ABSTRACT
Mainstream research in Web Services is currently looking at two main aspects, namely formally describing interactions among services, and finding and combining services. Much work made in the intelligent agents area is being applied to these issues. In this paper, we investigate the application of agent research to Web Services from a different perspective, that is, procedural learning. The final objective is to enable an adaptive system (an agent in our terminology) to discover or being fed with knowledge concerning how to solve a specific set of problems in a specific software or physical environment. Our work is a preliminary step into the issue, with the main objective of assessing how current Web Services technology can support a component, described in terms of beliefs, desires and intentions, dynamically adapting its behaviour to new environments.

KEYWORDS
Service-oriented computing, web services, intelligent agents, belief-desire-intention architecture.

1. INTRODUCTION
Mainstream research in Web Services (WS) is looking at two main aspects: first, formally describing interactions among services (possibly over long periods of time and having multiple real-world effects, including legally binding actions); second, finding and combining services (e.g., by extending the simple catalogue contained in UDDI repositories with semantically rich descriptions and using the latter for automated composition via planning and for formal verification). As observed for example in AgentLink III, 2004, and by M. N. Huhns, 2002, much work made in the intelligent agents area can be applied to these issues.

In this paper, we investigate the application of agent research to WS from a different perspective, that is, procedural learning. The final objective is to enable an adaptive system (an agent in our terminology) to discover or being fed with knowledge concerning how to solve a specific set of problems in a specific environment (no matter whether the latter is physical, e.g. as in the case of a personal assistant on board of a mobile device, or virtual, e.g. a manufacturing scheduler integrating with its peers in a virtual enterprise).

The main difference between procedural learning and approaches based on service discovery and planning is that the goals that have to be achieved are neither satisfied by a third-party, ad-hoc service (as it is envisaged in the world of composite services built by means of orchestration languages), nor via explicit reasoning by the agent on how to combine whatever elementary components are locally available. The first approach would imply the existence of additional computational servers in the target environment: this is often not appropriate especially when sensitive data would need to be exported from the agent trying to achieve its goals to an intermediate service. The second approach is computationally expensive, sometimes unable to reach a final solution without substantial human input (e.g. when multiple paths are equally possible), and practically not applicable when agents are on board of computationally limited devices but have to react in real-time.
Procedural learning is related to machine learning, in particular agents that learn from other agents via instruction or observation, to semantic web and semantic web services (in particular, to express goals and knowledge more flexibly that with current WS interfaces, e.g. WSDL), and to belief-desire-intention (BDI) and teamwork theories (which have been investigating, among other things, how agents and teams of agents can achieve multiple goals in the face of failures and rapidly changing environments).

Our work is a preliminary step into the issue, with the main objective of assessing how current WS technology can support a component, described in terms of beliefs, desires and intentions, that dynamically adapts its behaviour to new environments (namely, a CooBDI agent in our terminology). CooBDI agents (D. Ancona and V. Mascaldi, 2004) are suitable for modeling “procedural learning agents” which learn from the interaction with other agents. This is made possible by the main feature of CooBDI consisting of a built-in mechanism for retrieving plans from collaborative agents, for example when no local plans suitable for achieving a certain desire are available.

We have implemented the ideas behind the CooBDI theory by means of WS technologies. To this end, we have performed a number of simplifications. We ignore issues with semantics and drastically reduce the representation of desires to what is currently possible by means of WSDL (namely, strings). Agents' plans are expressed in BPEL4WS. The representation of the agent's beliefs remains implicit in the plan's description. A prototype, built on open-source software and available to the research community, shows the feasibility of our approach.

The paper is organised as follows: Section 2 describes the background of CooWS, namely CooBDI and the relationships between agents and WSs. Section 3 describes CooWS as an instance of the CooBDI model, and Section 4 summarises the related work and outlines the future directions of research.

2. BACKGROUND

Following the well known definition of N. Jennings, K. Sycara, M. Wooldridge, 1998, agents are usually characterised as computer systems situated in some environment, that are capable of flexible autonomous actions in order to meet their design objectives. Besides the “weak definition” given above, the notion of an intelligent agent as an entity which appears to be the subject of beliefs, desires, intentions (BDI) proposed by A. S. Rao and M. Georgeff, 1995, is well known and accepted by many researchers.

BDI-style languages allow agents both to be aware of the environment in which they are located thanks to the perception of external events that enter the agents' event queue, and to modify the environment thanks to a set of external actions with which every agent is equipped. They also support the sociality of agents by letting them exchange data. CooBDI and its CooAgentSpeak implementation (D. Ancona, V. Mascaldi, J. Hubner, and R. Bordini, 2004), together with JAM (www.marcush.net/IRS/irs_downloads.html), represent a step forward toward the implementation of the agents' cooperativity since they allow agents to exchange procedural knowledge (plans), besides mere data (beliefs).

