plan.*
- *Out — The list of output variables of the plan.*
- *X — The script of the plan, which is a sequence of control flow elements together with operations for manipulating the agents state.*

Variables occurring in the In- and Out-lists are defined as tuples, containing a name and a type $\vartheta := (Name, Type)$. The script X must support at least the following operations:

- **invoke(**$name$**)** — Invokes a plan for execution.
- **addGoal(**γ**)** — Adds a goal to the goal-base.
- **send(**μ**)** — Sends a message μ.
- **receive(**μ**)** — Receives a message μ.

3.2 Goals

A goal in this work corresponds to a usual *achievement goal*, and is related to a plan. This relation is defined via the plans and the goals signatures. Both must fit to each other by having the same name, and the *In*- and *Out*-lists must match to each other w.r.t. the sizes and types of the elements.

Definition 4. *(Goal) - A goal is a tuple γ = (Name, In, Out), where the name is an identifier, and In and Out are lists of data-elements.*

Note, that the in- and output elements are not restricted to variables here, but can contain any data elements.

3.3 Data and Beliefs

Data can be any (evaluated) expression and may be bound to a variable. A belief (fact) is data, which is located in the belief-base of an agent, and is referable via the name.

Definition 5. *(Fact) - A fact is a tuple θ = (Name, Value, Type). The Name is a unique identifier, the Value represents the content of the fact, and the Type belongs to a formal definition of a domain, e.g. to an ontology.*

As can be seen, adding a variable to the belief-base results in a fact.

3.4 Messages

The exchange of messages is one major aspect of business processes. For the purpose of the mapping, we have to deal with messages in multiple cases (send-tasks, events, etc.). We therefore summarize our minimum requirements on messages here.

Definition 6. *(Message) - A message is a tuple μ = (Name, Sender, Receiver, Content), with:*

- *Name — The id of the message.*
- *Sender — The unique id of the sender (agent).*
- *Receiver — The unique id of the receiver (agent).*
- *Content — A list of data.*

Although agents (and multi-agent systems) often contain many more aspects, no further elements are necessary for the mapping we will describe in the next section.

4 Mapping

The mapping we present is similar to that defined in the BPMN specification, which uses WS-BPEL [8] as target model. Since we use agents instead, the results will have several (desired) differences and advantages. For instance, agent frameworks that conform to the FIPA agent platform specification [1] provide infrastructure services that enable agents to find and cooperate with each other in a flexible manner. Thus, an application is easier to use and distribute among execution platforms. Another interesting point is, that agents can be modelled in terms of mental attributes, such as goals, beliefs and intentions, and hence their implementation can be quite intuitive and on a higher level of abstraction.

The approach we take is the definition of a graph transformation system, which is specified by a set of transformation rules, that map elements of the source model (BPMN) to elements of the target model (agents). The mapping is separated into a set of different tasks which have to be completed in a specific order. First of all, for any BPD, we apply a normalization process, that translates a

given graph into its canonical representation. Thereafter we check for syntactical inconsistencies by evaluating graph properties. Then we analyze the semantics of the BPD using an approach based on petri nets, as presented in [10]. Finally, if everything went well so far, the mapping itself is executed. In this work, we only deal with the final part, the mapping, focusing on the most relevant BPMN elements and their counterparts in the agents model (Subsection 4.1). There, all rules are in the form

$$LHS \Longrightarrow RHS,$$

where LHS (left-hand side) is a pattern that is searched in the source graph, and RHS (right-hand side) contains the elements that are added to the target model. The rules are not fully specified in that they do not provide $NACs$ (negative application conditions) or a reference model. The former is in most cases used to ensure that each element is mapped only once. The latter stores the progress of the mapping and may contain additional temporary elements, that simplify the mapping.

4.1 Mapping of Nodes

We initially start with providing the mapping-rules for nodes in a top to bottom manner. The most top-level nodes of a BPD are pools, representing the participants within a diagram. These are mapped directly to agents by applying this rule:

Rule 1 *(Pools) - Let $p \in O^P$ be a pool of a given BPD.*

$$p \Longrightarrow \Lambda := (p.name, \emptyset, \emptyset, \emptyset, \emptyset)$$

So far, an agent does not possess any specific knowledge in terms of plans, goals, intentions or beliefs. These elements are added by the application of subsequent rules. For each pool, there exists exactly one process that is translated into a plan and added to the agents plan-library.

Rule 2 *(Process) - Let p be a pool, and Λ be the agent, that was created from p. Let further be x the process of p, i.e. $p.process = x$.*

$$x \Longrightarrow \pi := (x.name, [\,], [\,], [\,])$$
$$\Pi_\Lambda := \Pi_\Lambda \cup \{\pi\}$$

The plan is given with a name, an empty in- and output-list, and an empty script, which has to be filled during the mapping of its control flow. The in- and output-lists depend on the start- and end-events of the process. If these are message-events, the message-properties are used to create the parameters and the results as follows:

Rule 3 *(Start-Event) - Let $e \in O_S^E$ be a start-event with $e.trigger = Message$. Let further π be the plan, that was created from the process, in which e is located.*

$$\forall p \in e.message.properties \Longrightarrow \vartheta := (p.name, p.type)$$
$$In_\pi := append(In_\pi, \vartheta)$$

Rule 4 *(End-Event) - Let $e \in O_E^E$ be an end-event with $e.result = Message$. Let further π be the plan, that was created from the process, in which e is located.*

$$\forall p \in e.message.properties \Longrightarrow \vartheta := (p.name, p.type)$$
$$Out_\pi := append(Out_\pi, \vartheta)$$

Start events of the type *Timer* and *Rule* may also be used in another context. Both should result in reactive behaviour, where the former requires that agents are aware of time, and the latter that they can monitor their beliefs w.r.t. certain conditions. Other end event-types refer to control flow, such as the termination or failure of a process, and are not detailed here.

Sub-processes are used to create hierarchical and reusable structures within a BPD. Either they refer to completely independent processes (*independent sub-process*), which may be defined outside the given BPD, or they can be included into a parent process (*embedded sub-process*). The *independent sub-process* is mapped onto a goal, revealing one major strength of multi-agent systems. In this case, the corresponding plan can be provided by any agent, if the target platform supports *Yellow Pages Services*. The *embedded sub-processes* is mapped onto a plan, because it contains an own workflow. Additionally an operation for invoking that plan is created. Note, that the *invoke*- and *addGoal*-operations have to be added into the correct place within the script. This is done during the control flow mapping, and hence they are not added into any script by the given rules:

Rule 5 *(Embedded Sub-Process Activity) - Let $x \in O_{Sub}^A$ be an activity, with $x.subProcessType = Embedded$. Let further be Λ the agent, that was created from the pool, in which x is located.*

$$x \Longrightarrow \pi := (x.name, [], [], [])$$
$$\Pi_\Lambda := \Pi_\Lambda \cup \{\pi\};$$
$$\mathbf{invoke}(\mathbf{x.name})$$

Rule 6 *(Independent Sub-Process Activity) - Let $x \in O_{Sub}^A$ be an activity, with $x.subProcessType = Independent$.*

$$x \Longrightarrow \gamma := (x.processRef.name, [], [])$$
$$\mathbf{addGoal}(\gamma)$$
$$\forall i \in x.inputPropertyMaps \Longrightarrow In_\gamma := append(In_\gamma, eval(i))$$
$$\forall o \in x.outputPropertyMaps \Longrightarrow Out_\gamma := append(Out_\gamma, eval(o))$$

Note that the elements of the property-maps, which are used to pass the data to the goal in Rule 6, are string expressions. Hence we cannot be more specific than interpreting the string, which is done using the *eval* function. Therefore it must conform to the syntax that is used to refer to a data-element in the script language of the target platform.

Communication in BPMN is specified by using *send* or *receive* tasks. These are simply mapped onto asynchronous speechacts. The rule for the send task is given exemplarily. The corresponding rule for the receive task is defined analogously. Only the *taskType* and the created operation is different.

Rule 7 *(Send-Task) - Let $x \in O_{At}^A$ be an activity with $x.taskType = Send$. Let further be μ the message, that should be sent (and is mapped from the message-attribute of x).*

$$x \Longrightarrow \mathbf{send}(\mu)$$

We note, that there are some nodes left that are not covered so far. We will not provide rules for each of them here, but discuss their mapping briefly. First of all, a very helpful BPMN node is the script-task, which may contain arbitrary code in its *script*-attribute. With this, it is easy to define functionality, that cannot be expressed otherwise. The mapping of reference nodes (task or sub-process of type *reference*) depends on the nodes they refer to. Some other elements are not supported yet, for instance the mapping of transactions, which is also an open issue in the mapping to WS-BPEL.

The next subsection describes, how these elements interact with each other by means of modifying and passing data.

4.2 Mapping of Data Flow

BPMN is not intended to model data [12, p. 34], and thus has to rely on other sources. Additionally, since the main purpose of BPMN is the visualisation of the work flow, data handling is not very comfortable. In most cases, it is completely hidden, and can only be specified within non-visible attributes of the nodes.

Data flow is addressed by assigning values to properties that can be attributed to every activity (task or sub-process) in a BPD. Since each relevant aspect of a property (i.e. its type, name or value) has to be given as string, the designer is free to choose any representation, and can simply adopt the specific language of the target model. The general approach for mapping the data handling of activities is given in Figure 2. In addition to the *operation* (which may be for instance an *invoke*), its local variables have to be created (as facts), and values have to be assigned to them. Assignments after the execution of the operation can be used to bind the results to globally accessible facts again for further processing. Since we utilise the belief-base of an agent for storing and accessing data, which in contrast to properties of activities is always globally accessible, the *cleanup* section removes them from the belief-base afterwards.

Some specific elements also possess messages (send- and receive tasks, events of type message), which also provide properties for capturing data. Messages are

```
[define facts]
[assignments]
<operation>
[assignments]
[cleanup facts]
```

Fig. 2. General data handling for activities

mapped onto their counterparts in the agent model, and in the case of the tasks are inserted before the speechact operation into the script, which passes them as parameter (see Rule 7). The assignments again assure that sent and received data is bound to the corresponding facts/variables.

The most specific mapping of data handling is given for goals, that result from *independent sub-processes* (see Rule 6). There, data is passed using the attributes *input-* and *outputPropertyMap*, which are sets of expressions. These map the properties between the two processes, and are used for this purpose in the agents model as well. Note, that the expressions are again represented as ordinary strings, and hence have to be encoded in the target language already.

The next subsection presents some issues concerning the mapping of the control flow.

4.3 Mapping of Control Flow

The mapping of control flow will be applied after the several BPMN elements have been mapped to equivalent elements of the agent description language. The purpose of the mapping of control flow is to arrange this collection of atomic elements into structures such as sequences, if-else blocks or loops. These structures then make up a plan's script. The mapping is quite intuitive to understand and at the same time complicated to realize and highly dependent on the targeted agent language's capabilities.

Basically, the flow objects contained in one pool will map to one plan. Sequence flows determine the order of execution. Gateways define the extend of structured elements, such as conditional blocks, parallel blocks or loops, depending on the gateway's type (AND, XOR, ...) and the number of incoming and outgoing sequence flows. In the case of BPMN also event handlers (intermediate events on an activity's boundary) have to be taken into account, which can complicate the resulting structured workflow by some amount, making it necessary to skip parts of the workflow in case an event handler is triggered.

Well-structured workflows can be mapped in quite a simple rule-based bottom-up approach. A set of rules is used to identify the few basic structures that make up a complex workflow – sequences, blocks, loops, etc. – and combine the target elements that have been mapped from the involved source elements to structures accordingly. After that, either the source model itself or a reference model, connecting the source model with the target model, has to be reduced by replacing

the successfully mapped structure with an atomic marker-node, referencing the new structure in the target model, so that a rule can not be applied twice for the same element. This process is repeated until the model can not be reduced any further.

However, in some cases, when the BPMN workflow is not well-structured, it is hard to map the control flow that is defined within BPMN to any kind of script, because the languages have a different power of expressiveness. Usually, graph-like languages such as BPMN allow to specify control flow, which is generally not reproduceable in block-oriented languages.

Some examples of such are workflows containing an AND-split gateway being followed by a OR-join gateway, resulting in a lack of synchronization and multiple instances of the workflow after the joining gateway, which can not be expressed easily in block-structured languages. Other examples are all kinds of overlapping blocks and loops and interconnected parallel workflows.

Some of these problems can be tackled by duplicating parts of the workflows, by spawning child-processes or by introducing auxiliary variables. Obviously, it is highly favorable if this is done programmatically and the user does not have to consider these workarounds.

These problems have been discussed in a number of papers and will not be reconsidered here. For further information, please refer to [13, 15, 19, 21].

4.4 Mapping-Example

In this section we will illustrate the mapping using the simple business process diagram introduced in Figure 1. In the course of this example we will use the identifiers found on each of the diagram nodes to refer to the nodes.

Besides the visual nodes the BPD also needs some non-visual attributes, such as properties and assignments, which will be given in Table 1. In the following we will use a notation like $prop = (name : type)$ for properties and $assign = (to \leftarrow from, assignTime)$ for assignments. Note also that in Figure 1 $a11.message$ is equal to $a21.message$ and $a12.message$ is equal to $a23.message$.

Element	**Properties**	**Assignments**
p1.process	$req : String, ans : String$	$req \leftarrow$ "$need\_meds\ldots$", $before$
a11.message	$msg\_req : String$	
a11		$msg\_req \leftarrow req, before$
a12		$ans \leftarrow msg\_ans, after$
p2.process	$req : String, ans : String$	
a21		$req \leftarrow msg\_req, after$
a22		$ans \leftarrow$ "$recipe\_for\_meds\ldots$", $after$
a23.message	$msg\_ans : String$	
a23		$msg\_ans \leftarrow ans, before$

Table 1. The example diagram's non-visual attributes

Firstly, both the Patient pool ($p1$) and the Doctor pool ($p2$) are mapped to agents (Rule 1), each holding a plan for the pool's process (Rule 2).

$$p1 \Longrightarrow \Lambda_{p1} := (\text{“}Patient\text{”}, \{\pi_{p1}\}, \emptyset, \emptyset, \emptyset)$$
$$\pi_{p1} := (\text{“}proc\_patient\text{”}, [\,], [\,], [\,])$$

The plan's *In*- and *Out*-lists are empty, since the start- and end-events in this simple example diagram are not of type *Message* and thus do not have a mapping. Also, the activities $a22$, $a31$ and $a32$, which are not further specified (e.g. as script-tasks), will simply be referred to by their *id*s.

The mapping of the first send-task would look like the following (Rule 7):

$$a11.message \Longrightarrow \mu_{req} := (\text{“}msg\_1\text{”}, \text{“}Patient\text{”}, \text{“}Doctor\text{”}, [msg\_req])$$
$$a11.assignments \Longrightarrow \mathbf{ass_{a11}}$$
$$a11 \Longrightarrow \mathbf{send}(\mu_{\mathbf{req}})$$

Since it will be up to the actual agent system how to realise the assignments between the global properties and the message properties we will leave this open and refer to the mapped assignments of activity x as $\mathbf{ass_x}$. The other send- and receive-tasks will map to similar structures and will not be explicitly stated here.

The mapping of the subprocess $a24$ would result in the following (Rule 5):

$$a24 \Longrightarrow \pi_{a24} := (\text{“}accounting\text{”}, [\,], [\,], [\,])$$
$$\Pi_{p1} := \Pi_{p1} \cup \{\pi_{a24}\}$$
$$\mathbf{invoke}(\mathbf{accounting})$$

Finally the control flow has to be mapped, which can be done using a set of rules like those described in Subsection 4.3. The results of the mapping can be found in Table 2: Each pool has been mapped to one agent – Λ_{p1} and Λ_{p2} – holding a plan for the pool's process and its subprocesses.

The properties of the process, i.e. the global variables, can be found in the agent's fact base Θ. The script elements that have been created from the various flow objects have been arranged in sequences and blocks and inserted into the plans' scripts[2]. In the next section we will have a look on other work that has been done in this area so far.

5 Related Work

Considering our longterm-goal, which is to use an intuitive graphical process notation for designing multi-agent systems, few related work exists. On the workflow level, the usage of petri nets and extensions (e.g. CPN) is studied for a

[2] We use the notation $x_1; x_2; x_3$ for sequences of script elements and $x_1 \parallel x_2$ for parallel execution, which are the only control structures needed in this example.

$\Lambda_{p1} = (\text{“}Patient\text{”}, \Pi_{p1}, \emptyset, \emptyset, \Theta_{p1})$
$\Pi_{p1} = \{\pi_{p1}\}$
$\Theta_{p1} = \{req, ans\}$
$\pi_{p1} = (\text{“}proc\_patient\text{”}, [\,], [\,], X_{p1})$
$X_{p1} = [\mathbf{ass_{p1.proc}}; \mathbf{ass_{a11}}; \mathbf{send}(\mu_{\mathbf{req}}); \mathbf{receive}(\mu_{\mathbf{ans}}); \mathbf{ass_{a12}}]$
$\mu_{req} = (\text{“}msg\_1\text{”}, \text{“}Patient\text{”}, \text{“}Doctor\text{”}, [msg\_req])$
$\Lambda_{p2} = (\text{“}Doctor\text{”}, \Pi_{p2}, \emptyset, \emptyset, \Theta_{p2})$
$\Pi_{p2} = \{\pi_{p2}, \pi_{a24}\}$
$\Theta_{p2} = \{req, ans\}$
$\pi_{p2} = (\text{“}proc\_doctor\text{”}, [\,], [\,], X_{p2})$
$X_{p2} = [\ \mathbf{receive}(\mu_{\mathbf{req}}); \mathbf{ass_{a21}}\ ;$
$\quad (\mathbf{a22}; \mathbf{ass_{a22}}; \mathbf{ass_{a23}}; \mathbf{send}(\mu_{\mathbf{ans}})) \parallel \mathbf{invoke}(\mathbf{accounting})\ ]$
$\pi_{a24} = (\text{“}accounting\text{”}, [\,], [\,], [\mathbf{a31}; \mathbf{a32}])$
$\mu_{ans} = (\text{“}msg\_2\text{”}, \text{“}Doctor\text{”}, \text{“}Patient\text{”}, [msg\_ans])$

Table 2. Results of the mapping

while, for example by Aalst et al. [2]. In [18], petri nets are even directly used to model multi-agent systems. Our approach differs from them in that it uses a notation that is more common to business users, and thus increases the accessibility to industry. As shown in [10], a semantical analysis is possible as well by translating business process diagrams into petri nets, but that can be done without any knowledge of the user about petri nets.

Our approach also relates to model driven architectures (MDA) in the general case, and in case for multi-agent systems, as for instance described in [5]. Again, the specification languages used there are usually used by software designers rather than by business users, and therefore are not suited to bridge the disconnect between industry and agent-oriented software design. On the other hand, they provide a set of interesting methodologies, and are more complete than our current work, and are thus a good reference point for further investigations.

As the importance and impact of SOA and related technologies starts to become clear in the area of agent technologies, a number of people have worked on different ways of incorporating knowledge and experiences of the different fields. Therefore we want to mention some work that is at least partially related to what we have presented in this paper. Casella et al. [7] and Mantell [16] have worked on creating tools to translate UML diagrams to BPEL. Mantell translates UML activity diagrams to executable BPEL code, while Casella et al. start from protocol diagrams designed in the UML extension Agent-UML [3] and create abstract BPEL processes. In addition to that, there exists work that incorporates web services into multi-agent systems, which in combination with the previously mentioned approaches would have a similar effect as our work, but is less straightforward. For instance, Bozzo et al. [4] apply the BDI paradigm in order to create adaptive systems based on webservices. Here, they start from a BDI-type multi-agent system (based on AgentSpeak [20]) and extend it to use web services as primitive actions. A similar work was developed by Vidal et

al. [23]. Walton [24] suggests to differentiate between agent interaction protocols and agent body, and therefore to allow web services (bodies) to be seamlessly incorporated into multi-agent system, and vice versa. Finally, for an overview on existing approaches regarding the benefits of combining web services and agency, see [9].

6 Conclusion

In this paper, we have proposed a way to close the gap that exists between the different roles which participate in the development of business applications, by defining a mapping from BPMN to agent concepts. Following this approach, we defined a set of (abstract) rules, showing how business processes can be modelled in terms of BDI-type agents. We also argued that the mapping is only one aspect out of a set of tasks that can be supported by a tool. We additionally proposed the usage of syntactical and semantical verification, combined with a normalisation of BPDs. In future, this may lead to a complete methodology for designing and implementing multi-agent systems. One aspect that is still missing is the data handling, which has to be incorporated in an appropriate manner.

6.1 Implementation

A transformation from BPMN to JIAC IV (Java-based Intelligent Agent Componentware) [22], a multi-agent framework based on the BDI paradigm, has lately been developed in the course of a diploma thesis [14].

For the purpose of this transformation a BPMN editor has been implemented using Eclipse GMF. It can be used to create and validate BPDs and to initiate the transformation. Since there is no XML schema or similar given for BPMN, the editor's domain model had to be created from scratch and thus is not compatible with other BPMN editors. However, it supports each single attribute given in the BPMN specification. The mapping has been implemented as a transformation tool using both a top-down pass through the BPMN model and a rule-based transformation, which can be subdivided in four stages: Normalization, element mapping, structure mapping and clean-up.

Although the mapping from BPMN to JIAC is not yet fully specified and there is still some work to do to support the transformation of unstructured workflows, the basics are working fine and simple BPMN diagrams can be transformed to JIAC multi-agent systems.

6.2 Future Work

After having defined a set of basic rules for a mapping from BPMN to agents there is still much work to be done in the future. First of all, we want to explore the mapping of the work flow as completely as possible. Therefore we will also identify any further requirements that are needed on the agents side in order to capture the desired functionality. To this end, it seems to be necessary to define or use an existing platform independent agent metamodel.

On a higher level, we plan to extend our approach to a complete methodology. The first thing to consider is the integration of data types as a fixed part of the design process. It is planned to combine BPMN with OWL [17] and OWL-S respectively, such that the extension is supported by well-founded concepts. An additional advantage is that this allows to define semantical services in terms of preconditions and effects.

Another interesting research task concerns the mapping into the opposite direction, i.e. from agents to BPMN. Since multi-agent systems are naturally complex, a good visualisation of their workflow, the interaction protocols and organisational structures would increase the understanding of existing systems significantly. Ideally, this would also help to identify in which parts the two models diverge in the power of expressiveness, and what extensions are needed in order to adjust this discrepancy.

References

1. Foundation for Intelligent Physical Agents (FIPA). Specification index, 2007.
2. W. Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
3. B. Bauer, J. P. Müller, and J. Odell. Agent UML: A Formalism for Specifying Multiagent Software Systems. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering, 1st International Workshop, AOSE 2000, Revised Papers*, volume 1957 of *LNCS*, pages 91–104. Springer-Verlag, 2001.
4. L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta. COOWS: Adaptive BDI Agents meet Service-Oriented Computing – extended abstract. In M. P. Gleizes, G. A. Kaminka, A. Nowé, S. Ossowski, K. Tuyls, and K. Verbeeck, editors, *Proceedings of the 3rd European Workshop on Multi-Agent Systems (EUMAS'05)*, pages 473–484. Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten, 2005.
5. A. BRANDO, V. SILVA, and C. LUCENA. A model driven approach to develop multi-agent systems. Technical report, Departmento de Informtica - Pontifcia Universidade Catlica do Rio de Janeiro - PUC-Rio, 2005.
6. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Havard University Press, Cambridge, MA, 1987.
7. G. Casella and V. Mascardi. From AUML to WS-BPEL. Technical Report DISI-TR-06-01, Dipartimento di Informatica e Scienze dell'Informatione, Università di Genova, 2006.
8. O. Committee. Web Services Business Process Execution Language (WS-BPEL) Version 2.0. Technical report, Oasis, 2007.
9. I. Dickinson and M. Wooldridge. Agents are not (just) web services: considering BDI agents and web services. In *Proceedings of the 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005), Utrecht, The Netherlands*, July 2005.
10. H. Endert, B. Hirsch, T. Küster, and S. Albayrak. Towards a Mapping From BPMN to Agents. In J. Huang, R. Kowalczyk, Z. Maamar, D. Martin, I. Müller, S. Stoutenburg, and K. Sycara, editors, *Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE-2007)*, volume 4504. Springer Verlag, May 2007.

11. S. Franklin and A. Graesser. Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer-Verlag.
12. O. M. Group. Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification dtc/06-02-01, OMG, 2006. `http://www.bpmn.org/Documents/OMGFinalAdoptedBPMN1-0Spec06-02-01.pdf`.
13. B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler. On Structured Workflow Modelling. In *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, pages 431–445, London, UK, 2000. Springer-Verlag.
14. T. Küster. Development of a Visual Service Design Tool providing a mapping from BPMN to JIAC. Master's thesis, Technische Universität Berlin, 2007.
15. R. Liu and A. Kumar. An Analysis and Taxonomy of Unstructured Workflows. In W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management*, volume 3649, pages 268–284, 2005.
16. K. Mantell. From UML to BPEL — Model Driven Architecture in a Web Services World. Technical report, IBM, 2005. `http://www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel/`.
17. D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language. W3C Recommendation, 2004. `http://www.w3.org/TR/owl-features/`.
18. D. Moldt and F. Wienberg. Multi-Agent-Systems Based on Coloured Petri Nets. In *ICATPN*, pages 82–101, 1997.
19. C. Ouyang, W. van der Aalst, M. Dumas, and A. ter Hofstede. Translating BPMN to BPEL. Technical Report BPM-06-02, BPMCenter.org, 2006.
20. A. S. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In R. van Hoe, editor, *Agents Breaking Away, 7^{th} European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96*, volume 1038 of *LNCS*, pages 42–55, Eindhoven, The Netherlands, January 1996. Springer Verlag.
21. W. Sadiq and M. E. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2):117–134, 2000.
22. R. Sesseler. *Eine modulare Architektur für dienstbasierte Interaktionen zwischen Agenten*. PhD thesis, Technische Universität Berlin, 2002.
23. J. M. Vidal, P. Buhler, and C. Stahl. Multiagent Systems with Workflows. *IEEE Internet Computing*, 8(1):76–82, January/February 2004.
24. C. Walton. Uniting Agents and Web Services. In *Agentlink News*, volume 18, pages 26–28. AgentLink, 2005.

Roles in Coordination and in Agent Deliberation: A merger of concepts

Guido Boella[1], Valerio Genovese[1], Roberto Grenna[1], and Leendert van der Torre[2]

[1] Dipartimento di Informatica, Università di Torino,
guido@di.unito.it, valerio.click@gmail.com, grenna@di.unito.it
[2] University of Luxembourg, Luxembourg
leendert@vandertorre.com

Abstract. In this paper we generalize and merge two models of roles used in multiagent systems which address complementary aspects: enacting roles and communication among roles in an organization or institution. We do this by proposing a metamodel of roles and specializing the metamodel to fit two existing models. We show how the two approaches can be integrated since they deal with complementary aspects: [1] focuses on roles as a way to specify interactions among agents, and, thus, it emphasizes the public character of roles. [2] focuses instead on how roles are played, and thus it emphasizes the private aspects of roles: how the beliefs and goals of the roles become the beliefs and goals of the agents. The former approach focuses on the dynamics of roles in function of the communication process. The latter approach focuses on the internal dynamics of the agents when they start playing a role or shift the role they are currently playing.

1 Introduction

In the last years, the usefulness of roles in designing agent organizations has been widely acknowledged. Witness not only the papers appeared at AAMAS, IAT, but also the creation of specialized workshops which have agent organizations (COIN, ROLES, AOSE, NorMAS, etc.) among their topics.

Many different models have been designed. Some of them use roles only in the design phases of a MAS [3], while other ones consider roles as first class entities which exist also during the runtime of the system [4]. Some of them focuses on how roles are played by agents [2], other ones on how roles are used in communication among agents in organizations [1].

This heterogeneity of the way roles are defined and used in MAS risks to be a danger for the interoperability of agents in open systems, since each agent entering a MAS can have a radically different notion of role. Thus, the newly entered agents cannot be governed by means of organizations regulating the MAS. Imposing to all agent designers a single notion of role is a strategy that cannot

have success. Rather, it would be helpful to design both multiagent infrastructures that are able to deal with different notions of roles, and to have agents which are able to adapt to open systems which use different notions of roles in organizations. This alternative strategy can be costly if it is not possible to have a general model of role that is compatible, or can be made compatible with other existing concepts.

In this paper we generalize and merge two models of roles used in multiagent systems, in order to promote the interoperability of systems. The research question is: How to combine the model of role enactment by [2] with the model of communication among organizational roles of [1]?

We answer these questions by extending to agents a metamodel of roles developed for object oriented systems [5]. The relevant questions, in this case, are: how to introduce beliefs, goals and other mental attitudes in objects, and how to pass from the method invocation paradigm to the message passing paradigm.

Then we specialize the metamodel to model two existing approaches and we show how they can be integrated in the metamodel since they deal with complementary aspects. We choose to model the proposals of [1] and [2] since they are representative of two main traditions. The first tradition is using roles to model the interaction among agents in organizations, and the second one is about role enactment, i.e., to study how agents have to behave when they play a role.

From one side, organizational models are motivated by the fact that agents playing roles may change, for example a secretary may be replaced by another one if she is ill. Therefore, these models define interaction in terms of roles rather than agents. In [1] roles model the public image that agents build during the interaction with other agents; such image represents the behavior agents are publicly committed to. However, this model leaves unspecified, how given a role, its player will behave. This is a general problem of organizational models which neglect that when, for example, a secretary falls ill, there are usually some problems with ongoing issues (the new secretary does not know precisely the thing to be done, arrangements already made etc.). So having a model of enacting and deacting agents surely leads to some new challenges, which could not be discussed, simulated or formally analyzed without this model.

In contrast, the organizational view focuses on the dynamics of roles in function of the communication process: roles evolve according to the speech acts of the interactants, e.g. the commitment made by a speaker or the commands made by other agents playing roles which are empowered to give orders. In this model roles are modeled as sets of beliefs and goals which are the description of the expected behavior of the agent. Roles are not isolated, but belongs to institutions, where constitutive rules specify how the roles change according to the moves played in the interactions by the agents enacting the roles.

