Agents are not (just) web services: considering BDI agents and web services

Ian Dickinson
Hewlett-Packard Laboratories
Filton Road, Stoke Gifford,
Bristol, UK
+44 117 312 8796
ian.dickinson@hp.com

Michael Wooldridge
Department of Computer Science
University of Liverpool
Liverpool, UK
+44 151 794 3667
mjw@csc.liv.ac.uk

ABSTRACT
Web services and software agents share a motivation of aiming to facilitate more flexible and adaptable I.T. systems. Web services are increasingly being used to provide active behaviour over the Internet, and promise end-user benefits that, in previous work, have been associated with agent systems. Thus it is natural to consider what relationships agents should have to web services. We argue that agents and web services are distinct. In our work, agents provide a distinctive additional capability in mediating user goals to determine service invocations. In this paper, we review some of the design choices for integrating agents and web services, and illustrate one approach using reactive planning to control web-service invocation by BDI agents.

Keywords
Web services. Software agents. Agent architectures. BDI.

1. INTRODUCTION
Increasing the flexibility of I.T. systems is a common strategy to cope with ever more complex and demanding user requirements. In most approaches, such flexibility derives from breaking larger units of functionality into smaller components that interact over networks to deliver a variety of end-user capabilities. An essential part of this strategy is to avoid rigid, highly interdependent linkages between the components. Two common approaches to creating flexible, loosely-coupled systems are asynchronous *message-passing* architectures and synchronous *remote procedure call* (RPC) architectures. Both styles have useful characteristics. *Software agents*, one strand of research and development into flexible I.T. systems, extend message-passing architectures to exhibit properties of being social, reactive, proactive and autonomous [1].

Throughout the 1990s, the idea of composing systems as collections of loosely-coupled software agents received considerable attention. More recently, other researchers have investigated the use of *web services*, initially based on the RPC approach, to meet similar goals of building flexible I.T. systems. The motivations of web service designers are similar to those of agent designers ([2], ch. 1), despite the differences in technology. To a large extent, the strict RPC metaphor has fallen out of favour, mostly due to interoperability problems. For example, the Web Service Interoperability Basic Profile suggests that web services be seen as accepting a XML document defining the input, and returning an XML document defining the output. While this change de-emphasises the RPC nature of the interaction, it is still very procedural in nature.

The increasing number of books, journals and conferences about web service technology suggest an accelerating take-up by the computing industry. This leads, in our view, to a number of questions for agent researchers:

- how should agents make use of web services?
- what use, if any, should web services make of agents?
- how can agents enhance the web services paradigm?

As web service architectures, standards and tools mature, we see an opportunity to revisit some of the assumptions implicit in current agent tools and platforms. For example, many of the infrastructure-centric capabilities of the FIPA specification suite [3] are very adequately covered by more recent web service standards. That web services have been more widely deployed suggests also that the web-service specifications have been tested more stringently in practice. If it is true that agent infrastructures might benefit from adopting web service technology, we also believe that some of the ideas from agent research are ideally suited to enhancing and extending service-oriented applications.

In the rest of this paper, we explore these complementary issues in the context of a BDI agent platform. The remainder of the paper is structured as follows: we first review, very briefly, the salient aspects of web services in section 2. In section 3, we consider current approaches to integrating web services and agents, and then in section 4 we show how we have approached the problem in our BDI platform. We conclude with some lessons learned from a prototype system.

2. Background: web services
There are numerous definitions of the term *web service*. The W3C defines a web service as: "*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network*" [4]. The W3C definition goes on to specify the use of SOAP [5] and WSDL [6] as communication standards, but this would preclude those web-services that are based on a REST-style [7] interface. In our work we don't limit consideration to only SOAP-based web services.

Service oriented architectures (SOA) describes an approach to building business applications based around web services. A very large number of standards, some of which are still being developed, describe many aspects of SOAs, from basic mechanisms for exchanging data, though service directories to aspects of security, message exchange patterns and business process description.

A common pattern for web-service implementation is familiar from RPC styles of distributed computing. A *service* exposes *operations*, these operations consume *inputs* and produce *outputs*. Service invokers communicate with web services over HTTP, and typically use XML as a meta-language for encoding inputs and

outputs. These elements are typically described in a machine-readable specification, such as WSDL [8], which is also encoded in XML.

