D 2.4.4
Guidelines for the integration of agent-based services and web-based services

Ian Blacoe (UniLiv)
David Portabella (EPFL)

with additional contributions from:
Shamimabi Paurobally (UniLiv), Thierry Moyaux (UniLiv), Ben Lithgow Smith (UniLiv), Valentina Tamma (UniLiv), and Michael Wooldridge (UniLiv).

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Deliverable D2.4.4 (WP2.4)
This deliverable offers guidelines on approaches to inter-operation and integration between agent services and web services, by examination of these research efforts and discussion of both the main points of correspondence and the main differences between the two paradigms, and where they may beneficially inform each other.

Document Identifier:	KWEB/2004/D2.4.4/v1.1
Class Deliverable:	KWEB EU-IST-2004-507482
Version:	v1.1
Date:	August 11, 2005
State:	Final
Distribution:	Public

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) – Coordinator
Institute of Computer Science,
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polythechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne.
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse, 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe - Montbonnot
Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi – Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom.
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street,
Liverpool, L69 7ZF
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer Science, University of Manchester,
Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed writing parts of this document:

University of Liverpool
École Polythechnique Fédérale de Lausanne (EPFL)

Changes

Version	Date	Author	Changes
0.1	01-03-2005	Ian Blacoe	Initial draft
0.2	01-05-2005	Ian Blacoe	Restructured
0.3	30-05-2005	David Portabella	First part of Section 3 added
0.4	10-06-2005	Ian Blacoe	Section 4 extended
0.5	29-06-05	Ian Blacoe	Revisions and corrections
0.6	07-07-05	Ian Blacoe	Added redrafted sections 4.3 and 4.6
0.7	14-07-05	Ian Blacoe	Added section 1-Introduction Revised sections 4.1, 4.2, 4.4
0.8	20-07-05	Ian Blacoe	Added section 4.8 -Dialogues Amendments from reviewers
0.9	02-08-05	David Portabella	Completed section 3 added
1.0	03-08-05	Ian Blacoe	Amendments and revisions to sections 1, 4.1, 4.2, 4.4, 4.6 and 4.7.
1.1	11-08-05	Ian Blacoe	Addressed reviewer's comments – amended section 1, created section 4, amended section 5, and created section 6.

Executive Summary

Recent developments in the web services research field, specifically regarding the coordination (choreography/orchestration) of services, are encountering many of the issues that have been addressed within the artificial intelligence and multiagent systems research communities, and may well benefit from this existing research. Furthermore, there have already been a number of research efforts aimed at enabling the inter-operation and/or the integration of agents (agent services) and web services.
This deliverable offers guidelines on approaches to such inter-operation and integration, by examination of these research efforts and discussion of both the main points of correspondence and the main differences between the two paradigms, and where they may beneficially inform each other.

Contents

1. Introduction

Autonomous agents and multiagent systems are designed to enable the construction of distributed systems based upon the principles of autonomy, rationality, pro-activity, re-activity, and social ability [Wooldridge95]. Multiagent systems consist of software entities that have independent knowledge and capabilities, and the ability to communicate with and request action from other agents, which they combine to achieve their goals. Communication between agents is intended to permit semantically rich, open, and dynamically constructed conversations – allowing them to communicate at a knowledge level. Long-standing research in the Artificial Intelligence field has produced and contributed to well-founded standards and technologies supporting these features, most notably the suite of FIPA standards that address agent and multiagent system architectures, and inter-agent communication languages and protocols [FIPA].

Web services have evolved from distributed object invocation systems, such as CORBA, and enable software systems to remotely invoke functionalities of independent service implementations using a range of complementary standard languages and protocols. These standards include web service description (WSDL), invocation (SOAP), advertisement and discovery (UDDI), composition (WSCDL) and process management (BPEL4WS). A significant strength of web services lies in the fact that they have evolved in an industrial domain, and address the needs of software developers in the construction of distributed and de-coupled systems.
More recently, a number of different research and standardisation efforts have addressed the issue of enriching the semantic descriptions of web services and the semantics of their communication – most notably OWL-S and WSMO. This has been driven by the recognition that automatic and dynamic composition of web services requires more information about the capabilities, actions, inputs and outputs, etc. than can be expressed with existing standards.

However, these differences do not mean that the two paradigms are incompatible, particularly as the sophistication of web services increases. This deliverable offers guidelines on approaches to such inter-operation and integration, by examination of these research efforts and discussion of both the main points of correspondence and the main differences between the two paradigms, and where they may beneficially inform each other.

The remainder of this document is organized as follows. In section 2, we examine relevant details of agent-based services, such as interaction protocols, service description formalisms, etc. However, basic concepts of agents and multiagent systems are covered in a previous KnowledgeWeb deliverable – D2.4.3. In section 3, we briefly overview web-based services, covering the basic architectural processes of description, discovery and invocation. We then go on to overview the main representation formalisms used in

the web service architectures, including those formalisms supporting the evolution of web services into semantic web services. Section 4 then overviews the main points of similarity and diversity between agent-based services and web-based services. This includes both comparison of underlying conceptualization issues and those points arising from the use of different formalisms and protocols in the two domains. Section 5 represents the main contribution of this deliverable, in which we present a number of issues, approaches and techniques relevant to the task of enabling integration between agent-based services and web services. Finally, section 6 presents a brief set of conclusions that aim to summarise the primary differences that remain between agent-based and web services, and to offer some guidelines on the interoperation and integration of these two service paradigms.

2. Agent Based Services

The following sub-sections and sections 4, 5 and 6 of this deliverable assume a basic familiarity with the underlying concepts of agents and agent systems. However, the characteristics and properties of autonomous agents and multiagent systems have been described in a previous Knowledge Web deliverable (D2.4.3), therefore we do not repeat them here and would refer interested readers to this deliverable.

2.1 Agent Capabilities

The capabilities of autonomous agents can be described in terms of the services that they offer. Under the FIPA [FIPA] reference architecture, agents register themselves with a Directory Facilitator (DF) agent, which acts as a 'yellow pages' for an agent platform. The DF is an optional component of an agent platform, and any one platform may support any number of DFs. Furthermore, DFs can be federated together to provide a combined yellow pages.

In order to make a service available to other agents the provider agent must register the service with an appropriate DF agent. Registering a service does not commit the agent to any future action – it may subsequently refuse to perform an advertised service for any reason. These registrations consist of an id and transport address, along with an optional description of the services offered by the agent. The DF does not place any restrictions on the content of any additional parameters of the description. The DF also enables modification and de-registration of agent descriptions, after verification of access permissions. The services offered by the DF – register, de-register, modify and search – must be requested by agents, using the `fipa-request` interaction protocol. A DF may also, optionally, provide a subscribe service – enabling agents to register an interest in one or more agent descriptions, and then be informed of any change in the DF entries for such descriptions. DF agents providing this service must implement the `fipa-subscribe` interaction protocol.

The DF allows agents to search for the registration of an agent offering a particular service – by attempting to match the description of the service sought with those registered. The DF cannot make any guarantees about the validity of any reply to a service request, as it does not restrict the information that can be registered. Searching of a federated DF is by default a depth-first search across the available DFs, starting with any DFs local to the requesting agent's platform. Federation of DFs is achieved by DFs registering their services with each other – using the reserved service type `fipa-df`. Searches across federated DFs can be constrained using the `max-depth` parameter – i.e. a DF will only forward searches to other DFs registered with it if the value of max-depth is greater than one, and when forwarding the search the DF then decrements max-depth by one. Results for searches can be limited using the `max-results` parameter of the search.

The search function provided by the DF enables searches to be performed for `fipa-agent-description` objects. A search is performed by specifying a template `fipa-agent-description` which is then matched against stored descriptions. Searches for services can be performed by specifying within the template `fipa-agent-description` a template `fipa-service-description` for the service sought, as a value of the `services` parameter.

Directory Facilitator agent descriptions are defined in the `fipa-agent-management` ontology, and are of the following form:

frame – *df-agent-description*
ontology – *fipa-agent-management*

Parameter	Description	Presence	Type	Reserved values
Name	The identifier of the agent.	Optional	agent-identifier	
Services	A list of services supported by the agent.	Optional	Set of service-description	
Protocols	A list of interaction protocols supported by the agent.	Optional	Set of String	
Ontologies	A list of ontologies known by the agent.	Optional	Set of String	fipa-agent-management
Languages	A list of content languages known by the agent.	Optional	Set of String	fipa-sl fipa-sl0 fipa-sl1 fipa-sl2
lease-time	The duration or time at which the lease for this registration will expire.	Optional	date-time	
Scope	Defines the visibility of this df-agent-description. The default global value places no restrictions on the visibility, whereas the local value means the description is not available to a search from a federated DF.	Optional	Set of String	global local

Directory Facilitator service descriptions are also defined in the `fipa-agent-management` ontology, and are of the following form:

frame – *df-service-description*
ontology – *fipa-agent-management*

Parameter	Description	Presence	Type	Reserved values
Name	The name of the service	Optional	String	
Type	The type of the service	Optional	String	fipa-df fipa-ams
Protocols	A list of interaction protocols supported by the service.	Optional	Set of String	
Ontologies	A list of ontologies supported by the service.	Optional	Set of String	fipa-agent-management
Languages	A list of content languages supported by the service.	Optional	Set of String	
Ownership	The agent owning the service.	Optional	String	
Properties	.	Optional	Set of property	

2.2 Interaction Protocols

FIPA specifications [FIPA] define a number of interaction protocols that model typical patterns of interactions, and specify the expected message exchange between agents participating in an interaction. By pre-specifying such protocols it enables agents to engage in meaningful conversations with other agents, simply by following a known protocol. The alternative would be for the agents to be sufficiently aware the meaning of the messages and of their goals, beliefs, etc. that such interactions arise spontaneously from the agents' choices – which requires a great deal of capability and complexity from the agent. The interaction protocol to be used in an agent interaction can be specified in the 'protocol' parameter of an ACL message.

The interaction protocols currently specified by FIPA are:

fipa-request

- Allows one agent to request another to perform some action. The receiver processes the request and decides whether to refuse it – sending a `refuse` to the initiator if so. If the decision is to accept the receiver may send an optional `agree` message – if `notification necessary` is true. Once the request has been agreed the receiver must send either:
  - `failure`: if it has failed to fulfil the request.
  - `inform-done`: if it has fulfilled the request and only wishes to indicate completion.
  - `inform-result`: if it has fulfilled the request and wishes to inform the initiator of the results.

fipa-query

- Allows one agent to request to perform some action on another agent. The initiator requests the participant to perform an inform action, using either `query-if` or `query-ref`. `query-if` is used when an agent wants to determine if a proposition is true or false, and `query-ref` is used when querying for some identified objects. The participant may either refuse the request, sending `refuse`, or agree to the request (maybe sending the optional `agree` message). The participant may then either send `failure` if the action fails, or reply with one of two inform message types once the action has succeeded:
  - `inform-t/f`: is used in response to `query-if`, stating the true or false.
  - `inform-result`: is used in response to `query-ref`, containing a reference to the queried objects.

fipa-request-when

- Allows one agent to request another to perform some action when some precondition evaluates to true. If the participant agrees to the request, it will then wait until the precondition becomes true before acting on the request, returning the appropriate result to the requester.

fipa-contract-net

- Allows one agent to take the role of a manager that wishes to have some actions performed by one or more agents and further wishes to optimise some function characterising the task. The characteristic may be expressed (in some domain specific way) as the price, time to completion, fair task distribution, etc. For any task a number of participants will respond with a `proposal` message, the remainder refusing, and negotiations continue with the proposers. A proposal will specify the preconditions a participant places on the task, such as price, time to completion, etc. Once a pre-specified deadline has passed the initiator evaluates the proposals, sending an `accept-proposal` to those participants selected, and a `refuse-proposal` to the remainder. As proposals are binding on participants, once one is accepted the participant is committed to performing the task. On completion of the task, the participant can send either a simple `inform-done`, or an `inform-result` to provide more explanation. Failure to perform the task will result in the participant sending a `failure` to the initiator.

fipa-iterated-contract-net

- This is an extension of the `fipa-contract-net` protocol that allows for multi-round iterative bidding. Following evaluation of proposals the initiator may issue a modified `call-for-proposals` to (some of) the proposers, in order to elicit improved bids. The iteration continues until either:
  - the initiator refuses all proposals and does not issue a new cfp.
  - all the participants refuse to make proposals.
  - the initiator accepts one or more of the bids.

fipa-brokering

- Designed to support brokerage interactions in a multi-agent architecture. A broker is an agent that offers communication facilitation services to other agents using knowledge about their preferences and requirements. Use of such agents can significantly simplify agent interaction, and also enable the multi-agent system to be adaptable and robust in dynamic situations.
  This initiator sends a message to the broker using the `proxy` communicative act, which contains: an expression denoting the target agents, the communicative act to be forwarded, and a set of proxy conditions. The broker first determines whether to accept or refuse, sending the appropriate message to the initiator. Once the broker has agreed to the action it attempts to find agents matching the description in the proxy message, returning `failure-no-match` if none is found. If such agents are found, the broker then engages in separate sub-protocol interactions with each of them – driven by the communicative act embedded within the proxy message. As the sub-protocols continue, the broker returns the responses to the initiator as `reply-message-sub-protocol` – either failure or success (as defined by the sub-protocol). When all the sub-protocols complete and the broker returns the final reply messages, the brokering interaction protocol ends. Any exceptions in the sub-protocol interactions will result in the broker returning a `failure-brokering` to the initiator.

fipa-recruiting

- Designed to support recruiting interactions in multi-agent systems. Like brokers recruiters facilitate communication between agents, however replies from target agents are sent directly to the initiator agent (or to some other specified receiver) rather than via the broker.

fipa-subscribe

- Allows an agent to request the receiving agent to perform some action on subscription, and subsequently when the referenced object changes. The participant may refuse to perform the subscription, or agree to it – either implicitly or explicitly (if specified). All subsequent successful performances of the action will result in an `inform-result` message to the initiator, containing a referring expression to the subscribed objects. If at any point after the acceptance of the subscription the participant fails to perform the action, it sends a `failure` message to the initiator which then ends the interaction. The initiator may also end the interaction at any point by using the `cancel` meta-protocol.

fipa-propose

- Allows an agent to propose to receiving agents that the initiator will perform the actions described in the `propose` communicative act. The receiving agents may accept or reject the proposal, using an `accept-proposal` or `refuse-proposal` as appropriate. Completion of the protocol with an `accept-proposal` would typically result in the initiator performing the specified actions and then returning a status response.

fipa-english-auction

- Designed to enable one agent to act as an auctioneer in order to determine the market-price of a good - by initially proposing a price below the supposed market-price and then gradually raising the price. The auctioneer initially issues a `call-for-proposals` message, multi-cast to all intended participants, and any participants wishing to bid for the good will reply with a `proposal` message. The auctioneer will then determine which proposals to accept and then responds to all proposers with an `accept-proposal` or `reject-proposal` as appropriate. The auctioneer follows each received proposal by issuing another cfp with an increased price, until no bidder is prepared to pay the price in the cfp, at which point the auction ends. If the last accepted price exceeds the auctioneers privately known reserve price the good is sold to the highest bidder, otherwise the good is not sold. Typically the auctioneer will use the `request` protocol to perform the sale of the good to the highest bidder.

fipa-dutch-auction

- This protocol operates in a similar way to the `fipa-english-auction` protocol, except that the auctioneer initially calls for proposals at a price above the supposed market-value and then gradually reduces the price until a bidder agrees to pay that price. As in the English auction, the auctioneer has a private reserve price below which the good will not be sold.

These interaction protocols are not intended to be an exhaustive list or to cover every desirable interaction type. Rather they are intended to specify interaction patterns, which may be elaborated upon within specific applications. This implies that following any of these protocols does not in itself guarantee interoperability, but that further agreement between agents may be required. Such agreements may be needed for common real-world issues in agent interaction, such as exception handling, lost messages, messages out of sequence, time-outs, etc.

2.3 Coordination Mechanisms

A typical approach in multi-agent systems to the coordination of agents is to use middle agents. Such middle agents are intended to facilitate communication between agents based on the services, information, etc. that they seek, and respecting any preferences, policies, etc. that the prospective participants may have. In an open service environment, agents can be distinguished into three general categories [Sycara02]: service providers, service requestors, and middle agents. Middle agents are those that assist service requestor agents in finding and selecting agent services from service providers. The most commonly utilised types of middle agents are brokers and recommenders. These are intended to store advertisements for services that agents wish to make available to the MAS, and to match these services to requests for services posed to them by agents. The

principal difference between brokers and recommenders is that the broker retains a role in the interaction after the initial contact between consumer and provider is made, whereas the recommender does not. Both advertisements of services and requests for services require specification in a manner that enables flexible and accurate matching.