[2] focuses, instead, on how roles are played by an agent , and, thus, on the private aspects of roles. Given a role described in terms of beliefs, goals, and other components, like plans, their model describe how these mental attitudes become the beliefs and goals of the agents. In this model roles are fixed descriptions,

so they do not have a dynamics like in the model of [1]. Moreover, when roles are considered inside organizations new problems for role enactment emerge: for example, how to coordinate with the other agents knowing what they are expected to do in their role, and how to use the powers which are put at disposal of the player of the role in the organization. The same role definition should lead to different behaviors when the role is played in different organizations.

In contrast, it specifies the internal dynamics of the agents when they start playing (or enacting in their terminology) a role or shift the role they are currently playing (called the activated role). So they model *role enacting agents*: agents that know which roles they play, the definitions of those roles, and which autonomously adapt their mental states to play the roles.

Despite the apparent differences, the two approaches are compatible since they both attributes beliefs and goals to roles. So we study by means of the metamodel how they can be combined to have a more comprehensive model of roles.

The paper is structured as follows. In Section 2 we describe the requirements on agents and roles in order to build a metamodel; in Section 3 we formally define the metamodel for roles together with its dynamics; in Section 4 we define the basic notions to model agents that play roles; Section 5 deals with the modeling of enacting agents as in [2]; Section 6 introduces and models roles to deal with coordination in organizations; in Section 7 we merge [2] and [1] into the framework introduced in Section 3; Conclusions end the paper.

2 Agents and roles

Since the aim of this paper is to build a metamodel to promote interoperability, we make minimal assumptions on agents and roles.

The starting point of our proposal is a role metamodel for object orientation. The relation of objects and agents is not clear, and to pass from object to agents we take inspiration from the Jade model [6].

Agents, differently than objects, do not have methods that can be invoked starting from a reference to the object. Rather, they have an identity and they interact via messages. Messages are delivered by the MAS infrastructure, so that agents can be located in different platforms. The messages are modeled via the usual send-receive protocol. We abstract in the metamodel from the details of the communication infrastructure (whether it uses message buffers, etc.).

Agents have beliefs and goals. Goals are modeled as methods which can be executed only by the agent itself when it decides to achieve the goal.

As said above, we propose a very simple model of agents to avoid controversial issues. When we pass to roles, however, controversial issues cannot be avoided.

The requirements to cope with both models of roles we want to integrate are:

- Roles are instances, associated in some way to their players.
- Roles are described (at least) in terms of beliefs and goals.
- Roles change over time.

– Roles belong to institutions, where the interaction among roles is specified.
– The interaction among roles specifies how the state of roles changes over time.

In [1] roles are used to model interaction, so agents exchange messages according to some protocol passing via their roles. This means that the agent have to act on the roles, e.g., to specify which is the move the role has to play in certain moment. Moreover, roles interact with each other.

[2]'s model specifies how the state of the agent changes in function of the beliefs and goals of the roles it plays. However, it does not consider the possibility that the state of the role change and, thus, it ignores how the agent becomes aware of the changes of beliefs and goals of the role.

To combine the two models we have to specify how the interaction between an agent and its role happens when the agent changes the state of the role or the state of the role is changed by some event. A role could be considered as an object, and its player could invoke a method of the role. However, this scenario is not possible, since the roles are strictly related to the institution they belong to, and we cannot assume that the institution and all the agents playing roles in the institution are located on the same agent platform. So method invocation is not possible unless some sophisticated remote method invocation infrastructure is used. Moreover, the role have to communicate with its player when its beliefs and goals are updated. Given that the agent is not an object, the only possibility is that a role sends a message to its player. As a consequence, we decide to model the interaction between the agent and the role by means of messages too.

Finally, we have to model the interaction among roles. Since all roles of an institution belongs to the same agent platform, they do not necessarily have to communicate via messages. To simplify the interaction, we model communication among roles by means of method invocation.

The fact that roles belong to an institution has another consequence. According to the powerJava model of roles in object oriented programming languages, roles, seen as objects, belong to the same namespace of the institution. This means that each role can access the state of the institution and of the sibling roles. This allows to see roles as a way to specify coordination [7].

In a sense, roles are seen both as objects, from the internal point of view of the institution they belong to, and as agents, from the point of view of their players, with beliefs and goals, but not autonomous. Their behavior is simply to:

– Receive the messages of their players.
– Execute the requests of their player of performing the interaction moves according to the protocol allowed by the institution in that role.
– Send a message to their players when the interaction move performed by the role itself or by some other role results in a change of state of the role.

3 A Logical Model for Roles

In [5] the model is structured in three main levels: universal, individual and dynamic; here we decide not to talk about the universal level an concentrate ourself on agents dynamics. We define the formalism of the framework in a way as much general as possible, this gives us an unconstrained model where special constraints are added later.

3.1 Individual level

The *individual* level includes in this paper some elements of the universal one and the elements of this level are individuals (or instances) of the types defined at the universal level. This level is composed by a *snapshot model* that describes in a particular moment the relationships between individual players contexts and roles, and a dynamic model which links snapshots and actions modeling how the system changes when an action is executed. In the formalization of the model we use *objects* as basic elements upon which the model is based; we refer to Section 4 for a complete discussion that underlines how the model can be used to grasp roles dynamics in MAS.

Definition 1 A *snapshot model* is a tuple

$$< \mathsf{O}, \mathsf{R\_types}, \mathsf{I\_contexts}, \mathsf{I\_players}, \mathsf{I\_roles}, \mathsf{Val}, \mathsf{I\_contraints}$$
$$\mathsf{I_{Roles}}, \mathsf{I\_Attributes}, \mathsf{I\_Operations}, \mathsf{I_{Attr}} >$$

where:

– O is a *domain* of objects, for each object o is possible to refer to its attributes and operations through $\pi_{I\_Attr}(o)$ and $\pi_{I\_Op}(o)$, respectively.
– R_types is a set of types of roles.
– I_contexts $\subseteq$ O is a set of institutions.
– I_players $\subseteq$ O is a set of actors.
– I_roles $\subset$ O is a set of *roles instances*.
– I_Attributes is the set of attributes.
– I_Operations is the set of operations.
– Val is a set of *values*.
– I_constraints is a set of integrity rules that constraint elements in the snapshot.

We usually refer elements in I_contexts, I_players and I_roles respectively, *institutions*, *actors* and *roles_instances*.
The snapshot model has also a few functions and relations:

– I_{Roles} is a *role assignment function* that assigns to each role R a relation on I_context x I_players x I_roles.
– I_{Attr} is an *assignment function* which it takes as arguments an object $d \in$ O, and an attribute p $\in \pi_{I\_Attr}(d)$, if p has a value $v \in$ Val it returns it, $\emptyset$ otherwise.

– I_AS ⊆ O x I_Attributes: is an attribute assignment relationship, through which we define what are the attributes assigned to an object in the defined snapshot.
– I_OS ⊆ O x I_Operations: is an operations assignment relationship, through which we define what are the operations assigned to an object in the defined snapshot.
– I_OT ⊆ O x R_types: is an type assignment relationship, through which we define the type of every role instance in the snapshot.
– I_PL ⊆ I_players x R_types: this relation states, which are the players that can play a certain role types.
– I_RO ⊆ I_roles x I_contexts: each role is linked with one or (potentially more) context.

Generally, when a role instance x is an individual of the type D, we write $x :: D$. If $a \in \pi_{I\_Attr}(x)$ we write x.a ∈ I_Attributes as the attribute instance assigned to object x, the same holds for elements in I_Operations.

The role assignment function I_{Roles} gives us the notion of an actor who plays a role within a specific context: if i is an institution, a an actor, and $o :: R$ a role, $(i,a,o) \in I_{Roles}(R)$ is to be read as: "the object o represents agent a playing the role R in institution i". We will often write $R(i,a,o)$ for this statement, and we call o the *role instance*.

Suppose we have a role instance employee, and that the value of the attribute salary is 1000 ∈ usually, instead of writing $I_{Attr}(\mathsf{employee}, \mathsf{salary}) = 1000$, we write

$$\mathsf{salary}(\mathsf{employee}) = 1000$$

3.2 The dynamic model

The dynamic model relies on the individual level and defines a structure to properly describe how the framework evolves as a consequence of executing an action on a snapshot. In Section 4 and 5, we describe how this model constraints agents' dynamics.

Definition 2 A *dynamic model* is a tuple

$$< \mathsf{S}, \mathsf{TM}, \mathsf{Actions}, \mathsf{Requirements}, \mathsf{D\_constraints}, \mathsf{I}_{\mathsf{Actions}}, \mathsf{I}_{\mathsf{Roles_t}}\ \pi_{\mathsf{Req}}, \mathsf{I}_{\mathsf{Requirements_t}} >$$

where:

– S is a set of *snapshots*.
– TM ⊆ S x ℕ: it is a time assignment relationship, such that each snapshot has an associated unique time t. For the sake of simplicity we define a discrete time through positive natural numbers.
– Actions is a set of actions.
– Requirements is a set of requirements for playing roles in the dynamic model.
– D_contraints is a set of integrity rules that constraints the dynamic model.

– $\mathsf{I}_{\mathsf{Actions}}$ maps each action from Actions to a relation on a set of snapshots P. $I_{Actions}(s,a,t)$ tells us which snapshots are the result of executing action a at time t from a certain snapshop. [1] This function returns a couple in TM that binds the resulting snapshot with time $t+1$. In general, to express that at time t is carried action a we write a_t.
– About I_{Roles_t} we say that $R_t(i,a,o)$ is true if there exists, at a time t, the *role instance* $R(i,a,o)$.
– $\pi_{Req}(t,R)$ returns a subset of Requirements present at a given time t for the role of type R, which are the requirements that must be fulfilled in order to play roles of type R.
– $I_{Requirements_t}$ is a function that, given (i,a,R,t) returns True if the actor a fills the requirement in $\pi_{Req}(t,R)$ to play the role R in the institution i, False otherwise. We often write $Req_t(i,a,R)$.

Intuitively, the snapshots in S represent the state of a system at a certain time. Looking at $\mathsf{I}_{\mathsf{Actions}}$ is possible to identify the *course* of actions as an ordered sequence of actions such that:

$$a_1; b_2; c_3$$

represents a system that evolves due to the execution of a, b and c at consecutive times. We refer to a particular snapshot using the time t as a reference, so that for instance $\pi_{I\_Attr_t}$ refers to π_{I_Attr} in the snapshot associated with t in TM.

Actions are described using dynamic modal logic [8], in paricular they are modelled through *precondition laws* and *action laws* of the following form:

$$\Box(A \wedge B \wedge C \supset \langle d \rangle \top) \quad (1)$$

$$\Box(A' \wedge B' \wedge C' \supset [d]E) \quad (2)$$

Where the □ operator express that the quantified formulas hold in all the possible words. *Precondition law* (1) specifies the conditions A,B and C that make an atomic action d executable in a state. (2) is an *action law*[2] which states that if preconditions A',B' and C' to action d holds, after the execution of d also E holds.

In addition we introduce *complex actions* which specify complex behaviors by means of *procedure definitions*, built upon other actions. Formally a complex action has the following form:

$$\langle p_0 \rangle \varphi \subset \langle p_1; p_2 ....; p_m \rangle \varphi$$

p_0 is a *procedure name*, ";" is the *sequencing operator* of dynamic logic, and the p_i's, $i \in [1,m]$, are procedure names, atomic actions, or test actions[3].

[1] Notice that given an action, we can have several snapshots because we model actions with modal logic in which, from a world it is possible to go to more than one other possible world. This property is often formalized through the *accessibility relationship*. Thus, each snapshot can be seen as a possible world in modal logic.
[2] Sometimes action laws are called *effect rules* because E can be considered the effect of the execution of d.
[3] Test actions are of the form $\langle \psi? \rangle \varphi \equiv \psi \wedge \varphi$.

Now we show some examples of actions that can be introduced in the dynamic model in order to specialize the model.

Role addition and deletion

For role addition and deletion actions we use, respectively $R, i \hookrightarrow_t \ a$, and $R, i \hookleftarrow_t \ a$. Then using the notation of dynamic logic introduced above, we write:

$$\Box(Req_t(i, a, R) \supset \langle R, i \hookrightarrow_t a\rangle\top)$$

to express that, if actor a fills the requirements at time t ($Req_t(i, a, R)$ is True), a can execute the role addition action that let him play role of type R.

The above definition gives us the possibility to model that a role assignment introduces a role instance:

$$\Box(\top \supset [R, i \hookrightarrow_t a]\exists x R_{t+1}(i, a, x))$$

or the fact that if a does not already play the role R within institution i, then the role assignment introduces exactly one role instance:

$$\Box(\neg\exists x R(i, a, x) \supset [R, i \hookrightarrow_t a]\exists! x R_{t+1}(i, a, x))$$

Methods

There are other actions through which is possible to change the model as well, for instance agents may assign new values to their attributes [5]. Again, the effects of such changes may depend on choices made earlier (e.g. in the case of delegation, changing the attribute value of an object may change the value of that attribute also in some roles he plays).

Here, we will focus on the case in which the attribute's values can be changed by the *objects themselves*. What we will do is to define *methods* of objects with which they can change attributes of their own or those of others. Actually, to simplify the model, we define one single primitive action: $\mathsf{set_t(o_1, o_2, attr, v)}$, which means that object o_1 sets the value of attr on object o_2 to v at time t. If $\mathsf{o_1}$ and $\mathsf{o_2}$ are autonomous agents, the $\mathsf{set(o_1, o_2, attr, v)}$ can be executed only when $\mathsf{o_1 = o_2}$.

Now, we will of course have that:

$$\Box(\top \supset [\mathsf{set_t(o_1, o_2, attr, v)}]\mathsf{attr}_{t+1}(\mathsf{o_2}) = \mathsf{v})$$

which means that in any state, after the execution of set, if the action of setting this attribute succeeds, then the relevant object will indeed have this value for that attribute.

Operations

Elements of our framework come with *operations* that can be executed at the individual level in order to change the model dynamically, the semantics of each operations can be given exploiting the actions defined for the dynamic model. Suppose, for instance, to have an object individual $x :: Person$ with $\mathsf{x.mail\_address}$ attribute, and an operation $\mathsf{x.change\_mail}$ that changes the value of $\mathsf{x.mail\_address}$ to its argument. Using the set primitive is possible to define how the model evolves after the execution of $\mathsf{x.change\_mail}$ operation trough the following axiom:

$$[\mathsf{x.change\_mail_t(s)}]\varphi \equiv [\mathsf{set_t(x, x, mail\_adress, s)}]\varphi$$

Where $\mathsf{x.change\_mail_t(s)}$ identifies the action carried by x at time t to execute the instance operation $\mathsf{x.change\_mail}$; objects can execute only operations that are assigned to them by $\mathsf{I\_OS}$ relation. In Section 5 we define exec of certain operations as complex actions because we have to describe a more complex semantics.

4 Roles in Multiagent Systems

Since here we have been talking about *objects* as cornerstone of the individual level, now in order to switch from objects to agents, it must be underlined that an *object* of the meta-model does not necessarily overlap with object in OO programming. We used the terms *object* to refer to individuals, and terms like *attribute* and *operation* to talk about state and bheavior of an entity.

In order to be as much general as possible, we define elements of the meta-model with only those features that are essential to talk about roles and leave the possibility to specify the abstract model depending on which account of role we want to grasp. This approach gives us the possibility to talk about object and agent using the same framework, and specifing each time which are the charateristics of role's player. In moving from objects to agents we need to state the following:

- *Attributes* are complex properties of the agent which describe its internal features as well as its mental attitudes (belief, goals, plans etc.).
- *Operations* are actions that the agent does in the system.
- *Agents* at individual level are supposed to be *autonomous* so they cannot be forced to execute an action from an external entity.
- The only way to interact between agents is through *message passing*.
- The *system* in which agents interacts is represented by a unique *institution*.
- *Role instances* are linked with one and only one *system*. In order to express this point we add into $\mathsf{I\_Constraint}$ of every snapshot the following integrity rule:

$$r \in \mathsf{I\_roles} \leftrightarrow \exists! c \in \mathsf{I\_contexts} :< r, c >\in \mathsf{I\_RO}$$

For the sake of generality, we prefer not to specify how agents reason on the basis of their mental attitudes; what we want to model is how mental attitudes evolve as a consequence of playing a role and what are the elements on which the agent have to carry out its resoning process.

It is important to understand that the meta-model is not a framework for agent specification, the elements listed above are the basic features that we think are foundamental to talk about role in MAS, but of course they are not sufficient to utterly model agents.

5 Enact and Deact Roles

In [2], the problem of formally defining the dynamics of roles, is tackled identifing the actions that can be done in a *open system* such that agents can enter and leave. In this setting roles have existence outside the agents in the implemented system, so "agents are not completely defined by the roles they play"[2]. This view leads to a definition of roles that sees them as strictly linked with a system (context), instantiable and with their own proper identity.

In [2] four operations to deal with role dynamics are defined: *enact* and *deact*, which mean that an agent starts and finishes to occupy (play) a role in a system, and *activate* and *deactivate*, which means that an agent starts executing actions (operations) belonging to the role and suspends the execution of the actions.

Although is possible to have an agent with multiple roles enacted simultaneously, only one role can be *active* at the same time.

Before diving into modeling the four basic operations to deal with roles, we need to match with our framework a few concept defined in [2], following we report a list of elements together with their definition and then how they fit in our meta-model:

- *Multiagent system*: In [2] roles are taken into account at the implementation level of *open MAS*, they belong to the system which can be entered or left by agents dynamically. In our framework is possible to view a system as a $\mathsf{context}$ to which are linked all roles that can be played by the agents.
- *Agent role*: A role is a tuple $\langle \sigma, \gamma, \omega \rangle$. Where σ are beliefs, γ goals and ω rules representing conditional norms and obligations. This definition specifies a role "in terms of the information that becomes available to agents when they enact the role, the objectives or responabilities that the enacting agent should achieve or satisfy, and normative rules which can for example be used to handle these objectives" [2]. With this view we define, for *roles* of our framework, a set of complex attributes $\{\mathsf{beliefs}, \mathsf{goals}, \mathsf{plans}, \mathsf{rules}\} \in \mathsf{I\_Attr}$ toghether with the *operations* that represent actions that an agent can carry out when it *activates* the roles instance chosing it from the set of roles it is playing.
- *Agent type*: We consider an agent type "as a set of agent roles with certain constraints and assume that an agent of a certain type decides itself to enact or deact a role". To talk about agent types we use *classes* introduced in the framework as a specification of agent instances at the individual level, with this in mind we use the PL relationship to link *agent classes* to *agent roles* (role's classes) so that the set of roles that an agent can enact (play), is constrainted by $\mathsf{I\_PL}$.
- *Role enacting agent*: "We assume that role enacting agents have their own mental attitudes consisting of beliefs, goals, plans, and rules that may specify their conditional mental attitudes as well as how to modify their mental attitudes. Therefore, role enacting agents have distinct objectives and rules associated to the active role it is enacting, and sets of distinct objectives and rules adopted from enacted but inactive roles". In our framework we define a *role enacting agent* as a instance x having a set of attributes A that represent the internal structures used to deliberate.

  $$A = \{\mathsf{beliefs_a}, \mathsf{objectives_a}, \mathsf{plans_a}, \mathsf{rules_a}, \mathsf{enacted\_roles}[], \mathsf{active\_role}\} \in \pi_{I\_Attr}(x)$$

  The $\mathsf{enacted\_roles}$ attribute is a role ordered record where each entry with index i corresponds to a triple $\langle \sigma_i, \gamma_i, \omega_i \rangle$ which represents the set of beliefs, objectives, plans and rules associated to roles instance i enacted by x.

As introduced above, the model in [2] identifies four operations to deal with role dynamics, in order to to grasp the foundamental ideas proposed in the cited paper, we redefine the *enact*, *deact*, *activate* and *deactivate* operations respecting their original meaning. Given a role enacting agent x, a role instance $i :: R$ played by x in context c such that,

$$\begin{gathered}\{\mathsf{beliefs_r}, \mathsf{objectives_r}, \mathsf{plans_r}, \mathsf{rules_r}\} \in \pi_{I\_Attr}(i) \\ \{\mathsf{beliefs_a}, \mathsf{objectives_a}, \mathsf{plans_a}, \mathsf{rules_a}, \mathsf{enacted\_roles}[], \mathsf{active\_role}\} \in_{SA} \pi_{I\_Attr}(x) \\ \{\mathsf{enact}, \mathsf{deact}, \mathsf{activate}, \mathsf{deactivate}\} \in \pi_{I\_Op}(x)\end{gathered}$$

Next we report the semantics of each operation exploiting the set primitive:

$$\begin{aligned}\langle \mathsf{x.enact_t(i)} \rangle \varphi \subset \langle \mathsf{R}, \mathsf{s} \hookrightarrow \mathsf{x}; \mathsf{set_t(x, x, beliefs_a, beliefs_a} \cup \mathsf{beliefs_r}); \\ \mathsf{set_t(x, x, enacted\_roles[i], <\ objectives_r, plans_r, rules_r\ >)} \rangle \varphi\end{aligned} \tag{3}$$

$$\langle \mathsf{x.deact_t(i)} \rangle \varphi \subset \langle \mathsf{R}, \mathsf{s} \hookleftarrow \mathsf{x}; \mathsf{set_t(x, x, enacted\_roles[i], null)} \rangle \varphi \tag{4}$$

$$\langle \mathsf{x.activate_t(i)} \rangle \varphi \subset \langle \mathsf{set_t(x, x, active\_role, enacted\_roles[i])} \rangle \varphi \tag{5}$$

$$\langle \mathsf{x.deactivate_t(i)} \rangle \varphi \subset \langle \mathsf{set_t(x, x, active\_role, null)} \rangle \varphi \tag{6}$$

In order to be coherent it must be respected a logical order in the execution of these operations, as in [2]:

- each operation $\mathsf{deact(i)}$ is preceded by a $\mathsf{enact(i)}$.
- each operation $\mathsf{deactivate(i)}$ is preceded by only one instruction $\mathsf{activate(i)}$ that is not preceded by another $\mathsf{activate(j)}$.

6 The public dimension of roles

In [9] roles are introduced inside institutions to model the interaction among agents. In [1] the model is specifically used to provide a semantics for agent communication languages in terms of public mental attitudes attributed to roles.

The basic ideas of the model are:

- Roles are instances with associated beliefs and goals attributed to them. These mental attitudes are public.
- The public beliefs and goals attributed to roles are changed by speech acts executed either by the role or by other roles. The former case accounts for the addition of preconditions and of the intention to achieve the rational effect of a speech act, the latter one for the case of commands or other speech acts presupposing a hierarchy of authority among roles.
- The agents execute speech acts via their roles.

This model has been applied to provide a semantics to both FIPA and Social Commitment approaches to agent communication languages [1]. This semantics overcomes the problem of the unverifiability of private mental attitudes of agents.

- In order to maintain the model simple enough, we model message passing extending the dynamic model with two actions (methods) send(x,y,sp) and receive(y,x,sp). Where send(x,y,sp) should be read as the action carried by x of sending a speech act (sp) to y and receive(y,x,sp) is the complementary action of y receiving the message from x. It must be underlined that arguments x and y can be agents or roles.
- A role only listens for the messages sent by the agents playing it:

  $\langle \mathsf{listen(r)} \rangle \varphi \subset \langle \mathsf{P}; \mathsf{played\_by(r,x)?}; \mathsf{receive(r,x,sp)}; \mathsf{D} \rangle \varphi$

  These rules define a *pattern* of protocol where P and D have to be read as possible other actions that can be executed before and after the receive.
- The reception of a message from the agent has the effect of changing the state of other roles. For example, a command given via a role amounts to the creation of a goal on the receiver if the sender has authority (within the system) over it.

  $\Box(authority_{sys}(r, \mathsf{request} \supset [\mathsf{receive(r,x,request(r,r',act))}]\mathbf{G}^{\mathsf{r}'}_{\mathsf{t}}(\mathsf{act}))$[4]

- To produce a speech act, the agent has to send a message to the role specifying the illocutive force, the receiver and the content of the speech act:

  $\langle \mathsf{communicate(a)} \rangle \varphi \subset \langle \mathsf{P}; \mathsf{send(x,r,sp)}; \mathsf{D} \rangle \varphi$

[4] request(r, r′, act) is a speech act that has to be read as following: role r asks to r′'s player to do act.

authority$_{sys}$(r, request) expresses that role r has the authority to make a request within system sys.

7 The combined model

The two models presented above model complementary aspects of roles: the public character of roles in communication and how agents privately adapt their mental attitudes to the roles they play.

In this section we try to merge the two approaches using the metamodel we presented. On the one hand, the model of [1] is extended from the public side to the private side, by using [2] as a model of role enacting. In this way, the expectations described by the roles resulting from the interaction among agents can become a behavior of agents and they do not remain only a description.

On the other hand, the model of [2] is made more dynamic. In the original model the role is given as a fixed structure. The goals of agent can evolve according to the goal generation rules contained in it, but the beliefs and goals described by the role cannot change. This is unrealistic, since during the activity of the agent enacting its role, it is possible that further information are put at disposal of the role and that new responsibilities are assigned, etc.

This problem can be solved by the merging with the model of [1] and by the addition of a further element, which is anyway necessary in [1]'s model.

First of all, in [2] roles cannot change since they are not related to a more extensive context. Instead, in [1], roles belong to institutions together with other roles. Sibling roles and the institution they belong to are the sources of changes for the role. Second, in [1], the changes of roles are described by the effects of the speech acts which can be performed via roles. These two elements can be added to [2]'s model without apparent contradictions.

The missing element is that both models do not consider the problem of how the player of a role become aware of the changes in the state of the played role as a consequence of the actions of other roles. Furthermore, in [2] a role is given as known by the agent playing the role. This is not a realistic assumption, in particular, when the state of the role changes over time, but also the way an agent comes to know the initial state of the role must be explicitly modeled. Otherwise, all roles instances must be assumed to be publicly known in advance.

In order to merge the two models within the same framework, we need to add (complex) actions which are able to grasp the dynamics introduced in [1] and [2]. Interactions among agents is done through message passing and, in particular, through actions send and receive introduced in section 6. Next we are going to introduce all the speech-acts and complex actions which are needed to grasp the combined model and then we introduce a running example to clarify their use defining a *course* of actions in the dynamic model defined in section 3.2.

An agent who wants to play a role within an *open system* has to ask to the system for a role instance; this process is handled by two speech act: ask_to_play(R) and accept_to_play(r,A), where the first one is sent from the agent to the system in order to ask to play a role of type R, whereas the second is sent from the system to the agent, together with the identifier of the role instance r and a set A of other role instances present in the system, in order to inform the agent with

which roles is possible to interact. Next we report the two *effect rules* associated:

$$\Box(\top \supset [\mathsf{receive}(s,x,\mathsf{ask\_to\_play}(R);\mathsf{send}(s,x,\mathsf{accept\_to\_play}(r,A)]\ \mathsf{played\_by}_{\mathsf{sys}}(r,x,s) \quad (7)$$

$$\Box(\top \supset [\mathsf{send}(x,s,\mathsf{ask\_to\_play}(R);\mathsf{receive}(x,s,\mathsf{accept\_to\_play}(r,A)]\ \mathsf{played\_by}_{\mathsf{ag}}(r,x,s)) \quad (8)$$

Where s is the system, x the agent, and r a role instance of type R. In this section we use x,y,z... to denote agents, s for the system and r, r', r''... for role instances. Notice that $\mathsf{played\_by}_{\mathsf{sys}}(r,x,s)$ and $\mathsf{played\_by}_{\mathsf{ag}}(r,x,s)$ refer to two different infrastructures; in Rule 7 is the system that, after having acknowledged the agent request, knows that x is going to play r, whereas in Rule 8 is the agent that becomes aware of the play relation between x and r. To link the two predicates with the logical model introduced in Section 3 we have that:

$$\mathsf{played\_by}_{\mathsf{sys}}(r,x,s) \wedge \mathsf{played\_by}_{\mathsf{ag}}(r,x,s) \rightarrow R(s,x,r)$$

When we are dealing with a single system we can omit s writing $\mathsf{played\_by}_{\mathsf{sys}}(r,x)$ and $\mathsf{played\_by}_{\mathsf{ag}}(r,x)$.