Advocates for web-services often emphasise *loose coupling* ([9] p76) of service components, to make applications more resilient, and their components more reusable, by minimising explicit data and control flow dependencies between services. Instead, elements of the service description are combined with a process description to determine which service operations are invoked in which sequence. A variety of approaches exist for constructing complex behaviours from loosely-coupled components. This is often termed web service *choreography* [10]. Such approaches define, at varying levels of abstraction, possible sequences of web-service invocation, and the dataflows between them.

2.1 Semantic web services

It may be observed that WS choreography and similar approaches concentrate on the *operational* aspects of web-service composition. While such choreography can add flexibility and resilience to an application architecture, it does imply that the actual services to be invoked are known, or discovered by the developer, at design time. There are situations in which such an operational approach is inadequate, for example:

- A service of a given type is required, but the identity and location of the provisioning service is not known (or cannot be known) at design time;
- A given capability is required, or some end effect needs to be achieved, but again the identity and location of the service, or services, that can fulfil the need are not known at design time;
- The process of invoking the service is more complex than simple RPC, for example it is strongly context-dependent, or it requires resource-management or negotiation.

To go beyond the standard choreographing of known services, we require additional information about what the service will do, should it be invoked. The service must be sufficiently self-describing that, if the caller subsequently decides to invoke that service, it is able to do so. To maximise flexibility, such descriptions need to be machine-processable, so that as much as possible of the flexible adaptation-to-circumstances takes place without human intervention. In particular, such adaptations should place a minimal burden on the end-user.

Following from recent work in the semantic web [11], one approach to this objective is termed *semantic web services* – the provision of semantic-web style *semantic descriptions* of services and processes. A number of semantic web services projects and standards initiatives are underway, but space does not permit a complete exploration here. Instead, we briefly outline one of the prominent technologies, OWL-S [12], as an exemplar that typifies the semantic web services approach.

An OWL-S description of a web service has three components:

- A *service profile*, that describes what the service does;
- A *process model* that specifies, in abstract terms, the operation of the service;
- A *grounding model* that specifies how other processes should invoke the service being described.

The service profile describes the preconditions that should exist prior to a service being invoked, the effects that occur as a result of invoking the service, and the explicit inputs to and outputs from the service. These *inputs, outputs, preconditions and effects* (IOPE) are described using a pre-defined OWL ontology. Using the IOPE descriptions, it is possible to use algorithmic approaches, including planning, to construct complex services from simpler components without depending on a human programmer.

2.2 Limitations of semantic web services

There are clearly scenarios in which the service descriptions provided by semantic web services research do provide effective solutions. For example, consider a supply-chain automation problem. Given a description of the required materiel for a certain production process, it is easy to imagine that a well-designed application could make use of semantic descriptions of component suppliers' ordering and estimation processes, and logistics providers' shipping and tracking processes, to ensure a smooth production supply. The supply chain manager process should be able to switch suppliers straightforwardly if one supplier forecasts a component shortage, or a delivery channel fails. The semantic descriptions of the services allow some robustness to variances in the interfaces to the different suppliers' services.

However, a more open-ended scenario presents greater challenges. In [12] section 2, it is suggested that OWL-S will help a user to locate a service that (i) sells airline tickets between two given airports, and (ii) accepts a certain type of credit card. We might speculate that the user's overall goal may be to get home in time for Thanksgiving, nevertheless the interaction is based around much more basic operations. This puts a strong onus on the user to decompose their own goals down to a level of necessary *basic actions*, which may then be performed by web services. But if the user has to perform this goal decomposition themselves, and form a suitable plan for achieving their goals, it is unclear how the automation provided by the web service is genuinely helping that person. If that user is able to analyse their own needs to that necessary degree, would it not would be simpler and easier for them simply to use a conventional travel web site to book their trip?

We propose that much of the putative benefit from flexible, advanced IT systems is largely contingent on increasing automation. We propose that more of this benefit will be delivered when users can specify the *goals they wish to achieve*, rather than the *actions they wish to perform*.

2.3 Software agents

A natural idiom for encoding and executing goals is through *software agents* [1]. For the purpose of this discussion, we restrict our attention to *deliberative agents* [13], that is, agents that have a symbolic knowledge representation, and which use symbolic reasoning to achieve their behaviour. In our work, we are particularly interested in ways that human users interact with agents, especially agents that behave autonomously. Such autonomous behaviour shifts the basis of the interaction from a *direct manipulation* model to a *delegation* model [14]. One advantage of deliberative agents over other approaches is that key elements of the user-agent interaction, for example the user's goals

or the agent's strategies, have an explicit representation. Crucially, this enables those objects themselves to be part of the dialogue. The user could, for example, critique the agent's strategy for achieving a given outcome, perhaps by refining or updating their own expressed goal.