One perspective on the application of agents in the Semantic Web environment, is that they exist as representatives of real-world entities – performing tasks and achieving goals on their 'owners' behalf. However, such collaborative interface agents [Lashkari94] do not provide the concrete services themselves, but rely on independent service providers, such as other agents or web services. In such a scenario, an agent would translate abstractly defined user goals into concrete semantic service descriptions that it then seeks to match with available services – through semantic-based dynamic discovery and matchmaking mechanisms.

The Semantic Web Fred system [SWF], developed by DERI, demonstrates this interface agent architecture in which the agents represent system users, and utilise available web services to provide both the underlying computational facilities, and the means for all agent to agent interaction. The architecture is built on top of the various WMSO specifications, using the defined web service and service choreography description languages to enable dynamic service discovery and composition by the user agents.

2.4 Matchmaking

Middle agents enable dynamic discovery of appropriate services by performing a *matchmaking* task, by using service request specifications and a stored set of service advertisements to determine which services provide the requested capabilities. It is recognised that such advertisements and requests need to be described in a common language, and that mechanisms are needed to determine structural and semantic matches of these descriptions – which may also require the reconciliation of semantically heterogeneous information in the descriptions.

An early approach to defining a language and matchmaking process that enables dynamic discovery of services that fulfil requirements was LARKS [Sycara02]. The agent capability description language in this system defined services in terms of: input and output variables and their types, constraints upon these input and output parameters, context, and optional fields for textual and concept descriptions. Terms used in the service descriptions could refer to definitions in domain ontologies, both to enable a semantically rich description of the service that could be globally understood, and to enable the matchmaking process to perform automatic inferencing over the utilised concepts to improve the quality of the matching. The matchmaking process itself was defined as a set of filters, each of which performed matching on difference elements of the descriptions:

- ***Context filter*** – the context field of a description is in the form of a set of keywords that describe the semantic domain of the service, and are used in order to reduce the size of the advertisement matching space. This is achieved by considering the semantic similarity between the keywords (based on trigger-pair word distance and subsumption relations), and selecting only those advertisements that have sufficiently similar context descriptions.

- ***Profile filter*** – this considers the entire service description, using a variant of the term frequency – inverse document frequency (TF-IDF) technique to determine the similarity between two service descriptions. However, this filter does not consider the location of terms within the description, or the semantics of the terms themselves.

- ***Similarity filter*** – the similarity between two service descriptions is determined on the basis of the semantic similarity calculated over pairs of input variables, output variables, and their constraints. This semantic similarity is calculated from the words and concepts used in the description fields, and the overall similarity consists of the average of the sum of the similarities among all pairs of inputs, outputs and constraints.

- ***Signature and constraint filters*** – these are intended to take account of the logical constraints within a service description, by determining semantic plug-in matches between descriptions. This is performed using sub-type inference rules, and concept and constraint subsumption testing.

In addition to these filters, the matchmaker computes a weighted associative network of the utilised concepts (using the WordNet [Miller90] ontology to determine associations), in which the concepts are connected by relations and each of these have a weight that indicates the (matchmakers) belief in that relation. These additional relations promote a deeper semantic understanding, and enable higher quality matchmaking.

More recently, this work on service description matchmaking in a multiagent environment has been applied to brokers performing matching of OWL-S web service descriptions [Paolucci04]. Paolucci *et al.* propose a broker that performs both discovery and mediation tasks, between clients (that the authors characterise as agents) and web services described using OWL-S. The broker's overall interaction protocol is first divided into two parts:

- ***Advertisement*** – in which the broker collects and stores service descriptions that are sent to it by service providers. This is the same advertisement process that is used in infrastructure registries, such as UDDI (though UDDI can only make limited use of the information provided by an OWL-S service profile).

- ***Mediation*** – in which the broker first receives a requester's query and attempts to match this request to one or more of the stored advertisements. The broker then re-formulates the requester's query into a service invocation for the selected provider, and then invokes the service. Finally, the broker receives the provider's response to the service invocation, computes the appropriate response, and then sends this response to the service requester.

These two interaction protocols require the broker to perform a number of reasoning tasks:

- To abstract from a query to determine the capabilities that a provider requires in order to answer that query. The authors propose the use of OWL-QL [Fikes03] to overcome the lack of explicit query support in OWL-S, and then use an algorithm that abstracts from variables in the query to their immediate ontological class to form an appropriate OWL-S service profile.
- Compare and match a required capability (or set of capabilities) with a stored set of advertised service capabilities. The broker utilises an existing OWL-S matchmaker [Paolucci03] (one of many such proposals), that exploits the ontological information to infer which advertisements satisfy a capability request.
- Transformation of the requester's query into a service invocation to send to the selected provider. Firstly, the broker must determine if it has all the information required by the selected service, and if not, obtain this information from the service requester. This transformation must generate an OWL-S service profile that reflects the semantic content of the query, and reflects the provider's requirements for the service request.

As with almost all Semantic Web applications, the broker will also be required to resolve (internally or by invoking an appropriate service) any syntactic and semantic heterogeneity between the ontologies used by the service requester and provider in the service descriptions and in their communication.

The authors note that OWL-S provides some constructs that support service brokering: the Profile providing a capability representation, and the Process Model and Grounding providing the basis for interaction between the broker and the provider and requester of the service. However, the authors also note an important shortcoming in the specification of OWL-S regarding it's inflexible treatment of service invocation – leading to a *Broker's Paradox* [Paolucci04]. This paradox occurs because the broker acts as a representative of the service provider but cannot be aware of the provider until the requester's query is received and processed. Therefore, the broker does not have prior access to the provider's Process Model on which its own model (as a representative) must depend. However, the Broker must still publish a Process Model so that the requester may interact with it.
Paolucci *et al.* propose a solution that enables the broker to dynamically modify its Process Model during an interaction, by extending OWL-S with an *exec* operator. The broker can then publish a provider-neutral Process Model, using which the requester

begins the interaction, and then modify this model following selection of an appropriate provider – using the provider's Process Model. The interaction between the requester and broker then continues using this new, provider-specific Process Model. The added *exec* operator enable the broker to execute the new Process Model, that it has returned as the output of another of its processes. Thus, it overcomes the broker's paradox by separating the task of service discovery from that of invocation and interaction.

2.5 FIPA Ontology Recommendations

The FIPA Abstract Agent Architecture and associated recommendations identify three areas in which ontologies have been used in the architecture:

FIPA Agent Management ontology

- Defines a number of frames representing objects and functions within the agent management domain. Definitions of objects include those for `agent-identifier`, `df-agent-description`, `service-description`, `search-constraints`, `ams-agent-description`, `ap-description`, `ap-service` and `property`. Definitions of functions include those for `register`, `de-register`, `modify` and `search`.

FIPA Quality of Service ontology

- This ontology defines a number of frames representing objects and predicates with the quality of service domain. It is intended to be used by agents wishing to communicate about issue regarding the quality of service (QoS) of offered or sought services. Definitions of objects include those for `qos`, `rate-value`, `time-value`, `probability-value`, `time-type`, `comm-channel`, `transport-protocol` and `property`. Definitions of predicates include those for `qos-information`, `time-constraint` and `qos-match`.

Message content ontologies

- The FIPA Agent Communication Language defines a message parameter ontology that is intended to indicate the ontology that should be used to interpret symbols within the content of the message. A severe restriction on the use of ontologies to define message content symbols is that provision is only made for one ontology parameter per message, and there is no means to link the ontology(s) declared in the parameter to those symbols used in the content. Further restrictions are made on the use of ontologies for message content in implementations of the FIPA abstract architecture. For example, in JADE, ontologies must be represented by a set of Java objects, each defining one concept within the ontology. Not only does this not enable use of ontologies in their native form, but there are also underlying semantic discrepancies between Description Logics and such objects that prevent accurate representation of the ontology in this form. Another feature of ontology support in

JADE is that ontology primitives are only provided for concepts, predicates and agent actions – restricting the ability to express knowledge within a domain.

The FIPA Ontology Service Specification outlines a possible extension to the abstract architecture, proposing a specialised Ontology Agent that provides ontology services to a community of agents on an agent platform. The proposed services offered by such an agent include:

- discovery of ontologies,
- maintenance of a set of available ontologies,
- translation of expressions between ontologies and/or content languages,
- responding to queries for relationships between ontologies or between terms,
- facilitating the identification of a shared ontology for communication between two agents.

However, this specification deals only with the communication protocols between agents and the OA – internal implementation and capabilities are left undefined.

In order to enable agents to communicate and manipulate the knowledge encoded in the ontologies a standard knowledge-model and meta-ontology must be adopted. FIPA uses the OKBC frame-based model, and specifies this model in the FIPA-Meta-Ontology specification. This model then serves as an interlingua for knowledge within the agent system, and all knowledge provided to or obtained from the OA must conform to this model.

3. Web Based Services

Web Services are self-contained, modular applications that can be described, published, located, and invoked over the Web. They have evolved from distributed object invocation systems such as CORBA and are independent of service implementation. They rely on several standard languages and protocols, mainly a web service description language (WSDL), advertisement and discovery (UDDI), invocation (SOAP) and process management (BPEL4WS). This work comes from industry efforts to focus mainly on standardization.
On the other hand, academic research, mainly the SWSI[1] and WSMO[2] initiatives, are focusing towards developing new paradigms aiming to solve the limitations of current solutions by providing semantic information to Web Services.

The W3C Working Group proposes a general view (see Figure 3.1) for the Web Services Architecture (on 11th February 2004) [Booth04]. It proposes the steps in the process of engaging a Web Service, independently of how it is done (either manually or automatically), which deals with the topics of service discovery, service description and service choreography explained in the following sections.

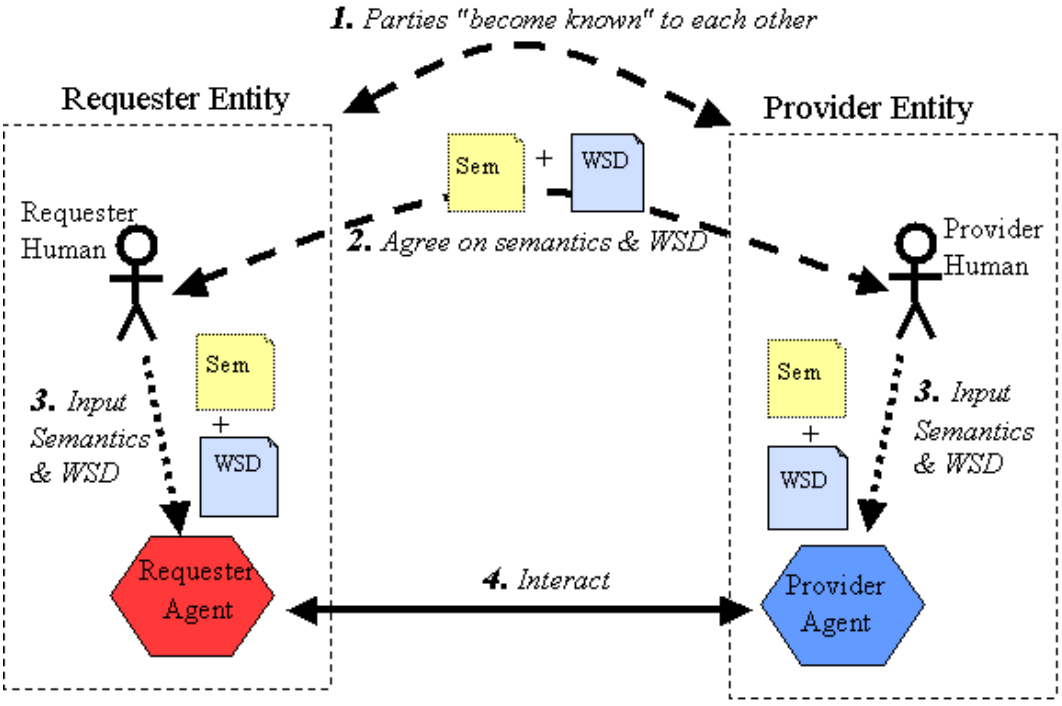

Figure 3.1: Web service architecture.

[1] Semantic Web Services Initiative (SWSI), http://www.swsi.org/
[2] Web Service Modeling Ontology (WSMO), http://www.wsmo.org/

3.1 Service Description

The description of the Web Services is a requirement for the automation of the tasks of discovery, invocation, interoperation, composition and execution monitoring. The first proposal in this direction appeared with the name of Web Service Description Language (WSDL), which specified the location of services, the signatures of the operations that they expose and some rudimentary natural language description.

But this language only defines how to access the service but it does not explain its capabilities, preconditions and effects. This means that the process integration of these traditional Web Services, such as Amazon or Google API, requires the developing of specific client for each of them. This does not scale in a dynamic world where clients need to adapt to changes and new needs.

DAML-S appeared first in 2001 and it was the first attempt to provide an ontology for annotating Web services with richer semantics. It is built upon DAML-OIL, a Web mark-up language that enables the creation of ontologies independently of the domain and is amenable to efficient reasoning procedures based on Description Logic. DAML-S lead to OWL-S in 2003 in the framework of an international effort towards the development of Semantic Web Services – the Semantic Web Services Initiative (SWSI), and it is widely supported by the industry with numerous tools and applications. As it will be explained in the section 3.4.4, this new ontology does not deprecate WSDL, but it is instead complementary.

WSMO is a parallel European initiative for the development of SWS. WSMO defines a Conceptual Framework to describe the different aspects related to Web Services (i.e. Ontologies, Goals, Web Services and Mediators), using a language, WSML, based highly in F-Logic.

The way used to describe SWS will have a clear impact on how much it will improve the task of automating discovery, composition, invocation and interoperation. While OWL-S is focusing more on improving current standards for Web Services, WSMO starts from scratch with a new conceptual framework.

3.2 Service Discovery

In service discovery, the basic idea is that service providers register their WS descriptions in public registries and requester agents query these registries looking for some functionality. Dustdar and Treiber [Dustdar04] classify Web Service Registries architectures into centralized, federated and decentralized. They also analyze their strengths and weaknesses regarding scalability, fault-tolerance, administrative overhead, complexity, and performance.

Independently of this classification and as mentioned previously, the discovery procedure is heavily dependent on the type of SWS description used. We hereby explain how the different initiatives can support this functionality.

In OWL-S the capabilities of the service are described using the Service Profile ontology, which is used to advertise the service. It has means to include non-functional information in an extensible way, although it defines by default some predefined types such as contact information of the provider, service categorization within the UNSPSC hierarchy and quality rating. But the most important feature for the service discovery task is that it allows a functional description of the service expressed in terms of the transformation produced by the service [Martin04]. It describes the inputs and preconditions required by the service, and the outputs and effects produced.
In this way, a requester agent looking for some functionality will itself produce a desired SWS definition using the OWL-S ontology, and query a registry to look for a match among the published services.

WSMO is based on the idea that the provider and the requester agent will not necessarily have the same points of view when publishing a SWS and requesting some functionality. SWS functionality is described in terms of service capabilities, while requester agents define goals that they desire to achieve. The service capability models the functionality of the SWS by means of preconditions, assumptions, post-conditions and effects. The preconditions and assumptions define the valid instances of the inputs and states of the world required by the service, and the post-conditions and effects describe the instances of the outputs and the state of the world that will be reached after executing the service.
In this way, a requester agent looking for some functionality will produce a requirement in terms of goals, and request for service discovery - that with the use of mediators will look for SWS that can fulfil this requirement.

[Verma04] propose a federated architecture of SWS registries using P2P technology in their METEOR-S system. Each registry has an associated hierarchical ontology for a specific domain. In comparison to the OWL-S and WSMO initiatives, they use only WSDL descriptions which are semi-automatically mapped to the specific registry ontology at the time of publication. At the time of discovery, agents can look for relevant services by filling an appropriate template built using ontological concepts. The algorithm for semi-automatically mapping the WSDL description to the registry ontology is based on linguistic similarity between concepts using WordNet [Miller90] and a structural similarity between the XML Schema defined complex data types and the ontological concepts of the registry ontology.
At another level, METEOR-S also defines an ontology to annotate the registries themselves and their interrelations.

3.3 Service Choreography and Orchestration

Once a service with the desired capabilities has been discovered, the client needs to interact with the server in order to invoke it and get the capability. The purpose of the Choreography is to describe this communication. On the other side, the WSMO initiative defines the Orchestration as the description of how a web service interacts with other web services in order to fulfil the requested capability.

The WSMO initiative has presented a first working draft this year [WSMO-CO][3], still in premature state. They propose to use the Abstract State Machine model, formerly known as Evolving Algebras [Gurevich93]. As the choreography describes a kind of "dialogue" between the client and the server, they need to be state-based. The state of the machine is represented by a signature made of an ontology where the concepts and relations can be changed in the next state of the machine based on transition rules. Depending on the mode of the concept, it can be changed by a Web Service instance, the environment, and/or the service. Transition rules take the form of either Guarded Transitions or Update Rules:

- Guarded transitions are `If-Then-Else`, `ForAll` and `Choose`.
- Update Rules are `add`, `delete` and `update`.