To enact a role, an agent, provided the identifier of the role instance it wants to enact, has to send a message to the role and to wait till the role replies with the information about the state of the role: its beliefs, goal, plans, etc. When the state is received, the agent can enact the role in the same way described by Rule 3 in Section 5. In order to model such interaction we introduce two complex actions tell_enact, accept_enact and two speech acts accept_enact and inform_enact. Following the specification of the complex actions:

$$\langle \mathsf{tell\_enact}(x,r)\rangle\varphi \subset \langle \mathsf{played\_by}_{\mathsf{ag}}(r,x)?;(\mathsf{send}(a1,r1,\mathsf{enact}(x,r))\rangle\varphi \quad (9)$$

$$\langle \mathsf{accept\_enactment}(r,x)\rangle\varphi \subset \langle \mathsf{receive}(r,x,\mathsf{enact}(x,r));\mathsf{played\_by}_{\mathsf{sys}}(r,x)?;\ \mathsf{send}(r,x,\mathsf{inform\_enact}(<\mathsf{beliefs}_r,\mathsf{objectives}_r,\mathsf{plans}_r,\mathsf{rules}_r>))\rangle\varphi \quad (10)$$

When the agent receives the specification of the role he wishes to enact, it can internalize them as in Rule 3:

$$\Box(\top \supset [\mathsf{receive}(x,r,\mathsf{inform\_enact}(<\mathsf{beliefs}_r,\mathsf{objectives}_r,\mathsf{plans}_r,\mathsf{rules}_r>))]\ \mathbf{B}^x(\mathsf{beliefs}_r) \wedge x.\mathsf{enacted\_roles}[r] =<\mathsf{objectives}_r,\mathsf{plans}_r,\mathsf{rules}_r>)^{5} \quad (11)$$

In this combined view is possible that role's specifications change dynamically, in that case it is up to the role to send a message to its player each time its state is updated:

$$\langle \mathsf{udpate\_state}(r,x)\rangle\varphi \subset \langle \mathsf{played\_by}_{\mathsf{sys}}(r,x)?;(\neg\mathbf{G}^r_t(q) \wedge \mathbf{G}^r_{t+1}(q))?;\ \mathsf{send}(r,x,\mathsf{inform\_goal}(q))\rangle\varphi \quad (12)$$

Last but not least, we need to model the deactment of a role respecting the formalization as in Rule 4, therefore we introduce two speech acts deact, ok_deact

and a complex action confirm_deact defined as follows:

$$\langle \mathsf{confirm\_deact}(r,x)\rangle\varphi \subset \langle \mathsf{receive}(r,x,\mathsf{deact});\mathsf{played\_by}_{\mathsf{sys}}(r,x)?;\ \mathsf{send}(r,x,\mathsf{ok\_deact})\rangle\varphi \quad (13)$$

After sending the ok_deact, the system will not consider anymore agent x as player of r:

$$\Box(\top \supset [\mathsf{confirm\_deact}(r,x)]\neg\mathsf{played\_by}_{\mathsf{sys}}(r,x) \quad (14)$$

If it is possible for the agent to deact the role, it will receive an ok_deact from its role:

$$\Box(\top \supset [\mathsf{receive}(x,r,\mathsf{ok\_deact})]x.\mathsf{enacted\_roles}[r] = \mathsf{null} \wedge \neg\mathsf{played\_by}_{\mathsf{ag}}(r,x)) \quad (15)$$

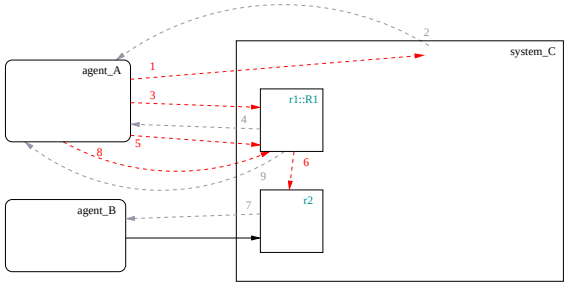


Fig. 1. Roles in MAS

Fig. 1 depicts two agents which interact through roles in an open system. At time t the system has already agent_B that enacts role r2 as represented by the black arrow which goes from agent_B to r2. The system evolves as following:

- At time t+1 agent_A asks to institution system_C to play a role of type R1:

$$\mathsf{send}_{t+1}(\mathsf{agent\_A},\mathsf{system\_C},\mathsf{ask\_to\_play}(R1))$$

- At time t+2 system_C replies to agent_A assigning to him the role instance r1:

$$\mathsf{send}_{t+2}(\mathsf{system\_C},\mathsf{agent\_A},\mathsf{accept\_to\_play}(r1,\{r2\}))$$

- At time t+3 agent_A wants to enact (internalize) role r1:

$$\mathsf{tell\_enact}_{t+3}(\mathsf{agent\_A},r1)$$

– At time t+4 role r1 receives the speech act from agent_A asking for enactment and accepts it, replying to agent_A with its specifications:

$$\mathsf{accept\_enactment}_{t+4}(\mathsf{r1}, \mathsf{agent\_A})$$

– Once that agent_A has enacted the role as in Rule 3 it decides, at time t+5, to activate it [6] and then to ask to the agent playing r2 to do an action act. In other words:

$$\mathsf{send}_{t+5}(\mathsf{agent\_A}, \mathsf{r1}, \mathsf{request}(\mathsf{r1}, \mathsf{r2}, \mathsf{act}))$$

When r1 receives a send from agent_A asking for an act of r2, first it checks if the sender has the authority in the system to ask such an act, if so r2 acquires the goal to do act:

$$\Box(\mathit{authority}_{\mathsf{sys}}(\mathsf{r}', \mathsf{act}) \supset [\mathsf{receive}(\mathsf{r}, \mathsf{agent\_A}, \mathsf{request}(\mathsf{r}, \mathsf{r}', \mathsf{act}))]\mathbf{G}^{\mathsf{r}'}(\mathsf{act}))$$

Is important to underline that because role internals are public to other roles in the same system, it is always possible for r1 to check or modify r2's goals. So, at time t+6 we have:

$$\mathsf{receive}_{t+6}(\mathsf{r1}, \mathsf{agent\_A}, \mathsf{request}(\mathsf{r1}, \mathsf{r2}, \mathsf{act}))$$

– Now that r2 has updated its internal state (i.e. its goals) it must inform its player agent_B:

$$\mathsf{update\_state}_{t+7}(\mathsf{r2}, \mathsf{agent\_B})$$

Where update_state is modelled as in Rule 12

– At time t+8 agent_A decides to deact the role r1:

$$\mathsf{send}_{t+8}(\mathsf{agent\_A}, \mathsf{r1}, \mathsf{deact})$$

– Finally, at time t+9, r1 confirm the deact:

$$\mathsf{confirm\_deact}_{t+9}(\mathsf{r1}, \mathsf{agent\_A})$$

8 Conclusions and Further Works

In this article we merged two represetative role's models in MAS by introducing a metamodel taken from [5] and adapting it to agents. In particular, we added representations of typical agents' mental attitudes and a framework to deal with message passing. The model has been specialized in order to describe both public and private dimensions of roles [1,2]. Finally, we merged the two dimensions defining a group of actions together with their semantics and we modelled a running example to show a possible course of events.

Further works point in two main directions: adapting the proposed metamodel to other roles approaches like [10], and introducing a formal proof theory of roles' actions dynamics and related apects starting from [8].

[6] Activating a role means to take into account its specification during the private agent deliberation process, so there is no need to introduce a public action in the dynamic model to represent the activation of a role.

References

1. Boella, G., Damiano, R., Hulstijn, J., van der Torre, L.: ACL semantics between social commitments and mental attitudes. In: International Workshops on Agent Communication, AC 2005 and AC 2006. Volume 3859 of LNAI. Springer, Berlin (2006) 30–44
2. Dastani, M., van Riemsdijk, B., Hulstijn, J., Dignum, F., Meyer, J.J.: Enacting and deacting roles in agent programming. In: Procs. of AOSE'04, New York (2004) 189–204
3. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. IEEE Transactions of Software Engineering and Methodology **12(3)** (2003) 317–370
4. Colman, A., Han, J.: Roles, players and adaptable organizations. Applied Ontology (2007)
5. Valerio Genovese: Towards a general framework for modelling roles. In Guido Boella, Leon van der Torre, Harko Verhagen, eds.: Normative Multi-agent Systems. Number 07122 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
6. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA-compliant agent framework. Software - Practice And Experience **31(2)** (2001) 103–128
7. Baldoni, M., Boella, G., van der Torre, L.: Roles as a coordination construct: Introducing powerJava. Electronic Notes in Theoretical Computer Science (ENTCS) Procs. of the First International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord 2005) **150** (2006) 9–29
8. M. Baldoni, C. Baroglio, A. Martelli, V. Patti: Reasoning about interaction protocols for customizing web service selection and composition. Journal of LOgic and Algebraic Programming, special issue on Web Services and Formal Methods,70(1):53-73 (2007)
9. Boella, G., van der Torre, L.: The ontological properties of social roles in multi-agent systems: Definitional dependence, powers and roles playing roles. Artificial Intelligence and Law Journal (AILaw) (2007)
10. Omicini, A., Ricci, A., Viroli, M.: An algebraic approach for modelling organisation, roles and contexts in MAS. Applicable Algebra in Engineering, Communication and Computing **16** (2005) 151–178

MAgtALO: Using Agents, Arguments, and the Web to Explore Complex Debates

Simon Wells and Chris Reed

School of Computing, University of Dundee,
swells@computing.dundee.ac.uk
chris@computing.dundee.ac.uk

Abstract. This paper introduces the MAgtALO system, a prototype environment for online debate that aims to provide a mechanism for supporting naturalistic dialogue. MAgtALO demonstrates how dialogue protocols can be harnessed to achieve two objectives: first, to support flexible intuitive interaction with data in complex, contentious domains in order to facilitate understanding and assimilation; and second, to provide mechanisms for structured knowledge elicitation that allow the resources in those domains to be expanded.

1 Introduction

Online argumentation systems are designed to support humans in arguing on specific topics. Over the past two decades there have been an enormous number of software systems developed to support such online argumentation. Many of these systems have remained within the confines of the academic laboratory, but some larger-scale projects have been deployed in the wild [6], [1].

Possibly spurred by the high-visibility arguments expressed in the *Iraq Study Group Report* and *The Stern Review on the Economics of Climate Change* which feature strong explicit argumentative structure, the online community has expressed interest in arguments and the processes through which they are developed. For example, two recent systems which stem from the online community demonstrate the growing appetite for argumentation, *convinceme.net* and *debatepedia.com*.

Convinceme.net, illustrated below, utilises paired message boards, one supporting and one attacking the topic of debate, and provides a Web 2.0 based environment for the construction and exploration of arguments so constructed. Both active participants within a debate and spectators can vote on specific posts and the relative positions of posts is determined based upon (i) the number of votes garned in open debate, (ii) the total number of votes cast in the head-to-head arguments, and (iii) a fixed number of points awarded to the arguments which become "*King of the Hill*" by attracting more votes that any other in a single debate.

Debatepedia.com, illustrated below, uses a wikipedia style interface as a tool for exploring complex topical debates. Structure and rules are imposed on contributors to encourage the construction of *logic trees* which are used to break a thesis down into a set of subquestions which can attract a variety of evidence pro and con. The aims of each tool are different, convinceme.net is developed as a source of competitive entertainment whereas debatepedia.com aims to facillitate the public understanding of complex domains.

Academic-oriented systems are generally built upon a sound argumentation theoretic foundation which provides a rich set of argumentative moves and structures. The presentation and framing of these systems however can be a barrier to wider public adoption. In contrast the online systems enjoy a broad user base but little foundation in argumentation theory leaving their users with an impoverished set of moves and tools that they can use.

This paper introduces MAgtALO (Multi-AGenT Argumentation, Logic and Opinion; also a Tagalog word for disagreement), a prototype system that aims to incorporate the strengths of each approach, building upon the foundations found in argumentation theory which provides theoretical models of argument and argumentative interaction during dialogue, and adopting the appealing and

intuitive interfaces and interaction mechanisms found in contemporary online-community originated argumentation systems. This is achieved through the adoption of a Web 2.0 interface, a multiagent system based backend and integration of representation formats for exchange of arguments and regulation of argumentative interaction. The goal is to support users in their engagement with complex domains in which there are multiple, conflicting points of view, allowing users to intuitively navigate the disagreement space, and facilitating the structured expansion of the argument resources available to the system through argument-based knowledge elicitation.

2 Theoretical Foundations

Argumentation theory provides the theoretical foundations which underlie the knowledge structure and interactions of MAgtALO agents. Agent knowledge is structured using the *Argument Markup Language* (AML) [13] which provides a way to relate statements to form argument structures. Agent interactions are specified using *Dialectical games* which have recently become a popular way of structuring inter-agent communications [10].

2.1 Dialectical Games

When a user interacts with MAgtALO the interaction proceeds according to a protocol which specifies what kinds of things can be said at each juncture in the dialogue. The protocol is a type of simple *Dialectical Game* developed specifically to underpin online argumentation in ill-structured and contentious domains. Dialectical games are turn-taking games which are used to structure the interactions between a dialogue's participants. Players use their turn to make moves which correspond to the kinds of things that they can say for example, asserting, conceding, &c. and the rules specify which moves are legal during any given turn. *Dialogical commitment* is recorded in stores associated with each player and is used as the basis for formulating some rules and as a way to record each player's position.

Dialectical games have been explored in philosophy as a way of analysing particular types of reasoning such as the fallacy of begging the question [8]. More recently, they have also been used as normative ideals for discourse in specific domains such as ethical discussion [15]. These philosophical investigations have, over the past decade or so, been providing rich resources for building inter-agent communication protocols. One of the earliest and best known is Mackenzie's game called DC [8]. DC specifies three types of rule; *Locutional rules* specify the types of moves available; *Commitment rules* specify how commitment stores are updated after a move; *Rules of dialogue* specify when moves are legal. The rules of DC are as follows:

Locutions

(i) Statements. 'P', 'Q', etc. and truth-functional compounds of statements: 'Not P', 'If P then Q', 'Both P and Q'.
(ii) Withdrawals. The withdrawal of the statement 'P' is 'No commitment P'.
(iii) Questions. The question of the statement 'P' is 'Is it the case that P?'
(iv) Challenges. The challenge of the statement 'P' is 'Why is it to be supposed that P?' (or briefly 'Why P?').
(v) Resolution Demands. The resolution demand of the statement 'P' is 'Resolve whether P'.

Commitment Rules

Statements, $\mathbf{CR}_S$: After a statement 'P', unless the preceding event was a challenge, 'P' is included in both participants' commitments.
Defences, $\mathbf{CR}_{YS}$: After a statement 'P', when the preceding event was 'Why Q?', both 'P' and 'If P then Q' are included in both participants' commitments.
Withdrawals, $\mathbf{CR}_W$: After the withdrawal of 'P', the statement 'P' is not included in the speaker's commitment. The hearer's commitment is unchanged.
Challenges, $\mathbf{CR}_W$: After the challenge of 'P', the statement 'P' is included in the hearer's commitment; the statement 'P' is not included in the speaker's commitment; and the challenge 'Why P?' is included in the speaker's commitment.
Questions and Resolution demands, $\mathbf{CR}_Q$ and $\mathbf{CR}_R$: These locutions do not themselves affect commitment.
Initial Commitment, $\mathbf{CR}_0$: The initial commitment of each participant is null.

Rules Of Dialogue

$\mathbf{R}_{Form}$**:** Each participant contributes a locution at a time, in turn; and each locution must be either a statement, or the withdrawal, question, challenge or resolution demand of a statement.
$\mathbf{R}_{Repstat}$**:** No statement may occur if it is a commitment of both speaker and hearer at that stage.
$\mathbf{R}_{Imcon}$**:** A conditional whose consequent is an immediate consequence of its antecedent must not be withdrawn.
$\mathbf{R}_{Quest}$**:** After 'Is it the case that P?', the next event must be either 'P', 'Not P' or 'No commitment P'.
$\mathbf{R}_{LogCall}$**:** A conditional whose consequent is an immediate consequence of its antecedent must not be challenged. $\mathbf{R}_{Chall}$**:** After 'Why P?', the next event must be either;

1. 'No commitment P'; or
2. The resolution demand of an immediate consequence conditional whose consequent is 'P' and whose antecedent is a conjunction of statements to which the challenger is committed; or
3. A statement not under challenge with respect to its speaker (i.e. a statement to whose challenge its hearer is not committed).

$\mathbf{R}_{Resolve}$**:** The resolution demand of 'P' can occur only if either;

1. 'P' is a conjunction of statements which are immediately inconsistent and to all of which its hearer is committed; or
2. 'P' is of the form 'If Q then R', and 'Q' is a conjunction of statements to all of which its hearer is committed; and 'R' is an immediate consequence of 'Q'; and the previous event was either 'No commitment R' or 'Why R?'.

$\mathbf{R}_{Resolution}$**:** After 'Resolve whether P', the next event must be either;

1. The withdrawal of one of the conjuncts of 'P'; or
2. The withdrawal of one of the conjuncts of the antecedent of 'P'; or
3. The consequent of 'P'.

2.2 Araucaria and the Argumentation Markup Language

Araucaria is an application used to mark up and analyse monologic argument based on the *Argumentation Markup Language* (AML) which is formulated in XML. The syntax of AML is specified in a Document Type Definition (DTD) which imposes structural constraints on the form of valid AML documents. AML was primarily produced for use in the Araucaria tool but has subsequently been adopted in other contexts such as within MAgtALO. AML is used to structure the internal knowledge of an agent such that natural language statements are related using argument theoretic concepts. This allows an agent to easily retrieve a supporting statement to use to defends its position if that position comes under attack. The benefit of adopting AML as the internal representation of agent knowledge is that Araucaria can be used as a graphical tool to construct

the agents knowledge by marking up existing natural language texts. Such an approach was used to provide initial agent knowledge so that MAgtALO agents could represent the views held by two prominent contributors to the ID card debate as garnered from their public statements on the subject.

3 The MAgtALO Architecture

MAgtALO consists of a multiagent system back-end and an AJAX-based web interface front-end. The web interface uses client side javascript to ensure that a responsive user interface is provided to the end-user. The interface is served from an Apache web server, MySQL database, and hypertext pre-processor (PHP) application stack running on a FreeBSD server. The multiagent system back-end uses the Jackdaw University Development Environment (JUDE) developed by Calico Jack Ltd [7]. JUDE is a Java based, lightweight, flexible and industrial strength agent development platform that takes a modular approach to agent development. Individual agents consist of a standard core module, provided by the agent framework, which is extended via dynamically loadable modules to provide domain specific capabilities. When a dialogue commences a number of agents are loaded to represent the various participants. Each agent loads a module which provides the capabilities required of an agent to act in the MAgtALO domain, e.g. respond to the users questions, interject when conflicting statements are made and defend its own position when attacked. The connection between the web interface and the agent system is achieved using a proxy web server agent which is provided as standard in the JUDE distribution. The proxy agent communicates HTTP traffic over a nominated port and routes messages from the web interface to the appropriate recipient agent. Similarly outgoing messages from the individual agents to the user are routed via the proxy agent to the web interface where they are displayed.

The key aspects of the MAgtALO system are representation of differing points of view within a specific topic, the capability for the user to engage in dialogue, according to a dialectical game protocol, with a number of agents and explore those differing points of view, and the ability to extend the system and provide new data for the system to use merely through interacting with the existing system. These aspects are explored in more depth in the following sections.

3.1 Points of View

MAgtALO uses agents in a multi-agent system to represent the views of participants. Pre-existing arguments can be analysed for their argumentative structure using tools like Araucaria [13]. The Argumentation Research Group at Dundee have conducted large scale analyses which are available in an online corpus (at araucaria.computing.dundee.ac.uk). The arguments in this corpus are stored using AML, the XML-based Argument Markup Language, and can easily be read into data structures. In this way, an agent can have its beliefs automatically

populated with propositions that correspond to real, analysed, natural text. In the same way, an agent can automatically acquire the argumentative relations between those propositions, as they are represented in the AML source analyses.

By taking several corpus arguments that are in the same domain, but that have been articulated by different authors, it is possible to have multiple agents storing – or "representing" – the points of view expressed in those different arguments. Of course, if multiple arguments are taken from the corpus and used to populate the beliefs of multiple agents, there is the risk that two arguments make the same (or almost the same) claim using different text. There is no reliable way to process natural language to identify such correspondences, so the MAgtALO team currently mark them manually where they arise. In this way, agents may end up with arguments that have overlapping sets of premises or conclusions.

Fig. 1. The main MAgtALO interface

With a multi-agent system populated by agents representing sets of arguments of specific authors, it becomes possible to interact with the data in interesting ways. The interface provides an indication of which points of view are represented. (The running example shown in the figures concerns a current debate

in the UK regarding the issue of identity cards. Two prominent contributors to the debate are Martin Linton, Labour MP for Battersea, and John Wadham, director of the lobby and campaign group Liberty. Some of the recent arguments of these two protagonists have been analysed and loaded into two agents in the system.) MAgtALO uses the metaphor of a meeting, and provides the user with the privileged position of chair. The user can ask individual agents for arguments pro or con a given claim, or can ask one agent to attack the arguments of another, or can ask whether an agent concurs with a claim, and so on. The user is thus responsible for the direction of the virtual discussion modulo the rules of engagement for arguments in meetings - i.e. the dialogue protocol.

Turn	User	Statement
1	System	New discussion started! With the point "Identity cards are a bad idea"
2	User	"I agree. They will undermine civil liberties"
3	John Wadham	"Yes, I agree. The real target is the people whom the home office has lost"
4	User	"Why do you think the current point is true John?"
5	John Wadham	"It will be black people and those from ethnic minorities who will find that they need to show their card most often"
6	Martin Linton	"No, I don't agree. If you are saying the police might discriminate against black people (even though it is illegal), that is a serious problem that needs to be tackled. But should not stop us having ID cards"
7	User	"I'd like to go back to the point that was made earlier... Identity cards are a bad idea"
8	User	"Do you agree with the current point Martin?"
9	Martin Linton	"No, I don't agree. They will not undermine civil liberties"
10	User	"No, I don't agree. Storing potentially vast amounts of sensitive data about every citizen in one place presents danges"

Fig. 2. An example dialogue between a user and two MAgtALO agents, one representing John Wadham of Liberty, and the other representing the Labour MP Martin Linton

3.2 Dialogue

One aim of MAgtALO is to enable the participants to engage in a discussion rather than an interrogation. This means that the protocol by which the players interact must allow for more sophisticated behaviours than just questioning the other players and thereby exploring a knowledge base. Each player must be able to interject with their own opinions, especially when something is said with which they disagree. To enable this kind of behaviour a simple dialogue game protocol was developed to govern the kinds of things that the players can say at each point in the dialogue. This protocol has been developed to ensure that each participant is fairly represented and that individual standpoints

can be investigated, whilst ensuring that the burden on the human participant does not become onerous. (Although there are many techniques and theories available in argumentation theory, rhetoric, and the communication sciences for explaining and structuring exchanges of this sort [4], dialogue games provide the right mix of abstraction from linguistic content and constraint on the role that such content plays dialectically. The abstraction is vital to obviate the need for natural language processing; the constraint is necessary to connect and structure the propositional content).

Dialogues begin from a fixed initial topic, for example, "*identity cards are a bad idea*" which is illustrated in turn 1 of figure 2. This topic does not necessarily represent any given participant's position but serves as the focus for the dialogue. Once the initial topic has been selected, the user is presented with the option to agree, disagree, or to find out where the other agents stand with respect to it. If the user selects either to agree or disagree with the initial point then they are invited to support their position with a reason such as that "*they will undermine civil liberties*". In figure 2 the user has opted to indicate their agreement with the initial point.

Although the user is nominally in control of the dialogue, agents may automatically interject after a statement is made if the agent has a sufficiently strong *desire to speak* regarding that statement. The function that currently calculates desire-to-speak is simple: it is the difference between the number of points in support and the number of points against the statement within an agent's knowledge base. If the value is around zero then the agent has mixed feelings regarding the point. If the value is greater (or less) than zero, then the agent has strong feelings for (or against) the point. Each agent has a threshold value which enables the strength of feeling for a given point to be determined individually. If the threshold is exceeded then the agent will automatically express its viewpoint in the dialogue at that point. Though it is possible to imagine more complex desire-to-speak functions, we have found that even such a simple mechanism provides engaging behaviour with appropriate threshold values. (Notice that there is a strong relationship between the desire-to-speak function and argument aggregation functions. Fox and Das [5] have demonstrated that very simple aggregation functions are often all that is required for appropriate automated reasoning in many situations). Automatic interjection enables the dialogue to proceed with a more natural rhythm. Without such a facility either the user must ask each agent for their view at each turn, or else the agents must all respond to each statement that the other agents and the user make. In either case the resulting dialogue seems artificial and stilted. Though sensitive to the threshold settings, automatic interjection can make the dialogue seem much more natural. This is illustrated in figure 2 in which the agent associated with John Wadham interjects with "*Yes, I agree. The real target is the people whom the home office has lost*" after the user has agreed with the initial point. This indicates that John Wadham has a strong desire to speak in agreement with the last statement made by the user.

Fig. 3. Dialogical interaction in MAgtALO

Once an agent has interjected, the dialogue game allows the user to either agree or disagree with the current point, the last point that was made during the interjection, or to question the agent that made the point to explore that agent's position. This can be as simple as asking, *Why?*, in order to get underlying reasons and so expose the basis for the agent's position. If the agent's point failed to persuade the user, further justification can be solicited. The focus of a dialogue generally follows the last point that was made, but by asking for further reasons the user is switching focus back to an earlier point to get extra, independent support for the point. This process of focus switching allows the user to return to any earlier point in the dialogue, simply by selecting the new focus-point from the dialogue transcript displayed on screen. Such a switch of focus is illustrated in turn 7 of the dialogue in figure 2 whereby the user indicates that they wish to return to an earlier point. In this case the earlier point is the initial point of the dialogue and the user further indicates that they wish to explore Martin Linton's position. The result of such focus switching is that the user is able to explore new threads of reasoning and expose different arguments for and against each point made rather than being locked into a particular path through the dialogue. Again this is an example of how the protocol enables a natural rhythm to be maintained in which, when the user is dissatisfied with the current position, they can return to the point of contention and explore it some more.

MAgtALO is not in the business of calculating a "solution" to a debate, or of evaluating points of view, or of persuading a user that a particular viewpoint is superior. Though such things may be interesting to investigate (as is hinted at, at least in part, in section 5), they are peripheral to the main focus, which is squarely upon providing a rich, flexible, but intuitive interface by which online users can interact with and explore complex debates, thereby gaining a deeper and more sophisticated understanding of the topic. One rather more direct additional benefit of using the theory of dialogue games as a foundation upon which to build such an interface is that the process of extracting structured knowledge from the user is made significantly easier.

3.3 Knowledge Elicitation

The process of uncovering a user's position on a given topic is a form of knowledge elicitation - what [15] refer to as the maieutic function of dialogue. MAgtALO uses a simple dialogue game protocol to expose this knowledge and to record it into the system in a structured fashion. Use of a dialogue game enables the underlying argumentative structure of the dialogue to be captured. This is because each statement is uttered in relation to some earlier statement. For example, offering justification for agreement with a position corresponds to an inference being drawn between the two points, one giving a conclusion and the other giving a reason in support of the conclusion. The use of a dialogue game protocol therefore helps to ensure that each new entry into the dialogue is

dialogically relevant. Such dialogical relevance is important to enable new information to be recorded for reuse in future dialogues. This approach to knowledge elicitation enables the user to express their position and underlying reasons, whilst avoiding the feeling that there is an interrogation occuring. The dialogues are not heavily weighted towards any given participant because any agent may interject at any point if their interjection threshold is exceeded. Meanwhile the user remains in control and moves the focus back and forth, following a natural path through the dialogue. These two elements help to ensure that the resulting dialogue feels natural to the user and thereby gives the user some incentive to continue with the discussion.

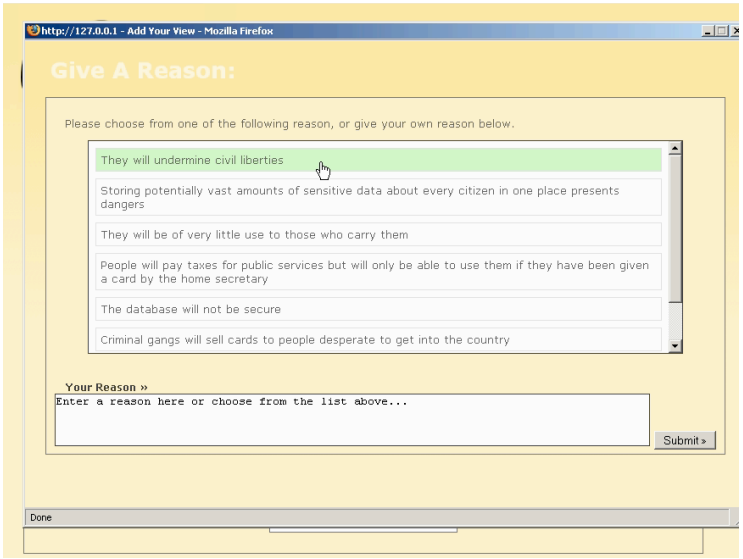


Fig. 4. Knowledge elicitation in MAgtALO

This argument-based knowledge elicitation has some interesting aspects. The amount of new, typed user input is minimised by allowing the user to select from previously recorded statements first, then allowing the user to type in new statements only if there is nothing appropriate already recorded. In the dialogue illustrated in figure 2 the user's views are represented by existing statements in the system until turn 10 at which point the user introduces a new statement as a reason for their disagreement. The benefit of this approach, as well as maintaining user interest by minimising typing, is that existing statements are reused, possibly in new ways so connections can be made between different threads of argument on a topic. Additionally, this approach avoids the need for natural language processing as propositional statements are recorded in their entirety. When statements are reused in new ways it is because the user has linked the statement to some point expressed within a dialogue. Rich, structured knowledge is thus accumulated through a lightweight, naturalistic interaction with the user. The knowledge collated during any given dialogue represents a user's position on the topic of the dialogue. This knowledge can then be reused in subsequent dialogues to provide the knowledge for a new agent representing the last user. Therefore each time a user engages in a dialogue within MAgtALO, there is a structured expansion of the knowledge base, which increases the number of agents who can potentially take part in future dialogues, as well as also increasing the size of the pool of statements from which the next user can select.

4 Argument Ontology

Although the use of AML has been widespread due to the popularity of Araucaria, there has not been a single shared, agreed notation for representing argumentation and arguments and this deficiency has been a major barrier in the deployment of practical argumentation systems. The *Argument Interchange Format* (AIF) [3] is a draft specification for representing information about arguments and exchanging it between argumentation tools and agent-based applications. Adoption of the AIF as the format for representing arguments within individual MAgtALO agents, and as a means to exchange information between MAgtALO installations has a number of benefits. Primarily it enables new online argumentation tools such as ArgDF [12] to be used as source of analysed argument with which to populate the agent's knowledge bases. Additionally, the AIF can be used to distribute argument data between MAgtALO installations so that a user interacting with one instance of MAgtALO, and thereby expanding the available pool of arguments through interactive knowledge elicitation, also provides data that further users interacting with other instances of MAgtALO can make use of. Finally, each dialogue, conceived as a sequence of utterances made by the user and agents in turn, can be recorded, shared, and analysed, both within MAgtALO and with other argument analysis tools.

4.1 The Argument Interchange Format

There have been a number of attempts to construct argument mark-up languages. For example two particularly relevant examples are, AML discussed in section 2.2 and used in the Araucaria system, and *ClaiMaker* which provides a suite of tools for publishing and contesting ideas and arguments. However there are limitations associated with these approaches which motivated the need to for a common interchange format. A number of limitations of existing approaches were identified [3]. Firstly, that each language was designed primarily to be used

by particular software tools, such as Araucaria using AML, rather than facilitating interoperability with other tools. A second limitation is that the existing languages were developed to support graphical in which the user graphically constructs diagrams showing the argumentative linkages between natural language sentences.

The AIF was developed to overcome these limitations with the following primary aims; firstly to facilitate the development of multiagent systems capable of argumentation-absed reasoning and interactions; and secondly to facilitate data interchange between tools that support argument manipulation, visualisation, and utilisation. Two aspects of the AIF that are of particular importance in relation to the MAgtALO system are its use in the representation of *monologic argument* and its use in the representation of *dialogic argument*.