Deliberative agents use symbolic structures, founded on predicate logic, to represent knowledge. In particular, logical formulae stand for mental attitudes in both the user and the agent, where mental attitudes include beliefs, desires, preferences and so forth. Often, modal operators, qualified by the name of the actor, distinguish (say) the agent's beliefs from the user's beliefs.

3. Agents and web services

There are many different ways in which to consider the relationship between software agents and web services. Drawing from the published literature, we identify the following common themes:

3.1 Theme 1: no conceptual distinction

One view (see, for example, [15]) is that agents and web services are not conceptually distinct. In this view, there is no *conceptual* difference between a web service and an agent: both are active building blocks in a loosely-coupled architecture. In such architectures, there is only an engineering problem of creating overall system behaviours from active components.

We reject this position, and suggest that there is a useful distinction between web-services and agents. Moreover, we propose that this distinction is useful to both the system designers and its users. If an agent is able to represent, mediate and proactively act to achieve a user's goals, it will manifest in the user-interface in a different way to non-agent components that do not have those properties. We propose that agents are necessarily those elements of the system that are most parsimoniously describable in terms of mental attitudes, particularly *intention* (the user's or the agent's).

Suppose we wish to represent the intent of the user, e.g. "Mary wishes (i.e. has a goal) to meet the product team in Paris", and we aim to use this intention to structure interactions between the system and the user, and perhaps between system components. There must be a locus for the representation of this intent in the system. Mary's digital travel assistant might represent her Paris-travel goal, and subsequently adopt an intention to assist with travel planning. The observable behaviours of a component that holds understands user goals and can adopt its own mental attitudes in response are distinct and different from deterministic components. Clearly a given software component can both represent intention and act as a web service, but this makes it different from traditional web services, which don't. Hence representing intention is, in our view, the key conceptual difference between agents and services.

3.2 Theme 2: bi-directional integration

A second theme in the literature is that agents and web-services can interoperate by either of them initiating communications [16], [17]. That is, agents can *invoke* web services, *and vice versa*. The work of Greenwood and Calisti [16] shows clearly that it is feasible for web-services to invoke an agent capability, providing that an appropriate WSDL to ACL mapping is in place. However, we view the invocation of agents by web services as problematic. The implication of the web-service to agent invocation is that the agent must expose pre-determined behaviours, for example named operations with known parameters. Suppose such exposed methods represent fixed, deterministic behaviours. This makes the invoking service easier to write, but violates the presumption of the autonomy of the agent. It is not clear why a software component that behaves in this deterministic manner can be termed an agent. If the invoked agent is not fixed and deterministic in its behaviours, the invoking web-service must behave in an agent-like manner to adjust to the agent's autonomous responses. If the behaviour of the web-service is not distinguishable from an autonomous agent, then we argue that it should be regarded conceptually as an agent, not a service.

In our model we regard web-services as more primitive than agents. If an agent is to behave plausibly autonomously, and respect its (and its user's) current intent, then it can only expose the most generic interfaces to other services – such as the delivery of a message or event.

Greenwood and Calisti also propose that agents exposing their capabilities as web services should use an adapter to translate between SOAP and ACL requests. An agent registers entries in a FIPA *directory facilitator* (DF) to advertise its abstract services, and it is these that are made available as web services via the adapter. This implies that agents advertise their capabilities and roles procedurally, using the operations they can perform, rather than declaratively describing their capabilities. This is also different from our approach.

3.3 Theme 3: agents invoke web services

A key proposal of web-service architectures is that simple (atomic) services can be composed together, in a workflow, to form complex composite behaviours. A number of researchers (e.g. [18], [19], [20]) have explored the use of AI planning to compose complex behaviours. Such planning is performed on behalf of a user to meet some set of goals. This suggests a layered view, as shown in Figure 1 (below).

In this view, agents primarily are responsible for mediating between users' goals, and the available strategies and plans. Agents invoke, or design, atomic or composite web services as necessary.