Previous work in the area of Orchestration is the BPEL4WS language. BPEL4WS [Thatte03] defines a model and a grammar for describing the behaviour of a business process based on interactions between the process and its partners. The specification is a result of a convergence of ideas from XLANG and WSFL specifications.

BPEL4WS distinguishes the public and private part of business processes. Executable business processes model the private process, i.e. internal behaviour of a participant in a business interaction. The public process is described using the business protocols, that specify the mutually visible message exchange behaviour of each of the parties involved in the protocol, without revealing their internal behaviour. The process descriptions for business protocols are called abstract processes. BPEL4WS defines mechanisms for dealing with business exceptions, processing faults and compensation mechanisms for unexpected cases.

A process definition is made of activities that can basically specify:

1) an action to invoke a web service,
2) the state of waiting and the action of replying a message,
3) the action of copying a value from a source to a destination variable, and
4) the indication of failure and process termination.

Complex structures such as sequence, switch and conditional are also allowed.

[3] We wish to thank James Scicluna for helping us in the understanding about the service choreography and orchestration research area.

Although formal Semantics of BPEL4WS have been provided using abstract state machines (ASM), BPEL4WS lacks form orthogonality as there are some workflow patterns that can be specified by similar constructs either from the XLANG or the WSFL style [Wohed03]. Moreover, it is a static approach not making use of semantics annotations for web services such as OWL-S. WS-CDL[4] is a new initiative of W3C, whose scope corresponds to the public processes of BPEL4WS.

WSMO Orchestration uses the notion of Goals for dynamically discovering and composing semantic services at runtime. Research in this area is still in progress, but is envisioned that would be based on a multi-agent asynchronous version of the ASM model.

Research on Service Composition, which partially overlaps with Service Choreography, has already achieved some results. The OWLS initiative defines the automatic composition of Web services as the problem of providing an "automatic selection, composition, and interoperation of Web services to perform some complex task, given a high-level description of an objective" [Martin04]. Automatic service composition techniques are most inspired by AI planning research. Here we explain two of the most promising ones.

SHOP2 [Wu03] is a Hierarchical Task Network (HTN) planner well-suited for working with the OWL-S Process Model. HTN planning is an AI planning methodology that creates plan by task decomposition. The tasks are decomposed into subtasks until the planning system finds atomic tasks and a global plan is found. The power of SHOP2 is that it can make use of inferencing and reasoning power at the time of the planning. The concept of task decomposition in HTN is very similar to the concept of process decomposition in OWL-S. This makes it possible that some types of OWL-S composite processes composition problem can be easily encoded as a SHOP2 planning problem, and so SHOP2 can be used with SWS descriptions to automatically generate a composition of Web services calls. The major drawback is that it does not support complex constructs like loops.

Situation calculus is a first-order logic language for reasoning about action and change, where the state of the world is expressed in terms of functions and relations relativized to a particular situation. Golog is an extended language of situation calculus for the specification and execution of complex actions in dynamic domains, but that didn't take into account information-gathering actions. An extended version of Golog with knowledge and sensing actions has been developed at the University of Toronto [McIlraith02]. In comparison to SHOP2, this solution supports complex constructs.

[4] Web Services Choreography Description Language, http://www.w3.org/TR/ws-cdl-10/

3.4 Service Representation Formalisms

3.4.1 SOAP

SOAP 1.2[5] is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. The specification has been produced by the XML Protocol Working Group[6], which is part of the Web Services Activity[7].

SOAP provides a distributed processing model that assumes a message originates at an initial sender and is sent to an ultimate receiver via zero or more intermediaries. It supports many message exchange patterns such as one-way messages, request/response interactions, and peer-to-peer conversations.
A SOAP message is composed of a body intended for the exchange of information between the sender and the ultimate receiver, and an optional header that can describes some aspects about the processing of the message such as the priority.

SOAP constitutes a basic building block for Web services and is therefore a common foundation for several Web services standards, in particular WSDL.

3.4.2 WSDL

WSDL (Web Services Description Language) describes Web Services as collections of "communication endpoints", which send and receive messages according to specified protocols, such as HTTP, or SOAP-RPC. The aim of WSDL is to automate the communication between Web services. This is done by distinguishing the abstract Web Service descriptions from the concrete data formats and protocols used to implement the Web Service. In fact, a WSDL binding is a mapping from the abstract description of a Web Service to its specific realization [WSDL].

Web Services are defined in WSDL as sets of endpoints. They are essentially network addresses linked to certain protocols and data format specifications. Each endpoint is associated with an interface that describes the message exchanges (operations) in which the endpoint can take part. In WSDL 1.0 there are four basic kinds of operations:

- a one-way message,
- a (two-way) request-response,
- a (two-way) solicit-response
- a (one-way) notification message.

[5] http://www.w3.org/TR/soap12-part1/
[6] http://www.w3.org/2000/xp/Group/
[7] http://www.w3.org/2002/ws/Activity

This has been improved in WSDL 2.0 by defining 8 basic operations and the possibility of being extensible. Message definitions normally employ XML Schema types and thus support a broad range of type definitions. Interfaces are reusable and can be bound to multiple endpoints. WSDL builds on SOAP by providing a binding for WSDL operations to SOAP messages and for WSDL endpoints to SOAP endpoints.

WSDL assumes a stateless client-server model of synchronous or uncorrelated asynchronous interactions [Thatte03].

3.4.3 UDDI

Universal Description, Discovery and Integration (UDDI)[8] is an OASIS specification for a framework for centralized online registries of Web Services. UDDI Version 3 [UDDI] focuses on the definition of a set of services supporting the description and discovery of:

1) businesses, organizations, and other Web services providers – *businessEntities*
2) the Web services they make available – *businessServices*
3) the technical interfaces which may be used to access those services – *bindingTemplates*

UDDI is based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP.

A *businessEntity* element, can be seen as a White Pages element describing the contact information for a business. It describes a business by a name, a key value, a categorisation, the services offered (*businessService* elements) and the contact information for the business. A *businessService* element describes a service using a name, a key value, a categorisation and multiple *bindingTemplate* elements. This can be considered to be analogous to a Yellow Pages element that categorises a business. A *bindingTemplate* element in turn describes the kind of access the service requires (phone, mailto, http, ftp, fax etc.), the key values and the *tModelInstances*.

tModelInstances are used to describe the protocols, interchange formats that the service comprehends, i.e. the technical information required to access the service. It is also used to describe the namespaces for the classifications used in categorisation. Many of the elements are optional, including most of the ones that would be required for matchmaking or service composition purposes.

Using UDDI, a Web service provider registers its advertisements along with keywords for categorisation. A Web services user retrieves advertisements out of the registry based on keyword search. So far, the UDDI search mechanism relied on predefined

[8] http://www.uddi.org

categorisation through keywords, but more recently specifications to use OWL in UDDI are emerging as a uniform way to express business taxonomies. Such use of OWL is expected to facilitate the service retrieval within the registry.

3.4.4 OWL-S

OWL-S [Martin04] defines a set of ontologies, based on OWL whose aim is to allow users and software agents to discover, invoke, compose and monitor Web services with a high degree of automation. The three main parts of its structure, which will be discussed below, are the *Service Profile*, the *Service Model* and the *Service Grounding*.

The *Service Profile* advertises the service and describes what it does. It includes the following types of information: the organisation that provides the service, the function it computes and the features that specify its characteristics. The functionality is described in terms of its Inputs, Outputs, Preconditions and Effects (commonly referred to as IOPEs). In contrast to WSDL, OWL-S defines the inputs and outputs in terms of OWL classes, which allows for a richer, class-hierarchical semantic foundation underlying the type specifications.

The *Service Model* describes how the service works with respect to its IOPEs and models the service as a set of processes. In the context of OWL-S, a process should not be seen as a program to be executed, but as a specification of how a client and a service may interact.

The *Service Grounding* specifies how to access the service. It can be seen as a mapping from an abstract specification of the elements present in the interaction, to a concrete one. This is done by mapping selected OWL-S ontology elements onto selected elements of a WSDL specification.

In brief, these mappings are based on several natural correspondences between OWL-S and WSDL:

- an OWL-S atomic process corresponds to a WSDL (v1.1) operation;
- the set of inputs and the set of outputs of an OWL-S atomic process each correspond to WSDL's concept of message;
- the types (OWL classes) of the inputs and outputs of an OWL-S atomic process correspond to WSDL's extensible notion of abstract type (and, as such, may be used in WSDL specifications of message parts).

3.4.5 WSMO

WSMO [Roman04] shares with OWL-S the vision that ontologies are essential to support automatic discovery, composition and interoperation of Web services. However, despite sharing a unifying vision, OWL-S and WSMO differ greatly in their details and their approach to achieve these results. Whereas OWL-S explicitly defines a set of ontologies that support reasoning about Web services, WSMO defines a conceptual framework which comprises the main elements needed to describe the different elements of Semantic Web Services, namely, Ontologies, Goals, Web Services and Mediators.

One key difference between OWL-S and WSMO is that WSMO defines the use of Goals, which express what the user wants modelled by means of post-conditions and effects, and they are decoupled from Web Service capabilities, which describes what the service provides. With this approach, the requester (which might be either a human or an agent) can create a goal request independently from any Web Service description.

Another difference is that while OWL-S does not make any distinction between types of Web services, WSMO places a lot of stress on the specification of mediators that are used to solve heterogeneity problems between different entities. There are four different types of mediators:

- ***ooMediators*** – used to solve terminological problems between different ontologies.
- ***ggMediators*** – used to link a goal to another one, defining the refinement of the former by the latter.
- ***wgMediators*** – link Web Service capabilities to goals that the web service (totally or partially) fulfils.
- ***wwMediators*** – link different services, and resolve protocol and process differences.

WSMO Choreography and Orchestration, which is ongoing research based on the Abstract State Machines methodology, is summarized in section 3.3.

4. Comparing agent services and web services

Despite their separate evolution, multiagent systems and web services both aim to provide open, dynamic, service-based architectures. These services are provided by self-contained functional elements can select, utilise and compose the functions exposed by other such elements, in addition to their own capabilities, in order to achieve their individual goals. However, web services focus on ways of 'packaging' and describing the functionality utilising standardised languages and formalisms for description, invocation, coordination, etc. Whereas research in agents focuses on ways to provide the service functionality based on principles of autonomy, social ability, proactive, reactive and goal-directed behaviour [Wooldridge95]. That is, agents provide a top-down, high-level and theoretical approach to a service oriented architecture, while web services offer a bottom-up, pragmatic approach. Due to this diversity in focus significant differences remain between the two architectural conceptualisations, for example:

- Service invocation in web services compared to service request in multiagent systems – i.e. remote procedure call compared to message-based communication. Giving the agent the choice of refusing the request for self-determined reasons (e.g. host system workload levels) – the agent has autonomy, and cannot be 'ordered' to perform the service [Huhns02].

- Agents are capable of extended conversations or dialogues, based on performative messaging that enables the receiver to know the *intent* of the sender, compared to the simpler synchronous, request-response communication used by web services.

- Ability of agents to set self-determined (sub-)goals as compared to web services use of goals that are fixed at design-time or obtained from service users. That is, agents are able to adapt to changing circumstances in their environment and optimise their behaviour according to these circumstances, e.g. using re-planning.

- Agents have awareness about the existence of other elements within their environment, whereas web services know only about themselves [Huhns02].

- "Agents are inherently communicative, whereas web services are passive until invoked" [Huhns02]. That is, agents are able to be both proactive and reactive, whereas web services are only reactive.

- "Agents are cooperative, and by forming teams and coalitions can provide higher level and more comprehensive services" [Huhns02].

- Web services have implicit world/domain knowledge, whereas agents are intended to base their knowledge and actions upon the observable behaviour of other agents. This is now being addressed in web services by proposals for trust and reputation models.

Current attention in web services focusing on coordination between services (automated discovery, choreography/orchestration, agreement/negotiation, etc.), has highlighted the requirements for more expressive, declarative (but static) specification of services – which is being addressed by the use of semantic web languages and approaches. The inclusion of semantic annotation in web services description and communication is leading to a degree of convergence with agent communication languages (ACL), and they are now encountering many of the same problems addressed in agents, e.g.:

- ontology-based, ACL communication over SOAP for web services - introducing performatives (i.e. richer message-typing)
- agent negotiation in WS-Agreement
- web service choreography / orchestration compared to coordination approaches in multiagent systems.

This leads to significant overlap in certain areas of each service domain:

- Communication of service requests and responses – i.e. the messages and their contents (ACL compared to SOAP).
- Service registries – yellow, white and green pages (FIPA DF compared to UDDI).
- Agent service description compared to web service description (FIPA agent-service-description compared to WSDL/OWL-S/WSMO).
- Agent interaction protocols and coordination compared to web service choreography.
- Negotiation between service clients and providers (e.g. in WS-Agreement).

Therefore, we believe that much of existing and on-going research in a number of agent-related areas is relevant to semantic web services:

- Communication based on use of performatives, interaction protocols, and expressive content languages (SL).
- Artificial Intelligence game theory that underlies marketplaces and auctions, negotiation techniques, and argumentation.
- Service requests instead of invocations – enabling services to refuse invocation (and potentially providing an explanation why), rather than simply failing to execute.
- Research on long-running transactions and dialogues (showing that even more expressivity in communication is needed).
- Development of persistence and stateful services may be informed by work on agents, that are inherently stateful.

Also agents can make use of research in semantic web services:

- Service description in OWL-S and WSMO is somewhat richer than in FIPA ACL.
- Work on web service choreography and orchestration overlaps with the body of work on agent coordination.

Therefore, three main scenarios exist for interacting agent services and web services (though the development metaphor may be advised by emerging Agent Orientated Software Engineering):

- Web services provide the more basic level functionality, and agent services provide higher-level functions. Agent services use web services, and act as choreographers and orchestrators to combine web services into added-value functions. Not consistent with expanding view of semantic web services, and does not account for an agent's own capabilities.

- Web service communication becomes equivalent (in many areas) to agent communication, so the distinction between agents and web services disappears. Agent techniques are applied 'behind the scenes' in the provision of web services – which can be seen as being agents in web service wrappers. So, such web services can provide 'normal' functions – as per current implementations; or they can offer 'agent' functions – where agent techniques are used in the service implementation and operation. This may enable one global conceptual model of services – based on semantic web service standards. Therefore, as the communication languages allow more agent-like behaviour in web services, we would see 'agent' web services and 'simple' web services co-existing and using the same communication regime.

- Web services and agent services remain separate, using different communication protocols and retaining different focus – creating a heterogeneous service space. They interoperate through gateways and translation processes – but these require many simplifying assumptions to overcome basic differences. This scenario results in two separate development tracks: one of declarative/functional services; and one of reactive (autonomous) services exhibiting agency.

5. Guidelines

In the following sub-sections we discuss various approaches to the integration of web service and agent services, and explore a number of ways in which research within the agent field may be applied to web services. Sections 5.1 Description, 5.2 Discovery, and 5.3 Invocation focus on the overlaps and miss-matches between web service and agent service paradigms – discussing existing approaches that contribute to the enablement of interoperation them. Sections 5.4 Coordination, 5.5 Agreement, 5.6 Decision-making, and 5.7 Dialogues focus on research from the agent and AI community that has the potential to make contributions to the integration of web service and agent services.

5.1 Description

In comparison to emerging web service frameworks, the FIPA Agent Management ontology *Service* description appears inadequate to describe a service. In particular, the *properties* list within the description lacks structure, and there is no provision to describe service *groundings*. In order to address this shortfall in expressiveness and the interoperation between the two paradigms, in 2003 FIPA proposed an Abstract Service Architecture [Dale03b]. This is intended to provide a consistent framework within which both agent-based and web-based services can be expressed, whilst maintaining the semantics of the service descriptions, and respecting the differences in modes of invocation and inter-operation. The architecture envisions agents being able to act as both providers and consumers of services, and separates the identity of the agent from that of any one of many services it may offer.
The proposal does not specify the role of web services in the architecture, however, it is implied that the web services are only providers of services. In this architecture any coordination of services must be performed by an external entity, i.e. an agent, which does not correspond to current views on web service choreography. Therefore, a useful extension of the architecture would be to view all participants, agents and web services, as both providers and consumers of services. However, we see no need to also separate the identity of the web service and the service it offers. The underlying model of a web service requires that one web identify correspond to one offered service, though there may be multiple groundings of the service. Furthermore, there is already sufficient provision to separate the identity of the web service provider in the non-functional parameters of web service descriptions.

FIPA Abstract Service Architecture

The proposed architecture defines a number of roles that can manipulate (i.e. search, examine, invoke, compose, monitor, etc.) services (see figure 5.1.1):

- ***Provider*** – Any component providing a service or set of services. Each provided service has a service description that is published in one or more registries.

- ***Requestor*** – Any component using a provided service. Requestors discover services by querying a registry and then examining the service descriptions to find services matching their needs.