4.2 Monologic Argument Representation

The primary mode of use of the AIF is to represent monologic arguments. In this mode, argument entities are represented as nodes in a directed graph (digraph) informally known as an *argument network*. Two primary types of node are supported, *information nodes* (I-nodes), which relate to argument content and are used to represent claims, and *scheme application nodes* (S-nodes), which are used to represent domain independent, stereotypical patterns of reasoning. Three sub-categories of S-node are currently supported, the *rule application node* (RA-node), the *preference application node* (PA-node), and the *conflict application node* (CA-node). The notion of support in an argument is supported in an AIF argumentation network using edges. If an edge runs from node A to node B then it is said that A supports B. Consequently an argument network can be constructed by connecting the requisite node types according to the specified support semantics of AIF.

4.3 Dialogic Argument Representation

MAgtALO does not currently make a record of the actual dialogue that occurs between the user and the agents, although any new statements entered by the user, or inferences drawn, are recorded into the knowledge base available in subsequent dialogues. It would of course be useful to record the dialogues that occur, so that the user make a record of their interaction, and so that analysis can be made of the ways that users interact with the topic under discussion. Such an ability could be incorporated in MAgtALO using the AIF both to record the entire dialogue and as the format for exchange of messages within a dialogue between the communicating agents. An extension to the AIF that supports such an application is due to Modgil and McGinnis [11] and proposes the introduction of *protocol interaction application* (PIA-nodes) to account for dialogue specific requirements along with adoption the *Lightweight Coordination Calculus* (LCC) [14] as a means to specify a protocol language.

5 Challenges & Directions

It has been suggested that argument provides a more intuitive and accessible means of presenting and assimilating complex data [5], and that structured argumentation can be applied to discussions of complex domains involving real risks [9]. In MAgtALO, both monologic argument structures and dialogic argument protocols are used to give the user intuitive control over navigation of a complex disagreement space. Presenting and organising material explicitly as arguments should mean that users find it easier to understand the relations between the various positions in comparison to sources which have a more discursive style (such as newspaper reports). One would expect the same to be true for other argument-based systems such as debatapedia. But providing an intuitive interaction metaphor with which the user is expected to be familiar (chairing a meeting), and allowing the user active participation in both directing the discussion and contributing to it, it is further expected that MAgtALO should offer an appreciable benefit over formats that allow little or no active participation with the material (such as reports from the traditional media) or that offer a weak, non-argumentative interaction model (such as or wiki pages and discussion boards). Although informal, small-scale evaluations conducted at Dundee suggest that this benefit is substantial, larger scale investigations are required. Testing these hypotheses on specific user groups is a key step for guiding both the MAgtALO project specifically, and the online argumentation research area in general.

From a technical perspective there are two key advances in the underlying representations that structure MAgtALO's immediate development. First is to allow the system to use a variety of different dialogue protocols, so that such protocols might be explored and evaluated, using both the representational style and evaluative approach of [16]. Second is to replace the existing machinery for processing arguments based on the Araucaria representation format AML, and instead equip agents in the system with the ability to import from, and export to, the argument interchange format [3]. The AIF represents a nascent standard for argument representation: by extending MAgtALO to support the AIF, it becomes one of a constellation of systems that can offer an interface to existing argument resources, and provide a means of creating new such resources. By moving to the AIF, it will also become easier to make use of argument computation services that are now under development, for connecting the linguistic, textual analysis, elicitation and interaction with underlying formal models and semantics. It will, for example, become feasible to compute acceptability of each agent's position according to one or more argumentation semantics [2], and provide this information to users as the dialogue progresses.

6 Conclusions

MAgtALO already represents the first example of an implemented online system that uses a closely specified argument-based dialogue protocol combined with a

rich monologic argument representation language to provide a tool for intuitive user exploration of a space of disagreement. As an additional benefit of the approach, it is possible to expand the argument resources through knowledge elicitation that is structured by the argument dialogue protocol. The continuing aim of the research is to use the advances in the theory of argumentation to push the practice of argumentation technology in providing tools and interfaces that have wide appeal.

7 Acknowledgements

A video which demonstrates MAgtALO in action and a live demonstration are of the system, are available online at arg.computing.dundee.ac.uk. The implementation work on MAgtALO was carried out by John Lawrence, an MSc student in the School of Computing during 2006. John can be contacted at mail@johnlawrence.net

References

1. K. Atkinson, T. Bench-Capon, and P. McBurney. Parmenides: Facilitating democratic debate. In *Lecture Notes in Computer Science*, pages 313–316. Springer Berlin / Heidelberg, 2004.
2. M. Caminada. Semi-stable semantics. In P.E. Dunne and T.J.M. Bench-Capon, editors, *Computational Models of Argument (Proceedings of COMMA 2006)*, pages 121–132. IOS Press, 2006.
3. C. Chesnevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *Knowledge Engineering Review*, 21(4):293–316, 2006.
4. F. H. van Eemeren, R. Grootendorst, and F. Snoeck Henkemans. *Fundamentals Of Argumentation Theory*. Lawrence Erlbaum Associates, 1996.
5. John Fox and Subrata Das. A unified framework for hypothetical and practical reasoning (2): Lessons from medical applications. In Dov M. Gabbay and Hans Jurgen Ohlbach, editors, *Practical Reasoning: Proceedings of the International Conference on Formal and Applied Practial Reasoning (FAPR-96)*, LNAI 1085. Springer, 1996.
6. Thomas F. Gordon and Nikos I. Karacapilidis. The zeno argumentation framework. In *International Conference on Artificial Intelligence and Law (ICAIL-97)*, pages 10–18, 1997.
7. Calico Jack Ltd. http://www.calicojack.co.uk, 2005.
8. J. D. Mackenzie. Question begging in non-cumulative systems. *Journal Of Philosophical Logic*, 8:117–133, 1979.
9. P. McBurney and S. Parsons. Risk agoras: Using dialectical argumentation to debate risk. *Risk Management*, 2(2):17–27, 2000.
10. P. McBurney and S. Parsons. Agent ludens: Games for agent dialogues. In *Game-Theoretic and Decision-Theoretic Agents (GTDT 2001): Proceedings of the 2001 AAAI Spring Symposium*, 2001.
11. S. Modgil and J. McGinnis. Towards characterising argumentation based dialogue in the argument interchange format. In *Proceedings of the Fourth International Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2007)*, 2007.

12. I. Rahwan, F. Zablith, and C. Reed. Laying the foundations for a world wide argument web. *Artificial Intelligence*, 2007.
13. C. Reed and G. W. A. Rowe. Araucaria: Software for argument analysis, diagramming and representation. *International Journal of AI Tools*, 14(3-4):961–980, 2004.
14. D. Robertson. Multi-agent coordination as distributed logic programming. In *Proceedings of the International Conference on Logic Programming*, 2004.
15. D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue*. SUNY series in Logic and Language. State University of New York Press, 1995.
16. S. Wells and C. Reed. A drosophila for computational dialectics. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005)*, 2005.

Enhancing Communication inside Multi-Agent Systems*

An Approach based on Alignment via Upper Ontologies

Viviana Mascardi†, Paolo Rosso‡, and Valentina Cordì†

†DISI, Università degli Studi di Genova, Via Dodecaneso 35, 16146, Genova, Italy
E-mail: mascardi@disi.unige.it, valentina.cordi@gmail.com
‡DSIC, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022, Valencia Spain
E-mail: prosso@dsic.upv.es

Abstract. This paper deals with a theoretical issue related to multi-agent system development and deployment, namely the need of a mechanism for aligning ontologies owned by agents, in order to allow them to communicate in a profitable way. Our approach exploits upper ontologies, i.e., ontologies which describe very general concepts that are the same across all domains, as a "lingua franca" among agents. This approach may overcome some problems that arise in various real scenarios, such as the impossibility for (or the lack of will of) an agent to disclose its own entire ontology to another agent, despite the need to communicate with it. In this paper we propose a comparison of seven existing upper ontologies, and an algorithm for aligning any two (or more) ontologies by exploiting an upper ontology as a bridge.

1 Introduction

In a paper that dates back to 2001, James A. Hendler predicted that

> in the next few years virtually every company, university, government agency or ad hoc interest group will want their web resources linked to ontological content - because of the many powerful tools that will be available for using it. [...] On top of this infrastructure, agent-based computing [...] will be a primary means of computation in the not-so-distant future. [9]

Hendler's vision has found a partial realisation: ontologies, web services, and the combination of both, i.e., semantic web services, are more and more exploited to share 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 find application in many commercial projects [14]. Proposals of integrating intelligent agents, web services, and ontologies, thus realising "agent-based semantic web services", are already around [6,20,11]. However, although it is probably true that almost every company, university, government agency *would want* their web resources linked to ontological content, and made available by exploiting an infrastructure based on software agents, it is also true that only a small subset of them *has* already implemented this vision.

* This work was partially supported by the research projects TIN2006-15265-C06-04 and "Iniziativa Software" CINI-FINMECCANICA.

One of the reasons for this delay with respect to Hendler's predictions, is that linking the organisation's web resources, that in most cases will contain knowledge represented by some domain dependent, ad-hoc ontology O_{Org}, to some reference ontological content O_{Ref}, requires to find mappings between the concepts of O_{Org} and O_{Ref}. Without an automatic, agent-driven means for finding these mappings, linking web resources to ontological contents becomes a very difficult and time-consuming activity, not only because of the time and human resources needed for finding them for the first time, but also for maintaining them, as the organisation's ontology O_{Org} will evolve during time, and this will require continuous updates to the mappings with O_{Ref}.

Finding formal statements that assert the semantic relation between two entities belonging to different ontologies (namely, finding an *alignment* of the two ontologies, [4,2]) is thus a key problem in many scenarios such as enterprise information integration, querying and indexing the deep web, merchant catalog mapping, etc. [8]. In all of these scenarios, one organisation wants to align its own entire ontology O with the entire ontology O' of another organisation. This is the most classical instance of the alignment problem, that requires that both ontologies O and O' are given in input to the alignment algorithm, and that are entirely available. The two organisations whose ontologies are involved in the alignment process, must have neither restrictions due to privacy issues, nor restrictions due to limited space or time resources that prevent them from sharing their whole ontology. There are also scenarios, however, where two organisations might want to perform an alignment of their ontologies in order to be able to interact, but either limiting the reciprocal disclosure of the ontologies to the minimum required for understanding each other, or disclosing portions of the ontologies in an incremental way. These scenarios are almost common inside multi-agent systems. In this paper, we face a theoretical issue strongly relevant for developing and deploying multi-agent systems (MASs) in the above scenarios: that of finding and automatic and incremental alignment of the agents' ontologies by exploiting "Upper Ontologies", namely ontologies which describes very general concepts that are the same across all domains.

The paper is organised in the following way: Section 2 describes two scenarios that motivate our approach, Section 3 provides some background on upper ontologies and ontology alignment, Section 4 discusses the algorithm that we propose, and Section 5 draws some conclusions and outlines the future directions of our work.

2 Motivation

In many applicative scenarios, the alignment of two ontologies may be performed directly, namely without any other ontology serving as a bridge, and offline, namely with both ontologies entirely known in advance. We will refer to this scenario as the "*classical*" one. However, there are situations where the assumptions made in the "*classical*" scenario, do not hold. Before considering two of these situations, we observe that ontologies may be developed and owned by a single human developer, by an institution

like a company, or by a software agent equipped with knowledge and algorithms suitable for managing an ontology in a (semi-)automatic way. In this paper, we attribute the capability of developing and possessing an ontology to human beings, software agents, and institutions indifferently. Also, coherently with the "strong definition" of agents [22], we attribute intentions, goals, and beliefs to them.

Personalised content provider. In this scenario, a provider of personalised content (news, commercial advertisements, etc.) is a software agent that provides content to the agents that access it, be them human or software, in a personalised way. The most accepted personalisation approach is the one where the user's interests and habits are learnt just by watching his/her behaviour, without any explicit request apart from request on feedback on the provided content. The MAS composed by the personalised content provider agent (PCPA) and its users is highly open and dynamic.

The PCPA is strongly motivated to make a good job because it usually has economic advantages in increasing the number of users that access its services. Also, it is motivated to make its users faithful to it, since a user that accesses the service on a regular basis can be better profiled and the PCPA can deliver most appropriate contents to him/her. The user will then be more satisfied with the offered service, and will tend to go on using it. As far as ontologies are concerned, it is very likely that the PCPA possesses a private ontology O_{PCPA} for categorising the contents that it manages and delivers.

On the other hand, the user does not want to spend time in training the PCPA, and s/he probably does not event want to reveal extra information to it, besides that concerning his/her interests which can be extracted from the queries s/he poses to the PCPA. The user may or may not have an explicit ontology defining the concepts that appear in the queries made to the PCPA. In case s/he (or it, if we consider software users acting on behalf of humans) has an ontology, s/he might not want to share it with the PCPA.

- **Alignment problem.** The alignment problem in this scenario is asymmetric: the PCPA would like to know all the ontologies that its users use, and align them with its own in order to provide the best answers to the users' queries. Instead, most of the users either do not have an explicit ontology (this means that they *do* have a reference ontology, but it is "hard-wired" inside their knowledge, or inside their code, or inside their brain), or they have it but are not interested in sharing it with the PCPA.
- **Proposed solution: incremental alignment.** What the PCPA may do to meet its objectives, is to consider the concepts that appear in queries issued by $User$ as a subset of the concepts of $User$'s ontology, O_{User}. The PCPA can make a partial alignment of the sub-ontology composed only by those concepts (without any relation) and O_{PCPA}, use this partial alignment to provide answers, and refine it as soon as new queries from $User$ (and thus, new concepts of O_{User}) arrive. In other words, the PCPA may perform a *partial, incremental, on-line alignment* between an (unknown and undisclosed) ontology O_{User} and O_{PCPA}. The alignment is partial because only a portion of O_{User} can be elicited from the users's queries. It is incremental, because as new portions of O_{User} arrive, the alignment is updated and enriched with new information. Finally, it is an on-line process, made of successive refinements performed at any interaction between $PCPA$ and $User$.

Virtual Enterprises. A virtual enterprise (VE) is a temporary consortium of autonomous, diverse and possibly geographically dispersed organizations that pool their resources to meet short-term objectives and exploit fast-changing market trends [3].

Due to the autonomy, distribution and heterogeneity of the enterprises belonging to a VE, it can be suitably conceptualised as a MAS. This MAS is neither open nor dynamic since the enterprises belonging to it are known in advance. It is reasonable to assume that, at least within the information and communication technologies domain, most or all of them possess a private ontology and are strongly motivated in sharing and aligning it with the ontologies of the other enterprises in the VE in order to improve the benefits of the VE as a whole.

This scenario would be the right one for adopting a classical approach to alignment, if there were no privacy restrictions on the ontologies of each individual enterprise. Unfortunately, this is not often the case. What happens in the real case, is that the enterprises want and need to exchange useful information among them, but they also want and need to protect (part of) their enterprise information, represented by the enterprise ontology. In fact, the VE is just a temporary consortium where each component also runs its own business, often in concurrence with other enterprises within the same VE.

The disclosure of the entire ontology to all the partners belonging to the VE, has economic and commercial advantages for the VE, but may be a serious disadvantage for the single enterprise.

- **Alignment problem.** The alignment problem in this scenario is symmetric: all the enterprises would like to have the means for interacting with each other in the best possible way, but they also would like to avoid disclosing their own ontologies.
- **Proposed solution: alignment via upper ontologies.** The solution to this instance of the alignment problem may come from the use of upper ontologies: each enterprise may align its own ontology with an upper ontology upon which all the enterprises agree. Then, each enterprise becomes able to communicate with any other enterprise by means of the upper ontology. This gives two advantages to both the individual enterprises and the VE:
  1. no enterprise has to disclose its own ontology: it may only disclose the corresponding portion of the upper ontology obtained by alignment;
  2. if there are n enterprises within the VE, the alignments required to allow any enterprise to interact with any other are only n (one alignment between each private ontology and the upper ontology), instead of the n^2 that would be required if any enterprise had to align its own ontology with all the other ones.

The analysis of the state of the art discussed in Section 3 shows that there are very few proposals of performing an incremental alignment of ontologies, and no implemented systems that exploit upper ontologies in the alignment process. However, the two motivating scenarios that we have identified, demonstrate that both incremental and upper ontology-based approaches would be extremely useful in many agent systems.

3 Background

In this section we introduce upper ontologies and a systematic comparison of seven of them, and we discuss the state-of-the-art of alignment techniques. To the best of our knowledge, there are no comparisons that consider all the upper ontologies discussed in Section 3.1. Thus, this section represents an original contribution of our work. For space constraints, we only report the synthesis of our comparison in form of tables. More information can be found in [13].

3.1 Upper Ontologies

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.

The definition of upper ontology (also named top-level ontology, or foundation ontology) given by Wikipedia [21] 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*".

In this section, we review the state-of-the-art in the field of upper ontologies by comparing seven of them based on dimension, implementation language(s), modularity, alignment with the WordNet lexical resource, and licensing. These software engineering criteria may prove 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. The choice of the upper ontologies we describe and compare, namely BFO, Cyc, DOLCE, GFO, PROTON, Sowa's ontology, and SUMO, is based on how much they are visible and used inside the research community. In fact 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 and National Information Exchange Models, that address the specific application domain of justice and public safety. Moreover, we have added PROTON and Sowa's ontology to those considered by Wikipedia.

The methodology followed to draw our comparison 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 seven upper ontologies under comparison, and integrating the obtained answers and suggestions.

Table 1 provides a short description of the seven upper ontologies, while Tables 2 and 3 provide an handy way for comparing them from a software engineering viewpoint. Few other comparisons among upper ontologies exist, and they just consider subsets of the upper ontologies that we have treated in this section. Most of these existing comparisons, such as Pease's comparison of DOLCE and SUMO [16,17], Onto-Med's comparison of GFO, DOLCE, and Sowa's ontology [10], and Grenon's comparison of DOLCE and BFO [7], take a philosophical perspective, and thus complement our work that is much more application-oriented. MITRE's comparison of SUMO, Upper Cyc, and DOLCE [18], compares the three upper ontologies according to a subset of our criteria.

BFO	BFO (`http://www.ifomis.org/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. It finds application mainly in the biomedical domain.
Cyc	The Cyc Knowledge Base KB (`http://www.cyc.com/`) 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. It finds application in natural language processing, network risk assessment, terrorism management.
DOLCE	DOLCE (`http://www.loa-cnr.it/DOLCE.html`) captures 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 an "ontology of particulars", i.e. an ontology of instances, rather than an ontology of universals or properties. DOLCE-Lite+ (`http://wiki.loa-cnr.it/index.php/LoaWiki:Ontologies#Modules_of_the_DOLite.2B_Library`) encodes the basic DOLCE ontology into OWL-DL and adds eight pluggable modules, including collections, social objects, plans, spatial and temporal relations, to it. DOLCE is used for multilingual information retrieval, web-based systems and services, e-learning.
GFO	GFO (`http://www.onto-med.de/ontologies/gfo.html`) 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. It is used in the biomedical domain.
PROTON	PROTON (PROTo ONtology, `http://proton.semanticweb.org/`) is a basic upper-level ontology providing coverage of the general concepts necessary for a wide range of tasks. The design principles are (i) domain-independence; (ii) lightweight logical definitions; (iii) consistence with popular standards; (iv) good coverage of named entities and concrete domains (i.e., people, organizations, locations, numbers, dates, addresses). It is used for semantic annotation, knowledge management systems in legal and telecomm. domains, business data ontology for semantic web services.
Sowa's	Sowa's ontology (`http://www.jfsowa.com/ontology/`) is based on [19]. The basic categories and distinctions have been derived from a variety of sources in logic, linguistics, philosophy, and artificial intelligence. 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 selecting an appropriate set of distinctions. No documented applications have been developed, but Sowa's ontology inspired the creation of many implemented upper ontologies.
SUMO	SUMO (`http://www.ontologyportal.org/`) and its domain ontologies [15] form one of the largest formal public ontology in existence today. SUMO is extended with many domain ontologies and a complete set of links to WordNet, and is freely available. It finds application in linguistics, knowledge representation, reasoning.

Table 1. Short description of the upper ontologies

	Developers	Dimensions	Language(s)
BFO	Smith, Grenon, Stenzhorn, Spear (IFOMIS)	36 classes related via the is_a relation	OWL
Cyc	Cycorp	About 300,000 concepts, 3,000,000 facts and rules, 15,000 relations (including microtheories)	CycL, OWL
DOLCE	Guarino and other researchers of the LOA	About 100 concepts and 100 axioms	First Order Logic, KIF, OWL
GFO	The Onto-Med Research Group	79 classes, 97 subclass-relations, 67 properties	First Order Logic and KIF (forthcoming); OWL
PROTON	Ontotext Lab, Sirma	300 concepts and 100 properties	OWL Lite
Sowa's	Sowa	30 classes, 5 relationships, 30 axioms	First Order Modal Language, KIF
SUMO	Niles, Pease, and Menzel	20,000 terms and 60,000 axioms (including domain ontologies)	SUO-KIF, OWL

Table 2. Comparison, Part I

	Modularity	Alignment with WordNet	Licensing
BFO	SNAP and SPAN modules	Not supported	Freely available
Cyc	"Microtheory" modules	Mapped to about 12,000 WordNet synsets	Commercial product; ResearchCyc and OpenCyc are freely available but are more limited than the commercial version
DOLCE	DOLCE is not divided into modules, while DOLCE-Lite+ is	Aligned with about 100 WordNet sysnsets	Freely available
GFO	Abstract top level, abstract core level, basic level	Not supported	Released under the modified BSD Licence
PROTON	Three levels including four modules	Not supported	Freely available
Sowa's	Not divided into modules	Not supported	Freely available
SUMO	Divided into SUMO itself, MILO, and domain ontologies	Mapped to all of WordNet v2.1 by hand	Freely available

Table 3. Comparison, Part II

3.2 Ontology Alignment

In [2], an alignment is described as

> "*a set of mappings expressing the correspondence between two entities of different ontologies through their relation and a trust assessment. The relation can be equivalence as well as specialisation/generalisation or any other kind of relation. The trust assessment can be boolean as well as given by other measures (e.g., probabilistic or symbolic measures)*".

Intuitively, a mapping can be described as a 5-tuple $< id, e, e', n, R >$ where:

- id is a unique identifier of the given mapping element;
- e and e' are the entities (e.g. tables, XML elements, properties, classes) of the first and the second ontology respectively;
- n is a confidence measure (typically in the [0,1] range) holding for the correspondence between the entities e and e';
- R is a relation such as equivalence, more general, disjointness, overlapping, holding between the entities e and e'.

In [5], ontology alignment approaches are classified into:

Local Methods — The main issue in aligning consists of finding to which entity or expression in one ontology corresponds another one in the other ontology. Local methods are the basic methods which enable to measure this correspondence at a local level, i.e., only comparing one element with another and not working at the global scale of ontologies. Very often, this amounts to measuring a pair-wise similarity between entities (which can be as reduced as an equality predicate) and computing the best match between them. Local methods exploit the definitions of similarity and of distance. In [5] such definitions are provided, as well as a detailed classification of local methods.

Global Methods — Once the local methods for determining the similarity are available, the alignment must be computed. This involves some kind of more global treatments, including:

- aggregating the results of local methods in order to compute the similarity between compound entities;
- developing a strategy for computing these similarities in spite of cycles and non linearity in the constraints governing similarities;
- organising the combination of various similarity algorithms;
- involving the user in the loop;
- finally extracting the alignments from the resulting similarity: indeed, different alignments with different characteristics can be extracted from the same similarity.

Since global methods are based upon local ones, in the following paragraph we briefly discuss local methods.

Local Methods.

Definition 1. (Similarity). *A similarity $\sigma : O \times O \rightarrow \mathbb{R}$ is a function from a pair of entities to a real number expressing the similarity between two objects such that:*

$$\forall x, y \in O, \sigma(x, y) \geq 0 \ (positiveness)$$
$$\forall x \in O, \forall y, z \in O, \sigma(x, x) \geq \sigma(y, z) \ \ (maximality)$$
$$\forall x, y \in O, \sigma(x, y) = \sigma(y, x) \ \ \ (symmetry)$$

The dissimilarity is the dual operation of the similarity.

Definition 2. (Distance). *A distance (or metrics) $\delta : O \times O \rightarrow \mathbb{R}$ is a dissimilarity function satisfying the definiteness and triangular inequality:*

$$\forall x, y \in O, \delta(x, y) = 0 \ iff \ x = y \qquad (definiteness)$$
$$\forall x, y, z \in O, \delta(x, y) + \delta(y, z) \geq \delta(x, z) \ (triangular \ inequality)$$

A (dis)similarity is said to be normalised if it ranges over the unit interval of real numbers [0 1]. Local methods introduced in the following classification use normalised measures.

1. **Terminological methods** compare strings. They can be applied to the name, the label or the comments concerning entities in order to find those which are similar. Terminological methods are further divided into string-based methods, that take advantage of the structure of the string as a sequence of letter, and language-based methods, that rely on using natural language processing (NLP) techniques to find associations between instances of concepts or classes.
2. **Structural methods** compare the structure of the entities. This comparison may either be a comparison of the internal structure of an entity (i.e., its attributes) or a comparison of the entity with other entities to which it is related.
3. **Extensional methods** compare the extension of classes, i.e., the set of their instances rather than their interpretation.
4. **Semantic methods** have model-theoretic semantics which is used to justify their results, and thus they are deductive methods. Examples are propositional satisfiability (SAT) and modal SAT techniques or description logic based techniques.

While the literature discusses many local and global "off-line" alignment methods, very few attempts have been made to propose "*incremental*" alignment methods [1,12], and none exploits upper ontologies for this purpose. The next section discusses our algorithm for incremental alignment based on upper ontologies. Even if the algorithm has not been implemented yet, our approach may be a preliminary contribution to a research field that is still unexplored.

4 Algorithm

The algorithm that we propose is based on two functions, $Align(O_1, O_2)$, where O_1 and O_2 are two ontologies, and $Merge(Al_1, Al_2)$, where Al_1 and Al_2 are two alignments. Both functions return an alignment. The algorithm consists of three steps: the concepts of the two ontologies to align are first "tagged" with concepts of a reference upper ontology by exploiting the $Align$ function (first two steps), and the two alignments obtained in this way are merged. The two agents that want to align their ontologies may exploit only the merged alignment, thus avoiding the disclosure of their private ontologies, and this process may take place following incremental steps, for coping with all those situations where one of the ontologies is not fully known in advance. We consider alignment of ontologies based on the equivalence relation R, which is thus dropped from the 5-tuple that represents the mapping element.

$Align(O_1, O_2)$ just computes an alignment between O_1 and O_2 by exploiting one (or a combination of) standard local method(s) among those introduced in Section 3.2.

Given two alignments $Al_1 = Align(O_1, O_{Bridge})$ and $Al_2 = Align(O_2, O_{Bridge})$, $Merge(Al_1, Al_2)$ computes the alignment Al between O_1 and O_2 starting from Al_1 and Al_2. A mapping element $\langle id, C_1, C_2, Conf \rangle$ belongs to Al iff $\exists \ C_{Bridge} \in O_{Bridge}$ such that $\langle id_1, C_1, C_{Bridge}, Conf_1 \rangle \in Al_1, \langle id_2, C_2, C_{Bridge}, Conf_2 \rangle \in Al_2$ and $Conf_1 * Conf_2 \geq Threshold$, where $Threshold$ is a configurable threshold.

Our algorithm consists of the following steps:

Al($\mathbf{O_1}$, $\mathbf{O_{Bridge}}$) Each concept $C_1 \in O_1$ is "tagged" with one or more concepts $\{C_{Bridge_1}, ..., C_{Bridge_k}\} \in O_{Bridge}$ by exploiting traditional mapping techniques based on a combination of string comparison, natural language processing techniques, and exploitation of linguistic resources such as common knowledge or domain specific thesauri. Namely, an alignment $Al(O_1, O_{Bridge})$ between O_1 and O_{Bridge} is computed. The tagging is represented as a set of mapping elements $\{\langle id_1, C_1, C_{Bridge_1}, Conf_1 \rangle, ..., \langle id_k, C_1, C_{Bridge_k}, Conf_k \rangle\}$.

Al($\mathbf{O_2}$, $\mathbf{O_{Bridge}}$) In the same way, an alignment $Al(O_2, O_{Bridge})$ between O_2 and O_{Bridge} is computed, resulting into a "tagging" of concepts of O_2 with concepts of O_{Bridge}.

Merge($\mathbf{Al_1}$, $\mathbf{Al_2}$) The alignment between O_1 and O_2, namely $Merge(O_1, O_2)$, is computed.

This algorithm may be used in three different ways, depending on the usage scenario and just skipping some of its steps:

Usage 1 - Classical way: This usage corresponds to the situation where two agents Ag_1 and Ag_2 agree to share their ontologies O_1 and O_2, to give them in input to the alignment function, and to take advantage of the computed output. Once both agents know the computed alignment $Align(O_1, O_2)$, they may communicate either using O_1 as their reference ontology, or using O_2, indifferently. In this case, only the first step of the algorithm is performed, with O_2 used instead of O_{Bridge}.

Usage 2 - Incremental way: This is the usage foreseen within the personalised content provider scenario. The PCPA Ag_1 computes $Al_1 = Align(O_1, O_{Upper})$, where

O_{Upper} is an upper ontology; when Ag_1 receives a message containing the concepts $\{C_{2_1}, \ldots, C_{2_n}\} \in O_2$ from the user agent Ag_2, it interprets these concepts as a sub-ontology $O_{2_{small}}$ of the implicit ontology O_2 used by Ag_2. $O_{2_{small}}$ is a degenerated ontology, since it has only concepts with no relations among them. Although degenerated, this is the only ontology that the PCPA Ag_1 may elicit from the user agent Ag_2. Ag_1 may then call $Al_2 = Align(O_{2_{small}}, O_{Upper})\ \forall C_{2_i}$, and merge the obtained partial alignment Al_2 with Al_1 by calling the function $Merge(Al_1, Al_2)$. As new messages from the user agent Ag_2 arrive, containing new concepts of the ontology used by Ag_2, the PCPA reiterates the computation of the alignment, and integrates the new alignment with the previously obtained one by making a union of the tuples belonging to the old and new alignments, just omitting those tuples that appear more than one time, with different identifier and confidence. In this case, the tuple with the best confidence may be kept, and the other one(s) may be skipped.