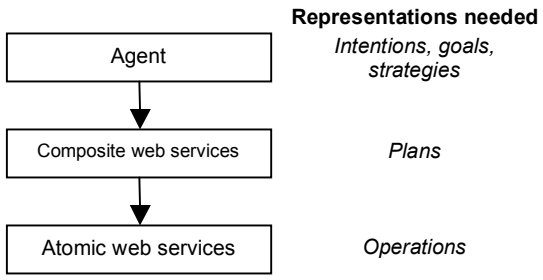

Figure 1: layered view of agent-ws interactions

A related approach is explored in [21], although in this work the authors seek to generalise the interface to web services from specific operations to generic operations that are analogous to speech acts. So a web service might have an *inform* operation, with an argument which has a similar role to `:content` in a FIPA ACL message.

Our approach broadly follows this third theme. Web services are invoked by agents as component behaviours, but autonomy and intent is only represented at the agent level. In the remainder of this paper, we explore how this general theme is embodied in our experimental BDI agent platform.

4. NUIN BDI AGENT PLATFORM

The Nuin agent platform [22] is an open-source Java implementation of a combination of a belief-desire-intention (BDI [23]) agent platform and semantic web techniques. A particular goal of the Nuin architecture was to make the platform easily extensible by agent developers. The outline architecture of Nuin is shown in Figure 2. A key extension point is the *abstract service boundary*. The original design intent for the abstract service boundary was as a means to add custom behaviours to the agent, written as Java plug-ins. For example, an incoming event might trigger a plan, which would delegate handling of the event to a GUI, incorporated as a plug-in capability.

This abstract service boundary provided a natural basis for extending the internal agent services to include external web services. So, for example, with the correct service binding in place, an *invoke* action from the agent script can directly call an operation on a web-service, and bind the result to a script variable. Moreover, this abstraction boundary also provides a natural place to encode *know-how* – the knowledge that an agent has of its own capabilities [24]. Currently, we use an RDF [25] knowledge base to store any local meta-knowledge an agent has about its own capabilities. The set of known web services may be fixed at design time, or dynamically extended at run time. Agents can dynamically create web-service bindings by fetching and parsing the WSDL service description.

4.1 Service descriptions

In order to determine which services to utilise to achieve a given goal or satisfy a given intent, the agent requires meta-data describing the available services. As an example, consider one aspect of a typical knowledge management application. As part of this application, the user can specify a search string to locate articles stored in the systems' database. While it could be said that the user's intent is to *perform a search with the given terms,* it is perhaps more accurate that the user's specific intent is to *locate a document relevant to a certain task*, where the task might be generating a customer bid. Indeed, the overall intent is to satisfy the customer's RFP, with the "locate-document" intent as a component of that overall goal. Suppose that the agent has access to a number of services, including a database search service, and a query-rewrite service. The query rewrite service has a number of tactics for modifying the user's query, for example performing WordNet [26] synset expansion or narrowing. We would like the agent to be able to offer strategic choices to the user, including the choice of the composite service of searching on the re-written query string. How does the agent know to offer this composite service to the user, to help satisfy the document location intent? The agent must be able to determine that a given service (including composite services) is:

- *relevant* to the user's intent
- strategically *useful* to meeting the user's goals
- *describable* to the user (when user assent is required before enacting the service)

One key role of a service description is to provide meta-knowledge that the agent will use to inform such decisions. A WSDL service description describes the *type signature* of operations. For example, the above query-expansion service takes a query string argument, and produces a new query string. It has `string` → `string` as a type signature. However, this type signature applies to many other string manipulating operations, so knowing the type signature alone of an operation is insufficient meta-knowledge to determine whether the operation is relevant to the current goal. Post-conditions in the service description can make the description more precise. The query-expansion service might perhaps state, as a post-condition, that the returned string is a `moreGeneralQuery` than the input string, assuming there is a suitable ontology in which levels of query generality are defined. This is the kind description that might be provided by an OWL-S semantic web service description. However, this still leaves open the issue whether *generalising a query* is a relevant and useful tactic to offer to the user. Such strategic knowledge does not fit conveniently into the OWL-S 1.1 framework. Our current approach is to encode strategic knowledge directly into the BDI agent's knowledge base.