- ***Registry*** – A component that stores and retrieves service descriptions on request from providers and requestors respectively. A registry acts as a 'yellow pages' for the services registered with it, though services may be registered with many registries and coverage of registries will overlap.

- ***Monitor*** – Any component, or function of a component, that can observe, and report or act on, the status of any current service invocation.

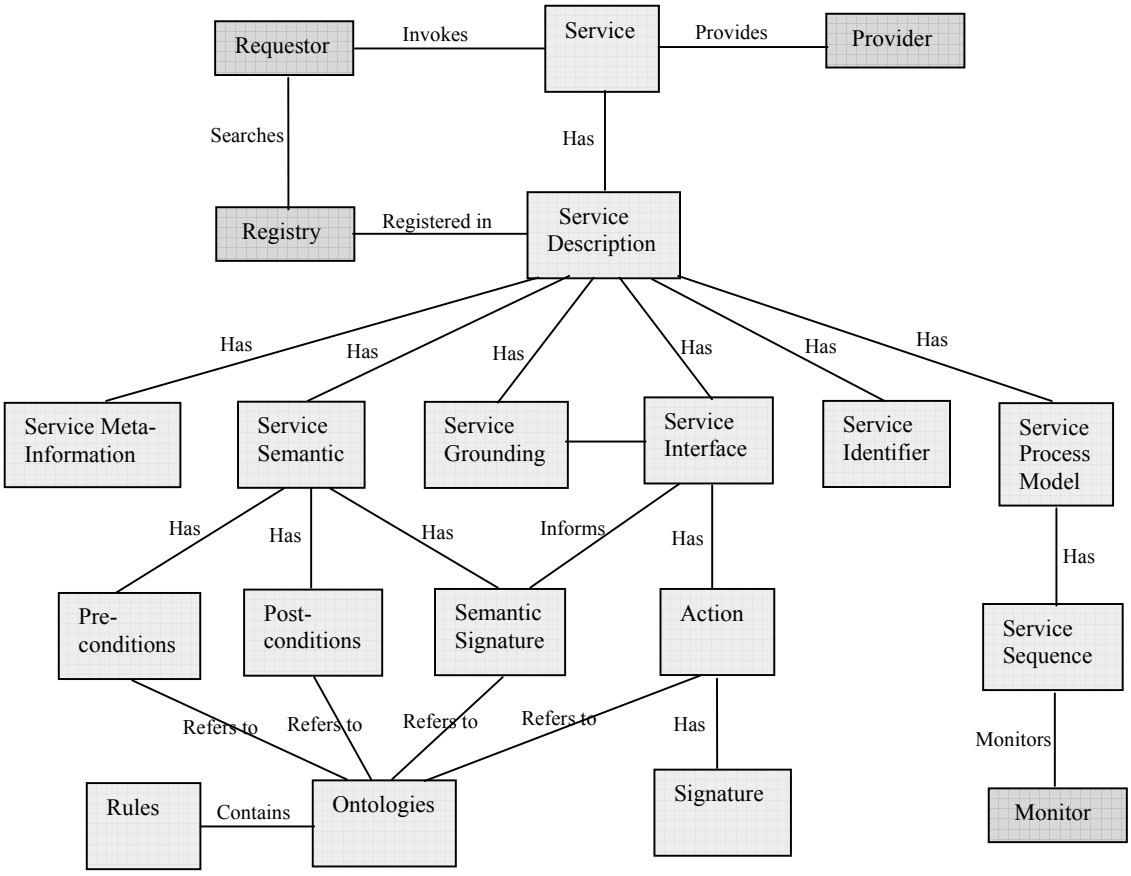

Figure 5.1.1: FIPA Abstract Service Architecture.

- ***Composer and Orchestrator*** – These roles are discussed within the architecture but not fully defined. However, the composer is intended to be a component that is able to inspect service descriptions, and then either automatically or semi-automatically composes the individual services into more complex ones. Such composition would involve using the service semantic section of the service description, matching outputs and post-conditions of one service into (some of) the inputs and pre-conditions of another service. Furthermore, any such matching may well involve resolving the semantic heterogeneity between the ontologies and rules utilised by each of the services in the composition. Due to this complexity the proposal envisions that this task may be performed in a non-dynamic and semi-automatic manner, whereas current web service architectures focus on dynamic service composition. Due to this it separates the role of orchestrator, as the invoker and monitor / controller of composed services, from that of composer. In current web service proposals these two roles are equivalent to the two separate operations of web service choreography and orchestration.

Within this architecture each *Service* has a *Service Description* which consists of the following primary components:

- ***Service Interface*** – described in terms of its component *Actions*, each of which have a distinct *Signature*.

- ***Service Identifier*** – A globally unique identifier for an individual service instance.

- ***Service Meta-information*** – Non-functional parameters describing the service in terms of its intended domain and usage, the providers name and contact details, etc.

- ***Service Grounding*** – Concrete groundings to a range of service technologies and frameworks, e.g. WSDL, SOAP, UDDI, etc. These descriptions provide the means to determine how a service should be used from a technical standpoint.

- ***Service Process Model*** – A description of the internal execution process of the service, including a *Service Sequence* description (or descriptions). It is unclear whether each action within the service has a separate *Service Process Model* or if the process describes the transitions between the actions. It is also unclear if each action is associated with a *Service Sequence*, or even if a process may contain multiple sequences.

- ***Service Semantic*** – This is a machine-manipulable description of the actions of a service, enabling other components in the architecture to perform a more advanced semantic interpretation of its function. This semantic description comprises a *Semantic Signature* and both *Pre-conditions* and *Post-conditions*, each of which may refer to published *Ontologies* and associated *Rules* to constrain the meaning of their

contained terms. The *Actions* of the *Service Interface* may also refer to these ontologies to define its vocabulary.

Zhao *et al.* of Manchester University have proposed an extension to the W3C Web Services Architecture (WSA) based on concepts from multiagent systems research [Zhao04]. The authors note the correspondence between the WSA core structure of distributed agents exchanging messages in the process of providing and consuming services, and the same underlying structure in multiagent systems (MAS). They also note that MAS architectures include the notion of agents collaborating in pursuit of dynamically changing goals to create intelligent and flexible software systems. Therefore, they propose that two agent systems constructs are added to the WSA to make it better suited to dynamic operation:

- *Agent Role Model* – introduces roles played by agents as encapsulations of dynamic behaviour and properties.
- *Agent Communication Model* – used to structure agent communication channels and organise the roles and responsibilities of agents in the communication.

The authors refer to the Web Services Architecture draft of 8th August 2003 [Booth03], that distinguishes five key models within the architecture:

1) **Message Oriented Model** – focused on the messages, message structures and message paths.

2) **Service Oriented Model** – focused on services, actions and supply chains.

3) **Resource Oriented Model** – focused on resources.

4) **Policy Model** – focused on architectural policies and constraints.

5) **Management Model** – focused on the management of web services.

A subsequent draft of the Web Services Architecture (on 11th February 2004) [Booth04] does not include the Management model as one of the five key models, however, the other four remain the same and this change does not substantially affect the proposed extension. The extension itself is based on a number of important criticisms of the WSA:

- the Message Oriented Model does not distinguish between types of agent and roles played by an agent when referring to: *sender*, *recipient*, and *intermediary*. This distinction is important, as designing a recipient as an agent would involve a static (specialisation) relation between *recipient* and *agent*; whereas designing a recipient as a role played by an agent involves an dynamic relation between the two.

- the Service Oriented Model represents the service provision and service request behaviours of an agent as two different relations between one agent entity and one service entity. This is inconsistent with the approach in the Message Oriented Model where the two behaviours are modelled as two separate concepts (see above).

- the *agent* concept plays a role in all of the WSA key models, but never as the core entity – so there is no model that focuses on agents and agent interactions. This makes it difficult to get a coherent view of agents and their roles in the architecture.

- the WSA 'hides' agent behaviour and focuses instead on message structures and routing. Agent behaviours in different contexts are represented as specialisations of *agent,* e.g. *sender*, *recipient*, *provider*, and *consumer*. In some of the models these specialisations are not even given as separate concepts, but are represented as relationships between *agent* and other model elements. This creates serious difficulties in modelling and managing dynamic behaviour change in agents.

- the Message and Service Oriented Models are described only in terms of their concepts and relationships – which gives the impression that everything is linked to everything else. However, there is no structure that describes how the concepts are interlinked into a communication channel.

Based on these criticisms, Zhao *et al.* propose the adoption of two additional key models, based on research within multiagent systems, within the WSA that address these shortcomings:

Agent Role Model

This model is formed from the composition of all the agent roles presented within the WSA, and is illustrated in figure 5.1.2. The diagram shows that an agent plays a particular role by accessing that role through a generic *Agent Role* interface. It also shows that each of the specific roles are implementations of the *Agent Role* interface.

The Agent Role Model (ARM) exhibits a Role Pattern, a well-known design pattern from object-oriented programming, that provides a number of features to the model:

- *Extensibility* – new roles can be added to the model without affecting existing inter-relationships.
- *Consistent viewpoint* – roles can only be accessed through the generic interface.
- *Lower coupling* – there is only ever a one-to-one relationship between an agent and its role.
- *Dynamic binding* – an agent is associated with a role dynamically at runtime, so a change of roles will not affect the agent itself.

These four characteristics enable the design of web service applications that have dynamic behaviour: each behaviour is separated out and defined as a specific agent role, which the agent can then dynamically change between dependent on its context. The authors [Zhao04] see the ARM as the focal point of the WSA, that the other four models converge towards. The intersections between the ARM and the other models can be used as integration points, and this central position of the ARM corresponds to the importance of agents as proactive actors in the WSA. However, such integration would require some modification of the current models with respect to the representation of agent behaviours as *Agent Role* types..

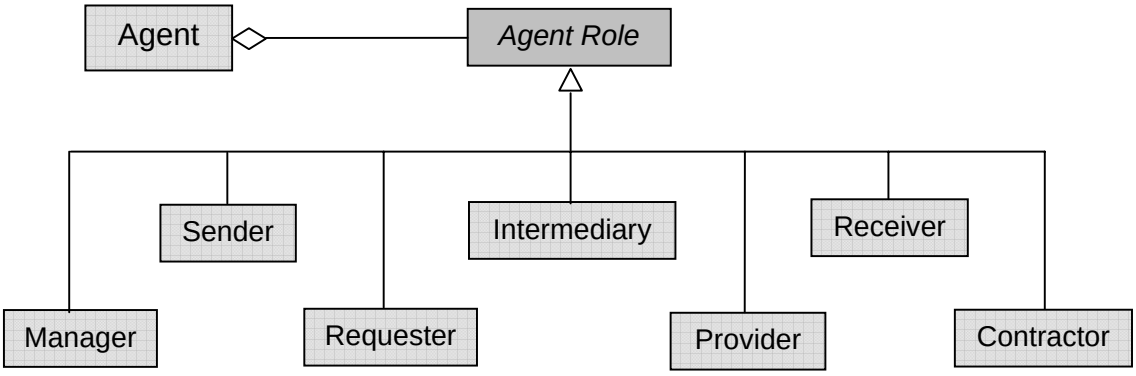

Figure 5.1.2: Agent Role Model in WSA

Agent Communication Model

This model is intended to explicitly capture *Agent Role* communication patterns within the WSA. Examination of the concepts and features relating to communication within the Message and Service Oriented Models, reveal an common underlying Proxy design pattern. This can be seen in the *intermediary* behaviour represented in the Message model. The Proxy design pattern requires clients of a component to communicate with a representative, which provides location transparency – a core part of many distributed and component-based systems (e.g. CORBA and COM). Such location transparency leads to *service transparency* – hiding from service users how the services are discovered, composed, invoked, etc.; and enabling dynamic service supply-chain formation based on user requirements.

Thus, the authors propose an 'agent-centred' view of the Web Services Architecture that focuses on the roles agents take on and the communication patterns of such roles. This is intended to 'facilitate the design of dynamic web service applications' by building upon the benefits of roles: dynamic, flexible, context-sensitive, and responsibility-driven.

Another approach to the integration of agent service descriptions and web service descriptions is translation-based, where (semi-)automatic processes attempt to represent descriptions in one formalism in terms of another description formalism. In the Agentcities Technical Recommendation, the point is made that due to the additional complexity and semantic richness of agent conversations, no generic mapping of agent to web service communication is possible – but that (semi-)automatic mapping is possible in one direction from agents to web services [Dale03a]. In this initial proposal, the translation between FIPA ACL `fipa-service-descriptions` and WSDL web service descriptions is achieved primarily by a manual initialisation of the agents, web services, and descriptions to be used by the gateway.

In the Whitestein Technologies AG implementation of this architecture, the Web Service Integration Gateway (WSIG) [Greenwood04], the translation achieved by a *ServiceDescriptionTranslator* component. The translation uses a mapping schema between the service description formalisms, but the translation cannot be one-to-one due to mutual inconsistencies between the different service descriptions; therefore, those elements with no direct mapping are encoded as optional properties. In each case, WSDL to FIPA, and vice versa, the gateway places the two forms of the service description in its internal UDDI and DF registries. In addition to translating the service descriptions, the WSIG performs bi-directional translation between FIPA ACL message and SOAP messages by use of a (pre-defined) schema that maps fields in one message format into equivalent fields in the other (see section 5.2).

Following from [Dickinson05], we can see such combined description and (to a lesser extent) translation approaches as being indicative of a view that agents and web services are not conceptually distinct (see, for example, [Breese98]. 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. However, we suggest that there is a distinction between web-services and agents, that 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 this behaviour 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).
The observable behaviours of a component that 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.
The Web Service Integration Gateway [Greenwood04] also proposes 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.

5.2 Discovery

Agentcities, a large-scale development and experimental platform for FIPA agent systems, produced a Technical Recommendation document [Dale03a] entitled "Integrating Web Services into Agentcities". This recommendation describes an architecture that enables agents on FIPA platforms to discover and access web services, and conversely enables web services to discover and invoke (or more properly – request invocation) of agent services. The recommendation makes a distinction between the agents providing services and the services themselves – using the term *FIPA Service* to identify a specific service provided by a specific agent. The recommendation also stresses that this architecture is only able to handle one-shot (atomic) web services, and it may not be easily extended to handle more complex interactions.
This reference architecture (see Figure 5.2.1) is built around two *gateways*, which utilise a translation approach between the two service description formalisms, as described in section 5.1.

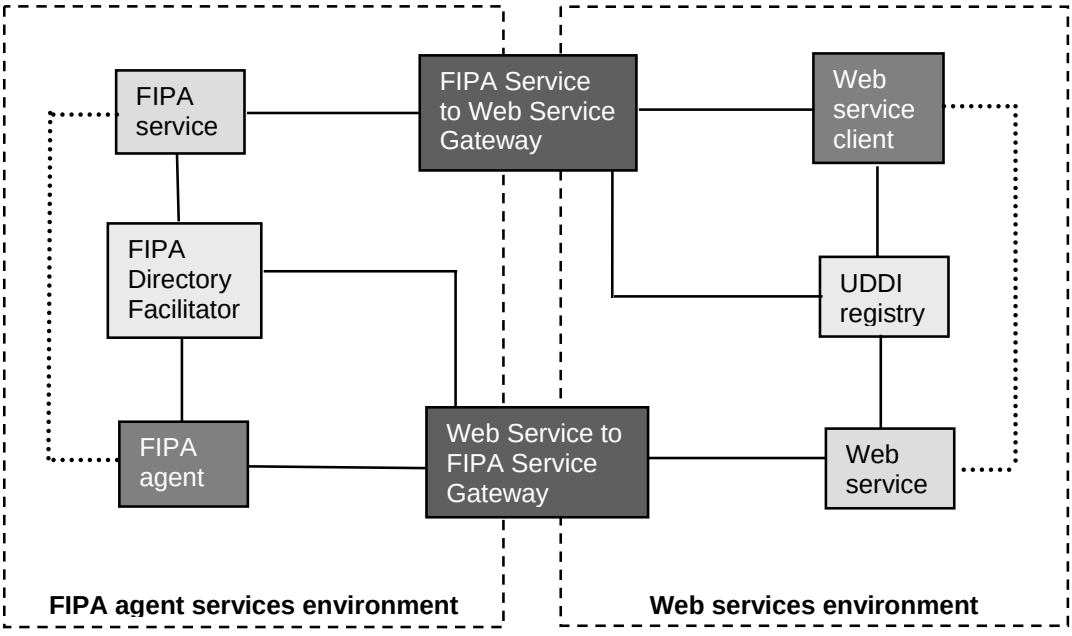

Figure 5.2.1: Adapted reference model for agent services and web services.

Web Service to FIPA Service Gateway (WStFSG)

- Intercepts `REQUEST` messages from agents that are intended for web services, and then translates this request from FIPA ACL into a SOAP message, before performing the SOAP invocation on behalf of the agent.
- Intercepts the return message resulting from a SOAP invocation, then translates the contents into FIPA ACL, before sending the message as an ACL `INFORM` to the originating agent.
- The gateway may optionally perform automatic querying of known UDDI servers, discovering service descriptions, translating them into `df-agent-description` instances, and then registering these with known DF agents – thus enabling agents to discover web services from within their own environment.