In this scenario, exploiting an upper ontology is very important for providing the right content to the user agent. In fact, the user agent might require something that is not included in the content provider's ontology. To make a trivial example, the user might look for content dealing with "marsupials", and the content provider might only possess the "mammal" concept in its ontology, with no sub-concepts. If the concepts in the user's message are directly aligned with those in the content provider's ontology, the mapping between marsupial and mammal is lost, since there is no way for the content provider to know what a marsupial is. Instead, if the reference upper ontology contains the information that a marsupial is a mammal (like SUMO does, `http://ontology.teknowledge.com/sumo-1.36classes.pdf`), a mapping between the marsupial concept belonging to the user agent's ontology O_2 and the mammal concept belonging to the PCPA's ontology O_1, may be found with a confidence lower than 1 (the two concepts are not equivalent).

Usage 3 - Upper ontology way: This is the usage foreseen within the Virtual Enterprise scenario. All the agents belonging to the VE, agree to use one upper ontology O_{Upper} as their "lingua franca" for knowledge exchange. Each agent Ag having a private ontology O calls $Align(O, O_{Upper})$. O_{Upper} is then used as the reference ontology for exchanging information among agents, without requiring to any agent to disclose its own private ontology. In this case, the first two steps of the algorithm are repeated n times - instead of twice - where n is the number of agents in the VE, and the last step of the algorithm that merges two alignments for finding a third one, is omitted.

4.1 Complexity

We evaluate the algorithm complexity when used in the three different ways identified in the previous paragraph.

Assumption. Given a concept $C_1 \in O_1$, finding a mapping between C_1 and any concept in O_2 by exploiting techniques based on a combination of string comparison, natural language processing techniques, and exploitation of linguistic resources, is in $\mathcal{O}(V_2)$, where V_2 is the number of concepts of O_2.

We obtain this result because we may consider that string comparison, NLP techniques and exploitation of a fixed-size thesaurus require the same amount of time $T_{string,NLP,thesaurus}$ whatever the two concepts to be mapped.

Usage 1 - Classical way: The time required for aligning O_1 with V_1 vertices and O_2 with V_2 vertices without exploiting an upper ontology is the time required by performing only the first step of the algorithm, namely $\mathcal{O}(V_1 * V_2)$.

Usage 2 - Incremental way: O_1 is the ontology owned by Ag_1 that wants to elicit O_2 owned by Ag_2, and align it with O_1. O_{Upper} is the reference upper ontology. The alignment between O_{Upper} and O_1 (first step of the algorithm) is done once and forall, and costs $\mathcal{O}(V_{Upper} * V_1)$. We do not count this time as part of the online incremental alignment process, since it is done offline, and only once.

Assuming that for any concept C_1 in O_1 there are at most k concepts in O_{Upper} that have a mapping with C_1 with a confidence greater that a given threshold, with k fixed, the concepts of O_{Upper} to which at least some concept in O_1 maps are at most in $k * V_1$, and identify a sub-ontology $O_{Upper}(O_1)$ of O_{Upper}. When j new concepts of O_2 arrive as part of a message from Ag_2 to Ag_1, they may be aligned only with the concepts in $O_{Upper}(O_1)$ (second step of the algorithm). Since the concepts in $O_{Upper}(O_1)$ are $k * V_1$, aligning j concepts of O_2 with them requires $\mathcal{O}(j * V_1)$ (k is a constant, thus we drop it from the time complexity).

After this alignment has been computed, it must be merged with the one between O_{Upper} and O_1. The merge step requires $\mathcal{O}(j * V_1)$.

To make a comparison with the "classical" approach, let us suppose that a message containing all the concepts of O_2 arrives from Ag_2 to Ag_1. This will require that all the V_2 concepts of O_2 are aligned with those in O_1. The complexity of finding this alignment is $\mathcal{O}(V_1 * V_2)$ (second step of the algorithm). The last step of the algorithm requires once again $\mathcal{O}(V_1 * V_2)$ resulting into a $\mathcal{O}(V_1 * V_2)$ overall time complexity. If we exclude the first step of the algorithm, the time required for aligning O_1 and O_2 by exploiting O_{Upper} as a bridge is the same required for aligning O_1 and O_2 without using O_{Upper}.

Upper ontology way: If V is the number of concepts of O, and V_{Upper} is the number of concepts of O_{Upper}, finding an alignment between O and $Upper$ requires $\mathcal{O}(V * V_{Upper})$.

In the Virtual Enterprise scenario, finding an alignment between any two ontologies $O_1, ..., O_n$ via O_{Upper} requires $\mathcal{O}(n * V_{max} * V_{Upper})$ where V_{max} is the number of concepts belonging to the largest ontology in the set $\{O_1, ..., O_n\}$. In fact, the alignment step of the algorithm must be performed not only twice, but n times, and the merge step is not needed, since all virtual enterprises will communicate using concepts from O_{Upper}. We may use V_{max} as an upper bound for the number of concepts of $O_1, ..., O_n$, and consider that any alignment between one ontology in this set, and O_{Upper} requires at most $\mathcal{O}(V_{max} * V_{Upper})$ time. Since the alignments to compute are n, we obtain the stated complexity result.

If we did not exploit O_{Upper} as a bridge among $O_1, ..., O_n$, alignments between any two couples of ontologies $O_1, ..., O_n$ should be computed to allow virtual enterprises to communicate, and this would require to compute $n * (n-1)$ alignments, instead of n.

The choice of using O_{Upper} in order to obtain a gain in complexity depends on the size of O_{Upper}. If the size of all the ontologies in $O_1, ..., O_n$ is comparable to that of O_{Upper}, let us name this size S, the complexity of the algorithm that uses O_{Upper} as a bridge is in $\mathcal{O}(n * S^2)$, while the complexity of the algorithm that does not use O_{Upper} as a bridge is in $\mathcal{O}(n^2 * S^2)$. If O_{Upper} is much larger than $O_1, ..., O_n$, it might be better to compute the $n*(n-1)$ small alignments between each couple of ontologies, instead of computing n very large alignments between any ontology and O_{Upper}.

In this second case, however, there is a real gain only if the system is closed and no new ontologies will be aligned in subsequent moments. In fact, if a new ontology O_{new} needs to be aligned, and no upper ontology O_{Upper} has been used, O_{new} must be aligned with n ontologies instead than with only one. In an open system, such an approach may become soon unacceptable.

4.2 Implementation

The implementation of the algorithm and its testing are under way. In this section we describe the approach that we are following; changes are still possible according to the experimental results that we will obtain and that will drive our future work.

We have chosen to exploit an API for ontology alignment developed by J. Euzenat, with contributions from other researchers. [1] The API provides services for

- storing, finding, and sharing alignments;
- piping alignment algorithms;
- manipulating alignments;
- comparing alignments.

The Alignment interface provides the following methods:

NameEqAlignment compares the equality of class and property names and aligns those objects with the same name;

EditDistNameAlignment uses an editing (or Levenshtein) distance between entity names. It builds a distance matrix and chooses the alignment based on that distance;

SubsDistNameAlignment computes a substring distance on the entity name;

StrucSubsDistNameAlignment computes a substring distance on the entity names and aggregates this distance with the symmetric difference of properties in classes.

The pseudo-code for the $align$ and $merge$ functions is given below. Ontologies $O1$ and $O2$ are loaded from two URIs (the $loadOntology$ method is provided by Euzenat's API). $AlignmentMethod1$ ... $AlignmentMethodN$ will be chosen among the alignment methods provided by the API and listed above, and $combine$ will combine the results of the alignments performed with different methods, for example by pipelining them. The $merge$ function implements the algorithm introduced in the beginning of Section 4, in an iterative way.

[1] `http://alignapi.gforge.inria.fr/`


```
public Alignment align(URI uri1, URI uri2) {
  OWLOntology O1 = loadOntology(uri1);
  OWLOntology O2 = loadOntology(uri2);

  AlignmentProcess A1 = new AlignmentMethod1(O1, O2);
  ...
  AlignmentProcess AN = new AlignmentMethodN(O1, O2);

  Alignment A = combine(A1, A2, ..., AN);

  return A;}

public Alignment merge(Alignment A, Alignment B){
  Alignment M = null;

  for each mapping m in A
   for each mapping n in B
          if (m.C2 is equal to n.C2)
                then
                add(<id, m.C1, n.C2, m.Conf*n.Conf>) to M;
  return M;}
```

The output file would consist of XML structures like the one shown below.

```
<Alignment>
 <map>
   <Cell>
     <identifier
       rdf:datatype='http://www.w3.org/2001/XMLSchema#integer'>
       Id </identifier>
     <entity1 rdf:resource='O1#Concept1'/>
     <entity2 rdf:resource='O2#Concept2'/>
     <measure
       rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>
       Conf </measure>
   </Cell>
 </map>
 ...
</Alignment>
```

Each correspondence (`map`) is made of an integer identifier, two references to the aligned entities, and a confidence measure ([0,1]) in this correspondence.

5 Conclusions

The work proposed in this paper originates from the observation that, in many situations, agents need to "learn" new knowledge that must be understood and added to the already possessed knowledge on-the-fly, in order to communicate in a profitable way with other agents in an open, dynamic system. The approaches for coping with this integration need to approximate and optimize the newly acquired knowledge. In

fact, the new knowledge cannot be "precise" since it comes from heterogeneous agents with which agreement neither on the terms to be used in the communication, nor on their meaning, had been made before. The common knowledge shared among agents, that is often implicit even to the agents themselves (in case of human agents), must be constructed via interaction, in an incremental way.

Starting from these considerations, we have designed an algorithm for enhancing communication among heterogeneous agents via incremental ontology alignment and exploitation of upper ontologies. The agents that may benefit from implementing such an algorithm are "semantic web agents" equipped with an ontology and able to communicate via the net, be it an intranet, like in the Virtual Enterprise scenario, or the Internet, like in the Personalised Content Provider scenario. Both issues that characterise our approach, i.e., incremental alignment and use of upper ontologies, are original contributions, as well as the comparison among BFO, Cyc, DOLCE, GFO, PROTON, Sowa's ontology, and SUMO, that we have drawn.

At the time of writing, implementation and testing of our algorithm are still under way, but we will soon receive feedback from the implementation results, and this 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 work is entirely aimed at completing the implementation of the algorithm and systematically describing our experimental results. Afterwards, one extension of our approach that might be interesting to explore is whether the seven upper ontologies that we have described in our paper may be themselves aligned into one "upper-upper ontology".

Acknowledgments

We want to acknowledge all the researchers that helped in drawing the comparison of Section 3.1 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 thank A. Locoro for her thoughtful advices.

References

1. P. A. Bernstein, S. Melnik, and J. E. Churchill. Incremental schema matching. In *VLDB'2006, 32nd International Conference on Very Large Data Bases, Proceedings*, pages 1167–1170. VLDB Endowment, 2006.
2. P. Bouquet, J. Euzenat, E. Franconi, L. Serafini, G. Stamou, and S. Tessaris. Specification of a common framework for characterizing alignment. Technical Report D2.2.1, NoE Knowledge Web project, 2004.
3. H. Davulcu, M. Kifer, L. R. Pokorny, C. R. Ramakrishnan, I. V. Ramakrishnan, and S. Dawson. Modeling and analysis of interactions in virtual enterprises. In *RIDE-VE'99, 9th International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises, Proceedings*, pages 12–18. IEEE Computer Society, 1999.
4. J. Euzenat. An API for ontology alignment. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *International Semantic Web Conference, Proceedings*, volume 3298 of *Lecture Notes in Computer Science*, pages 698–712. Springer, 2004.

5. J. Euzenat, T. Le Bach, J. Barrasa, and P. Bouquet et al. State of the art on ontology alignment. Technical Report D2.2.3, NoE Knowledge Web project, 2005.
6. N. Gibbins, S. Harris, and N. Shadbolt. Agent-based semantic web services. In *WWW '03, 12th International Conference on World Wide Web, Proceedings*, pages 710–717. ACM Press, 2003.
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. A. Y. Halevy. Why your data won't mix. *ACM Queue*, 3(8):50–58, 2005.
9. J. A. Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.
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, Germany, 2006.
11. L. Kerschberg, M. Chowdhury, A. Damiano, H. Jeong, S. Mitchell, J. Si, and S. Smith. Knowledge sifter: Agent-based ontology-driven search over heterogeneous databases using semantic web services. In M. Bouzeghoub, C. A. Goble, V. Kashyap, and S. Spaccapietra, editors, *Semantics for Grid Databases, First International IFIP Conference on Semantics of a Networked World: ICSNW 2004, Revised Selected Papers*, volume 3226 of *Lecture Notes in Computer Science*, pages 278–295. Springer, 2004.
12. J-S. Lee and K-H. Lee. XML schema matching based on incremental ontology update. In X. Zhou, S. Y. W. Su, M. P. Papazoglou, M. E. Orlowska, and K. G. Jeffery, editors, *Web Information Systems - WISE 2004, 5th International Conference on Web Information Systems Engineering, Proceedings*, volume 3306 of *Lecture Notes in Computer Science*, pages 608–618. Springer, 2004.
13. V. Mascardi, V. Cordí and P. Rosso. A comparison of upper ontologies. Technical Report DISI-TR-06-21, University of Genoa, 2006.
14. S. Munroe, T. Miller, R. A. Belecheanu, M. Pechoucek, P. McBurney, and M. Luck. Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. *The Knowledge Engineering Review*, 21(4):345 – 392, 2006. Cambridge University Press.
15. I. Niles and A. Pease. Towards a standard upper ontology. In C. Welty and B. Smith, editors, *FOIS 2001, 2nd International Conference on Formal Ontology in Information Systems, Proceedings*, pages 2–9. ACM Press, 2001.
16. 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.
17. A. Pease and C. Fellbaum. Formal ontology as interlingua: The SUMO and WordNet linking project and GlobalWordNet. To appear.
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. J. F. Sowa. In *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing, 1999.
20. K. P. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. *IEEE Internet Computing*, 8(3):66–73, 2004.
21. Wikipedia. Upper ontology – Wikipedia, the Free Encyclopedia, 2006. [Online; accessed 31-July-2007].
22. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

A Service-Oriented Approach for Curriculum Planning and Validation

Matteo Baldoni[1], Cristina Baroglio[1], Ingo Brunkhorst[2], Elisa Marengo[1], Viviana Patti[1]

[1] Dipartimento di Informatica — Università degli Studi di Torino
c.so Svizzera, 185, I-10149 Torino (Italy)
{baldoni,baroglio,patti}@di.unito.it, elisa.mrng@gmail.com
[2] L3S Research Center, University of Hannover
D-30539 Hannover, Germany
brunkhorst@l3s.de

Abstract. We present a service-oriented personalization system, set in an educational framework, based on a semantic annotation of courses, given at a knowledge level (what the course teaches, what is requested to know for attending it in a profitable way). The system supports users in building personalized curricula, formalized by means of an action theory. It is also possible to verify the compliance of curricula w.r.t. a model, expressing constraints at a knowledge level. For what concerns the first task, classical planning techniques are adopted, which take into account both the student's initial knowledge and her learning goal. Instead, curricula validation is done against a model, formalized as a set of temporal constraints. We have developed a prototype of the planning and validation services, by using -as reasoning engines- SWI-Prolog and the SPIN model checker. Such services will be supplied and combined as plug-and-play personalization services in the Personal Reader framework.

1 Introduction and Motivation

The birth of the Semantic Web brought along standard models, languages, and tools for representing and dealing with machine-interpretable semantic descriptions of Web resources, by giving a strong new impulse to research on personalization. The introduction of machine-processable semantics makes the use of a variety of reasoning techniques for implementing personalization functionalities possible, widening the range of the forms that personalization can assume. So far, reasoning in the Semantic Web is mostly reasoning about knowledge expressed in some ontology. However personalization may involve also other kinds of reasoning and knowledge representation, that conceptually lie at the logic and proof layers of the Semantic Web tower.

Moreover, the next Web generation promises to deliver Semantic Web Services, that can be retrieved and *combined* in a way that satisfies the user. It opens the way to many forms of *service-oriented personalization*. Web services provide an ideal infrastructure for enabling *interoperability* among personalization applications and for constructing Plug&Play-like environments, where the user can

select and combine the kinds of services he or she prefers. Personalization can be obtained by taking different approaches, e.g. by developing services that offer personalization functionalities as well as by personalizing the way in which services are selected, and *composed* in order to meet specific user's requirements.

In the last years we carried on a research in the educational domain, by focussing on *semantic web* representations of learning resources and on *automated reasoning* techniques for enabling different and complementary personalization functionalities, e.g. curriculum sequencing [6, 7] and verification of the compliance of a curriculum against some course design goals [5]. Our current aim is to implement such results in an organic system, where different personalization services, that exploit semantic web reasoning, can be combined to support the user in the task of building a curriculum, based on *learning resources* that represent courses.

While in early times learning resources were simply considered as "contents", strictly tied to the platform used for accessing them, recently, greater and greater attention has been posed on the issue of *re-use* and of a *cross-platform* use of educational contents. The proposed solution is to adopt a *semantic annotation* of contents based on standard languages, e.g. RDF and LOM. Hereafter, we will consider a *learning resource* as formed by *educational contents* plus *semantic meta-data*, which supply information on the resources at a *knowledge level*, i.e. on the basis of concepts taken from an ontology that describes the educational domain. In particular we rely on the interpretation of learning resources as *actions* discussed in [6, 7]: the meta-data captures the *learning objectives* of the learning resource and its *pre-requisites*. By doing so,one can rely on a classical theory of actions and apply different reasoning methods -like *planning*- for building personalized curricula [6, 7]. The modeling of learning resources as actions also enables the use of model checking techniques for developing a validation service that detects if a user-given curriculum is compliant w.r.t an abstract model, given as a set of constraints. In the following we present our achievements in the implementation of a Planning service and a Validation service that can interoperate within the Personal Reader Framework [18].

Curriculum planning and validation offer a useful support in many practical contexts and can be fruitfully combined for helping students or teaching institutions. Often a student knows what competency he/she would like to acquire but has no knowledge of which courses will help him/her acquiring it. Moreover, taking courses at different Universities is becoming more and more common in Europe. As a consequence, building a curriculum might become a complicated task for students, who must deal with an enormous set of courses across the European countries, each described in different languages and on the basis of different keywords.

The need of personalizing the sequencing of learning resource, w.r.t. the student's interests and context, has often to be *combined* with the ability to check that the resulting curriculum *complies* against some abstract *curricula specification*, which encodes the *curricula-design goals* expressed by the teachers or by the institution offering the courses. Consider a student, who wants to build

a valid curriculum with the support of our automatic system. The student can either use as a basis the suggestion returned by the system or he/she can design the curriculum by hand, based on own criteria. In both cases a personalized curriculum is obtained and can be given in input to the validation service for checking the compliance against a curricula model. Curricula models specify general rules for building learning paths and can be interpreted as constraints designed by the University for guaranteeing the achievement of certain learning goals. These constraints are to be expressed in terms of knowledge elements, and maybe also on features that characterize the resources.

Consider now a university which needs to certify that the specific curricula, that it offers for achieving a certain educational goal, and that are built upon the courses offered locally by the university itself, respect some European guidelines. In this case, we could, in fact, define the guidelines as a set of constraints at an abstract level, i.e. as relations among a set of competencies which should be offered in a way that meets some given scheme. At this point the verification could be performed automatically, by means of a proper reasoner. Finally, the automatic checking of compliance combined with curriculum planning could be used for implementing processes like cooperation among institutes in curricula design and integration, which are actually the focus of the so called *Bologna Process* [15], promoted by the EU.

While SCORM [2] and Learning Design [19, 20] represent the most important steps in the direction of managing and using e-learning based courses and workflows among a group of actors participating in learning activities, most of the available tools lack the machine-interpretable information about the learning resources, and as a result they are not yet open for reasoning-based personalization and automatic composition and verification. Given our requirements, it is a natural choice to settle our implementation in the Personal Reader (PR) framework. The PR relies on a service-oriented architecture enabling personalization, via the use of semantic *Personalization Services*. Each service offers a different personalization functionality, e.g. recommendations tailored to the needs of specific users, pointers to related (or interesting or more detailed/general) information, and so on. These semantic web services communicate solely based on RDF documents.

The paper is organized as follows. Section 2 describes our approach to the representation and reasoning about learning resources, curricula, and curricula models. The implementation of the two services and their integration into the PR Framework is discussed in section 3. We finish with conclusions and hints on future work in Section 4.

2 Curricula representation and reasoning

Let us begin with the introduction of our approach to the representation of learning resources, curricula, and curricula models. The basic idea is to describe all the different kinds of objects, that we need to tackle and that we will introduce hereafter, on the basis of a set of predefined *competencies*, i.e. terms identifying

specific *knowledge elements*. We will use the two terms as synonyms. Competencies can be thought of, and implemented, as concepts in a shared ontology. In particular, for what concerns the application system described here, competencies were extracted by means of a semi-automatic process and stored as an RDF file (see Section 3.1 for details).

Given a predefined set of competencies, the initial knowledge of a student can be represented as a set of such concepts. This set changes, typically it grows, as the student studies and learns. In the same way, a user, who accesses a repository of learning resources, does it with the aim of finding materials that will allow him/her to acquire some knowledge of interest. Also this knowledge, that we identify by the term *learning goal*, can be represented as a set of knowledge elements. The learning goal is to be taken into account in a variety of tasks. For instance, the construction of a personalized curriculum is, actually, the construction of a curriculum which allows the achievement of a learning goal expressed by the user. In Section 3 we will describe a *curricula planning service* for accomplishing this task.

2.1 Learning resources and curricula

A *curriculum* is a sequence of *learning resources* that are homogeneous in their representation. Based on work in [6, 7], we rely on an *action theory*, and take the abstraction of resources as *simple actions*. More specifically, a learning resource is modelled as an action for acquiring some competencies (called *effects*). In order to understand the contents supplied by a learning resource, the user is sometimes required to own other competencies, that we call *preconditions*. Both preconditions and effects can be expressed by means of a *semantic annotation* of the learning resource [7]. In the following we will often refer to learning resources as "courses" due to the particular application domain that we have considered (university curricula).

As a simple example of "learning resource as action", let us, then, report the possible representation (in a classical STRIPS-like notation) of the course "databases for biotechnologies" (*db_for_biotech* for short):

ACTION: db_for_biothec(),
PREREQ: relational_db, EFFECTS: scientific_db

The prequisites to this action is to have knowledge about *relational databases*. Its effect is to supply knowledge about *scientific databases*.

Given the above interpretation of learning resources, a *curriculum* can be interpreted as a *plan*, i.e. as a sequence of actions, whose execution causes transitions from a state to another, until some final state is reached. The *initial state* contains all the competences that we suppose available before the curriculum is taken, e.g. the knowledge that the student already has. This set can also be empty. The *final state* is sometimes required to contain specific knowledge elements, for instance, all those that compose the user's learning goal. Indeed, often curricula are designed so to allow the achievement of a well-defined *learning goal*.

A transition between two states is due to the application of the action corresponding to a learning resource. Of course, for an action to be applicable, its preconditions must hold in the state to which it should be applied. The application of the action consists in an *update* of the state. We assume that competences can only be added to states. Formally, we assume that the domain is monotonic. The intuition behind this assumption is that the act of using a new resource will never erase from the students' memory the concepts acquired insofar. Knowledge grows incrementally.

2.2 Curricula models

Curricula models consist in sets of constraints that specify desired properties of curricula. Curricula models are to be defined on the basis of knowledge elements as well as of learning resources (courses). In particular, we would like to restrict the set of possible sequences of resources corresponding to curricula. This will be done by imposing constraints on the *order* by which knowledge elements are added to the states (e.g. "a knowledge element α is to be acquired before a knowledge element β"), or by specifying some *educational objectives* to be achieved, in terms of knowledge that must be contained in the final state (e.g. "a knowledge element α must be acquired sooner or later"). Therefore, we represent a curricula model as a set of *temporal constraints*. Being defined on knowledge elements, a curricula model is *independent* from the specific resources that are taken into account, for this reason, it can be *reused* in different contexts and it is suitable to open and dynamic environments like the web.

The possibility of *verifying the compliance of curricula to models* is extremely important in many applicative contexts, as explained by examples in the introduction. In some cases these checks could be integrated into the curriculum construction process; nevertheless, it is important to be able to perform the verification independently from the construction process. Let us consider again our simple scenario concerning a university, which offers a set of curricula that are proved to satisfy the guidelines given by the EU for a certain year. After a few years, the EU guidelines change: our University has the need to check if the curricula that it offers, still satisfy the guidelines, without rebuilding them.

A natural choice for representing temporal constraints on action paths is linear-time temporal logic (LTL) [14]. This kind of logic allows to verify if a property of interest is true for all the possible executions of a model (in our case the specific curriculum). This is often done by means of model checking techniques [12].

The curricula as we represent them are, actually, Kripke structures. Briefly, a Kripke structure identifies a set of states with a transition relation that allows passing from a state to another. In our case, the states contain the knowledge items that are owned at a certain moment. Since the domain is monotonic (as explained above we can assume that knowledge only grows), states will always contain *all* the competencies acquired up to that moment. The transition relation is given by the actions that are contained in the curriculum that is being checked.

It is possible to use the LTL logic to verify if a given formula holds starting from a state or if it holds for a set of states.

For example, in order to specify in the curricula model constraints on *what* to achieve, we can use the formula $\Diamond\alpha$, where $\Diamond$ is the eventually operator. Intuitively, such a formula expresses the fact that a set of knowledge elements will be acquired sooner or later. Moreover, constraints concerning *how* to achieve the educational objectives, such as "a knowledge element β cannot be acquired before the knowledge element α is acquired", can, for instance, be expressed by the LTL temporal formula $\neg\beta\ U\ \alpha$, where U is the *weak until* operator. Given a set of knowledge elements to be acquired, such constraints specify a partial ordering of the same elements.

2.3 Planning and Validation

Given a semantic annotation with preconditions and effects of the courses, classical planning techniques are exploited for creating *personalized curricula*, in the spirit of the work in [6, 7]. Intuitively the idea is that, given a repository of learning resources, which have been semantically annotated as described, the user expresses a *learning goal* as a set of *knowledge elements* he/she would like to acquire, and possibly also a set of already owned competencies. Then, the system applies planning to build a sequence of learning resources that, read in sequence, will allow him/her to achieve the goal.

The particular planning methodology that we implemented (see Section 3.3 for details) is a simple *depth-first forward planning* (an early prototype was presented in [3]), where actions cannot be applied more than once. The algorithm is simple:

1. Starting from the initial state, the set of *applicable* actions (those whose preconditions are contained in the current state) is identified.
2. One of such actions is selected and its application is simulated leading to a new state.
3. The new state is obtained by adding to the previous one the competencies supplied as effects of the selected action.
4. The procedure is repeated until either the goal is reached or a state is reached, in which no action can be applied and the learning goal is not satisfied.
5. In the latter situation, backtracking is applied to look for another solution.

The procedure will eventually end because the set of possible actions is finite and each is applied at most once. If the goal is achieved, the sequence of actions that label the transitions leading from the initial to the final state is returned as the resulting *curriculum*. If desired, the backtracking mechanism allows to collect a set of alternative solutions to present to the user.

Besides the capability of automatically building personalized curricula, it is also interesting to perform a set of verification tasks on curricula and curricula models. The simplest form of verification consists in *checking the soundness* of

curricula which are built by hand by users themselves, reflecting their own personal interests and needs. Of course, not all sequences which can be built starting from a set of learning resources are lawful. Learning dependencies, imposed by courses themselves in terms of preconditions and effects, must be respected. In other words, a course can appear at a certain point in a sequence only if it is *applicable* at that point, therefore, there are no *competency gaps*. These implicit "applicability constraints" capture precedences and dependencies that are innate to the nature of the taught concepts. In particular, it is important to verify that all the *competencies*, that are necessary to fully understand the contents, offered by a learning resource, are introduced or available before that learning resource is accessed. Usually, this verification, as stated in [13], is performed manually by the learning designer, with hardly any guidelines or support.

Given the interpretation of resources as actions, the verification of the *soundness of a curriculum*, w.r.t. the learning dependencies and the learning goal, can be interpreted as an *executability check* of the curriculum. Also in this case, the algorithm is simple:

1. Given an initial state, representing the knowledge available before the curriculum is attended, a simulation is executed, in which all the actions in the curriculum are (virtually) executed one after the other.
2. An action (representing a course) can be executed only if the current state contains all the concepts that are in the course precondition. Intuitively, it will be applied only if the student owns the notions that are required for understanding the topics of the course.
3. If, at a certain point, an action that should be applied is *not applicable* because some precondition does not hold, the verification fails and the reasons of such failure can be reported to the user.
4. Given that all the courses in the sequence can be applied, one after the other, the final state that is reached must be compared with the learning goal of the student: all the desired goal concepts must be achieved, so the corresponding knowledge elements must be contained in the final state.

This latter task actually corresponds to another basic form of verification, i.e. to check whether a (possibly hand-made) curriculum allows the *achievement of the desired learning goal*. These forms of basic verifications can be accomplished by the service described in Section 3.4.

Another interesting verification task consists in checking if a *personalized curriculum is valid w.r.t. a particular curricula model* or, following Brusilovski's terminology, checking if the curriculum is *compliant against the course design goals* [11]. Indeed, a personalized curriculum that is proved to be executable, cannot automatically be considered as being *valid* w.r.t. a particular *curricula model*. A curricula model, in fact, imposes further constraints on *what* to achieve and *how* achieving it. We will return to this kind of verification in Section 3.4.

3 Implementation in the Personal Reader Framework

The Personal Reader Framework has been developed with the aim of offering a uniform entry point for accessing the Semantic Web, and in particular Semantic Web Services. Indeed it offers an environment for designing, implementing and realizing Web content readers in a service-oriented approach, for a more detailed description, see [18] (`http://www.personal-reader.de/`).

In applications based on the Personal Reader Framework, a user can select and combine —plug together— which personalized support he or she wants to receive. The framework has already been used for developing Web Content Readers that present online material in an embedded context [10, 1, 17]. Besides

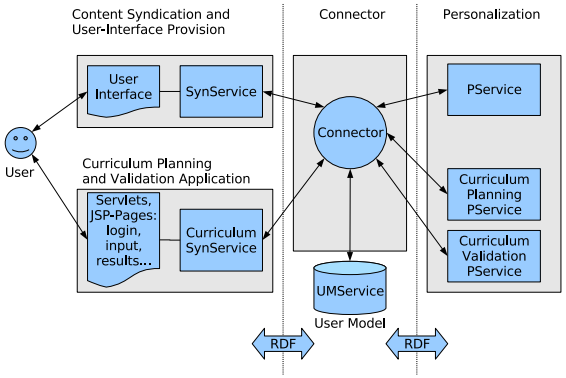


Fig. 1. Personal Reader Framework Overview

a user-interface, as shown in figure 1, a Personal Reader application consists of three types of *services*. *Personalization services* (PService) provide personalization functionalities: they deliver personalized recommendations for content, as requested by the user and obtained or extracted from the Semantic Web. *Syndication Services* (SynService) allow for some interoperability with the other services in the framework, e.g. for the discovery of the applications interfaces by a portal. The *Connector* is a single central instance responsible for all the communication between user interface and personalization services. It selects services based on their semantic description and on the requirements by the SynService. The Connector protects –by means of a public-key-infrastructure (PKI)– the communication among the involved parties. It also supports the customization and invocation of services and interacts with a user modelling service, called the *UMService*, which maintains a central user model.