An earlier anonymous reviewer of this paper suggested that encoding the strategic knowledge in the agent's KB simply introduces a knowledge acquisition bottleneck into the design process. This is a fair criticism, but serves to underline a fundamental difficulty. The semantic web services descriptions cannot express context-dependent knowledge. A given web service might be useful to one user, given his or her goals and preferences, but (relatively) useless to another user with similar goals but whose context is different. The process of mapping from high-level goals to services to invoke must draw on knowledge of both the service capability and the user's context. In order to remain general, the web service description cannot be too specific to any given user's goals. This remains an area where additional research is required.

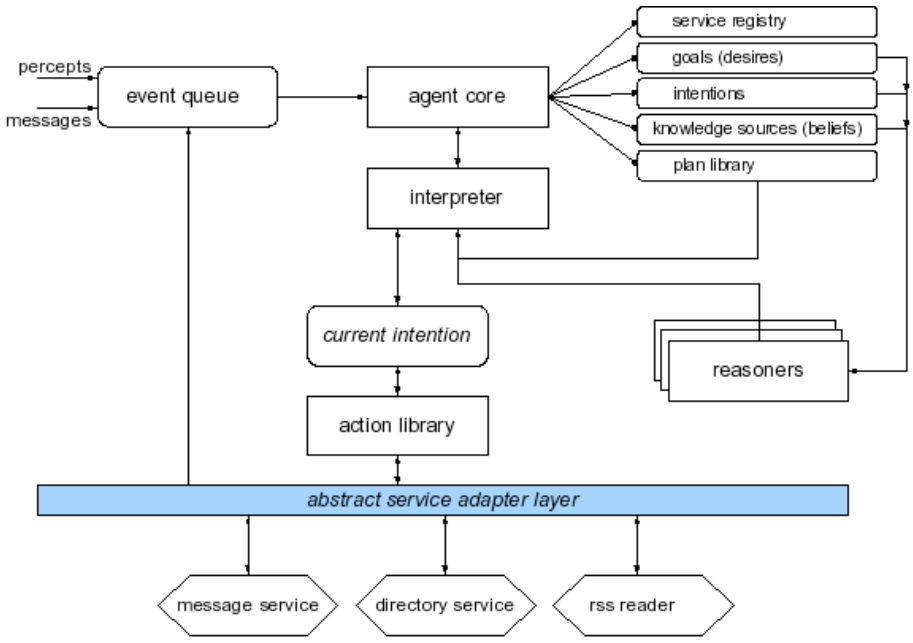

Figure 2: outline Nuin architecture

We note that the Web Services Modelling Ontology (WSMO) [27] includes the concept of a *wgMediator*, which is claimed to encode the mapping between a goal and a web service. However, the details of the definition, use and semantics of mediators are unclear in the current version of WSMO, so we have not investigated this further.

4.2 Integrating web services in BDI agents

A goal of BDI architectures is *practical reasoning*: an attempt to achieve effective computational performance in autonomous systems by balancing consideration of how to act with acting. BDI agents commit to a course of action, represented by an intention, based on their current beliefs about the world and their current goals. In order to be able to react to the changing state of the world, it is important that a BDI agent be able to adopt new intentions, and drop or modify existing ones if they are no longer relevant.

In a typical BDI architecture, such as PRS [28] or AgentSpeak(L) [29], the agent's starting state includes a plan library, which the agent uses to control its behaviour, rather than utilising planning from first-principles. In typical practical reasoning approaches, an intention is either the *intention-to* perform a given plan, or the *intention-that* the post-conditions of a given plan become true.

In reactive planning, a complete plan to achieve a given goal is not constructed *a priori* and then executed. Instead, a library of general pre-defined (i.e. defined at compile time) plans is provided to the agent, and the agent performs one or more of these plans in response to its perceptions of the environment. Thus the agent reacts to actual conditions of the world. There are two principal advantages of reactive planning: it is computationally more efficient, since a large search-space does not have to be explored, and it does not require the planner to have available a symbolic model of the possible effects of actions and the initial state of the world. By contrast, *first-principles planning* [30] approaches require full knowledge of the initial world-state and of the changes that will be brought about by performing a given action. The initial-state requirement can be mitigated to some extent by explicitly planning information-gathering steps into the plan [31], but it remains the case that planning from first principles is computationally expensive ([30], §3.4).