FIPA Service to Web Service Gateway (FStWSG)

- Intercepts SOAP messages from web service clients that are intended for FIPA services, and then translates the message into ACL, before sending the message to the appropriate agent as a `REQUEST`.
- Intercepts `INFORM` messages from agents that are intended as invocation responses to web services, then translates the message into a SOAP method return, before sending the SOAP message to the originating web service on behalf of the agent.
- The gateway may optionally perform automatic querying of known DF agents, discovering agent descriptions and translating them into UDDI service registrations, and then registering these with known UDDI servers.

This architecture is separated into two implementation components – the Web Service Agent Gateway; and the SZTAKI service wrapper agents:

- The Web Service Agent Gateway enables existing agent services to be deployed as web services, by translating between FIPA ACL messages and SOAP invocations. The initial step is to process the WSDL file of the web service, and from this to generate a specific ontology and agent deployment code for the web service proxy agent.

- The SZTAKI service wrapper agents enable agents to access existing web services, by generating wrapper/proxy agents for each of the web services required, and translating between ACL and SOAP. Once the wrapper agent is deployed, agents access the web service by sending a `REQUEST` message to this proxy – specifying the web service operation required and the input parameters. This information is specified using the generated ontology, which defines classes for the operations, parameters and outputs of the web service that are used in conversations between the invoking agent and wrapper agent.

The Web Service Integration Gateway (WSIG) [Greenwood04] is a further development of this Agentcities architecture by Whitestein Technologies AG, built as an extension to the JADE multiagent system development platform. It provides a bridge between agent-based services, expressed as FIPA Agent Management ontology *Service* instances and registered with a Directory Facilitator agent, and web-based services, expressed as WSDL descriptions and registered with a UDDI registry. The gateway is constructed as a stand-alone application, and manages the translation between the formalisms internally, rather than having a wrapper agent on the agent platform as a proxy for each of the web services being translated – which the authors see as being a more scalable solution. This is achieved by the gateway generating a web service stub from the WSDL file of each web service registered with it, and then activates this stub to handle any invocations of the web service.
The WSIG maintains both a Directory Facilitator registry of services and a UDDI registry stack, and provides automatic bi-directional translation between *Service* instances and WSDL descriptions. For each service registration received by the gateway the service is translated into the other formalism, and then the two equivalent descriptions are stored in their respective registries.
As part of the translation process the gateway creates grounding end-point stubs for each of the agent-based services exposed as WSDL descriptions. Therefore, when a web service sends a SOAP invocation to the WSIG, it translates this synchronous call into an asynchronous FIPA ACL communication with the agent providing the invoked service. Conversely, when an agent sends a FIPA `REQUEST` to the gateway with the intention of invoking a web service, the WSIG transforms this asynchronous request into a synchronous SOAP invocation on the web service required. From the perspective of the client, either agent or web service client, the requested / invoked service is of the same type as the invoker – i.e. a web service addresses its invocations only as SOAP messages and vice versa. Therefore, the web service client need not be aware of the fact that the invocation is via the WSIG. However, an agent client must be aware of this, as the addressee of the web service invocation `REQUEST` message is the gateway itself rather than the required service – this is specified as part of the ACL message content.

An example of the usage of an agent service / web service gateway can be seen in [Buhler04], in which the gateway enables services provided by agents to be included in BPEL4WS specified workflows as if they were web services. The authors see agents having a symbiotic relationship with web services, where agents can utilise web services as externally defined agent behaviours. In this scenario, because the workflow calls a proxy agent for the web service, rather than calling the web service directly, an opportunity is created for the agent to perform intelligent action related to the service request using other knowledge and capabilities of the agent – instead of simply invoking the service without consideration. This example also serves to highlight some shortcomings in agent/web service interaction – noting that it is insufficient to simply place web service invocation parameters in the content field of ACL messages, and that an ACL content language would enable better encoding of such parameters for use in open agent environments.

Another approach for integrating agent services and web services based on translation between description and communication formalisms is that proposed by Iqbal *et al.* [Iqbal04]. Here the authors envisage an 'Autonomous Distributed Service System' in which:

- agent services can be offered in a web service environment, by use of WSDL descriptions extended with ontology definitions,
- and web services can be offered in an agent environment, by use of proxy agents representing web services. However, the authors observe that this proxy agent approach is "neither scalable nor efficient and non-dynamic as well".

This proposed architecture aims to incorporate not only agent services and web services, but grid services as well – which are themselves based on standard and extended web service languages and formalisms. In addition to using translation (coupled with wrapping and simplifying assumptions) to enable agent services to exist in a web service environment, the architecture incorporates features of agent systems into its unified service environment:

- **Autonomy** – the entities of the system are self-regulating, and manage their own behaviour.

- **Adaptability** – the entities within the system, and the system as whole, are able to adapt their behaviour in response to dynamic events.

This theme in the literature (as characterised in [Dickinson05]) is based on the proposition that agents and web-services can interoperate by either of them initiating communications [Greenwood04], [Good99]. That is, agents can invoke web services, and vice versa. The Web Service Integration Gateway [Greenwood04] (described earlier) 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.
We generally regard web-services as being 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.

5.3 Invocation

Invocation of web services is built upon a Remote Procedure Call (RPC) approach. A client 'orders' a service to be executed, and can then assume that the service will execute as advertised (providing the correct input parameters have been provided). Any failure to execute the service by the provider simply results in no response being returned to the client. In this scenario the onus is on the client to determine that the service has not executed correctly, and then to undertake corrective action – such as re-invoking the same service or finding another appropriate service.
In multiagent systems the same concept of service invocation does not exist. Instead, a client requiring the performance of a service would send a request message to the provider agent asking for the service to be performed. In this scenario the provider agent may chose not to execute the service as requested, e.g. due to its workload the agent is unable to provide the service; or the agent may use a recently cached result of an identical query rather than re-execute the query and pay any associated costs. However, the provider agent does not simply have to fail to execute a service, as would be the case with a web service. The agent can utilise the richer communication style embodied in FIPA ACL to converse with the client agent, in order to seek further inputs for the service, explain why the service cannot be executed, etc.

The primary focus of research into semantic representations for web services has been to use such semantic descriptions to facilitate web service discovery and composition, by both human developers and by other web services. However, a closely related problem is that of how these web services go about communicating with each other. The process models in which these services are used, and their service profiles, are becoming increasingly sophisticated in comparison to mark-up languages for semantic web services.

Willmott *et al.* [Willmott05] propose the adaptation of language structures and protocols from the agent research area to enrich the semantic of web service to web service communication. Agent communication languages have been developed to solve very similar problems to those being encountered in the web service arena, that is, arbitrary, semantically well-founded communication between autonomous systems. Of particular interest are languages such as FIPA ACL, which are based upon a 'speech act' paradigm and have well developed structure. These languages enable the structuring of arbitrary communication by sub-dividing the problem into levels that each have clear language semantics, such as context, protocols, speech acts, content expressions and domain references. They also provide linkages to higher level agent coordination patterns that might be used by web service designers. The authors identify three specific areas where web service communication could benefit from more specific semantics:

- The intent of the component invoking a web service is only partially and indirectly encoded in the name of the method called and its association with any triggered process.

- A single interaction with a service may consist of a lengthy sequence of messages, whose semantics is not captured by the current focus on atomic request-response interactions.

- There may be specific relationships between the arguments of a web service, which can only be inferred from the triggered processes, and these relationships can only be re-defined by creating new functions.

In addition, the application of agent communication approaches would address the significant redundancy likely across applications. This is because different functions in different service descriptions will, at a high level of abstraction, have similar intuitive meanings, e.g. requests, information statements, etc. That is, in current web service approaches the function definitions are necessarily domain and application specific (being directly linked to specific process models), which inhibits reuse.

In order to use FIPA ACL as a web service communication language the authors propose the following architecture components:

- a set of language ontologies, specified in OWL, that encode the grammar of FIPA ACL, FIPA SL and supporting elements.
- XML message representations of messages in these languages, utilising the OWL specifications.
- a logical message structure enabling these messages to be carried over SOAP.
- a WSDL specification of SOAP endpoints that support the message structure.

Using these components an algorithm can then transform each term of an ACL message into an automatically generated instance of the appropriate class of the language ontologies. Instance identifiers are generated automatically and are used to specify the relations between the different class instances. The content field of the ACL message is encoded in the same way, using the FIPA-SL ontology, and links to domain ontology class instances. The XML message is then formed of these class instances and properties, and is sent in the body of a SOAP message to invoke the required web service.
In order to preserve the asynchronous basis of agent communication, the required responses to SOAP invocations are treated simply as acknowledgements of message receipt. The invoked web service carries out the invoked process in a separate thread of control, returning a response at the end of this process – i.e. the web service handles the invocation in its own time, not simply as a call and response.

This approach focuses on specifying the grammatical structure of the FIPA ACL representations, rather than attempting to capture the meaning of the language elements (e.g. *inform* or *request*). This is because the authors see the modelling of the grammar as an essential first step, so that a well-defined message structure is available, and because there is no agreement on how best to capture language semantics. The authors also note that OWL/RDF are not fully adequate to express such grammatical and language parsing constraints, but that this approach enables a immediate link to be made between language

elements and elements in domain ontologies. Finally, issues remain regarding the mixing of elements from different ontologies whilst preserving class relationships – which has been side-stepped in the current model by allowing any grammar element to be replaced by a reference to an instance from a domain ontology (though this does not allow type-checking and communicating services would need to apply pre- and post-processing on messages).

One potential benefit of using such a generic web service communication language as that described above, is that service descriptions (e.g. WSDL/OWL-S groundings) can be de-coupled from specific method calls. By declaring in the service description which content languages, ontologies and protocols are supported, other services can communicate with the described service by generating messages that fit these constraints. However, such communication requires that services have a common interpretation of arbitrary statements in the content language used. In addition, most applications will need to employ a number of different content languages to express different elements of the communication, and any generic communication framework would need to support this use of multiple languages (and their diverse underlying semantic models).

5.4 Cooperation and Coordination

Building the Semantic Web relies on technologies that permit the various components – ontologies, reasoning engines, agents and web services – to work together harmoniously. These interactions need to be managed according to a theory that is understood and agreed upon by all the components (e.g., web services or agents). Coordination is the process of managing the possible interactions between activities and processes, and a mechanism to handle such coordination interactions is known as a coordination regime. A successful coordination regime will prevent negative interactions occurring (e.g., by preventing two processes from simultaneously accessing a non-shareable resource), and wherever possible will facilitate positive interactions (e.g., by ensuring that activities are not needlessly duplicated). Often, coordination is not *required* for the agents to be successful in their tasks, but there may be a global benefit to be gained by adopting such coordination. There has been significant work in the multiagent systems research area on coordination between agents [Wooldridge02], and so we adopt the convention of referring to the processes which need to coordinate as *agents* – though they may equally be any other type of processing entity (such as a web service.)

Effective coordination mechanisms require the sharing of knowledge about activities, resources and their properties. Typically, this sharing is achieved statically, by hard-coding at design time the coordination mechanism in the agents. However, in more open systems, where the processes and resources of the system are not known at design time, this approach is often impossible. In such systems, it may be desirable to allow the relevant processes to communicate their intentions with respect to future activities and resource utilisation, and get them to *reason* about coordination – with the goal of

preventing negative interactions, and facilitating positive interactions. This is a *dynamic* approach to coordination, since the coordination requirement is handled at *run-time*, rather than design time. The communication implied by this solution requires an agreed common vocabulary for coordination, with a precise semantics, that is, an ontological approach to dynamic coordination.

In the following sub-sections we present an ontological approach to coordination among autonomous entities[9]. These sub-sections firstly review the coordination research area, particularly regarding agent coordination research; then present our coordination ontology; and finally provide details of a test-bed implementation of this ontology-based coordination.

5.4.1 Coordination

The coordination problem is that of *managing relationships between the activities of agents* [Malone94]. Coordination is essential if the activities that agents engage in can *interact* in any way. Consider the following examples.

- *You and I both want to leave the room, and so we independently walk towards the door, which can only fit one of us. I graciously permit you to leave first.*

  In this example, our activities need to be coordinated because there is a resource (the door) which we both wish to use, but which can only be used by one person at a time.

- *I intend to submit a grant proposal, but in order to do this, I need your signature.*

  In this case, my activity of sending a grant proposal depends upon your activity of signing it off – I cannot carry out my activity until yours is completed. In other words, my activity *depends* upon yours.

- *I obtain a soft copy of a paper from a Web page. I know that this report will be of interest to you as well. Knowing this, I pro-actively photocopy the report, and give you a copy.*

  In this case, our activities do not strictly need to be coordinated – since the report is freely available on a Web page, you could download and print your own copy. But, by pro-actively printing a copy, I save you time.

Coordination, defined in this way, subsumes the well-known and widely studied concept of *synchronisation* [Ben-Ari90]. Synchronisation is primarily concerned with the restricted case of ensuring that processes do not destructively interact with one another, while the concept of coordination is actually much broader than this. Standard solutions to synchronisation problems involve *hard-wiring* coordination regimes into program code. Thus, for example, a JAVA method may be flagged as `synchronized` by a programmer, indicating that a certain access regime is enforced whenever this method is

[9] This description is substantially based upon work produced by the University of Liverpool within the OntoGrid project – FP6-511513 [OntoGrid]

invoked. However, in large-scale, dynamic, open systems, like the Semantic Web, such hard-wired regimes are too limiting. We really want computational processes to be able to *reason about* the coordination issues in their system, and resolve these issues *autonomously*.

Building agents for semantic web applications that can reason about coordination issues dynamically, means that we must first identify the possible interaction relationships that may exist in these applications. So, the goal here is to derive and formally define the possible interaction relationships that may exist between activities. Prior work on this topic—von Martial [vMartial92] puts forward a high-level typology for coordination relationships. He suggests that relationships between activities could be either *positive* or *negative*. Positive relationships "are all those relationships between two plans from which some benefit can be derived, for one or both of the agents plans, by combining them" [vMartial90]. Such relationships may be *requested* (I *explicitly* ask you for help with my activities) or *non-requested* (it so happens that by working together we can achieve a solution that is better for at least one of us, without making the other any worse off). Von Martial distinguishes three types of non-requested relationships:

- *The action equality relationship*: We both plan to perform an identical action, and by recognizing this, one of us can perform the action alone, and so, save the other effort.

- *The consequence relationship:* The actions in my plan have the side-effect of achieving one of your goals, relieving thus you of the need to explicitly achieve it.

- *The favour relationship:* Some part of my plan has the side effect of contributing to the achievement of one of your goals, perhaps by making it easier (e.g., by achieving a precondition of one of the actions in it).

Another significant body of relevant work is *Partial Global Planning* [Durfee88], in which agents develop and exchange plans of local activity in order to identify possible interactions (positive or negative). The ideas were refined in Decker's subsequent work on *Generalised Partial Global Planning* (GPGP) in the TÆMS test-bed [Decker95]. GPGP makes use of five techniques for coordinating activities:

1) *Updating non-local viewpoints*: Agents have only local views of activities, and so, sharing information can help them achieve broader views. In his TÆMS system, Decker uses three variations of this policy: communicate no local information, communicate all information, or an intermediate level.

2) *Communicate results*: Agents may communicate results in three different ways. A minimal approach is where agents only communicate results that are essential to satisfy obligations. Another approach involves sending all results. A third is to send results to those with an interest in them.

3) *Handling simple redundancy*: Redundancy occurs when efforts are duplicated. This may be deliberate – an agent may get more than one agent to work on a task because it wants to ensure the task gets done. However, in general, redundancies indicate wasted resources, and are therefore to be avoided. The solution adopted in GPGP is as follows. When redundancy is detected, in the form of multiple agents working on identical tasks, one agent is selected at random to carry out the task. The results are then broadcast to other interested agents.

4) *Handling hard coordination relationships*: "Hard" coordination relationships are essentially the "negative" relationships of von Martial. Hard coordination relationships are thus those that threaten to prevent activities being successfully completed. Thus a hard relationship occurs when there is a danger of the agents' actions destructively interfering with one another, or preventing each others actions being carried out. When such relationships are encountered, the activities of agents are rescheduled to resolve the problem.

5) *Handling soft coordination relationships*: "Soft" coordination relationships include the "positive" relationships of von Martial. Thus, these relationships include those that are not "mission critical", but which may improve overall performance. When these are encountered, then rescheduling takes place, but with a high degree of "negotiability": if rescheduling is not found possible, then the system does not worry about it too much.

5.4.2 Coordination ontology

Based on these approaches, we have designed an ontology for coordination, aimed at enabling agents to reason about the relationships of their activities to the activities of other agents. So, the fundamental purpose of the ontology is to answer the following questions:

- what is a *coordinable activity*?
- what *coordination relationships* do such activities have to one another?