The BDI model is characterised by the following concepts: beliefs (the agent's knowledge about the world); desires (objectives to be accomplished); intentions (stacks of plans currently under execution); plans (“recipes” representing the procedural knowledge of the agent). Most BDI systems also include an event queue where both events (either perceived from the environment or generated to notify an update of its belief base) and internal subgoals (generated by the agent itself while trying to achieve a main goal) are stored. The typical BDI execution cycle is characterized by the observation step, where new events – that generate new desires - are perceived; the generation of the plan instances that can be adopted to cope with the new desires (relevant plans); the insertion of the chosen plan instance on top of the intention stack whose execution generated the desire (in case of plan nesting); and finally the execution of the topmost plan of one intention stack.

In most existing BDI-style languages the plan library is static: agents can neither extend their procedural knowledge at run time, nor exchange plans. CooBDI, instead, overcomes these limitations by introducing cooperations among agents to retrieve external plans for achieving desires. It also extends plans with access specifiers and extends intentions to take into account the mechanism for retrieving external plan instances. The CooBDI execution cycle is also modified to cope with all these issues.

- **CooBDI cooperation strategy.** The cooperation strategy of an agent A includes the set of agents which is expected to cooperate with (partner agents), the plan retrieval policy that states when an external plan...
should be looked for, namely only when no plans suitable for achieving a desire are locally stored, or in any case, and the plan acquisition policy that states what should be done with a retrieved external plan, namely, whether it should be saved in the local plan library, discarded or used to replace those plans with the same trigger (i.e., the desire that fired the adoption of the plan).

• **CooBDI plans.** CooBDI plans share with “classical” BDI ones the trigger, the precondition that the current state must satisfy for the plan to be applicable, a body of actions to perform, an invariant condition that must hold during the whole plan execution, a set of actions to be executed if the plan execution terminates successfully and a set of actions to be executed in case of plan failure. Besides these components, a plan also has an access specifier that determines the set of agents which can share it. It ranges over three values: private (the plan cannot be shared), public (the plan can be shared with any agent) and only(TrustedAgents) (the plan can be shared only by the agents contained in the TrustedAgents set).

• **CooBDI intentions.** Intentions are characterized by “standard” BDI components plus components introduced to manage the external plan retrieval mechanism.

• **CooBDI execution cycle.** The execution cycle of CooBDI departs from the classical one to take into account both desires generation and cooperations. It is characterized by three macro-steps: 1. process the event queue; 2. process suspended intentions; 3. process active intentions.

Here, we only discuss what happens when relevant plans for a desire (those plans whose trigger “matches” with the desire itself) must be generated: 1) The intention that generated the desire is suspended. 2) The local relevant plans for the desire are generated and associated with the intention. 3) According to the cooperation strategy, the set of the agents expected to cooperate to retrieve relevant plans is defined. 4) A plan request for the desire is created and sent to all these agents.

In the design and implementation of CooWS we have exploited most of the state-of-the-art WSs languages and technologies, in particular SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), BPEL (Business Process Execution Language) and UDDI (Universal Description, Discovery, and Integration). The exploitation of WSs for agent-based applications is perceived as an hot-topic by the agent community, as stated in the last edition of the AgentLink III Roadmap, 2004:

> Agents cannot be directly invoked like objects but can be assigned tasks by their owners. Nevertheless, they may be constructed using a wide range of technologies, including object technology, Web Services and others. [...] While Web Service technologies define conventions for describing service interfaces and workflows, we need more powerful techniques for dynamically describing, discovering, composing, monitoring, managing, and adapting multiple services in support of virtual organisations, for example.

### 3. COOWS

A CooWS agent adopts the following metaphor inspired by CooBDI.

• **Beliefs.** The variables local to the BPEL processes that constitute the body of the agent's plans can be considered as a metaphor for the agent's beliefs local to that plan, that are not explicitly represented.

• **Desires.** Desires may be either messages structured according to the FIPA ACL standard ([http://www.fipa.org/](http://www.fipa.org/)), or unstructured Java strings.

• **Actions.** There are two kinds of actions, those that may appear inside the BPEL specification of the agent's plan body (plan's actions), and those that must be executed in case of success or failure of the achievement of a desire (success and failure actions). Plan's actions are thus only external ones, and they consist in the invocation of a WS by means of the BPEL invoke statement. There are three types of plan's actions: 1) delivery of ordinary events, discussed later in this section; 2) achievement of new desires, that supports the plan nesting mechanism; and 3) invocation of existing WSs. Cooperative requests for plans are managed transparently to the agent, and do not belong to the set of actions that can be programmed by the user. Success and failure actions are associated with each external event that enters the event queue, and state what to do in case the management of that event succeeds (resp. fails).