Given that a BDI agent will select a course of action based on its environment (determined by incoming messages or sensed percepts), a key issue in BDI approaches is what the agent should do if it determines that it has more than one possible course of action. In PRS, the interpreter only proceeds once there is exactly one relevant plan to follow. If more than one plan is relevant at a given step, the interpreter treats this as a problem that can be solved by a meta-level evaluation of the choices. This recursion continues until a single course of action has been selected. In AgentSpeak(L), Rao abstracts this plan-selection problem into pre-specified *evaluation functions*, which select a single event to process, or a plan to adopt, given multiple choices. While the evaluation functions encapsulate the abstract requirement, Rao does not offer a practical solution to the representation of evaluation functions.

With Nuin, we have decided not to adopt the recursive approach of PRS, since in our experience it creates conceptual (and debugging) difficulties for the agent programmer. We have allowed for variable evaluation functions in the interpreter architecture, following AgentSpeak(L), but this is also unsatisfactory in general. We would like the agent's choice of action to be, at least in-part, determined by the agent's current mental state (i.e. current beliefs, desires and intentions). This is not well-defined if the decision procedure is contained within the interpreter. Indeed, Rao's evaluation functions in AgentSpeak(L) do not take the agent's current mental state as a parameter. We do define in the agent script language actions for modifying the current mental state, for example adopting or dropping a goal or intention.

The general problem, then, is how to define strategic knowledge that the agent can use both to select its own course of action, and to converse with the user in terms that match the user's

conceptualisation of the domain. Our current experimental approach is to utilise a structured goal language, in which strategic knowledge is encoded into a declarative goal structure. EaGLe [32] is one example approach. Like EaGLe, we define a small number of goal refinement operators (for example: *all* sub-goals, *any* sub-goal, *sequence* of sub-goals, *perform* a plan). These goals are stored in an RDF knowledge base, which can then be augmented by the user. For example, the agent may presume to achieve goal *g* by sub-goals *g0*, *g1* and *g2* in order. A given user may override the reduction of *g* as *g2* then *g0*, ignoring *g1*. This is only a rather crude first step, but will allow us to explore the, and refine, the user's ability to influence the agent's behaviour through entering a dialogue around such strategic choices.

One particular reason for using an RDF representation to encode and store the goals themselves is to permit the use of semantic web technologies to allow goals, or goal strategies, to be shared. We have not yet investigated this in detail, but one way to at least mitigate the knowledge acquisition problem alluded to above, would be to allow users in a community to share strategies. In particular, sharing strategies that map the pre- and post-conditions of newly introduced services to general goal conditions, perhaps related to a shared upper ontology, would help an agent community quickly integrate new capabilities. With this in mind, the use of URI's for symbols, and other RDF modelling commitments, becomes especially valuable.

Whether or not this particular approach is shown to work effectively, we argue that the current semantic web services approaches, on their own, lack a standard means to allow an agent to relate its intentions in pursuit of stated user goals to the capabilities of services. We don't argue necessarily that either OWL-S or WSMO should be extended to cover this need, just that there is a currently unmet requirement.

4.2.1 Interleaving planning and acting

Reactive planning, as exhibited by practical-reasoning agents, mixes planning with acting. The changes to the world state, combined with the agent's current intention set, determine the choice of next action. Reactive planning uses libraries of pre-defined plans, which are activated by the agent's perceptions of its environment.

While the reactive planning approach has some advantages, it does suffer from a significant drawback when actions have side-effects. The Nuin interpreter allows chronological backtracking through side-effect free actions, but it does not permit any backtracking through actions that have side-effects. For internal actions, the side-effecting nature of an action is part of the action description. For web-services, it is not clear, *a priori,* whether an operation is safe to repeat or backtrack through. For example, the operation of determining the weather forecast for a certain city is probably idempotent, but booking a plane ticket is not. By definition, REST-style web services using the HTTP GET method should be idempotent[1]. In general, however, idempotency is not known. We currently understand that neither OWL-S nor WSMO allow for action idempotency to be specified in a service description.

[1] Though this is not always the case; the web service API provided by Amazon.com is RESTful, but includes side-effecting operations.

For some actions, it may be possible to specify a *compensation* action, to perform if the agent wishes to reverse some partially-completed action. This might mean, for example, cancelling a non-committed transaction in a transactional system. In general reversing an action is a difficult and open research question. It does suggest, however, that some applications, even if predominantly using a reactive planning approach, may require some online planning to plan ahead before committing to a course of action.