In the sub-sections that follow, we give an overview of the ontology: the key concepts, the slots associated with these concepts, the relationships between these concepts, and axioms. From the ontological perspective, there exist a number of ontologies that define notions similar to these ones we define in the coordination ontology, such as process, and activity. One direction to explore is the possible integration of our ontology with OWL-S [Martin04], and its first order logic representation of process theory based on PSL [PSL]. For example, perhaps an *AtomicActivity* is a sub-class of an OWL-S process, and similarly for OWL-S composite processes.

Agents

The concept *Agent* relates to the processing entities in the system, i.e., the things that perform the actions requiring coordination – this role may also be filled by web services. In the coordination ontology, agents have just one slot: *id*, which is a string representation of the unique identifier for the agent (e.g., a URI).

Processes and Activities

Our next concept is *Process*. A process is an activity that changes the state of the environment in some way. It may be terminating or non-terminating, and be carried out by a human or other agent, or be a natural (physical) process. The process concept has two sub-classes: the most important of which is that of a *CoordinableActivity*. A coordinable activity is a process that can be managed in such a way as to be coordinated with other coordinable activities. For example, executing the process of invoking a web service would be a coordinable activity, in the sense that the invocation of such a service can be managed so as to coordinate with other invocations. For example, suppose we have two agents, both of which want to invoke the same web service, with different parameters. Then, in general, the agents could manage their invocations so as not to interfere with one another.

Not all processes of interest to a system are coordinable – hence we have the *NonCoordinableActivity* concept. We intend this concept to capture all those processes whose coordination is not possible by the agents within the system to which a particular knowledge base refers. This will include at least the following two types of process (although we do not represent these as concepts):

- *Natural events*: These are physical processes that will take place irrespective of what any agent in the system does. An extreme example would be the decay of an atom, caused by essentially random quantum events. Clearly, such processes cannot be coordinated with other processes: they will take place (or not take place) irrespective of what the agents in the system do.

- *External processes*: These are processes – either physical world processes or natural processes – which are simply outside the control of the system, in that they cannot be managed by the agents in the system. Notice that such processes may be coordinated by entities *outside* the system: the point is, that for the purposes of the system to which the knowledge base refers, they cannot be coordinated.

Another way of thinking about the distinction between a coordinable and a non-coordinable activity is that there is always an agent (i.e., a software agent within the system) associated with a coordinable activity, whereas there is no such agent associated with a non-coordinable activity.

We think of particular *CoordinableActivity* as being arranged into an and/or tree hierarchy of activities, with *AtomicActivity*s as leaves of the tree. Thus a *Coordinable-Activity* is *composedOf* possibly many other *Activity*s, and may be:

- a *ConjunctiveActivity*: in this case, it is composed of a number of other activities, which must *all* be successfully completed in order for the overall activity to be completed;
- a *DisjunctiveActivity*: it is composed of a number of other activities, of which *at least one* must be successfully completed in order for the overall activity to be completed;
- a *AtomicActivity* – in which case the activity is composed of *no* further activities. (The set of *CoordinableActivity*s of which this activity is composed is empty.)

We can further identify the following sub-classes of *AtomicActivity*:

- *ConcludedCoordinableActivity*: an activity that has taken place in the past, and is now fully concluded.
- *ContinuingCoordinableActivity*: this is an activity that is currently in progress.
- *ScheduledCoordinableActivity*: this is an activity that it is expected *will* take place, in the sense that it is scheduled for execution by some agent.
- *SuspendedCoordinableActivity*: this is an activity that whose status is undetermined.

Let us briefly consider slots and properties of our concepts. A *CoordinableActivity* will have the following slots:

- *actor*: an *Agent*, i.e., the agent that intends to carry out, or has carried out this activity;
- *earliest start date*: either a date or null, with a date indicating the earliest date at which the activity may begin; null indicates that this information is not known;
- *latest start date*: either a date or null, with a date indicating the latest date at which the activity may begin; null indicates that this information is not known;
- *expected duration*: either a natural number, indicating the number of milliseconds the activity is expected to take, or null indicates an unknown duration;
- *latest end date*: either a date or null, with a date indicating the latest date at which the activity may end; null indicates that this information is not known;
- *actual start date*: either a date or null, with a date indicating the date at which the activity actually began; null indicates that this information is not known;
- *actual end date*: either a date or null, with a date indicating the date at which the activity actually ended; null indicates that this information is not known;
- *final status*: an enumeration type, either succeeded, failed, or null.

There are a number of axioms that may be introduced at this point. With respect to *Conjunctive* and *DisjunctiveActivity*s, we have the following:

- a *ConjunctiveActivity* has successfully terminated if all its components have successfully terminated;

- a *DisjunctiveActvity* has successfully terminated if at least one of its components has successfully terminated.

With respect to the relationship between scheduled activities and their successful completion, we have the following:

- if an activity is scheduled, then it should have a null actual start date and actual end date.
- if an activity is concluded, then the final status must be non-null;
- if an activity started before its earliest start date, then it has failed;
- if an activity started after its latest start date, then it has failed.

Resources

Next, we have the *Resource* concept. The idea of this concept, as we discussed in the introduction, is that a resource is something that may be required to expedite an activity. Thus, we have a one-to-many relationship between *AtomicActivity*s and *Resource*s. Note that we regard this set as being fixed, for any given activity.

The *Resource* concept has the following slots:

- *viable*: a Boolean value, indicating whether the resource is still in a state to be used; a value of false here would indicate that the resource could not be used by any activity (even if these activities *Require* it). Another simple way to think about *viable* is that it indicates whether a resource is "broken" or "working".
- *consumable*: a Boolean value, which indicates whether the use of the resource will reduce subsequent availability of the resource in some way; more precisely, whether the repeated use of the resource in activities would make the resource non-viable.
- *shareable*: a Boolean value, indicating whether a resource may be used by more than one agent at any given time.
- *cloneable*: a Boolean value, indicating whether or not the resource is cloneable (= true), or unique and not-cloneable (= false). An example of a cloneable resource would be a dataset or a digital document. An example of a unique resource would be a physical artefact produced as the output of a particular experiment, or a human being.
- *owner*: either an *Agent* (in which case this is the agent that owns the resource), or null (in which case the semantics are that the resource may be used by any agent at no cost). If a resource is owned by an agent, and another agent wishes to use this resource, then it may be necessary to enter into negotiation over the exploitation of the resource.

Interdependencies between Activities

We now turn to the interrelationships that exist between activities. Our first concept is that of an *Interdependency*. The interdependency concept has the following slots:

- *source* and *target*: both slots are *Activities*, the idea being that these are the two activities which are interdependent.

- *isHard*: a Boolean value, which indicates whether the relation is "soft" (= false) or "hard" (= true), with the following semantics:
    - a *hard* relation is one which will materially affect the success or otherwise of the activities;
    - a *soft* relation is one which *may* affect the activities, positively or negatively, but will not affect whether they are successful or not.

Sub-classes of *CoordinationRelation* are:

- *NegativeCoordination*: an interaction which, if it occurs, will lead to a reduction in the quality of the solution or the utility of the participants;
- *PositiveCoordination*: an interaction which, if it occurs, will lead to an increase in the utility of the participants or the quality of the solution.

We have a further sub-class of *NegativeCoordination*:

- *FatalCoordination* is a hard coordination relationship which, if it occurs, will inevitably lead to the failure of one or more of the component activities. Note that instances of *FatalCoordination* relationships are always *hard*.

As sub-classes of *FatalCoordination*, we have:

- *MutuallyExclude*: an instance of this relationship will exist between two *Atomic-Activity*s iff:
    - they both *Require* some resource *r*,
    - the actual or scheduled usage of *r* by both activities overlaps;
    - *r* is non-shareable.

  The idea is thus that these two activities will be mutually exclusive, in the sense that they cannot possibly both succeed, as they require access to a resource that cannot be shared.

- *ResourceContention*: an instance of this relationship will exist between two *Atomic-Activity*s iff:

- they both *Require* some resource *r*;
- resource *r* is consumable.

The idea here is thus that one of the activities (the earlier one) could prevent the successful completion of the other activity, by depleting it or rendering it unviable. We do not require that *ResourceContention* relationships are *hard*, although, of course, they could be.

- *Disables*: one activity will disable another if the occurrence of it will definitively prevent the occurrence of the other.

Sub-classes of *PositiveCoordination* are:

- *ConditionallyFeeds*: in such a coordination, the occurrence of activity *A*1 will subsequently make possible the occurrence of activity *A*2, but it is nevertheless possible that *A*2 could not occur (i.e., the occurrence of *A*1 is a sufficient but not necessary event for the occurrence of *A*2);

- *Enables*: the occurrence of activity *A*1 is both necessary and sufficient for the occurrence of *A*2;

- *IsSubsumedBy*: activity *A*1 is subsumed by activity *A*2 if *A*2 contains all the activities of *A*1;

- *Subsumes*: the inverse of *IsSubsumedBy*;

- *Favours*: an activity *A*1 favours another activity *A*2 if its prior occurrence will subsequently improve the overall quality of *A*2. We include this as a "catch all". This is a *soft* relationship.

Operational Relationships

In order to *resolve* a coordination relationship between two activities, we may have to appeal to the *operational relationships* that exist between the agents that will carry them out. Intuitively, operational relationships exist between agents that carry out activities, and by understanding these relationships, it can help to resolve the coordination relationships. The main concept here is *OperationalRelationship*. This concept has two slots, both of which are *Agents*: *source* and *target*.

Sub-classes of *OperationalRelationship* include:

- *LegalAuthority*: this sub-class indicates that *source* has legal authority over *target* (of course, this begs the question of what "legal authority" means in the context of semantic web services and processes, but this is outside the scope of our current work, and is left as a placeholder for the future);

- *ContractualAuthority*: this indicates that *source* has contractual authority over *target* (i.e., that both agents "belong" to the same organisation, and that in the context of this organisation, *source* should take precedence over *target*);

- *ProducerConsumer*: this indicates that *source* is the *owner* of a *Resource* that is to be used by *target*;

- *ConsumerProducer*: the inverse of *ProducerConsumer*;

- *Peer*: two agents that work as peers, i.e., that neither has any authority over the other.

5.4.3 Implementation

We have implemented our prototype with the plug-in JessTab 1.1 [Eriksson05] in Protégé 3.0 [Protégé]. JessTab is a plug-in integrating the inference engine Jess (version 6.1 [Friedman-Hill05] in our case) with Protégé, so that Jess can carry out inferences on the knowledge base in Protégé. More precisely, JessTab enables Jess to work with a Protégé knowledge base, i.e., Jess can:

a) access the ontology and the instances represented in Protégé,
b) directly manipulate these ontology and instances,
c) infer new facts deduced from these ontology and instances,
d) perform all the other programming tasks permitted by Jess, such as calculating or launching Java operations.

In our prototype, we use these capabilities of JessTab in the following way. We first design an ontology for our agents in OWL [OWL] using Protégé. For this proof of concept we restrict our attention to few concepts and types of coordination and we do not implement the whole ontology described in Section 3. In our implementation, concepts in the ontology are translated into Jess facts, whilst the coordination strategy is translated into a set of Jess rules. In our ontology , we create the class *Agent*, with subclasses *Provider*, *Requester* and *Registry*, as well as the classes required by these three types of *Agent*s, i.e., *RegistryMemory*, *Intention* and *Resource*, which are now outlined.

- A *RegistryMemory* is related to an instance of *Registry* by the property "hasMemory". Every instance of *RegistryMemory* represents either a *Requester* and

one of its *Intention*s, or a *Provider* with one of its capabilities and associated *Resource*s.

- The second element, *Intention*, is related to instances of *Agent* to describe one of the activities planned by a particular *Agent*. Besides an *Agent*, an *Intention* may also be linked to a *RegistryMemory* to enable a *Registry* to memorize an *Agent*'s *Intention*.

- The third class is *Resource*, which contains the name of a resource (we assume this name is a unique identifier for this resource) and the flag "isShareable".

After the creation of the classes of this ontology, we populate the ontology with instances. In our example, we instantiate one resource *Provider*, two resource *Requesters* and one *Registry*. These first two steps related to ontology building do not require JessTab, but only Protégé. Finally, we add Jess rules to "animate" our instances. These rules implement the choreography between the instanciated *Agent*s. These three stages are detailed in the following subsections.

Classes in the ontology

As noted, we basically deal with three different classes, namely *Provider, Requester* and *Registry*. Each time a *Provider* or a *Requester* registers to a *Registry*, this *Registry* records the information sent by this *Provider/Requester* by creating a *RegistryMemory*, which means that a *RegistryMemory* is very similar to a *Provider/Requester* and thus to an *Agent* (of course, only from an ontological viewpoint). To record a *RegistryMemory*, *Registry* has an object property called "has-Memory" listing instances of *RegistryMemory* used by this *Registry*. "hasMemory" is the only property of interest in a semantic *Registry*, even if a *Registry* inherits all the properties of an *Agent*, namely "hasName", "hasCapabilities", "hasGoals", "hasIntention" and "hasResource".

It is worth noting the difference we make between "hasGoals" and "hasIntention": in a similar way to the BDI (Belief, Desire and Intention) architecture [AOSG], an agent desires to achieve its multiple goals, and as a result, this agent selects and adopts the appropriate intention (which is a plan of actions in the BDI architecture). In our ontology, we translate this in the following way: an *Agent* has a property "hasGoals" pointing to several instances of *Intention*, and one property "hasIntention" pointing to one of these instances of *Intention*. This latter *Intention* represents what current action this *Agent* currently tries to achieve. Indeed, this is the main difference between a *RegistryMemory* and an *Agent*: only an *Agent* has a property "hasIntention", while both *RegistryMemory* and *Agent* have a property "hasGoals". A semantic *Registry* uses the property "hasGoals" to register in one of its *RegistryMemory*s what it knows about an *Agent*'s planned activities. In other words, there is one *RegistryMemory* per *Agent* (normally, this *Agent* should be a *Requester*), and as many properties "hasGoals" per *RegistryMemory* as the *Agent* communicates to the *Registry*.

Finally, all *Agent*s may have object properties "hasResource" of type *Resource*, and "hasCapabilities" of type *string*. As previously stated, there is one *RegistryMemory* per *Agent* (this *Agent* should now be a *Provider*), and as many properties "hasResource" and "hasCapabilities" per *RegistryMemory* as the *Agent* communicates to the *Registry*.
A *Resource* describes a resource, such as a CPU, a hard drive, a printer, etc. and data-type properties "hasCapabilities" of type string, e.g., saving-information, calculating, printing, etc. In addition, *RegistryMemory* has two datatype properties "agent-Name" and "agentRef" to respectively record the name (which is the string an *Agent* records in its datatype property "hasName") and the address of the agent.

Example instances for our ontology

As a case study, we have implemented a system with four agents, in which "Requester 1" and "Requester 2" look for the non-shareable resource called "Printer", while "Provider Printer" manages this resource. Requester 1 has "intention1", which describes the fact that this agent has scheduled to use Printer from the date 5 and for a duration of 10 time units. Figure 4.4.1 displays this Requester 1's intention to use Printer, as well as Requester 2's.

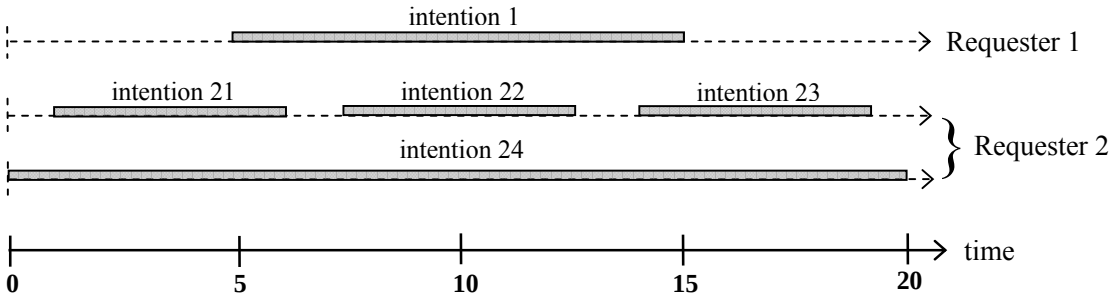

Figure 4.4.1: Gantt chart of Requester 1's and Requester 2's schedules

In this figure, we can also see that the property "hasGoals" of Requester 2 points to four intentions, namely intention21, 22, 23 and 24, and that intention24 is at the same time as intention21, 22 and 23.We assume that Agent2 has not seen this overlapping in its own schedule, and the registry should thus detect this clash among Agent2's goals. The registry should also detect the clashes between Agent2's goals and Agent1's.

Orchestration implemented in the prototype

The Jess program roughly adopts the following four steps. By "roughly", we mean that these steps are interlaced in practice, while we are now presenting them sequentially:

- *Step 1*: The Protégé classes, instances and templates are translated into Jess. This is performed by the JessTab command (mapclass :THING) that translates the root node of the Jess ontology, as well as all its children up to the instances, into Jess. Note that this command is an addition of JessTab to Jess.