3.1 Metadata Description of Courses

In order to create the corpus of courses, we started with information collected from an existing database of courses. We used the Lixto [9] tool to extract the needed data from the web-pages provided by the HIS-LSF (http://www.his.de/) system of the University of Hannover. This approach was chosen based on our experience with Lixto in the *Personal Publication Reader* [10] project, where we used Lixto for creating the publications database by crawling the publication pages of the project partners. The effort to adapt our existing tool for the new data source was only small. From the extracted metadata we created an RDF document, containing course names, course catalog identifier, semester, number of credit points, effects and preconditions, and the type of course, e.g. laboratory, seminar or regular course with examinations in the end, as illustrated in Figure 2.

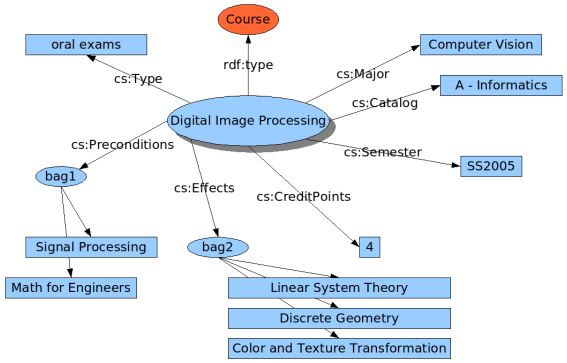


Fig. 2. An annotated course from the Hannover course database

The larger problem was that the quality of most of the information in the database turned out to be insufficient, mostly due to inconsistencies in the description of prerequisites and effects of the courses. Additionally the corpus was not annotated using a common set of terms, but authors and department secretaries used a slightly varying vocabulary for each of their course descriptions, instead of relying on a common classification system, like e.g. the ACM CCS for computer science.

As a consequence, we focussed only on a subset of the courses (computer science and engineering courses), and manually post-processed the data. Courses are annotated with prerequisites and effects, that can be seen as knowledge concepts or competences, i.e. ontology terms. After automatic extraction of effects

and preconditions, the collected terms were translated into proper English language, synonyms were removed and annotations were corrected where necessary. The resulting corpus had a total of 65 courses left, with 390 effects and 146 preconditions.

3.2 The User Interface and Syndication Service

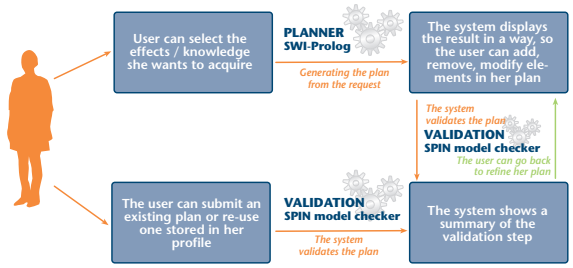


Fig. 3. The Actions supported by the User Interface

In our implementation, the user interface (see figure 3) is responsible for identifying the user, presenting the user an interface to select the knowledge she wants to acquire, and to display the results of the planning and validation step, allowing further refinement of created plans. The creation of curriculum sequences and the validation are implemented as two independent Personalization Services, the "Curriculum Planning PService", and the "Curriculum Validation PService". Because of the plug-and-play nature of the infrastructure, the two PServices can be used by other applications (SynServices) as well (Fig. 3). Also possible is that PServices, which provide additional planning and validation capabilities can be used in our application. The current and upcoming future implementations of the Curriculum Planning and Validation Prototype are available at `http://semweb2.kbs.uni-hannover.de:8080/plannersvc`.

3.3 The Curriculum Planning PService

In order to integrate the Planning Service as a plug-and-play personalization service in the Personal Reader architecture we worked at embedding the Prolog reasoner into a web service. Figure 4 gives an overview over the components in the current implementation. The web service implements the Personalization

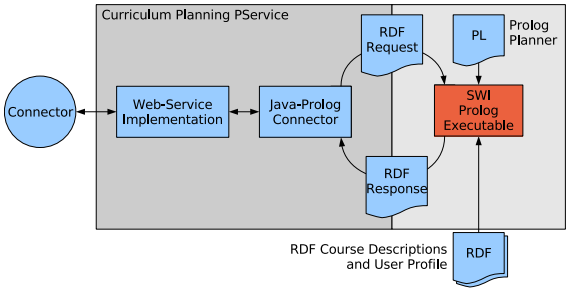


Fig. 4. Curriculum Planning Web Service

Service (*PService* [18]) interface, defined by the Personal Reader framework, which allows for the processing of RDF documents and for inquiring about the services capabilities. The *Java-to-Prolog Connector* runs the SWI-Prolog executable in a sub-process; essentially it passes the RDF document containing the request *as-is* to the Prolog system, and collects the results, already represented as RDF.

The curriculum planning task itself is accomplished by a reasoning engine, which has been implemented in SWI Prolog[3]. The interesting thing of using SWI Prolog is that it contains a semantic web library allowing to deal with RDF statements. Since all the inputs are sent to the reasoner in a *RDF request document*, it actually simplifies the process of interfacing the planner with the Personal Reader. In particular the request document contains: a) links to the RDF document containing the database of courses, annotated with metadata, b) a reference to the user's context c) the user's actual learning goal, i.e. a set of knowledge concepts that the user would like to acquire, and that are part of the *domain ontology* used for the semantic annotation of the actual courses. The reasoner can also deal with information about credits provided by the courses, when the user sets a credit constraint together with the learning goal.

Given a request, the reasoner runs the Prolog planning engine on the database of courses annotated with prerequisites and effects. The initial state is set by using information about the user's context, which is maintained by the User Modelling component of the PR. In fact such user's context includes information about what is considered as already learnt by the student (attended courses, learnt concepts) and such information is included in the request document. The Prolog planning engine has been implemented by using a classical depth-first search algorithm [22]. This algorithm is extremely simple to implement in declarative languages as Prolog.

[3] `http://www.swi-prolog.org/`

At the end of the process, a *RDF response document* is returned as an output. It contains a list of plans (sequences of courses) that fulfill the user's learning goals and profile. The maximum number of possible solutions can be set by the user in the request document. Notice that further information stored in the user profile is used at this stage for adapting the presentation of the solutions, here simple hints are used to *rank higher* those plans that include topics that the user has an expressed special interest in.

3.4 The Curriculum Validation PService

In order to verify if a curriculum is valid w.r.t. a curricula model, we adopt *model checking* techniques, by using SPIN. To check a curriculum with SPIN, this must be translated in the Promela language. *Competencies* are represented as *boolean variables*. In the beginning, only those variables that represent the initial knowledge of the student are true. *Courses* are implemented as actions that can modify the value of the variables. Since our application domain is monotonic, only those variables, whose value is false in the initial state, can be modified.

The Promela program consists of two processes: one is named *CurriculumVerification* and the other *UpdateState*. While the former contains a representation of the curriculum itself, and simulates its execution, the latter contains the code for updating the state (i.e. the set of competencies achieved so far) step by step along the simulation of the execution of the curriculum. The two processes communicate by means of two channels, *attend* and *feedback*. The notation *attend!courseName* represents the fact that the course with name *courseName* is to be attended. In this case the sender process is *CurriculumVerification* and the receiver is *UpdateState*. *UpdateState* will check the preconditions of the course in the current state and will send a feedback to *CurriculumVerification* after updating the state. On the other hand, the notation *feedback?feedbackMsg* represents the possibility for the process *Curriculum* of receiving a feedback of kind *feedbackMsg* from the process *UpdateState*.

Given these two processes, it is possible to perform a test, aimed at verifying the possible presence of competency gaps. This test is implemented as a *deadlock* verification: if the sequence is correct w.r.t. the action theory, no deadlock arises, otherwise a deadlock will be detected. The *curricula model* is to be supplied apart, as a set of temporal logic formulas, possibly obtained by an automatic translation process from a DCML representation. Notice that curricula can contain branching points. The branching points are encoded by either conditioned or non-deterministic *if*; each such *if* statement refers to a set of alternative courses (e.g. *languagesEnvironmentProg* and *programmingLanguages*). Depending on the course communicated by the channel *attend*, it updates the state. The process continues until the message *stop* is communicated. Then the learning goal is checked.

Let us see how to use the model checker to verify the *temporal constraints* that make a curricula model. Model checking is the algorithmic verification of the fact that a finite state system complies to its specification. In our case the

specification is given by the curricula model and consists of a set of temporal constraints, while the finite state system is the curriculum to be verified.

SPIN allows to specify and verify every kind of LTL formulas and it also allows to deal with curricula that at some points contain alternatives. This makes the system suitable to more realistic application scenarios. In fact, for what concerns curricula written by hand, users often do not have a clear mind and, thus, it is difficult for them to write a single sequence. In the case of curricula built by an automatic system, there are planners that are able to produce sets of alternative solutions gathered in a tree structure.

The following are examples of constraints, expressed as LTL formulas, that could be part of a curricula model:

(1) $\neg jdbc \,\mathsf{U}\, (sql \wedge relational\_algebra)$,
(2) $\neg op\_systems \,\mathsf{U}\, basis\_of\_prog$,
(3) $\neg basis\_of\_oo \,\mathsf{U}\, basis\_of\_prog$,
(4) $\Diamond basis\_of\_prog \supset \Diamond basis\_of\_java\_prog$,
(5) $\Diamond database$,
(6) $\Diamond web\_services$.

The first constraint means that before learning *jdbc* the student must own Knowledge about *sql* and about *relational algebra*. The following two constraints are of the same kind but involve different competencies. Constraint (4) means that if the student acquires knowledge about "basis of programming", he/she will also have knowledge about "basis of java programming" but the two events are not temporally related. Constraints (5) and (6) mean that soon or later knowledge about databases and web services must be acquired.

4 Conclusion, Further and Related Works

In this work we have described the current state of the integration of semantic personalization web services for Curriculum Planning and Validation within the Personal Reader Framework. The goal of personalization is to create sequences of courses that fit the specific context and the learning goal of individual students. Despite some manual post-processing for fixing inconsistencies, we used real information from the Hannover University database of courses for extracting the meta-data. Currently the courses are annotated also by meta-data concerning the schedule and location of courses, like for instance room-numbers, addresses and teaching hours. As a further development, it would be interesting to let our Curriculum Planning Service to make use also of such metadata in order to find a solution that fits the desires and the needs of the user in a more complete way.

The Curriculum Planning Service has been integrated as a new plug-and-play personalization service in the Personal Reader framework. In the current implementation, the learning goal corresponds to a set of hard constraints; that is to say that the planner returns only plans that satisfy them *all*. A different choice would be to consider the constraints given by the goal as *soft* constraints, and allow the return of plans which do satisfy the goal only partially. This

would be approapriate, for instance, in the case in which a student would like to acquire a range of competencies of interest but it is not possible to build, on top of a given repository of course descriptions, a curriculum for achieving them all. Nevertheless, it would be possible to build a curriculum for achieving *part* of them. In some circumstances, it would anyway be helpful for the student to receive this information as a feedback. Of course, in this case many questions arise, e.g. the issue of ranking the goals based on the actual interest of the requestor, so to know what can possibly be discarded and what is mandatory. From an implementation perspective, the spirit of the SOA infrustructure given to the Personal Reader is, indeed, meant to easily allow extensions by adding new Personalization Services. We can, therefore, think to develop and add a soft-goal planning service, to be used in these circumstances. The new planner would inherit the wrapping and interaction part from the current planning service but implement an algorithm like for instance [16].

The Curriculum Validation Service has been designed. An early prototype of the validation system based on the model checker SPIN has been developed [5] and is currently being embedded in the same framework. The choice of relying on SPIN, rather than developing a simpler and ad hoc checking system, is due to the need of rapidly developing a prototype. For this reason we have decided to rely on already exisiting and well-established technology. The engineering of the developed services should be tailored to the specific kinds of constraint that can be used to design the model. Analogous considerations can be done for the planning algorithm. The one that has been used is the simplest that can be thought of. Of course, there are many possible optimizations and extensions (e.g. the adoption of soft goals mentioned above) that could be done, and many algorithms are already available in the literature. Our choice has been motivated by the desire of quickly testing our ideas rather than developing a system thought for real use.

The Personal Reader Platform provides a natural framework for implementing a service-oriented approach to personalization in the Semantic Web, allowing to investigate how (semantic) web service technologies can provide a suitable infrastructure for building personalization applications, that consist of re-usable and interoperable personalization functionalities. The idea of taking a service oriented approach to personalization is quite new and was born within the personalization working group of the Network of Excellence REWERSE (Reasoning on the Web with Rules and Semantics, http://rewerse.net).

Writing curricula models directly in LTL is not an easy task for the user. For this reason, we have recently developed a graphical language, called DCML (Declarative Curricula Model Language) [8, 4], inspired by DecSerFlow, the Declarative Service Flow Language by van der Aalst and Pesic [23]. DCML allows to express the temporal relations between the times of acquisition of the concepts. 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. At the same time, a rigorous and precise meaning is also given, due to the logic grounding of

the language. Moreover, in [4] we represent curricula as UML activity diagrams and include the possibility of handling the concurrent attending of courses. Also in this case curricula can be translated in Promela programs so that it becomes possible to perform all the kinds of verification that we have described.

DCML, besides being a graphical language, has also a textual representation. We are currently working at an integration of this new more sophisticated solution into the Personal Reader Framework by implementing an automatic system for translating DCML textual representations into LTL, for translating curricula (activity diagrams) in Promela, and then run the checks.

Another recent proposal for automatizing the competency gap verification is done in [21] where an analysis of pre- and post-requisite annotations of the Learning Objects (LO), representing the learning resources, is proposed. In this approach, whenever an error will be detected by the validation phase, a correction engine will be activated. This engine will use a "Correction Model" to produce suggestions for correcting the wrong curriculum, by means of a reasoning-by-cases approach. The suggestions will, then, be presented to the course developer, who is in charge to decide which ones to adopt (if any). Once a curriculum will have been corrected, it will have to be validated again, because the corrections might introduce new errors. Melia and Pahl's proposal is inspired by the CocoA system [11], that allows to perform the analysis and the consistency check of static web-based courses. Competency 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 by knowledge elements.

Acknowledgement 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`).

References

1. F. Abel, I. Brunkhorst, N. Henze, D. Krause, K. Mushtaq, P. Nasirifar, and K. Tomaschweski. Personal reader agent: Personalized access to configurable web services. Technical report, Distributed Systems Institute, Semantic Web Group, University of Hannover, 2006.
2. Advanced Distributed Learning Network. SCORM: The sharable content object reference model, 2001. http://www.adlnet.org/Scorm/scorm.cfm.
3. M. Baldoni, C. Baroglio, I. Brunkhorst, N. Henze, E. Marengo, and V. Patti. A Personalization Service for Curriculum Planning. In E. Herder and D. Heckmann, editors, *Proc. of the 14th Workshop ABIS*, pages 17–20, Hildesheim, Germany, October 2006.
4. M. Baldoni, C. Baroglio, and E. Marengo. Curricula Modeling and Checking. In *Proc. of AI*IA 2007: Advances in Artificial Intelligence*, volume 4733 of *LNAI*, pages 471–482. Springer, 2007.
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 the Semantic Web Personalization Workshop*, pages 53–62, Budva, Montenegro, 2006.
6. M. Baldoni, C. Baroglio, and V. Patti. Web-based adaptive tutoring: An approach based on logic agents and reasoning about actions. *Artificial Intelligence Review*, 1(22):3–39, 2004.
7. M. Baldoni, C. Baroglio, V. Patti, and L. Torasso. Reasoning about learning object metadata for adapting SCORM courseware. In L. Aroyo and C. Tasso, editors, *Int. Workshop on Engineering the Adaptive Web, EAW'04*, pages 4–13, 2004.
8. M. Baldoni and E. Marengo. Curriculum Model Checking: Declarative Representation and Verification of Properties. In *Proc. of 2nd Eur. Conf. EC-TEL*, volume 4753 of *LNCS*, pages 432–437. Springer, 2007.
9. R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *VLDB*, pages 119–128. Morgan Kaufmann, 2001.
10. R. Baumgartner, N. Henze, and M. Herzog. The personal publication reader: Illustrating web data extraction, personalization and reasoning for the semantic web. In *ESWC*, pages 515–530, 2005.
11. 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.
12. O. E. M. Clarke and D. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 2001.
13. Juri L. De Coi, Eelco Herder, Arne Koesling, Christoph Lofi, Daniel Olmedilla, Odysseas Papapetrou, and Wolf Sibershi. A model for competence gap analysis. In *Proc. of WEBIST 2007*, 2007.
14. E. A. Emerson. Temporal and model logic. In *Handbook of Theoretical Computer Science*, volume B, pages 997–1072. Elsevier, 1990.
15. European Commission, Education and Training. The Bologna process. `http://ec.europa.eu/education/policies/educ/bologna/bologna_en.html`.
16. E. Giunchiglia and M. Maratea. SAT-based planning with minimal-actions plans and "soft" goals. In *Proc. of AI*IA 2007: Advances in Artificial Intelligence*, volume 4733 of *LNAI*. Springer, 2007.
17. N. Henze. Personal readers: Personalized learning object readers for the semantic web. In *12th International Conference on Artificial Intelligence in Education, AIED05*, Amsterdam, The Netherlands, 2005.
18. N. Henze and D. Krause. Personalized access to web services in the semantic web. In *The 3rd International Semantic Web User Interaction Workshop (SWUI, collocated with ISWC 2006*, November 2006.
19. IMSGlobal. Learning design specifications. Available at `http://www.imsglobal.org/learningdesign/`.
20. R. Koper and C. Tattersall. *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*. Springer Verlag, 2005.
21. M. Melia and C. Pahl. Automatic Validation of Learning Object Compositions. In *Information Technology and Telecommunications Conference IT&T'2005: Doctoral Symposium*, Carlow, Ireland, 2006.
22. S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.
23. W. M. P. van der Aalst and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In Mario Bravetti and Gialuigi Zavattaro, editors, *Proc. of WS-FM*, LNCS, Vienna, September 2006. Springer.

Integrating Agents, Ontologies, and Web Services to Build Flexible Sketch-based Applications

Giovanni Casella[1,2] and Vincenzo Deufemia[2]

[1] Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova
Via Dodecaneso 35, 16146, Genova, Italy
casella@disi.unige.it
[2] Dipartimento di Matematica e Informatica – Università di Salerno
Via Ponte don Melillo, 84084 Fisciano (SA), Italy
deufemia@unisa.it

Abstract. We present an approach based on web services, for building open and dynamic agent societies aimed at hand-drawn sketch recognition. The approach exploits ontologies to enable agents to agree on message semantics and service purposes, standard web services languages to represent agent interaction protocols in a suitable way to be exchanged and handled by agents and web services to expose low-level recognition services. The communication mechanisms that characterize our approach, as well as the modular architecture allow agent societies to self-organize at run time, for gaining the capability of recognizing new domain languages, thus obtaining new flexible sketch-based applications.

1 Introduction

Sketching provides a natural way for humans to design (buildings, software, electronic circuits, and so on), to communicate and cooperate, to share ideas, to transfer information. As an example, sketching allows an architect, or engineer to quickly specify a design. Architects make exploratory sketches before making more definitive schematic design drawings and models, and finally construction and fabrication drawings. Mechanical engineers make sketches as part of a process that also includes calculations, material specifications, and detailed design drawings.

Computers can support users in the sketching process only if they are able to understand the sketch semantics, i.e., they can recognize what the user sketches represent. However, since hand-drawn input tends to be highly variable and inconsistent sketch interpretation turns out to be a difficult task. Sketch recognition systems must robustly cope with the variations and ambiguities inherent in hand drawings, facing the task of grouping a user's pen strokes into clusters representing intended symbols, the task of identifying several symbols in one single stroke, and so on.

In [1] we have exploited intelligent agents to face the diagrammatic sketch recognition problem. The use of agents was inspired by the observation that the

"virtual blank sheet" where the user draws represents a dynamic and unpredictable environment, and the entities devoted to recognize the symbols of some language must be responsive, pro-active, situated, autonomous, and social. In [2] we have presented a Multi-Agent System (MAS) that makes use of collaborating intelligent agents to coordinate a set of heterogeneous symbol recognizers and to generate a sketch interpretation.

This system presents many features suitable for sketch understanding, but it also shows several limitations. In particular, even if the MAS can be built to recognize any domain language, it is impossible to change the domain language while the system is running. This is mainly useful when the users exploit sketches to express new ideas (e.g., an interior architect may wish to add a new shape in its sketched design to represent a new decorative element, such as a flower vase) or when the set of symbols to be recognized can evolve, as for example, in the domain of hieroglyph recognition where new symbols can still be discovered. Another limitation is that if we have a MAS able to recognize a set of symbols S_1, such as use case diagrams, it is not possible to use it to recognize a set of symbols S_2, such as finite state machines, even if S_1 and S_2 share some symbols. Moreover, the MAS does not allow users to integrate interfaces customized to specific tasks. As an example, if the MAS has been designed to facilitate students in the specification of assignment's solutions, it cannot be changed, at run time, into a system that enables teachers to specify solutions to assignments and automatically evaluate student work.

These limitations are mainly due to the lack of flexibility of the agent organization and interactions that, once specified at design time, cannot be adapted to the domain and users' needs at run-time. In this paper, we face these problems by giving to the agents in the MAS suitable means

- to look for the services provided by the other agents,
- to interact with the other agents following heterogeneous interaction protocols, and
- to be able to understand the message semantic and the meaning of the services offered by each agent.

In this way it is possible to recognize new languages and to realize new sketch-based applications changing the interactions between existing agents and/or adding new agents at run-time. In particular, we propose to use ontologies to enable our agents to agree on message semantics and service purposes, and standard web service languages to represent AIP in a way suitable to be exchanged and handled by agents. Web services have also be used to expose low-level recognition services.

The paper is organized as follows. Section 2 introduces the sketch understanding problem and our previously proposed solution. In Section 3 we present the proposed open society for sketch understanding. Section 4 describes how agents publish, reason and learn agent interaction protocols, while Section 5 defines web services for sketch recognition. Finally, Section 6 contains a discussion of related work and conclusions.

2 Sketch Recognition and a MAS to Support it

Sketches are informal drawings created by people to represent abstract concepts and acquired by computers in the format of point chains. The pen trajectory on the screen between each pair of pen-down and pen-up operations, i.e., a unit of user's original sketch input, is named *stroke*. As an example, the human stick figure depicted in Fig. 1(a) is composed of four strokes.

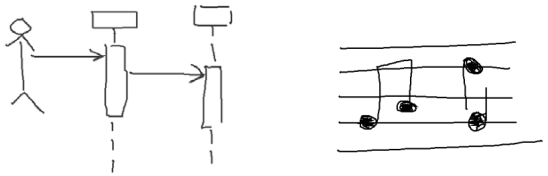


Fig. 1. Two examples of sketches: a sequence diagram (a) and a musical score (b).

Usually, the interpretation of a sketch is performed by classifying the strokes into primitive geometric objects, such as line, arc, and ellipse, and by clustering the primitive shapes into a set of intended symbols. As an example, a human stick is recognized by clustering five line strokes and one ellipse stroke properly related. However, it is worth to note that, according to the user drawing style, the shape of the abstract concepts, namely the symbols, can be drawn using a varying number of strokes (e.g., the two upper rectangles in Fig. 1(a) have been drawn with one stroke and three strokes, respectively), can contain ambiguities (e.g., the recognizer can associate the head of the leftmost note in Fig. 1(b) both to its right and left stems), or be incomplete. These issues make the recognition of sketches a very critical task.

To minimize the recognition mistakes many systems have constrained the user's drawing style (e.g., enforcing users to carefully draw each symbol with a single stroke) making the recognition process easier. However, in order to be really usable and useful in practice, sketch recognition systems should not place constraints on how the users can draw symbols. Indeed, users should be able to draw without having to worry about where to start a stroke, how many strokes to use, in what order to draw the strokes, etc. Beside this, in order to be flexibly adapted to new needs and visual domain languages the recognition system should be able to easily integrate new symbol recognizers without needing to change any other component.

The multi-agent approach to sketch recognition proposed in [2] implements a sketch recognition system having the above features. In particular, it

- manages the variation in drawing style by an ink parsing process that groups and segments the user's strokes into clusters of intended symbols;
- solves the ambiguities due to the possible membership of one stroke to more than one symbol, by analyzing the objects surrounding the ambiguous parts, i.e., the context around them;
- coordinates the behavior of the symbol recognizers in such a way to detect and solve conflicting interpretations of symbols;
- integrates in a seamless way heterogeneous symbol recognizers in order to exploit different techniques for recognizing different symbols.

The recognition approach is based on a MAS composed by intelligent cooperating agents with specific tasks. The MAS contains a set of Symbol Recognizer Agents (SRAs) devoted to recognize the symbols (i.e., their shapes) of a given domain. Each SRA uses an internal recognition algorithm to recognize all the instances of a particular symbol in the sketch, and is able to exchange feedback messages with other SRAs in order to compute contextual information. In particular, when an SRA recognizes a symbol S the belief that S is a correct interpretation increases if other SRAs have recognized symbols that are related with S.

The set of recognized symbols, together with the collected feedbacks, are sent to the Sketch Interpreter Agent (SIA) that analyzes and solves the possible arising conflicts, where a conflict occurs when two or more recognized shapes share one or more strokes.

In the following we exemplify the behavior of the MAS on the UML Use Case Diagram notation. This notation is characterized by the graphical symbols depicted in Fig. 2 and a set of permitted relations between them. Such diagrams are composed of use cases, actors, and connectors among them. In particular, there is only one type of relationship that may occur between actors and use cases; it is visualized like a solid line, named communication link. Four types of relationships between use cases are supported by UML: communication, inclusion, extension (visualized as the inclusion but with label $<<extend>>$), and generalization. The only type of relationships that may hold among actors is generalization.

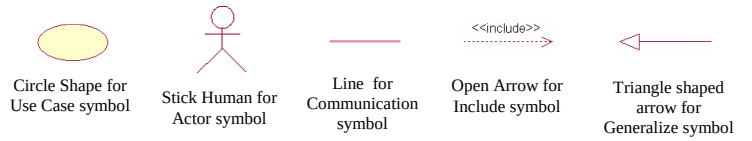


Fig. 2. Use Case Diagram symbols.

Let us suppose that the user draws a Use Case diagram composed by an actor that participates through a communication symbol to a use case, as shown

in Fig. 3. Each SRA in the MAS analyzes the sketch to recognize a particular Use Case symbol. The Actor SRA recognizes actor A_1 (surrounded by a dotted box in the figure). The Communication SRA recognizes C_1 and C_2, where C_1 is part of the actor A_1. Use Case SRA recognizes U_1 and U_2, while Include SRA recognizes I_1. Generalize and Extend SRAs do not recognize any symbol. For each recognized symbol each SRA requests a feedback to the proper SRAs. In particular, the following feedback messages are exchanged:

- Communication SRA obtains a feedback for C_1 from Use Case SRA since it recognized U_1 (and vice versa);
- Include SRA obtains a feedback for I_1 from Use Case SRA since it recognized U_1 (and vice versa);
- Communication SRA obtains a feedback for C_2 from Actor SRA since it recognized A_1 (and vice versa);
- Use Case SRA obtains a feedback for U_2 from Communication SRA since it recognized C_2 (and vice versa).

Finally, each SRA sends the recognized symbols and their feedback to the SIA agent that has to detect and solve conflicts. In particular, A_1, C_1, and U_1 are in conflict, while U_2 and C_2 are considered "unambiguous symbols" because are not in conflict. The unambiguous symbols and the collected feedback are used by the SIA to solve the conflicts. In particular, the SIA applies the following reasoning: A_1 has a feedback from C_2 that is unambiguous, while C_1 and I_1 have a feedback from U_1, but C_1, I_1, and U_1 are in conflict. Then A_1 has been correctly recognized, while C_1 and U_1 have been misrecognized. After the conflict resolution the SIA interprets the sketch as an Actor A_1, a Communication C_2, and a Use Case symbol U_2. The SIA reasoning and the SRAs behavior are detailed in [1].

3 An Open Society for Sketch Understanding

Fig. 4 shows the proposed society of agents and services for sketch understanding, named *AgentSketch*. The society extends the MAS proposed in [1] to support the building of flexible sketch-based applications that can be easily applied across a variety of domains. Indeed, it can be configured to recognize different domain languages and can be extended with new functionalities by adding new agents to the society at run-time.

The features previously described are possible thanks to the proposed architecture, composed by agents and web services, and to the use of suitable ontologies and web service (WS) languages. The latter play a central role to enable agents developed by different organizations and with heterogeneous internal behaviors to interact and to join the society at run-time.

We have identified four groups of agents to build the society:

- **Symbol Recognition Group**: Agents belonging to this group, namely *Symbol Recognizer Agents* (SRAs), are able to recognize a particular domain symbol and to collaborate with other SRAs in order to obtain contextual information on their recognized symbols. Each SRA interacts with a Shape Recognizer Web Service to recognize a symbol.
- **Sketch Interpretation Group**: Agents that belong to this group, namely *Sketch Interpreter Agents* (SIAs), are able to coordinate the recognition process of a set of SRAs by reasoning on the recognized symbols and elaborating an interpretation of the whole sketch.
- **Domain Expert Group**: A *Domain Expert Agent* (DEA) is able to reason on the sketch interpretations provided by SIAs to face a domain specific task. For example, a DEA could support the user to correctly design circuits by reasoning on the diagrams representing them. Another DEA could be able to reason on Use Case diagrams to help the user to enhance their clarity.
- **Intelligent Interface Group**: *Intelligent Interface Agents* (IIAs) are able to interact with the user in order to enable him/her to draw a diagrammatic sketch and to support him/her in solving a particular task working on the sketch interpretation. An IIA is designed to work on a particular domain language and is customized for a particular purpose. Each IIA collaborates with a SIA to obtain the interpretation of the user sketch and with one or more DEAs to perform some tasks on a previously interpreted sketch.