5. Discussion

A central hypothesis of our work is that explicitly referencing goals and intentions provides a more cogent and flexible foundation for human-assisting agent dialogues. Deliberative agents provide the representational tools to store and manipulate such mental attitudes, and this distinguishes a software agent from a complex web service.

Nevertheless, web services are being increasingly deployed as units of active behaviour on the web. Our work has shown how a BDI-style agent, using a reactive planning approach, can mediate between the representations of the user's and the agent's mental attitudes, and the operational semantics of the web service. Crucial to this mediation is the provision of knowledge about the web-services to be invoked. Current semantic web service descriptions provide some, but not all, of the necessary knowledge. The particular issue that we found is the need for strategic knowledge, which can assist the agent to make, or suggest to the user, decisions about choices of which service to invoke.

In contrast with web-service composition techniques based on planning, reactive planning requires fewer runtime computational resources and does not require a complete model of the symbolic effects of actions and the world's initial state. However, reactive planning does risk over-commitment to ultimately non-viable courses of action, which can be problematic if the actions themselves are side-effecting on the world. We anticipate, therefore, that some agent applications will always require the ability to plan ahead in time and consider future courses of action without performing actions. Attempting to use reactive-planning for web service selection has highlighted that the current semantic web service descriptions do not provide a means to describe side-effects, or failure recovery actions.

We have experimented with explicitly encoded meta-knowledge, added directly to the agent's knowledge base, to assist the process of mapping between the user's (highly context-dependent) goals and the (context independent) semantic descriptions of service capabilities. Once strategic knowledge is a first-class object in the agent's knowledge base, we can enlist the user's assistance to adjust the agent's strategy either by directly modifying strategy parameters, or by updating the original goal. We view this as a crucially important aspect of human-agent interaction, and our current approach is just a preliminary step. As they encoded in RDF/OWL, we could in principle allow such strategic knowledge to be shared among members of a community. We have not fully investigated this yet, but it is one possible approach to mitigating the arguably high cost of acquiring strategic knowledge.

6. Conclusions and future work

There seems little doubt that web services will be part of the I.T. systems landscape for the foreseeable future. Current web services approaches and standards provide much of the general

infrastructure currently handled by agent platforms. We conclude that agent software should evolve to concentrate on higher-level capabilities that integrate web-service components into highly flexible solutions to better meet user needs in the face of increasingly complex computational systems. In our view, the essential role of software agents in this scenario is to encapsulate user's intentions. An essential element of this evolution will be determining the mechanisms by which strategic knowledge about the uses of a given web service, relative to user goals, is encoded and made available.

Our future work will investigate better means of adding strategic knowledge into the agent platform, to allow the agents to make more automated decisions. We will also add support for directly utilising semantic web service descriptions in OWL-S

7. REFERENCES

1. Wooldridge M. & Jennings N. "Intelligent Agents: Theory and Practice". *Knowledge Engineering Review*. Vol. 10:2. 1995. pp. 115-152.

2. Zimmermann O, Tomlinson M, Peuser S. *Perspectives on Web Services* Springer, 2003.

3. Foundation for Intelligent Physical Agents (FIPA). *Specification index*. 2004. http://www.fipa.org/specifications/index.html

4. Booth, David , Haas, Hugo, McCabe, Francis, Newcomer, Eric, Champion, Michele, Ferris, Chris, and Orchard, David. *Web Services Architecture*. (W3C Working Group Note) 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

5. W3C. *Simple Object Access Protocol (SOAP) 1.1*. 2000. http://www.w3.org/TR/SOAP/

6. Kantor, P. B. , Boros, E., Melamed, B., & Meńkov, V. "The Information Quest: A Dynamic Model of User's Information Needs". In: *Proceedings of tthe 62nd Annual Meeting of the American Society for Information Science*. American Society for Information Science. 1999. pp. pp. 536-545. http://aplab.rutgers.edu/ant/papers/quest/quest.htm

7. Fielding R. & Taylor R. N. "Principled Design of the Modern Web Architecture". *ACM Transactions on Internet Technology*. Vol. 2:2. 2002. pp. 115-150.

8. Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. *Web Services Description Language (WSDL) 1.1*. 2001. http://www.w3.org/TR/wsdl

9. Singh M, Huhns M. *Service-Oriented Computing* Wiley, 2004.

10. Burdett, D. & Kavantzas, N. *WS Choreography Model Overview (working draft)*. 2004. http://www.w3.org/TR/2004/WD-ws-chor-model-20040324/

11. W3C. *W3C Semantic Web Activity*. 2001. http://www.w3.org/2001/sw/

12. Pretschner, A. & Gauch, S. "Ontology Based Personalized Search". In: *Proceedings 11th International Conference on Tools with Artificial Intelligence. TAI 99*. IEEE Computer Society. pp. 391-398.

13. Wooldridge M . *Reasoning About Rational Agents* MIT Press, 2000.

14. Negroponte N. Agents: From Direct Manipulation to Delegation. In: Bradshaw J., ed. *Software Agents*. AAAI Press , pp. 57-66, 1997.

15. Breese JS, Heckerman D & Kadie C. "Empirical Analysis of Predictive Algorithms for Collaborative Filtering". *Proceedings of the 14th Conference on Uncertainty in....* 1998.

16. Greenwood, D. & Calisti, M. "An Automatic, Bi-Directional Service Integration Gateway". In: *Proc. Workshop on Web Services and Agent-Based Engineering (WSABE'2004)*. 2004. http://www.agentus.com/WSABE2004/program/7.pdf

17. Good, N., Schafer, J. B., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J., & Riedl, J. "Combining Collaborative Filtering With Personal Agents for Better Recommendations". In: *Proc. 16th National Conference on AI (AAAI-99)*. AAAI Press, 1999. pp. 439 – 446.

18. Sirin, E. & Parsia, B. "Planning for Semantic Web Services". In: *Proc. Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*. 2004. http://www.ai.sri.com/SWS2004/final-versions/SWS2004-Sirin-Final.pdf

19. Horrocks I. Reasoning With Expressive Description Logics: Theory and Practice. In: Andrei Voronkov, ed. *Proc. 18th Int. Conf. on Automated Deduction (CADE-18)*. Springer Verlag, pp. 1–15, 2002.

20. Pistore, M., Barbon, F., Bertoli, P., Shaparau, D., & Traverso, P. "Planning and Monitoring Web Service Composition". In: *Workshop on Planning and Scheduling for Web and Grid Services*. 2004. http://www.isi.edu/ikcap/icaps04-workshop/final/pistore.pdf

21. Gibbins, N., Harris, S., & Shadbolt, N. "Agent-Based Semantic Web Services". In: *Proc. Twelfth International World Wide Web Conference*. ACM, 2003. http://eprints.ecs.soton.ac.uk/7278/

22. Dickinson, I. *Nuin: the Jena Agent Framework*. 2004. http://www.nuin.org

23. Rao, A. & Georgeff, M. "BDI Agents: From Theory to Practice". In: *Proc. First Int. Conf on Multi-Agent Systems (ICMAS-95)*. 1995.

24. Singh M. *Multiagent Systems: a Theoretical Framework for Intentions, Know-How, and Communications* Springer-Verlag, 1994.

25. World Wide Web Consortium (W3C). *The Resource Description Framework (RDF)*. 2004. http://www.w3.org/RDF/

26. Miller G. "WordNet: an on-Line Lexical Database". *International Journal of Lexicography*. Vol. 3:4.

27. Lara, R., Roman, D., Polleres, A., & Fensel, D. "A Conceptual Comparison of WSMO and OWL-S". In: *European Conference on Web Services (ECOWS 2004)*. 2004. http://www.uibk.ac.at/~c703225/papers/conceptualcomparison.pdf

28. Georgeff, Mike and Ingrand, François. *Research on Procedural Reasoning Systems (Final Report – Phase 2)*. SRI International. 1990.

29. Rao, A. "AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language". In: *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '96)*. Springer-Verlag, 1996. pp. 42–55.

30. Ghallab M, Nau D, Traverso P. *Automated Planning : Theory & Practice* Morgan-Kaufmann, 2004.

31. Kuter, U., Sirin, E., Nau, D., Parsia, B., & Hendler, J. "Information Gathering During Planning for Web Service Composition". In: *Proc. Third International Semantic Web Conference 2004 (ISWC2004)*. http://www.mindswap.org/papers/ISWC04-Enquirer.pdf

32. Sycara, K., Lewis, M., Lenox, T., & Roberts, L. "Calibrating Trust to Integrate Intelligent Agents into Human Teams". In: Proc. 31st Annual Hawaii International Conference on System Sciences. IEEE, 1998. pp. 263 – 268.