- *Step 2*: Every agent sends (i.e., asserts) a registration message to every registry. This message contains the description of this agent, one of its capabilities, one of its goals and one of its resources. The agent sends several messages to register all its capabilities, goals and resources, and can write "none" if it does not have one of these features.

- *Step 3*: Every registry receives these messages and saves their content by creating *RegistryMemory*s. One *RegistryMemory* is created for each registering *Agent*, and this *RegistryMemory* is almost a copy of *Agent* reconstructed from the registration messages. In practice, the JessTab commands make-instance and slot-set add instances and slots in the Protégé base, and then map instance converts this information into Jess, so as the consistency is maintained between Jess and Protégé knowledge bases.

- *Step 4*: Every semantic *Registry* detects clashes between non-shareable resources. Intention21, 22, 23 and 24 in Figure 2 represent the four possible types of clash with intention1. For example, intention1/intention21 is a conflict in which the beginning of the time interval represented in intention1 overlaps the end of intention21. This conflict is characterized by the following conjunction: (i) the starting date requested by Requester 1 is later (greater) than the starting date requested by Requester 2, (ii) the starting date requested by Requester 1 is earlier (lower) than the ending date requested by Requester 2, (iii) the ending date requested by Requester 1 is later (greater) than the ending date requested by Requester 2. Notice that (i) and (ii) mean that Requester 1's starting date is in the time interval requested by Requester 2, while (ii) and (iii) mean that Requester 2's ending date is in the time interval requested by Requester 1. A separate Jess rule is programmed to detect each of these four possible clashes. We call ***1*** the rule detecting the conflict intention1/intention21, ***2*** for intention1/intention22, etc.

Results

The execution trace in JessTab is displayed in Figure 3, in which we can see seven conflicts, each one beginning with the name of the rule that detected it followed by some explanations. For example, the first conflict was detected by the rule ***2***, and is thus of the type intention1/intention22, but this first conflict is not between Requester 1's intention1 and Requester 2's intention22. In fact, this conflict is due to the fact that Requester 2 wants to use Printer both over [1;6] and [0;20], and thus, the former time interval is included in the latter while Printer is non-shareable.

Conversely to this, the second clash is between two different requesters. In other words, inter-agent as well as intra-agent conflicts are detected. We have checked that it is possible to add more instances of resources, providers, requesters and registries and JessTab still detects the conflicts.

5.5 Agreement

The dynamic formation of business relationships that is implied in the automated use of web services for the outsourcing of business processes depends on three main factors [Dan04]:

- *Interoperability* – access to services must be based on open standards for service-oriented architectures (SOA), such as those in web services and grid services.
- *Service-level agreement* – to ensure quality of service (upon which the decision to outsource was made) the client and provider should jointly define a service-level agreement as a part of the service contract.
- *Automated management* – the whole life-cycle of a business relationship should be provided with automated support, from creation of the service offering, through definition of the SLA (including possible negotiation), to monitoring of the service to ensure compliance with the SLA. In order to enable this, the SLA (and any other service-related agreements) must be specified in machine-processable forms.

WS-Agreement [Andrieux04] specifies an XML-based language for creating contracts, agreements and guarantees from offers between a service provider and a client. An agreement may involve multiple services and includes fields for the parties, references to prior agreements, service definitions and guarantee terms. Here the service definition is part of the terms of the agreement and is established prior to the agreement creation. The guarantee terms specify the service levels that the parties are agreeing to and may be used to monitor and enforce the agreement. A service provider publishes an agreement template describing the service and its guarantees. Negotiation then involves a service consumer retrieving the template of agreement for a particular service from the provider and filling in the appropriate fields. The filled template is then sent as an offer to the provider. The provider decides whether to accept the offer, depending on its resources.

WS-Agreement specifies an ability by the parties to negotiate about the service-level parameters in any agreement document, however it does not provide sufficient means to achieve this. One approach is to extend WS-Agreement by utilising techniques from multiagent systems, where negotiation is a well established research area [Paroubally05b]. In this approach, the authors note that current web services work fails to benefit from the agent communities research in negotiation for coalition formation between parties with heterogeneous information needs. Furthermore, given the view that multi-agent systems are groups of agents that interact through cooperation, coordination and negotiation to satisfy their individual or common goals, web service and agent

technology can complement each other for efficient provisioning and management of services. Since automated negotiation can effectively help in resolving conflicts typically over resource allocations and in setting up agreements such as for service provision between autonomous entities, this approach develops such negotiation mechanisms between web services. These mechanisms address the current limitation of semantic web services to support dynamic negotiation strategies for task and resource allocation. Their aim is to successfully deploy negotiation and interaction mechanisms between web services in the same way that autonomous agents negotiate in a collaborative or competitive distributed system.

There are a number of significant shortcomings [Paurobally05a] in WS-Agreement:

- *Limited Message Types* – The first significant weakness lies in the fact that WS-Agreement messages are limited to two types – *offer* and *agree* – and constrained according to a template a service provider publishes. The WS-Agreement specification is only used at the last stage in a transaction where the parties close their interaction with a contract specified as a WS-Agreement. The *offer* and *agree* templates are not sufficient or appropriate for modelling negotiation, auctions or the contract net protocol [Smith81].

- *No Interaction Protocols* – Even with a more varied set of messages (speech-acts), WS-Agreement still suffers from the lack of an interaction protocol specified between parties. That is, even if we increase the WS-Agreement schema with various speech-acts, there is no concept of how to sequence messages through interaction protocols to form a valid conversation. This is the second significant weakness. There is only a two step conversation, an *offer* followed by an *agree*. Without an adequate set of speech-acts and specification of how to construct interaction protocols, the usefulness of a WS-Agreement exchange is limited to cases such as buying from catalogues, with take-it or leave-it offers from the seller or buyer.

- *Lack of Semantics* – On the whole, WS-Agreement is a specification with vague and unclear semantics. The WS-Agreement specification only defines a higher-level template for agreements and offers. There is the need of a language to express the elements in the service description terms and guarantee Terms. Thus there is no indication of how to access or provision a service from an agreement.

In the following sub-sections we show how we adapt the Rubinstein's alternating offers protocol for web service negotiation. Rubinstein's [Rubinstein82] protocol of alternating offers is a well-known strategic negotiation model. In this model two parties *A1* and *A2* participate in the negotiation process and makes offers and counteroffers. Rubinstein first proposed this model of alternating offers to describe how two players could share or partition a pie of size 1. He assumed that the players cannot opt out of the negotiation. He considered situations of fixed discount costs over time and fixed discounting factors. He later extended this protocol to deal with incomplete information scenarios. In our case

also, we assume the web services have incomplete information about their opponents' preferences. The features of negotiation normally include a communication language, an interaction protocol and the decision process used to determine responses, concessions and criteria for agreement [Singh05]. In the following sections, we adapt each of these three components to the web services domain for the case of the Rubinstein's alternating offers protocol.

Alternating Offers Negotiation Language

The actions in this model are: *offer, counter-offer, agree, reject*. These actions are typically called speech-acts [Searle69] and in our XML definition we type them as `wssa` (for Web Service Speech Act). Each of these actions have as parameters the sender, receiver, service description, the process to be executed and the agreement terms such as deadline of carrying out the agreement, penalty of not meeting the agreement or conditions for de-committing to the agreement. Below, we show the XML definition of a counter-offer.

```
<wssa:counter-offer>
  <wssa:Name> "xsd:NCName" </wssa:Name>
  <wssa:Parties>
    <wssa:Sender> xs:anyType </wssa:Sender>
    <wssa:Receiver> xs: anyType </wssa:Receiver>
  </wssa:Parties>
  <wssa:service>
    <wssa:Name=" xs:NCName" wssa:ServiceName=" xs:NCName">
    <xsd:any> . . . </xsd:any>
  </wssa:service>
  <wssa:Agreement wssa:Name=" xs:NCName">
    <wssa:Deadline> xs: Time </wssa:Deadline>
    <wssa:Penalty> xsd:integer </wssa:Penalty>
    <wssa:Reward> xsd:integer </wssa:Reward>
    <wssa:Decommitment> xsd:any </wssa:Decommitment>
  </wssa:Agreement>
</wssa:counter-offer>
```

The other actions, *offer, agree, reject* can be similarly defined in XML. In the next section, we show the interaction protocol for sequencing these actions. For example, an offer can be followed by a counter-offer but not by another offer, or an agree cannot follow a rejection.

Alternating Offers Negotiation Protocol

The parties can act in the negotiation only at discrete time points in the set $\boldsymbol{T} = \{0, 1, 2, \ldots\}$. At each instant $\boldsymbol{t}$ ($\boldsymbol{t} \neq 0$) in the negotiation, if the negotiation has not yet terminated, the agent, whose turn it is to respond, may send a counter-offer, agree, or reject. If an offer or counter-offer made by the agent *A1* at time instant $\boldsymbol{t}$ is accepted by *A2* then the negotiation terminates. If *A2* rejects the offer/counter-offer the process ends in a conflict. We express the bilateral protocol in XML so that it is in a language that web services can understand.
A protocol in XML consists of a `name`, a set of `state`s and `transition`s.
First we define XML templates for states and transitions. As in finite state automata, transitions, here through exchanging speech-acts or messages, lead to a change in states – from source state to target state. For example, in the source state *offered*, sending an *agree* message triggers the target state *agreed*. We define the type `state` in XML as having a name, a boolean attribute (whether the state holds or not), and optionally includes the service that triggered the state, the recipients and any action needed.

```
<xsd:element name = "State"/>
  <xsd:complexType>
    <xsd:attribute name="xs:Name" type="xs:boolean"/>
    <xsd:sequence>
      <xsd:element name="Initiator" type="wsag:Initiator"/>
      <xsd:element name="Respondent" type="wsag:Respondent"/>
      <xsd:element name="Process" type="wsdl:Operation"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The type `Transition` is defined in XML as having attributes name, source and target states, sender (perpetrator) and recipients of the transition. A transition that initialises a conversation may not have a source state and therefore the source state is an optional field. In contrast, the target state is a compulsory field. A `Transition` also has an optional `Process` field to encode relevant information such as offering to carry out an action $\boldsymbol{\alpha}$, where the transition is the speech-act name and $\boldsymbol{\alpha}$ is the process attribute. Using the definitions of states and transitions, we show below a partial specification in XML of the salient parts of the Rubinstein's bilateral protocol. It can be seen that we are now able to specify which service sent the last message and whose turn it is now to respond and what are the valid responses.

```
<Protocol name="BilateralProtocol"
  <States>
    <State Name="offered(X)" >
    <!-- X sent an offer-->
      <Initiator> X </Initiator>
      <Respondents> Y </Respondents>
      <Process> α </Process>
    </State>
    <State Name="agreed(Y)" >
    <!-- Y sent an agree-->
      <Initiator> Y </Initiator>
      <Respondents> X </Respondents>
      <Process> α </Process>
    </State>
  </States>
  <Transitions>
    <Transition Name = Y.counter-offer>
    <!-- Y sends a counter-offer, for Y to do α-->
      <SourceInteraction href="offered(X)"/>
      <DestinationInteraction href="offered(Y)"/>
      <Trigger trigger-process "Y.offer">
      <!-- offer for Y itself to perform α-->
        <Sender> Y </Sender>
        <Recipients> X </Recipients>
        <Action> α </Action>
      </Trigger>
    </Transition>
    <Transition Name = X.reject>
    <!-- X sends a rejection from an offered state-->
      <SourceInteraction href="offered(Y)"/>
      <DestinationInteraction href="rejected(X)"/>
      <Trigger trigger-process "X.reject">
      <!-- rejection from X for Y itself to perform α-->
        <Sender> Y </Sender>
        <Recipients> X </Recipients>
        <Action> α </Action>
      </Trigger>
```