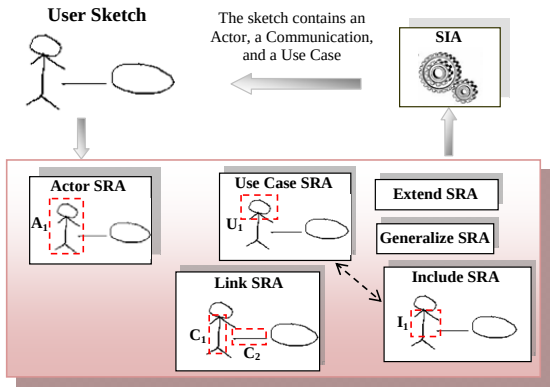


Fig. 3. Use Case diagram interpretation.

In order to support agent interactions we have included in our architecture two services, the "*Agent Directory Service*" and the "*Ontology Agent Service*" introduced by the "Abstract Architecture Specification" [3] and by the "Ontology Service Specification" [4], respectively.

An agent uses the Agent Directory Service to register itself and the services that it is able to provide in the "*Agent Directory*". Moreover, an agent queries

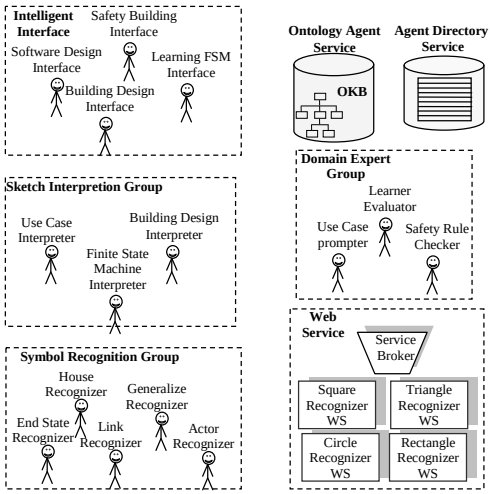


Fig. 4. The AgentSketch society architecture.

the Agent Directory to find other agents able to provide the services that it needs.

We have included in our framework a shared ontological knowledge base, namely AgentSketch OKB, to enable agents to agree on message semantics and service purposes. The Ontology Agent Service supports our community of agents providing services to discover and browse the ontology, and to add instances to the ontology concepts. The OKB will be described in Section 3.1.

Finally, a set of Web Services support SRAs to recognize hand-drawn shapes by offering suitable shape recognition implementations. Agents can find these services through the "Service Broker" also included in our society. Both the Agent Directory and the Service Broker enable agents to discover services, but while the first is suitable to contain agent's information (name, description, interaction protocols) and is used by all the agents, the latter is suitable to contain WSs information (address, WSDL description, and so on) and is used only by SRAs. The WSs of AgentSketch are detailed in Section 5.

The AgentSketch society depicted in Fig. 4 includes WSs able to recognize some basic shapes (i.e., square, arrow, circle, and so on), SRAs able to recognize symbols (i.e., Actor, Generalize, End State, and so on), SIAs able to interpret sketch belonging to different domain languages (i.e., Use Case, Finite State Machine, and so on), IIAs useful for different tasks (i.e., Software Design, Building Design, Learning), and finally, DEA able to furnish domain-specific services (i.e., prompt hints to enhance a Use Case, check if a building design satisfies a set of

security constraints, check if a Use Case diagram is correct for learning purposes). New agents can be added at run-time to increase the society capabilities.

3.1 AgentSketch Ontological Knowledge Base

As stated in [5] to become a member of a society an agent must agree to adhere to the constraints of the system, and in return the agent can benefit from the other members of the society, e.g., their knowledge or services. When an agent enters in the AgentSketch society it must agree to use the AgentSketch OKB to properly communicate with other agents and to understand the services they provide. AgentSketch OKB is shown in Fig. 5.

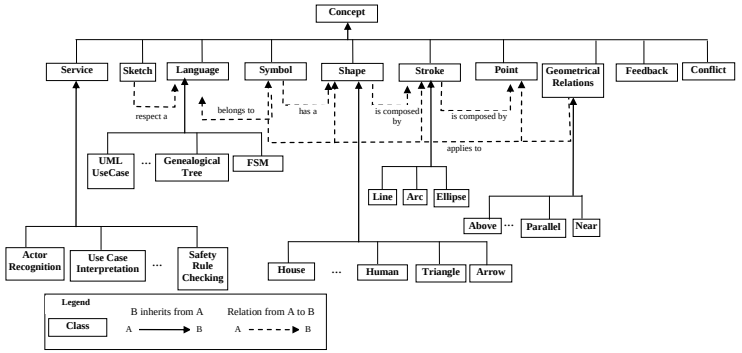


Fig. 5. AgentSketch ontological knowledge base.

Since in AgentSketch society the recognition of a single shape is performed by a single agent, the ontology does not contain low-level concepts about shape recognition, such as shape structure, aggregate of shapes, shape features, and so on. Moreover, the agents do not discuss about their intentions or goals (intentions and goals of each kind of agent are implicitly defined by the services they offer as detailed in the following) so these concepts are not modeled in the ontology.

A common issue is that usually an agent also has an internal ontology used to represent its knowledge and to perform reasoning. In order to use the internal ontology and the shared ontology in a consistent way the agent needs to semantically relate its internal ontology with the shared one. In literature many general approaches for ontology mapping are available [6].

In AgentSketch OKB the class *Concept* acts just as the ontology root: all other classes represent concepts. The class *Service* represents a service offered by an agent. Example of services are: to recognize a particular symbol, to interpret a sketch belonging to a given language, to check if a diagrammatic sketch

satisfies the language constraints, to suggest users a sketch re-arrangement, and so on. The *Sketch* class represents a diagrammatic hand-drawn sketch, i.e., a diagram belonging to a particular language. A *Language* represents a visual domain language and can be described in terms of symbols and rules (i.e., UML Use Case, Genealogy Trees, and so on). In particular, a *Language* is composed of a set of symbols, while the rules define allowed relationships between symbols. A *Symbol* belongs to a particular *Language* and has a particular *Shape*. For example, the Include symbol belongs to the UML use case diagrams and its shape is an Arrow. A *Stroke* represents a user stroke (or a segment of it) and it is composed of a point chain. *Geometrical relations* apply to symbols, shapes, strokes, and points. *Above*, *Under*, *LeftOf*, *Parallel*, *Near* are instances of geometrical relations. The *Feedback* concept represents a feedback sent by an SRA to another about symbol recognition in order to compute contextual information. The *Conflict* concept represents a conflict between two or more recognized symbol interpretations.

3.2 Symbol Recognition and Sketch Interpretation Groups

An SRA is designed to interact with a Shape Recognizer WS for recognizing a particular domain symbol and to collaborate with other SRAs to compute contextual information (feedback exchange). In particular, the WS includes an algorithm to recognize a shape, while the SRA is able to handle the recognized shape as a domain symbol extracting the meaningful features and interacting with other agents to obtain the symbol context. Considering for example the use case diagrams, the SRA devoted to recognize the Include symbol interacts with the WS providing the shape recognition service for the Arrow shape, and collaborates with the SRA devoted to recognize the Use Case symbol.

The role of SIAs is to interpret diagrammatic hand-drawn sketches according to a domain language. The main task of the SIA is to interact with a set of SRAs, to collect their recognized symbols, and to build a coherent sketch interpretation solving the arising conflicts. When a SIA enters in the AgentSketch society it queries the Agent Directory and the OKB to find the SRAs able to recognize the symbols of the domain language. For each SRA it retrieves, from the agent directory, the AIP that it has to follow in order to obtain the service. When the SIA has found the properly SRAs it registers to the Agent Directory and advertises the domain language interpretation service, with the AIP to follow, that it is able to provide. Finally, the SIA browses the OKB to find the language it is able to recognize, if not found then it is added to the instances of the Language concept and the symbols to the instances of the Symbol concept. Using the OKB, the agents can understand the language that the SIA is able to recognize and the symbols that belong to this language.

To obtain the SRA service the SIA has to follows the AIP depicted in Fig. 6 and represented using AUML [5]. The first message sent by the SIA is a request containing: a sketch identifier $sk(id)$, the set of SRAs for the feedback exchange $relatedSRA(sr)$, and the geometrical relations $symb\_rel(r)$ that are allowed between the symbol recognized by the SRA and the shapes recognized by

the related SRAs. The geometrical relations are extracted by the language rules about symbols relationships. As an example, from the rule "a Communication symbol can be connected to an Actor symbol" we extract the relation "the *bounding box* of an actor symbol can be *near* the end or the start point of a Communication symbol". In order to exchange feedback SRA must be able to check these geometric relations starting from the shape recognized by the WS. The SRA can autonomously decide (*outer alternative fragment*) to provide the service ($accept(sk(id))$) or not ($reject(sk(id))$) (e.g., based on the amount of sketches handled at that time by the SRA). If the request is accepted the loop fragment is executed until the protocol ends. In the loop three cases can happen (*inner alternative fragment*):

1. The SIA sends a message, $inform($"$strokes(set)$, $sketch(id)$"$)$, to inform the SRA that the user has drawn a set of strokes that has to be analyzed.
2. The SIA sends a message to the SRA, $request($"$recognized\_shapes$, $sk(id)$"$)$, to request the set of recognized shapes, and the SRA replies sending the shapes with the message $inform($"$recognized\_shapes(set)$, $sk(id)$"$)$.
3. The SIA informs the SRA that the user has terminated the drawing by sending the message $inform($"$sketch\_finished$, $sk(id)$"$)$.

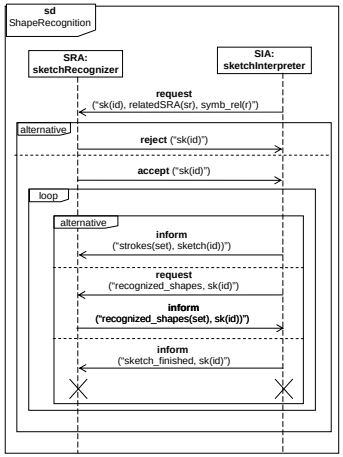


Fig. 6. AUML interaction protocol for the recognition services provided by an SRA.

When a new SRA joins the AgentSketch society it registers to the Agent Directory Service its description (name, address, and so on) and advertises its

recognition service. The SRA adds a standardized textual representation (detailed in Section 4) of the AIP shown in Fig. 6 to its service description. The SRA also adds to the service description the name of the symbol that it is able to recognize. Finally, the SRA browse the AgentSketch OKB and, if it is not present, add the symbol that it is able to recognize to the instances of the *Symbol* concept. Using the OKB another agent can understand the symbol that the SRA is able to recognize.

3.3 Intelligent Interface Agent Group

The main goal of an Intelligent Interface Agent (IIA) is to handle complex interactions with the user in order to enable him/her to draw a diagrammatic sketch and to present him/her feedbacks on the sketch interpretation process (performed by a SIA). Moreover, the IIA can interact with one or more DEA to offer to the user some domain specific services.

To find a SIA able to interpret a given domain language the IIA queries the Agent Directory and the OKB. The SIA retrieves the AIP published by the SIA and follows it to obtain the service. The IIA provides the SIA with the needed information about the sketch (drawn stroke and their attributes, such as spatial coordinates). Moreover, the suitable DEAs are found queering the Agent Directory. An IIA also offers to the users "Intelligent Symbol Manipulations" features. This features support users to easily modify the sketch (for example moving a symbol S while the system automatically re-arrange the symbols related to it) and prevent the violations of the language constraints.

3.4 Domain Expert Agent Group

A DEA is an agent designed to work on the model represented by a diagrammatic sketch (sketch semantic). For example, a DEA could be designed to analyze a diagram representing a building in order to check if some safety rules are satisfied and to prompt some suggestions. The services offered by DEAs are domain-specific and the Agent Interaction Protocols to follow to obtain these services can be very different. However, each DEA registers its provided service and the AIP to obtain it in the Agent Directory.

3.5 An agent society for Use Case Diagram understanding and reasoning

In this section we exemplify the agent society behavior of the intelligent sketch-based application for designing use case diagrams.

Fig. 7 shows the components composing the society and some agent interactions of an application for use case design. In particular, for each shape representing a use case diagram symbol a suitable WS must be available, and for each symbol an SRA must be added to the society. Each SRA searches the suitable WS by querying the Service Broker (arrow 1), and then registers itself in the

Agent Directory (arrow 2). A use case diagram SIA has also to be added to the AgentSketch society. This SIA queries the Agent Directory for finding the SRAs able to recognize the use case diagram symbols and for registering its interpretation service (arrows 3 and 4). Finally, an IIA handles the user interactions by finding the suitable SIA through the Agent Directory (arrow 5). If the user needs advanced domain specific services, one or more DEAs can be added to the society, even at run-time, and the IIA can find them using the Agent Directory (arrow 6).

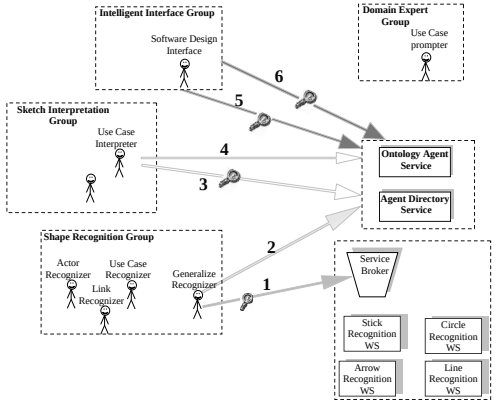


Fig. 7. An agent society for use case diagram recognition-based application.

4 Enabling Agents to Exchange Agent Interaction Protocols

As detailed in the previous sections new agents joining AgentSketch society need to interact with other agents in order to request their services. This can be accomplished only if the agent that need the services follow the AIP associated to them.

Many AOSE methodologies (for example GAIA [7]) take AIP as their starting point to design MASs. Indeed, interacting agents are implemented according to the designed AIPs. AgentSketch society enables agent development at different times and from different development groups by exploiting the advertisement of offered services, and related AIPs to follow in order to obtain these services. In particular, agents can find a service and the AIP to follow by looking into the agent directory. Obviously, the AIPs have to be represented in a standardized unambiguous way, so that the agents can easily handle. A widespread visual

notation used to represent and design AIPs is AUML interaction diagrams [8], an extension of UML sequence diagrams. AUML diagrams are visual diagrams conceived to design MAS by humans and are not suitable for representing AIP in a precise computer processable way. In order to be processed in an automatic way by computers, textual notations are still widely considered to have some significant advantages.

In [9] we have proposed to represent AIPs using a widespread standard textual notation designed for Web Services: the Web Services Business Process Execution Language [10]. WS-BPEL is layered on top of WSDL [11] and provides a language for the formal specification of business protocols describing the mutually visible message exchange of each of the parties involved in the protocol, without revealing their internal behavior. In particular, we have detailed the translation process from AUML to WS-BPEL and we have realized a AUML2WS-BPEL Translator[1], to obtain the automatic translation of an AUML AIP to a WS-BPEL document. In our agent society the AUML2WS-BPEL Translator, available as a set of Java library, can be used by an agent providing a service (i.e., an SRA, a SIA, or a DEA) to generate the WSBPEL representation of the AIP that an agent, that needs the service, namely service consumer has to follow.

The AUML2WS-BPEL Translator can also be used to obtain a Prolog representation of an AIP represented in BPEL. The AIP Prolog representation, as described in [9], can be used to semi-automatically generate the programming code needed by a service consumer agent to execute the AIP. The generated code can be executed in JADE[2] by means of the DCaseLP [12] libraries.

In the AgentSketch society, an agent that needs a service can use the translator to generate the suitable code to handle the AIP published in the Agent Directory by the service provider. For example, an IIA can find a service offered by a DEA in the Directory Agent and can generate the code needed to handle it.

Fig. 8 summarizes the AIP exchange process. The service provider generates the WS-BPEL representation of the AIP associated to a service and stores it in the Agent Directory. The service consumer looks for the service in the Agent Directory and retrieves the BPEL representation of the AIP to follow.

Fig. 8. AIP exchange process.

[1] AUML2WSBPEL Translator, http://www.disi.unige.it/person/MascardiV/Software/AUML2WS-BPEL.html

[2] http://jade.tilab.com/

5 Shape Recognizer Web Services

The implementation of Shape Recognizers (SR) as WSs allows us to integrate in AgentSketch shape recognizers implemented by anyone and in any language, physically stored anywhere, running on any platform, and replaceable with a minimal effort. The use of WSs is also motivated by the low quantity and the simple structure of data to be exchanged between SR and SRA and by the strong availability of standards, design patterns, and development tools. Moreover, the results obtained in the composition of WSs could be exploited to realize complex SRs as composition of simple ones.

Each Shape Recognizer Web Service can internally be based on a different shape recognition approach (for example, Ladder [13] and Sketch Grammars [14] represent suitable choices), however, all WSs must expose the same interface.

The operations that each WS has to expose are described in the following in a "Java like" style:

- **void *start_new_sketch*(Integer *sketch_id*)**
  This operation is used to inform the SR that a new sketch, identified by a number, namely the *sketch_id*, is started. The SR initializes itself to handle the shape recognition associated with the sketch and allocates all the needed resources.
- **void *input_strokes*(Vector *strokes_info*, Integer *sketch_id*)**
  This operation is used to give a set of strokes in input to the SR. The strokes are associated to a sketch identified by the *sketch_id* parameter. Each stroke is represented by an element of the Vector $strokes\_info$ where each element contains the following fields:
  - *Integer stroke_id*: represents the unique *id* in a sketch associated to the stroke
  - *Vector points*: represents the set of key points that characterize the stroke. Each point is represented by its position (x, y coordinates) and by its drawn time t.

  The *input_strokes* operation is called every time the user adds new strokes to the sketch or modify some strokes (for example moving or resizing them). If the SR receives new information about previously known strokes (it can happen if the user moves, resize or modify them), it updates the information associated to it.
- **Vector *input_ns_get_rsymbols*(Vector *strokes_info*, Integer *sketch_id*)**
  This operation is similar to the previous one, but it is used to give a set of strokes in input to the SR and at the same time to ask the set of recognized symbols. The set of recognized symbol is represented by a Vector of $recognized\_symbol\_info$ where each element represents a recognized symbol and it is composed by the following fields:
  - *Integer recognized_symbol_id*: the unique *id* of the symbol;
  - *String recognized_symbol_name*: the recognized symbol name;
  - *Vector stroke_id_vector*: a vector containing all the strokes *id* used to recognize the symbol;

- *ShapeGeometricAttributes spg*: each shape has associated a set of attributes computed by the SR and defined in the sketch ontology.

– **void *delete_input_strokes*(Vector *strokes_id*, Integer *sketch_id*)**
This operation is invoked when the user deletes some strokes (*strokes_id*) from the sketch (*sketch_id*).
– **Vector *get_recognized_symbol*(Integer *sketch_id*)**
This operations is used to ask to the SR the set of recognized symbol for a given sketch *sketch_id*. The output is the same of the *input_ns_get_rsymbols* operation.
– **void *end_sketch*(Integer *sketch_id*)**
This operation is used to inform the SR that a sketch is finished and all the resource associated with it can be released.

As described in Section 3, WSs advertise agents about their capabilities by means of a Service Broker.

6 Related Work and Conclusions

In the last two decades several approaches have been proposed for the recognition of freehand drawings but few of them exploit agent technology. QuickSet uses a suite of agents for multimodal human-computer communication [15], whereas the approach proposed in [16] uses a system for graphic unit recognition, where singular agents may specialize in graphic unit-recognition, and multi-agent systems can address problems of ambiguity through negotiation mechanisms. EsQUIsE is an interactive tool for free-hand sketches to support early architectural design [17]. The same system has been extended with the possibility of interpreting vocal information [18]. In particular, the graphical inputs are interpreted by either rule-based agents or model-based agents, while the spoken inputs are interpreted by model-based vocal agents.

Regarding the use of ontologies, Zheng and Sun proposed in [19] a sketch understanding process driven by domain knowledge bases. Their framework allows users to easily adapt the hierarchical understanding process to any domain through the definition of visual concept ontologies.

In this paper we have not concentrated on sketch recognition issues (the suitability and the effectiveness of the agent-based approach for sketch recognition, which is the backbone of the current proposal, have already been discussed in [2]) but we have focused to the flexibility issues of sketch recognition-based system. In particular, we have presented a framework that combines Web Services and Ontologies, for building open and dynamic agent societies aimed at hand-drawn sketch recognition. Ontologies enable agents to agree on message semantics and service purposes, standard web services languages to represent agent interaction protocols in a suitable way to be exchanged and handled by agents, and WSs to expose low-level recognition services. The flexibility of the agent organization and interactions allows us to recognize new languages and to realize new sketch-based applications changing the interactions between existing agents and/or adding new agents at run-time.

References

1. Casella, G., Costagliola, G., Deufemia, V., Martelli, M., Mascardi, V.: An agent-based framework for context-driven interpretation of symbols in diagrammatic sketches. In: Proc. of VL/HCC 06, Brighton, UK, IEEE CS Press (2006) 73–80
2. Casella, G., Deufemia, V., Mascardi, V., Costagliola, G., Martelli, M.: An agent-based framework for sketched symbols intepretation. (To appear in Journal of Visual Languages & Computing.)
3. Foundation for Intelligent Physical Agents: FIPA abstract architecture specification. http://www.fipa.org/specs/fipa00001/SC00001L.html (2002)
4. Foundation for Intelligent Physical Agents: FIPA ontology service specification. http://www.fipa.org/specs/fipa00086/XC00086D.html (2001)
5. Walton, C.: Agency and the Semantic Web. Oxford University Press (2006)
6. Choi, N., Song, I.Y., Han, H.: A survey on ontology mapping. ACM SIGMOD Record **35**(3) (2006) 34–41
7. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems **3**(3) (2000) 285–312
8. Huget, M.P., Odell, J.: Representing agent interaction protocols with agent UML. In: Proc. of AAMAS'04, IEEE CS Press (2006) 1244–1245
9. Casella, G., Mascardi, V.: Intelligent agents that reason about web services: a logic programming approach. In: Proc. of International Workshop on Applications of Logic Program. in the Semantic Web and Semantic Web Services, Seattle, WA, USA (2006) 55–70
10. Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N., Liu, C.K., Thatte, S., Yendluri, P., Yiu, A., eds.: Web Services Business Process Execution Language (WS-BPEL). Version 2.0. (2005)
11. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C Note. (2001)
12. Gungui, I., Martelli, M., Mascardi, V.: DCaseLP: a prototyping environment for multilingual agent systems. Technical Report DISI-TR-05-20, DISI, Univ. of Genova, Italy (2005)
13. Hammond, T., Davis, R.: LADDER, A sketching language for user interface developers. Computers & Graphics **29**(4) (2005) 518–532
14. Costagliola, G., Deufemia, V., Risi, M.: Sketch Grammars: A formalism for describing and recognizing diagrammatic sketch languages. In: Proc. of ICDAR'05, IEEE Press (2005) 1226–1230
15. Cohen, P.R., Johnston, M., McGee, D., Smith, I., Pittman, J., Chen, L., Clow, J.: Multimodal interaction for distributed interactive simulation. In: Proc. of IAAI'97, AAAI Press (1997) 978–985
16. Achten, H.H., Jessurun, A.J.: An agent framework for recognition of graphic units in drawings. In: Proc. of eCAADe'02, Warsaw (2002) 246–253
17. Juchmes, R., Leclercq, P., Azar, S.: A freehand-sketch environment for architectural design supported by a multi-agent system. Computers & Graphics **29**(6) (2005) 905–915
18. Azar, S., Couvreury, L., Delfosse, V., Jaspartz, B., Boulanger, C.: An agent-based multimodal interface for sketch interpretation. In: Proc. of MMSP-06, British Columbia, Canada (2006)
19. Zheng, W.T., Sun, Z.X.: Knowledge-based hierarchical sketch understanding. In: Proc. of ICMLC'5. (2005) 2838–2843

Extending the FIPA Interoperability to Prevent Cooperative Banking Frauds

Mauricio Paletta[1] and Pilar Herrero[2]

[1] Departamento de Ciencia y Tecnología.
Universidad Nacional Experimental de Guayana.
Av. Atlántico. Ciudad Guayana. Venezuela.
[2] Facultad de Informática. Universidad Politécnica de Madrid.
Campus de Montegancedo
S/N. 28.660 Boadilla del Monte. Madrid. Spain.

Abstract. Electronic bank transactions are very common today. Services given by an Automatic Teller Machine (ATM), for example, are very popular and widely used by bank clients. Unfortunately, in the same way as the use of these devices is increasing, the proliferation of different frauds to try to violate these systems to steal user's money is also increasing. Sometimes, the modus operandi used by the delinquents depends on different factors, such as the country or the city where fraud is committed or, as in the case of ATMs, the model or location of these devices. Since the detection of these modus operandi is not easy and they could be different from a bank institution to another, having both an environment capable of following up the swindler agents learning processes and a way to prevent the cooperation between these agents to share the learned knowledge, would be very useful to discover different modus operandi before crimes are committed. In this paper, a framework designed to follow up the swindlers' agents learning process and to share the knowledge between the agents is presented. This framework is based on the FIPA (Foundation for Intelligent Physical Agents) specifications and it emphasizes on the swindler agents learning process to fulfil the human-like agent behaviour and a realistic interaction with the environment.

1 Motivation

Electronic banking frauds have attracted significant international attention, since individuals and organizations have lost billions of dollars worldwide. Electronic banking speeds up transactions and creates new "promising" services, altering banking operations, and dramatically expanding the reach of financial institutions. Services given by an Automatic Teller Machine (ATM), for example, are very popular and widely used by bank clients. In fact, this kind of transaction is leading the current payment system.

Given the inherent nature of electronic banking in eliminating paper documentation and traditional identity verification processes, a new dimensional amount of risky situations have arisen. The proliferation of different frauds to try to violate these systems to steal user's money is also increasing.

Bank institutions are continuously receiving claims from victims of this type of crimes and the only thing they can do is to inform their employees and clients about one particular modus operandi once it is discovered.

Detection of procedures to commit these crimes is not easy and once it is discovered, criminals find another more skilful way to proceed. Sometimes, the modus operandi involves different people and techniques making them more difficult to detect. These also could depend on different factors: the country or city where the fraud is committed, the model or location of an ATM, etc. This hinders bank institutions to inform their employees and clients about crimes on time before they can be committed.

In this sense, having an environment capable of following up the swindler agents learning processes and to prevent the cooperation among these agents, with the purpose to share learned knowledge, would be very useful for bank institutions, discovering, for example, how criminals improve their modus operandi day by day and how they "create" new techniques and schemes to avoid the bank systems devoted to detect this kind of frauds.

If each bank institution could have one of these specialized agents which have learned the modus operandi from previous committed frauds, and all these agents could communicate among each other, then institutions can share learned knowledge and have more chance to avoid crimes. This is explained in more detail in this paper.

FIPAL (a FIPA agent-based framework designed to follow up the swindlers' intelligent agents Learning process) is also presented in this paper. It is based on the Foundation for Intelligent Physical Agents (FIPA)[1] specifications and it emphasizes on the swindler agents learning process to fulfil the human-like agent behaviour and a realistic interaction with the environment. In order to accomplish these necessities, the FIPAL-KBEL (Knowledge Base Experience Language for FIPAL), has been created as a new XML language based approach. This paper also describes how the FIPAL-KBEL language allows a flexible representation of the knowledge as well as a more effective learning process.

2 Related Work

The specifications of the Foundation for Intelligent Physical Agents (FIPA) constitute an interoperability model covering all elements from the agent architecture to the application domains. In this model, the agent system interoperability is based on the use of a common Agent Communication Language (ACL) [5] and supported by an Abstract Architecture [6] which can be used to abstract the internal architecture of each agent.

The FIPA abstract architecture provides some mechanisms that could be used to enact the communication process among heterogeneous agent systems

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

to achieving interoperability in message representation and transportation. However, the technological support provided by FIPA alone is not sufficient to achieve interoperability. Agents also need to have the knowledge to contextualize their acts of communication within the multiagent environment in which they take place [15].

Extensions of the FIPA specifications for intelligent agents could be seen in some research works, such as in [15] where the authors proposed an approach to extend the FIPA interoperability model to deal with agent social issues, like social requirements on agent conversations and communication languages. Examples of FIPA compliance, on the other hand, can be seen in other works, such as in [7] where Lynden et al introduced LEAF, a software toolkit for developing learning multiagent systems, Pokahr et al [12] present Jadex, a software framework for the creation of goal-oriented agents based on the FIPA specifications and following the BDI (Belief-Desire-Intention) [14].

By the other side, a no less important factor is the one related to the intelligent learning process. Learning plays a fundamental role in many of human activities since experience, including both achievements and errors [2]. It seems that this is the fundamental property that allows humans to adjust themselves to the different changes in the environment [1], [2], [3].

However, none of the previous authors has focused on the main objective from the perspective of this paper: to design and develop an open and flexible framework based on the learning process to be applied to the electronic banking fraud.

In the next section, the way in which an intelligent agent has been structured will be presented, by means of an open and flexible architecture using the FIPA specifications and based on the Learning and environment interaction processes (FIPAL). FIPAL is the evolution of a previous architecture called IVAL (An Open and Flexible Architecture based on the IVA Learning Process) which can be reviewed in detail in [10] and [11].

3 A FIPA extension designed to follow up the swindlers' agents learning process

3.1 The FIPAL structure

According to the FIPA in [6], the existence of a FIPA Abstract Architecture does not prohibit the introduction of elements useful to make a good agent system; it merely sets out the minimum required elements. Based on this affirmation and considering the BDI fundamentals, the FIPAL architecture has been designed as it can be seen in Fig. 1. There are five service modules:

1. The Service Directory Service (SDS): its basic role is to provide a consistent means by which agents and services can discover services [6].
2. The Social Issue Service (SIS): it contains those services needed to achieve the interaction with the environment based on stimuli reception and acting (see Sect. 3.2 for more detail).