• **Plans.** Plans are defined by a unique plan identifier, a trigger, a body, and an access specifier. The trigger is the desire for which the plan has been defined; the body is a string (mBodyID) that uniquely identifies the serviceName of a BPEL process; the access specifier (CooAccessSpecOnlyTrusted, CooAccessSpecPrivate, and CooAccessSpecPublic) implements the three access policies described in Section 2. Our choice of including a reference to a BPEL process file instead of the file itself as the plan body
is due to efficiency reasons, since specifications of BPEL processes may be very large (just to make an example, the WSDL representation of the web service offered by eBay is 35000 lines of code). The instance of a plan is the BPEL process identified by the mBodyID field of the plan, that executes on the BPEL engine.

- **Intentions.** An intention contains a stack of desires, a boolean attribute defining the intention's state (either active or suspended), and the success and failure actions. The set of relevant plans currently available to the agent is associated with the corresponding desire on the stack. The set of relevant plans is generated as follows: the plans are first looked for in the local plan library. If the set of local relevant plans is empty, or if the retrieval strategy is to always retrieve external plans, the retrieval of external plans is started: a request of cooperation is delivered to the partner agents, the agent goes on with its activities, and, when answers to the cooperation request enter its event queue, it associates the received plans with the desire. Intentions are suspended when there are no active plans associated with their topmost desire. If an intention remains suspended for more than a given amount of time, it fails.

- **Events.** There are four kinds of events: cooperation, ordinary, plan outcome, and achieve events. Here we only discuss the first two ones, since plan outcome and achieve events are related to the plan nesting mechanism which is not central in this paper. A cooperation event is either a request, characterised by sender, receiver, unique request identifier, and desire for which the request has been issued, or a provide event, characterised by sender, receiver, unique request identifier, and set of plans that are relevant for the desire appearing in the corresponding request. Ordinary events consist of the reception of messages from other agents. Events are managed by a first-in-first-out event queue.

- **Retrieval and acquisition strategies.** They are integer fields that can assume values corresponding to the different types of retrieval and acquisition strategies described in Section 2.

- **Agent.** An agent is characterised by the agent’s identifier, its partners, the agent’s plan library, its retrieval and acquisition strategies, its set of intentions and its event queue, respectively. An agent is equipped with methods for handling incoming events, according to their type (cooperation, ordinary, plan outcome, and achieve).

The implementation of the CooWS platform, downloadable from the web site http://coows.altervista.org, relies entirely on opensource tools that include ActiveBPEL, Apache Tomcat and Axis, jUDDI, UDDI4J, and MySQL.

4. RELATED AND FUTURE WORK

In this paper we have described CooWS, an implemented system, that represents a preliminary step towards an adaptive BDI framework based on WS. The advantages of integrating CooBDI and WS are twofold. On the one hand, CooWS provided an hint to implement BDI-style agents that are situated in a real software environment, the Web. On the other hand, it made possible to understand some limitations of current WS technology with respect to our long term research, namely the limited support given to adaptivity of procedural knowledge, and gave hints on an architecture both for learning agents and the environment in which they act.

The adoption of semantic WSs as a means for specifying the behaviour of agents and MASs is envisaged by P. Buhler and J. M. Vidal in two papers, Semantic Web Services as Agent Behaviors, 2003, and Adaptive Workflow = Web Services + Agents, 2003. In the first paper, a technique for providing agent software with dynamically configured capabilities is discussed. These capabilities, described with DAML-S, can represent atomic or orchestrated WSs. The DAML-S specification can be transformed into an executable program written in a composition language. When executed, the composite service will be available as a semantically described behaviour within a FIPA compliant agent. In the second paper, Buhler and Vidal advance the idea that BPEL can be used as a specification language for expressing the initial social order of a MAS, which can then intelligently adapt to changing environmental conditions. While our proposal differs from the one contained in Adaptive Workflow = Web Services + Agents, where BPEL is used to express knowledge at the MAS level, and not at the level of the single agent as we do, it shares many similarities with the one described in Semantic Web Services as Agent Behaviors. Another proposal that is driven by motivations similar to ours is that by K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan, 2004, who suggest to extend the OWL-S Model Processing Language with a new statement called exec. The exec statement takes a
process model as input and then executes it in order to support a broker agent in both discovery and mediation.

Although much work is currently done on agents that reason on specifications of WSs in order to match services in an intelligent way, we were not able to find proposals to adopt process languages for WSs to specify the procedural knowledge of the single agent or of the overall MAS, besides the three discussed above.

As far as our current and planned activities are concerned, we are working on the implementation of “digital butlers” that query Google (which can be accessed as a web service) to arrange travels and to organise meetings for their principals. The plans available to the digital butlers do not cover all the requests that may arrive from their principals, and the lack of plans for coping with an incoming request fires the collaborative exchange of plans. In the future, we are willing to explore: 1) the ability to integrate an ontology into the system, so that matching between desires and triggers of plans can become more sophisticated than a simple comparison of strings; in particular, we plan to follow the proposal of N. Srinivasan, M. Paolucci, K. Sycara, 2004 of using OWL-S plus UDDI to implement an advanced matchmaker; 2) the ability to dynamically update the set of trusted partners following reputation mechanisms such those suggested by J. Sabater, 2003.

REFERENCES