3. The Learning Service (TLS): it contains a knowledge repository and those services needed to achieve the knowledge reasoning (see Sect. 3.3 for more detail).
4. The Communication Service (TCS): it contains the aspects of message communication between agents, as the message structure, message representation and message transport (see Sect. 4 for more detail).
5. The Control of Services (COS): it synchronizes the execution of the remainder elements and has the algorithms to react, deliberate and control the plans of the agent. It represents the FIPAL agent core since all the interaction among the other four elements passes through it. Related plans are represented using a kind of Teleo-Reactive (T-R) sequence [9].

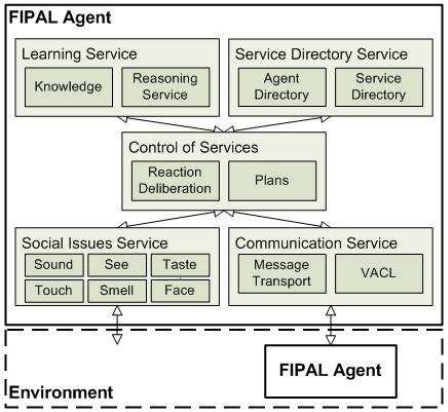


Fig. 1: The FIPAL structure

Following the FIPA specifications, the descriptions of all these services are registered in the Service Directory element inside the SDS module.

3.2 The FIPAL social issues

As it could be seen previously, the FIPAL structure is endowed with social issues abilities provided by the corresponding service module. One of the fundamental roles of this module is to interact with the environment. In this sense, a FIPAL agent has six components or set of services each of them with the capabilities to interchange information with the environment in order to receive a stimulus and give an answer according to the kind of interaction the agent is doing.

Five of the six kinds of interaction or social issues correspond to the human being senses. The other remaining social aspect corresponds with a service capable to represent facial expression. In this sense, a FIPAL agent tries to follow a social behaviour in the same way as the human beings do it. This six set of services are the following:

1. Sound, to interact with their surroundings by means of hearing (listening and speaking). The services associated to this component are the following:
   - Listening( ): start listening from the environment in order to receive auditory stimuli; with this service the FIPAL agent acquires the hearing ability.
   - StopListening( ): stop receiving auditory stimuli from the environment.
   - Talking(message): with this service the FIPAL agent can act saying an oral message to the environment.
2. See, to perceive the environment by means of the sight. The services associated are the following:
   - Seeing(): start seeing from the environment in order to receive visual stimuli; with this service the FIPAL agent acquires the visual ability.
   - StopSeeing(): stop receiving visual stimuli from the environment.
   - SeeFaceExpression(): with this service the FIPAL agent can act observing the facial expression of the most nearby agent.
3. Touch, to interact with their surroundings by means of the touch (touching and manipulating), with the following services:
   - Touching(): start touching objects into the environment in order to receive sense of touch stimuli; with this service the FIPAL agent acquires the sense of touch ability.
   - StopTouching(): stop receiving sense of touch stimuli from the environment.
   - Taking(): with this service the FIPAL agent can act taking the most nearby object.
   - Putting(object): with this service the FIPAL agent can act putting into the environment the object indicated.
4. Smell, to interact with their surroundings by means of the smell by using the following services:
   - Smelling(): start smelling from the environment in order to receive olfactory stimuli; with this service the FIPAL agent acquires the ability to smell.
   - StopSmelling(): stop receiving olfactory stimuli from the environment.
5. Taste, to interact with their surroundings by means of the taste by using the following services:
   - Tasting(): start tasting from the environment in order to receive taste stimuli; with this service the FIPAL agent acquires the taste ability.
   - StopTesting(): stop receiving taste stimuli from the environment.
6. Face, to represent and interpret the facial expressions such as happiness, fright, fears, etc.
   - Setting(expression): with this service the FIPAL agent can act by setting the indicated facial expression.

By the other side, FIPAL pays a lot of attention to the learning topic, including all an agent has to learn, how it should learn it and how knowledge should be handled. This is the main reason that has motivated the definition of a new XML-based approach for knowledge representation, known as FIPAL-KBEL. The following section describes how FIPAL-KBEL is used to represent the knowledge and how the learning process is carried out.

3.3 The knowledge representation and learning process in FIPAL

The FIPAL basic architecture is provided with learning abilities, thanks to the corresponding Learning Service (Fig. 1). One of the fundamental purposes of this service is to maintain the knowledge learned by the agent. In this sense, a knowledge representation technique was necessary to achieve this objective.

Due to its simplicity and flexibility, XML[2] (Extensible Markup Language) has been used as the basic representation language for covering the FIPAL functionalities, particularly knowledge representation. Since XML is a universal and web-based data format, it is appropriate for an independent platform and world wide use and has become a widely accepted standard data interchange technology, so its general usability is guaranteed for the next years [4].

Based on the previous statements, FIPAL-KBEL (Knowledge Base Experience Language for FIPAL) has been designed to fulfill the requirements related to knowledge representation. FIPAL-KBEL follows the rules representation [13] and, for instance, it defines the requested labels to structure a knowledge rule.

```
<?xml version="1.0"?>
<FIPAL-KBEL:Knowledge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="FIPAL-KBEL.xsd"
 xmlns:FIPAL-KBEL="…">
 <FIPAL-KBEL:Experience id="…">
  <FIPAL-KBEL:Stimulus source="…">
   <FIPAL-KBEL:Data>…</FIPAL-KBEL:Data>
  </FIPAL-KBEL:Stimulus>
  <FIPAL-KBEL:Service name="…">
   <FIPAL-KBEL:Parameter name="…" type="…">…</FIPAL-KBEL:Parameter>
   <FIPAL-KBEL:Parameter name="…" type="…">…</FIPAL-KBEL:Parameter>
   … more parameters from this service
   <FIPAL-KBEL:Result type="…">…</FIPAL-KBEL:Result>
  </FIPAL-KBEL:Service>
  <FIPAL-KBEL:Service name="…">
   <FIPAL-KBEL:Parameter name="…" type="…">…</FIPAL-KBEL:Parameter>
   <FIPAL-KBEL:Parameter name="…" type="…">…</FIPAL-KBEL:Parameter>
   … more parameters from this service
   <FIPAL-KBEL:Result type="…">…</FIPAL-KBEL:Result>
  </FIPAL-KBEL:Service>
  … more services from this experience
 </FIPAL-KBEL:Experience>
 … more experience from this knowledge base
</FIPAL-KBEL:Knowledge>
```

Fig. 2: The FIPAL-KBEL document structure

Since the agent associated with the FIPAL architecture is acting according to the received stimuli, the relation between what it is saw, heard, touched, smelled or tasted and the given reactions needs to be represented and stored. Any stimulus consists on a tuple <data, type> that represents the information

[2] http://www.w3.org/XML/

coming from the environment. The reaction of the agent consists on many executed services with their corresponding parameters, if any; for example, in one service associated with the ability of talking, the parameter must be the string that the agent wants to say. Additionally, each service is accompanied by its corresponding executed result or reaction. In this way, each service is given by a pair $<service\_name(p_1, p_2, \cdots, p_n), result>$ where p_i $1 \leq i \leq n$, is the i^{th} parameter.

Fig. 2 can be reviewed to detail the labels of the FIPAL-KBEL document structure. As it can be seen, FIPAL-KBEL represents each of this knowledge rules inside the label "<FIPAL-KBEL:Experience>". Any rule has a different identification number, "id" attribute, inside this label and it is used to indicate the corresponding sequence into the knowledge base. This is useful to follow exactly what happened before and after the stimulus arrived. It is important to highlight that the result of a service indicates whether not only the service was executed properly, but also if it was appropriate or still unknown.

The reasoning process occurs when a new stimulus arrives and the agent looks for an adequate answer; it consults the learning service component to know whether there is any experience with the present state or stimulus. If a knowledge rule associated with the received stimulus is found, all the services identified in the rule with the appropriate result value are executed in the same order they are executed when the same stimulus is received during the learning process. Fig. 3 shows in detail this algorithm executed by the Control of Services component.

```
Experience E
Stimulus I

For all Stimuli received:
  I = SocialIssuesService.StimulusReception()
  E = LearningService.Consulting(I)
    If E is not null
      For all E.Services: E.Si
        ControlOfServices.Calling(E.Si,p1,…,pn)
    For all new Services: Si
      E = new Experience(I,Si)
      LearningService.Learn(E)
```

Fig. 3: The FIPAL reasoning and learning general algorithm

If a different service is executed after the evaluation of the knowledge rule, this new information is considered to reinforce or give feedback to the knowledge according to what it was learned. This learning strategy continues until a new stimulus is received. This means that all the services executed after a stimulus is received and before a new stimulus arrives are taken in consideration to upgrade the knowledge rule associated with the first stimulus. To better understand the previous algorithm, Fig. 4 shows an UML collaborative diagram between the elements that conform the FIPAL structure (Fig. 6 in section 5 shows the UML class diagram related to these concepts).

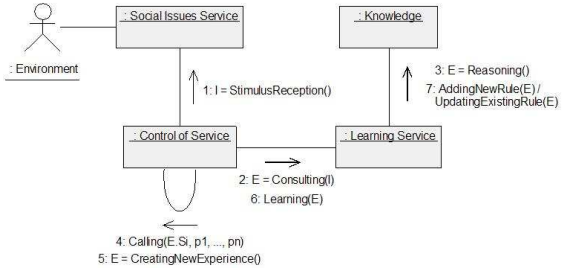


Fig. 4: UML collaboration diagram of the FIPAL learning and reasoning general process

Next section describes the way in which the FIPAL agents can interact each other to prevent banking frauds cooperatively.

4 The FIPAL interoperability to prevent cooperative banking frauds

Supposing there is at least one FIPAL agent in any bank institution which is learning the modus operandi used by the delinquents to commit the frauds in this institution, and supposing that these FIPAL agents are connected among each other (see Fig. 5), then the FIPAL agents can share the experience acquired with the aim to detect the possible fraud on time before it can be committed.

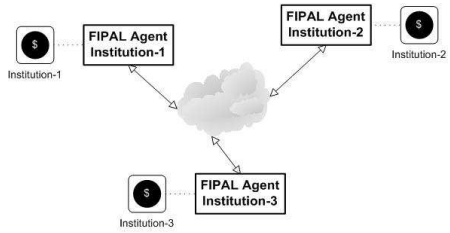


Fig. 5: Cooperative banking frauds

In order to cooperate with each other, any agent structured with FIPAL can respond to the service "Consulting(stimulus)" which is associated with the component Learning Service (see Fig. 1). The objective of this service is to find

an experience based on the given stimulus and, as it occurs with the rest of services the agent can serve, the description of this service is registered in the Service Directory element of the FIPAL structure.

Once the Communication Service of the agent receives the request of executing the service "Consulting(stimulus)", the Learning Service element looks for an experience associated with the given stimulus by using the algorithm showed in Fig. 3. If there is a knowledge rule associated with that stimulus, the agent prepares an answer based on the services related to it. Only that services which have the corresponding results with a set value indicating if its execution is or not appropriate, will be considered for the answer. This is useful to inform the agent who sent the request of the service what it is or not appropriate to do in response to that stimulus.

A FIPAL agent can ask for help at anytime, particularly when it doesn't have any experience to an actual situation. Once the agent receives the answers from other agents, if any, it can execute the services with an appropriated value in their corresponding results and avoid executing services with an inappropriate value. All this information is used for this agent to learn how must act in presence of this stimulus in the future.

In order to achieve the communication among FIPAL agents, the FIPA ACL abstract message structure was adopted to define the XML based language FIPAL-VACL (Virtual Agent Communication Language for FIPAL). XML allows agent developers to extend sensor or effectors classes with their own ACL if they so desire, and the text basis of XML minimises functional coupling between sensors and effectors of different agents [8].

A scenario in which it is possible to review the way a couple of FIPAL agents can interact between each other to prevent banking frauds cooperatively is described in the next section.

5 A case study: Cooperative Banking Frauds

It should be pointed out that this work is currently in progress and therefore FIPAL will continue evolving. FIPAL is developing using C# programming language and, at this moment, efforts have been concentrated on having an appropriated graphical interface for the IVA. Fig 6 shows a UML class diagram with the concepts related to FIPAL and the relations among them. As it can be seen, these concepts and relations are related to the FIPAL structure showed in Fig. 1.

Several scenarios have been raised with the aim of validating the FIPAL interoperability described in the previous sections. The scenario presented in this section corresponds to a simulation of real situation which has been happening in Venezuela related to the Automatic Teller Machine (ATM) banking fraud.

The environment where the simulation is carried out is an ATM precinct with a couple of machines: ATM-1 and ATM-2. There are three agents: the delinquent D1, a second delinquent D2 and the client C1. The fraud is carried out because while the client C1 is initiating the transaction in ATM-2, the delinquent D1,

who is using ATM-1, has programmed this ATM to intercept the transaction of ATM-2 acting itself as if it were ATM-2. Then, D1 prevents C1 from using ATM-2 pretending it does not work, while D2 is already finishing in ATM-1 the transaction that C1 initiated in ATM-2.

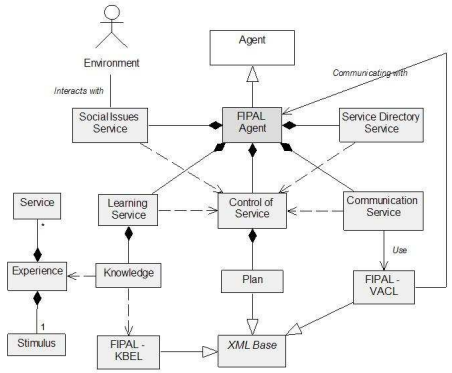


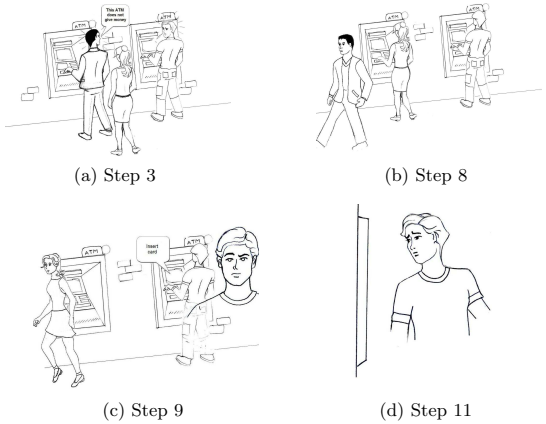
Fig. 6: UML class diagram of the FIPAL concepts

The scenario described previously can be summarized through the following steps:

1. C1 arrives to the ATM room while D1 is using the ATM-1 and the ATM-2 is free.
2. C1 is located in front of ATM-2. It introduces its card and its password while D2 enters in the ATM room.
3. D2 is in the ATM-1 queue, behind D1; D1 moves towards C1 with a serious facial gesture and tell him that the ATM he is intended to use (ATM-2) counts the money but it will not be able to get the cash from the ATM (see Fig. 7-a).
4. C1 reacts to D1's comment by cancelling the transaction which he was carrying on.
5. Once C1 has given the order of cancelling the transaction in the ATM-2 and it has waited for the confirmation of the cancellation, it decides to go to the ATM-1.
6. D1's transaction in ATM-1 is over and it moves towards the ATM-2; D2, which was waiting behind D1, starts using ATM-1 and C1 places behind D2 to use the ATM-1 as soon as possible.
7. D1, with a warning facial expression, tells C1 that the ATM-2 is making a transaction.

8. C1, with a worry expression, goes to the ATM-2 to cancel the transaction, while D1 gets out from the ATM room (see Fig. 7-b).
9. C1 can not observe any kind of transaction movement in the ATM-2 and therefore it decides to move again towards the ATM-1 where D2 is still finishing its transaction before leaving the ATM room (see Fig. 7-c).
10. Once C1 is located again in front of ATM-1, it introduces its card, password and the rest of the data to start a new transaction.
11. ATM-1 alerts to C1 that it has already taken out the maximum amount permitted by the bank; C1 reads this message and realises of the fraud (see Fig. 7-d).

With the aim of explaining better the simulation it is assumed that there are two bank institutions: B1 and B2. In B1 the FIPAL agent associated to the client C1 (for convenience, it will be used B1:C1 to identify this agent) has learned the modus operandi previously described, but in B2 the agent associated to C1 (B2:C1) doesn't have any experience yet. The simulation is going to be divided in two different parts: 1) the B1:C1's learning process and 2) the B2:C1's learning process using the B1:C1's experience. Both of parts are described by means of a series of steps each of them with a brief description of the simulated situation and the explanation of how the FIPAL architecture associated to C1 (B1:C1 or B2:C1) tackles this scene or step.



(a) Step 3 (b) Step 8

(c) Step 9 (d) Step 11

Fig. 7: Simulated situation of some steps of the scenario. The ATM-1 is the ATM on the left side being used by D1 in (a); the ATM-2 is the ATM on the right side being used by C1 in (a), (b) and (c); D2 is the woman

5.1 The B1:C1's learning process

Step 1: C1 arrives to the ATM room while another client (D1) is using the ATM-1 machine and the ATM-2 is free. **FIPAL:** The environment perception starts once the B1:C1 agent executes the service "Seeing()" from the See set of services. In response, B1:C1 receives a set of visual stimuli associated with the specific environment situation. In this case, the stimulus received would be <"ATM-1 busy", seen> and <"ATM-2 free", seen>.

Step 2: C1 is located in front of ATM-2. It introduces its card and its password while another client (D2) enters in the ATM room. **FIPAL:** Once received the stimuli and according to section 3.3, since B1:C1's knowledge about the fraud is initially null, B1:C1's reasoning about the fraud is null and therefore, it decides to ask for help by sending to the other FIPAL agents which are interconnecting into the cooperation banking network, a FIPAL-VACL message with the request of the service "Consulting(<"ATM-2 free", seen>)". Since there are not FIPAL agents with experience about this fraud, B1:C1 does not received any answer and therefore it didn't react to the delinquent intentions based on its experience; B1:C1 keeps its main purpose "to use the ATM machine as soon as possible" (based on its plans) and therefore it tries to go to the ATM-2 and to use this machine; B1:C1 executes the services "Putting(card)" and "Putting(password)" from the Touch set of services. In this moment, B1:C1 starts learning about what is happening, and as a consequence a knowledge rule is created based on the last stimulus received as well as on the actions executed; B1:C1 updates the knowledge with this new rule, modifying therefore the FIPAL-KBEL document.

Step 3: D2 is in the ATM-1 queue, behind D1; D1 moves towards C1 with a serious facial gesture and tell him that the ATM he is intended to use (ATM-2) counts the money but it will not be able to get the cash from the ATM. **FIPAL:** As D2 is in the ATM-1 queue, B1:C1 realises that the environment has changed and then it receives the stimulus: <"new client in ATM-1", seen>; B1:C1 doesn't react to this environment; moreover, the oral message that D1 sends to B1:C1 is captured by the Listen() service as an FIPAL-VACL message with the stimulus: <"ATM does not give money", listened>.

Step 4: C1 reacts to D1's comment by cancelling the transaction which he was carrying. **FIPAL:** Due to B1:C1 receives the stimulus <"ATM does not give money", listened> and it doesn't have experience on this respect, it decides to ask for help again following the same action as it is described in the Step2 and, in this case the sent FIPAL-VACL message contains this solicitude: "Consulting(<"ATM does not give money", listened>)". As it has no experience on this respect, B1:C1 reacts executing the service "Setting(worry)" from the Face set of services and C1's expression change to "worried". In the same way, B1:C1 reacts executing the service "Putting(cancel)" from the Touch set of services and the transaction is cancelled. B1:C1 updates its knowledge with this new rule, modifying therefore the IVAL-KBEL document.

Step 5: Once C1 has given the order of cancelling the transaction in the ATM-2 and it has waited for the confirmation of the cancellation, it decides to go to the ATM-1. **FIPAL:** The See module realises that the environment

has changed and then it receives the stimulus: <"transaction cancelled", seen>; B1:C1 tries to makes a decision based on its experience but once more it has no experience on this respect and once more it doesn't receive any help from another FIPAL agent.

Step 6: D1's transaction in ATM-1 is over and it moves towards ATM-2; D2, which was waiting behind D1, starts using ATM-1 and C1 places behind D2 to use ATM-1 as soon as possible. **FIPAL:** B1:C1 realises the changes in the environment and it receives the corresponding stimulus without carrying out any reaction.

Step 7: D1, with a warning facial expression, says to C1 that the ATM-2 is making a transaction. **FIPAL:** The oral message that D1 sends to B1:C1 is captured by the Sound module with the stimulus: <"ATM-2 processing", listened>; B1:C1 tries to react with this new stimulus received.

Step 8: C1, with a worry expression, goes to the ATM-2 to cancel the transaction, while D1 gets out from the ATM room. **FIPAL:** B1:C1 reacts: 1) executing the "Setting(worry)" service from the Face module (with the aim of changing the face expression), 2) executing the "Putting(cancel)" service from the Touch set of services (with the aim of cancelling the transaction), 3) executing the "Seeing()" service from the See module (with the aim of observing the situation); B1:C1 introduces a new knowledge rule and updates its learning.

Step 9: C1 cannot observe any kind of transaction movement in ATM-2 and therefore it decides to move again towards ATM-1 where D2 is still finishing its transaction before leaving the ATM room. **FIPAL:** B1:C1 receives the stimulus <"Insert card", seen> from the See module and it reacts executing the service "Setting(normal)" from Face; the agent goes back to ATM-1; B1:C1 introduces a new knowledge rule and updates its learning.

Step 10: Once C1 is located again in front of ATM-1, it introduces its card, password and the rest of the data to start a new transaction. **FIPAL:** See receives the stimulus <"ATM-1 free", seen>; B1:C1 reacts executing the services: "Putting(card)", "Putting(password)" and "Putting(options)" from Touch; B1:C1 updates its learning with the new knowledge rule.

Step 11: ATM-1 alerts to C1 that it has already taken out the maximum amount permitted by the bank; C1 reads this message and realises of the fraud. **FIPAL:** See receives the stimulus <"it has withdrawn the maximum quantity of money permitted for day", seen>; B1:C1 reacts executing "Setting(worry)" from Face; B1:C1 introduces, again, a new knowledge rule and updates its learning. At this moment it is important to mention that, due to the fact that the final result for B1:C1 was negative, all the knowledge rules previous the last one and in which the services results are not setting yet, must be changed in order to set a non acceptable value into the corresponding services results.

5.2 The B2:C1's learning process using the B1:C1's experience

Step1: C1 arrives to the ATM room while D1 is using the ATM-1 and the ATM-2 is free. **FIPAL:** The environment perception starts once the B2:C1 agent executes the service "Seeing()" from the See set of services. In response, B2:C1

receives a set of visual stimuli associated with the specific environment situation. In this case, the stimulus received would be <"ATM-1 busy", seen> and <"ATM-2 free", seen>.

Step2: C1 (located in front of ATM-2) introduces its card and its password while D2 enters in the ATM room. **FIPAL:** Once received the stimuli and according to section 3.3, since B2:C1's knowledge about the fraud is initially null, B2:C1's reasoning about the fraud is null and therefore, it decides to ask for help by sending to the other FIPAL agents which are interconnecting into the cooperation banking network, a FIPAL-VACL message with the request of the service "Consulting(<"ATM-2 free", seen>)". Since B1:C1 has experience regarding to this stimulus, it responds to the solicitude sent by B2:C1 by sending a FIPAL-VACL message with the services "Putting(card)" and "Putting(password)", because this was what B1:C1 learned once it received this stimulus. In response to its solicitude, B2:C1 receives the message previously described and therefore it executes the services "Putting(card)" and "Putting(password)" from the Touch set of services. In this moment, B2:C1 starts learning about what is happening, and as a consequence a knowledge rule is created based on the last stimulus received as well as on the actions executed until now; B2:C1 uses the Learning Service component in order to update the knowledge with this new rule, modifying therefore the FIPAL-KBEL document.

Step3: D2 is in the ATM-1 queue, behind D1; D1 moves towards B2:C1 with a serious facial gesture and tell him that the ATM he is intended to use (ATM-2) counts the money but it will not be able to get the cash from the ATM. **FIPAL:** As D2 is in the ATM-1 queue, B2:C1 realises that the environment has changed and then it receives the stimulus: <"new client in ATM-1", seen>; B2:C1 doesn't react to this environment; moreover, the oral message that D1 sends to B2:C1 is captured by the "Listen()" service with the stimulus: <"ATM does not give money", listened>.

Step4: As B1:C1 has learned that cancelling the transaction due to a comment like D1 has done is not good, B2:C1 decides to continue with the transaction normally. **FIPAL:** Due to B2:C1 receives the stimulus <"ATM does not give money", listened> and it doesn't have experience on this respect, it decides to ask for help again following the same action as it is described in the Step2 and, in this case the sent FIPAL-VACL message contains this solicitude: "Consulting(<"ATM does not give money", listened>)". At this point it is important to mention that once B1:C1 found the rule that correspond with the stimulus <"ATM does not give money", listened>, it realises that this experience is associated with the services "Setting(worry)", "Putting(cancel)" and "Putting(options)" but, the results of the two first services are not acceptable. In this sense, B1:C1 answers with a message which only contains the execution of the service "Putting(options)" with the intention to inform to B2:C1 to continue with the transaction. Once B2:C1 receives the answer it reacts executing the service "Putting(options)" following the indications given by B1:C1. At this point, B2:C1 updates the knowledge with this new knowledge rule, modifying therefore the FIPAL-KBEL document.

Step5: The C1's transaction was successful and it receives the money from the ATM-2, so it takes its money and gets out from the ATM room. **FIPAL:** B2:C1 receives the stimulus: <"Taking money", seen>; B2:C1 reacts: 1) executing the service "Setting(happy)" from Face, and 2) executing the service "Taking(money)" from Touch; B2:C1 introduces a new knowledge rule and updates its learning. Due to the fact that the final result for C1 was positive, all the knowledge rules previous the last one and in which the services results are not setting yet, must be changed in order to set an acceptable value into the corresponding services results.

6 Discussion

In this paper it is presented FIPAL, a framework designing to follow up the swindlers' agents learning process. This framework is based on an open and flexible architecture, according to the FIPA specifications, that emphasizes on the swindlers' agents learning and environment interaction processes.

FIPAL is based on the FIPA architecture as well as on some previous research made on this direction, such as the design an implementation of a learning architecture for IVA named IVAL [10][11]. Both IVAL and FIPAL emphasize on the swindlers' agents learning process to fulfill not only more human-like agent behavior but also a more realistic interaction with the environment.

The implementation of the FIPAL architecture, complemented with the FIPAL-KBEL language, has already been evaluated in different simulations with successful results. Based on the evaluation of the model as well as on the results obtained, we can conclude that FIPAL, starting with an empty knowledge base, learns from the experience of interacting with the environment. This learning process is quite similar to human learning process since they are born.

In this paper we have concentrated in describing the way in which FIPAL agents can communicate or interact among each other to prevent banking frauds cooperatively. A complete scenario of a real ATM fraud situation was presented in this paper, in which was possible to verify that an agent, structured with FIPAL, not only is capable of learning the modus operandi used by the delinquents to commit the fraud but also cooperate with other agents which have not learned yet this particular modus operandi and therefore, both bankers and clients can be trained to react to this kind of banking frauds in almost real-time.

It is important to point out that FIPAL has not been implemented yet as final applications but it is part of the future work related to this research.

References

1. Bonasso R. P., Kortenkamp D.: An Intelligent Agent Architecture In Which to Pursue Robot Learning. In Proc. Workshop on Robot Learning, Rutgers University, New Brunswick, NJ (1994)
2. Buczak A. L., Cooper D. G., Hofmann M. O.: Evolutionary agent learning. International Journal of General Systems, Vol. 35 N 2 (2006) 231–254

3. Caicedo A., Thalmann D.: Virtual Humanoids: Let Them Be Autonomous without Losing Control. In Proc. Fourth International Conference on Computer Graphics and Artificial Intelligence, Limoges, France (2000)
4. de Vries A: XML framework for concept description and knowledge representation. Artificial Intelligence; Logic in Computer Science; ACM-class: I.7.2; E.2; H.1.1; G.2.3; arXiv:cs.AI/0404030 v1 (2004)
5. Foundation for Intelligent Physical Agents: FIPA ACL Message Structure Specification. SC00061, Geneva, Switzerland (2002) http://www.fipa.org/specs/fipa00061/index.html
6. Foundation for Intelligent Physical Agents: FIPA Abstract Architecture Specification. SC00001, Geneva, Switzerland (2002) http://www.fipa.org/specs/fipa00001/index.html
7. Lynden S., Rana O.: LEAF: A FIPA Compliant Software Toolkit for Learning based MAS. In Proc. First International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS'02, Bologna, Italy, ACM 1-58113-480-0/02/0007 (2002)
8. Maher M. L., Smith G. J., Gero J. S.: Design Agents in 3D Virtual Worlds. In Proc. Workshop on Cognitive Modeling of Agents and Multi-Agent Interactions (IJCAI) (2003) 92–100
9. Nilsson, N. J.: Teleo-Reactive Programs for Agent Control. Journal of Artificial Intelligence Research, 1 (1994) 139–158
10. Paletta M., Herrero P.: Learning from an Active Participation in the Battlefield: A New Web Service Human-based Approach. In Proc. International Workshop on Agents, Web Services and Ontologies Merging (AWeSOMe'06) in the Second OTM Federated Conferences and Workshops (OTM 2006), Montpellier, France. LNCS 4277, Springer (2006) 68–77
11. Paletta M., Herrero P.: Banking Frauds: An Agents-Based Training Framework to Follow-up the Swindlers Learning Process. Special Issue of International Transactions on Systems Science and Applications, Vol. 3, No. 2 (2007) (to be published)
12. Pokahr A., Braubach L., Lamersdorf W.: Jadex: A BDI Reasoning Engine. Multiagent Systems, Artificial Societies, and Simulated, Organizations International Book Series, Vol. 15, 10.1007/b137449, ISBN: 978-0-387-24568-3, Springer (2005) 149–174
13. Post E.: Formal reductions of the general Combinatorial Problems. American Journal of Mathematics 65 (1943) 197–268
14. Rao A. S., Georgeff M. P.: Modeling Rational Agents within a BDI-Architecture. In Proc. Second International Conference on Principles of Knowledge Representation and Reasoning, San Mateo, CA, USA, Morgan Kaufmann publishers Inc. (1991) 473–484
15. Soriano F. J., Reyes J., Gómez G., Amo F.: Extending the FIPA Interoperability Model to Deal with Agent Social Issues. In Proc. 6th. World Multiconference on Systemics, Cybernetics and Informatics, Orlando, USA (2002)

Author Index