```
        </Transition>
      </Transitions>
    </Protocol>
```

Alternating Offers Negotiation Strategies

There are a number of attributes that negotiation strategies ideally seek to satisfy so that the parties interact productively and fairly. These attributes include efficiency (no wastage of resource), stability (no incentive to deviate from agreed strategies), simplicity (low computational and bandwidth costs), distribution (no central decision maker) and symmetry (no arbitrary bias against any web service). Here we show a strategy suitable for use in negotiation between web services.
Let the set of all possible agreements be denoted by $\boldsymbol{S}$. The outcome of the negotiation is an agreement $\boldsymbol{s}$ reached at time $\boldsymbol{t}$ and is denoted by the ordered pair $(\boldsymbol{s}, \boldsymbol{t})$. A disagreement denotes a rejection or a negotiation that continues forever without reaching an agreement. Each web service would prefer to agree on an outcome that is most favourable to it. The utility functions guide the web services in the process of distinguishing favourable outcomes from unfavourable ones. A web service's negotiation strategy defines what the web service should do on receiving an offer. Let $\boldsymbol{S}$ denote the set of all possible agreements that can be reached at time $\boldsymbol{t} = \{0, 1, 2, \ldots\}$ and the offers that a web service can receive denoted as $\{s0, s1, s2, \ldots\}$. A web service's strategy can be defined as:

$$\boldsymbol{f}: \{s0, s1, s2, \ldots\} \rightarrow \boldsymbol{S} \cup \{reject\}$$

- where *reject* denotes rejecting an offer or counter-offer.

The utility function assigns a numerical value to the actions of the web services. One of the main problems is to determine a strategy that would yield maximum utility for the web services. Formally the utility function can be defined as:

$$\boldsymbol{U}: \{\boldsymbol{S} \cup \{reject\} \times \boldsymbol{T}\} \rightarrow \boldsymbol{R}.$$

For every service $\boldsymbol{i}$ $\{\boldsymbol{i} = 1, 2\}$ and time period $\boldsymbol{t}$, *Possible(**i**,**t**)* is the set of all agreements that web service $\boldsymbol{i}$ prefers to rejection. If this set is non-empty then the agreement ***s (i, t)*** that belongs to *Possible (**i**, **t**)* is called the ***Best Possible Agreement*** if it satisfies:

$$\boldsymbol{U^i}(\boldsymbol{s}(\boldsymbol{i}, \boldsymbol{t})) = max_{\boldsymbol{s} \in Possible(\boldsymbol{i},\boldsymbol{t})} \boldsymbol{U^i}(\boldsymbol{s})$$

Analogously the ***Worst Possible Agreement*** is defined as $\boldsymbol{s}(\boldsymbol{i}, \boldsymbol{t})$ if it satisfies:

$$\boldsymbol{U^i}(\boldsymbol{s}(\boldsymbol{i}, \boldsymbol{t})) = min_{\boldsymbol{s} \in Possible(\boldsymbol{i},\boldsymbol{t})} \boldsymbol{U^i}(\boldsymbol{s})$$

If *S1* is the strategy adopted by web service *A1* when the strategy adopted by web service *A2*, then the pair *(S1, S2)* forms an *equilibrium solution* if neither can deviate from their

strategy without losing utility. It is one of the main objectives of game theoretic problems to determine equilibrium solutions.

Having analysed existing proposals for enabling negotiation between web services, we have in particular found the WS-Agreement model to be limited and not expressive enough for profitable semantic web service negotiation. Thus, in this paper we show our specification of negotiation actions and protocols to enable interaction between web services and we give an overview of our model for strategic decision-making.

5.6 Decision making[10]

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 [Pretschner99] 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.

[10] This work was undertaken by Ian Dickinson and Mike Wooldridge under the auspices of KnowledgeWeb. Ian Dickinson is employed by Hewlett-Packard, whom we gratefully acknowledge.

A natural idiom for encoding and executing goals is through software agents [Wooldridge95]. For the purpose of this discussion, we restrict our attention to deliberative agents [Wooldridge00a], 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 [Negroponte97]. 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.

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. [Sirin04], [Horrocks02], [Pistore04]) 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, in which 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. A related approach is explored in [Gibbins03], 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.

The authors' approach broadly follows this 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 section, we explore how this general theme is embodied in their experimental BDI agent platform.

Nuin BDI Agent Platform

The Nuin agent platform [Dickinson04] is an open-source Java implementation of a combination of a belief-desire-intention (BDI [Rao95]) 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 5.6.1. 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.

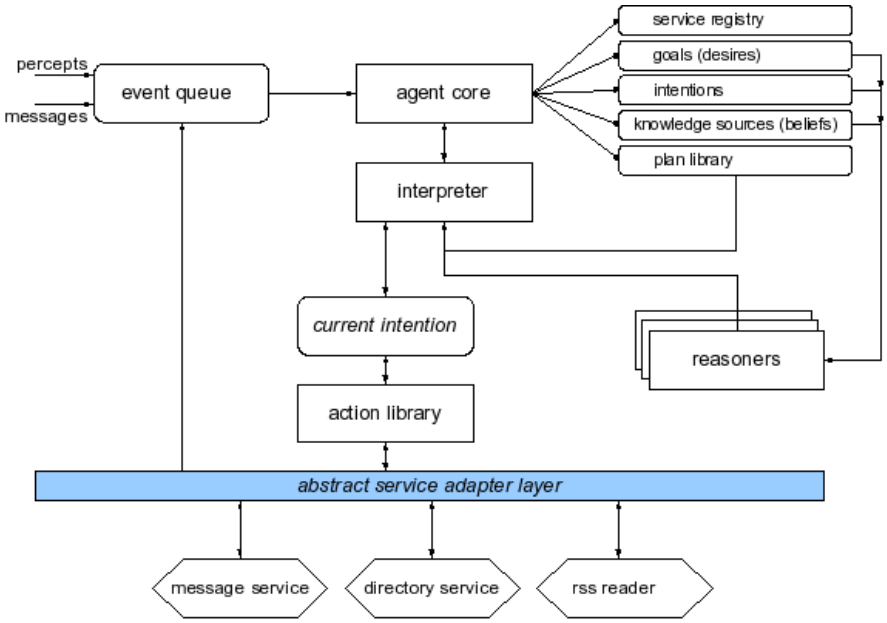

Figure 5.6.1: Outline Nuin architecture

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 [Singh94]. Currently, we use an RDF [RDF] 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.

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 request for proposals (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 [Miller90] 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.

However, encoding the strategic knowledge in the agent's KB may simply introduce a knowledge acquisition bottleneck into the design process, which 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.

We note that the Web Services Modelling Ontology (WSMO) [Lara02] 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.

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 [Georgeff90] or AgentSpeak(L) [Rao96], 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 [Ghallab04] 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 [Kuter04], but it remains the case that planning from first principles is computationally expensive ([Ghallab04], §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 [Sycara98] 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, 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.

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 . 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.

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.

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.

5.7 Integrating agent platforms and Semantic Web toolkits

To ensure that agents are properly able to understand and utilise knowledge encoded on the Semantic Web they require the capability to parse and process Semantic Web languages such as RDF and OWL. This can be achieved by usage of semantic web toolkits (such as Jena [Jena]) within agents constructed within multiagent system development environments (such as JADE [JADE]). However, the simple ability to extract knowledge encoded in these languages is not sufficient for agents to fully integrate into the Semantic Web arena. This is due to the fact that there may exist many semantic and conceptual miss-matches between Semantic Web representations and internal agent representations of the same knowledge – introducing semantic heterogeneity that in turn will lead to errors in computation. In addition, the continual process of translation between knowledge formalisms imposes a considerable workload upon agents whose main sources and consumers of knowledge conform to Semantic Web standards.

One approach to this problem would be for agents that are intended to interact with Semantic Web entities and knowledge to natively use Semantic Web languages for their internal knowledge representation. In this scenario, agent knowledge would be represented by RDF statements that reference OWL concepts and properties. Building agent knowledge upon sharable ontological definitions enhances their ability to interact with other entities on the semantic web – using existing ontology mediation (mapping, merging, etc.) techniques. Furthermore, it simplifies the task of interacting with semantic web services whose own descriptions are built on top of ontologies expressed in these standard languages.

The usage of semantic web languages to encode agent knowledge has been explored in a number of research systems, for example the SERSE system [Tamma05] in which RDF is used to represent the agent's knowledge about the web resources at its disposal. This approach has also been explored in the NUIN agent platform [Dickinson04] (see section 5.6), where RDF and OWL form the basis for all of the knowledge representation within the agents. Furthermore, the use of RDF has been extended to include a representation of the goals and the strategies adopted and pursued by the agents – so that they may be communicated and reused.

However, much research in agent systems utilises higher-level logics (first-order, modal, temporal, etc) to encode agents knowledge and reasoning [Wooldridge00a]. It is unclear to what extent description-logic-based formalisms such as OWL can represent such constructs as beliefs, desires and intentions. The NUIN system retains use of first-order for BDI representation and reasoning, and implements resource bounded reasoning over these constructs. This is because the designers belief is that agents require the richer

representation provided by FOL more than they require the computational tractability provided by description logics [Dickinson04].

5.8 Dialogues

Formal dialogue games, adopted from augmentation theory, have been studied in philosophy for at least 2500 years [Aristotle28]. They are interactions between players (two or more) in which each player can make a 'move' in the game by making utterances (public statements) that conform to a specific set of rules for that game. Modern research has used such games to study non-deductive forms of reasoning, and to provide game-theoretic semantics for intuitionistic, classical and quantum logic. In the artificial intelligence research field these dialogue games have been used to model complex human interactions, and in agents research the main application is for the design of structured agent communication/interaction protocols [McBurney02a].
The use of formal dialogue games in agent interaction protocols provides a framework for the design of structured conversations by supporting the definition of dialogue protocols. These protocols are formed from rules that constrain what can, cannot and must be said in a dialogue, and when they things are said [McBurney02b]. Such protocols are intended to enable agents, in an orderly and efficient manner, to engage in dialogues through which they acquire new information – that may lead to them changing their beliefs or preferences. To achieve this agents require means to question or challenge another agent's statements. It may be expected that the more information, relevant to the interaction, passed between participating agents, the more likely the interaction will succeed in its aims. This increased chance of success is the point of using information-rich dialogue games for interactions, as compared to less rich economic interaction mechanisms, such as auctions [Parsons98]. Although, such mechanisms typically have the advantages of simplicity and analytical tractability [Sandholm99], the provision for exchange of information is limited. For example, in auctions the participants normally have no means to provide reasons for their acceptance or rejection of a bid.

From research into human dialogues, argumentation theorists Walton and Krabbe proposed the following model of primary dialogue types [Walton95], that has been influential in the application of dialogue game protocols to agent interactions. This dialogue model is based on the information that participants have at the dialogue commencement, the participants' individual goals for the dialogue, and their shared dialogue goals.
The model defines the following typology:

- ***Negotiation dialogues*** – in which participants bargain over the division of some scare resources between themselves. The global goal of such dialogues is a division of the resource acceptable to all participants, which may be in conflict with the goals of individual participants.

- ***Persuasion dialogues*** – in which one participant seeks to persuade another to believe/endorse some proposition that they do not currently believe.

- ***Information seeking dialogues*** – in which one participant seeks to get another participant (that is believed by the first to know the answer) to answer some question.

- ***Inquiry dialogues*** – in which participants collaborate to jointly seek the answers to questions (where these answers are not known to any one participant).

- ***Deliberation dialogues*** – in which participants collaborate to determine a course of action to be adopted in some situation. The participants share responsibility for the decision; or at least share a willingness to discuss their shared responsibility. Again, the best result for the group may conflict with individual preferences or goals. In addition, no one participant may have all the necessary information to make the 'best' decision.

- ***Eristic dialogues*** – in which participants argue verbally to air perceived grievances.

It is important to note that most 'real' dialogues involve a mixture of the dialogue types, where a dialogue proceeds from one type to another, forming sequences or loops, or embedding one type of dialogue within another.

McBurney and Parsons define a model of a generic formal dialogue game, that consists of the following elements [McBurney02c]:

- ***Commencement rules*** – that define the circumstances under which the dialogue begins.

- ***Locution rules*** – that define what utterances are permitted, which typically include those permitting participants to assert propositions, question or contest prior assertions, and to justify contested assertions. Such rules may also permit participants to assign degrees of commitment to their assertions, i.e. enabling one to *propose* something rather than *asserting* it.

- ***Combination rules*** – that define the context (i.e. when and where within a sequence of locutions, actions, etc.) within which particular locution may, may not, or must be uttered.

- ***Commitment rules*** – that define the circumstances under which participants express commitment to a proposition. An assertion of a proposition indicates to the other participants that the speaker has (some level of) commitment to the proposition. Formal dialogue systems typically use publicly available commitment stores for each participant [Hamblin70].

- ***Termination rules*** – that define the circumstances under which a dialogue ends.

The commitments referred to above are dealt with in a number of different ways in dialogue game research. However, as our interest is from the perspective of interactions between autonomous agents (that the agents enter into in order to achieve some goal), such commitments can reasonably be defined in terms of future actions or propositions external to the dialogue itself [McBurney02a].

McBurney, Parsons and Wooldridge [McBurney02b] propose a set of desiderata for designers of agent interaction protocols that are based upon formal dialogue games. These desiderata are themselves based on research in agent interaction, critical assessment of auction mechanisms, and elements from argumentation theory and political theory. It is assumed that the agents participating in a dialogue have autonomously decided to participate, are free to enter and leave a dialogue when they choose, and that within a dialogue an agent is autonomous, and is not compelled to accept or reject any proposition. Furthermore, it is assumed that the specification (referred to by the authors as a *Dialectical System*) of a dialogue game protocol consists of a set of discussion topics, the syntax for locutions concerning these topics, and commencement, locution, combination, commitment and termination rules (as defined above). The thirteen desiderata are:

- ***Stated dialogue purpose*** – A dialectical system should have one or more publicly-stated purposes, and its locutions and rules should facilitate the achievement of these purposes. Such purposes may be categorised using a model of dialogue types, such as that described previously. The purposes need to be stated so that all agents are aware of them before participation, and the dialogue will be successfully resolved when these stated purposes are achieved.

- ***Diversity of individual purposes*** – A dialectical system should permit participating agents to achieve their own private purposes that are consistent with the overall dialogue purpose. These purposes may conflict, as in a negotiation dialogue, or coincide, as in a inquiry dialogue.

- ***Inclusiveness*** – A dialectical system should not preclude participation by any agent that is qualified and willing to participate. Due to their autonomy, all agents can be seen as deserving equal treatment, and having a moral right to be included in decisions that may affect them [Bohman97]. In addition, inclusion of affected parties in decisions can improve the quality of the decision outcomes [Fiorino89].

- ***Transparency*** – Dialogue participants should be aware of the rules and structure of the dialectical system before the dialogue commences. Any reference from such

dialogues to an external entity should be explicitly stated, e.g. when commitments made inside a dialogue imply a subsequent real-world obligation.

- ***Fairness*** – A dialectical system should treat all participants equally, or should explicitly state any inequalities in such treatment. For example, agents may play different roles within a dialogue, and these roles may have different rights and responsibilities, which the specification should make known to all participants.

- ***Clarity of argumentation theory*** – A dialectical system should conform to a stated theory of argumentation, e.g. Principles for Rational Mutual Inquiry [Hitchcock91] or persuasion dialogue rules (as in [Eemeren92]), so that all participants can be aware of the rules of inference and procedure, can adhere to their obligations, and have reasonable expectations of the responses of other agents in the dialogue.

- ***Separation of syntax and semantics*** – In a dialectical system the syntax should be defined separately from the semantic. Primarily this is because semantic verification of agent communication is a complex issue [Wooldridge00b], as any sufficiently complex agent can mimic the required internal states. So specification of the syntax separately at least enables verification of conformance to the protocol syntax, even if the semantics cannot be completely verified.

- ***Rule-consistency*** – The rules and the locutions of a dialogue protocol should be internally consistent, and should not lead to deadlocks or infinite loops.

- ***Encouragement of resolution*** – Normal termination (or resolution) of a dialogue should be assisted and facilitated by the rules and locutions of the system.

- ***Discouragement of disruption*** – The rules of a dialectical system should attempt to preclude any disruptive behaviour by participants, e.g. repeatedly issuing the same locution. However, a balance must be struck between stopping such behaviour and yet permitting freedom of expression [Krabbe01], and this balance will differ from application to application.

- ***Enablement of self-transformation*** – A dialectical system should permit the participants to dynamically alter their knowledge, goals, valuations, degrees of belief, etc. during the course of a dialogue based on information received from others - that is, the system should permit participants' self-transformation [Forester99]. For example, participants can *retract* a statement they have previously made in the same dialogue. Such transformation is the essential point of engaging in dialogues, as without it one agent could not persuade another to change beliefs, intended actions, etc.

- ***System simplicity*** – The rules and locutions of the system should be as simple as is possible, whilst remaining consistent with the previous desiderata. Each locution

should have a clear purpose in the dialogue, and the rules should lead to efficient resolution.

- ***Computational simplicity*** – A dialectical system should minimise computational requirements, both for the participants and the system itself, whilst remaining consistent with the previous desiderata.

In addition to these desiderata the authors note that they have not specified that any such dialectical system should be a realistic model of human dialogues, and if fact such systems could be applied to dialogue that humans could not undertake, e.g. hundreds of participants undertaking simultaneous negotiations over multiple resources [McBurney02b]. For specific types of dialogue there may be additional appropriate desiderata, for example, in negotiations by purely self-interested agents over a scarce resource it may well be desired that the outcome be Pareto optimal (i.e. any other outcome would leave at least one participant worse off). However, desired outcomes, such as Pareto optimality, may be unachievable due to other operational constraints upon the dialogue agents, e.g. time constraints, resource constraints, specific dialogue rules, social constraints, etc.

These desiderata (specifically those on *Clarity of Argumentation Theory* and *Enablement of Self-transformation*) as based upon a particular view of joint decision-making by autonomous entities. A distinction can be made between market-place (or rational choice) models and models of social decision-making (deliberative democracy) [Bohman97]. In the market-place models, each participant is seen as entering the process with a fully-formed set of beliefs, preferences, utilities, etc. These models do not allow for such beliefs to be dynamically determined during the interaction, nor do they allow for a group view (in terms of wider social consequences) of the decision-making issues to be formed. Deliberative democracy models, in contrast, focus upon the way in which beliefs, etc. can be formed and changed through the process of interaction, by participants undergoing self-transformation [Forester99]. This transformation occurs by argument and debate – i.e. by participants sharing information, challenging and defending assertions, and by persuasion or joint consideration.

Dialogue game protocols have mainly been proposed for autonomous agent interactions, however, an agent communication language is required to enable the agents to interact in dialogues. Although, the FIPA Agent Communication Language has been designed as a generic agent language, it could potentially be used as the communication mechanism in dialogue games. Therefore, it is interesting to assess the ACL with respect to the desiderata described above:

- *Stated dialogue purpose* – The language is mainly intended for use in purchase negotiations, and there does not seem to be any means to declare the purpose of other dialogue types.
- *Diversity of individual purposes* – Enabled.

- *Inclusiveness* – Enabled.
- *Transparency* – Enabled.
- *Fairness* – Enabled.
- *Clarity of argumentation theory* – The language has no explicit underlying argumentation theory, and the implicit theory has only limited ability to question and contest received information (by use of *not-understood*).
- *Separation of syntax and semantics* – These are not separate, as the syntax is defined in terms of the semantics.
- *Rule-consistency* – Enabled.
- *Encouragement of resolution* – Resolution is not discouraged, but no rules exist for dialogue termination.
- *Discouragement of disruption* – This is not addressed by any specific rules, however, the conditions imposed by the language semantics may limit disruptive behaviour.
- *Enablement of self-transformation* – This is limited, due to the sincerity constraints imposed on agent utterances by the language semantics, and the fact that there is no means for agents to retract previous assertions, qualify assertions, or express degrees of belief.
- *System simplicity* – The language locutions consist of both substantive locutions (e.g. *accept-proposal*) and procedural locutions (e.g. *propagate*), which should be explicitly defined as different classes of locution.
- *Computational simplicity* – This is difficult to assess, as the ACL is basically just a message format system. However, any internal proof mechanisms used by the agents (e.g. to ensure sincerity) must use the first-order model logic of the ACL semantics [Labrou99], and this is only semi-decidable.

Therefore the key weakness of FIPA ACL in respect of its support for the use of formal dialogue games between agents is that it has limited support for formal argumentation and for self-transformation by agents. This is in part due to the fact that the ACL is based on a market-place (or rational choice) model of decision-making, due to its intended use in purchase negotiation interactions, and does not attempt to model a theory of argumentation. This would seem to suggest that the FIPA ACL cannot be used in the more complex, argumentation-based dialogue games described above.

Development of the thirteen desiderata above, and analysis of various concrete protocols using the desiderata, lead to the proposal of a set of guidelines for designers and users of agent interaction protocols based on argumentation [McBurney02b]:

1) The protocol should embody a formal and explicit theory of argument.

2) The rules for the protocol should ensure that the reason(s) for conducting the dialogue are stated within the dialogue at its commencement.

3) The protocol should include locutions which enable participants to:
   a) formally enter a dialogue,
   b) request information,
   c) provide information,
   d) request arguments and reasons for assertions,
   e) provide arguments and reasons for assertions,
   f) challenge statement and arguments,
   g) defend statement and arguments,
   h) retract previous assertions,
   i) make tentative proposals,
   j) express degrees of belief in statements,
   k) express degrees of acceptability or preferences regarding proposals,
   l) formally withdraw from a dialogue.

4) The protocol syntax should be defined in observable terms, so that its conformance can be verified without reference to internal states or mechanisms of the participants.

5) The rules of the protocol should seek to preclude disruptive behaviour.

6) The rules of the protocol should indicate circumstances under which a dialogue terminates.

7) The rules of the protocol should identify any difference in formal roles and the rights and duties pertaining to these.

This discussion of dialogues in multiagent systems demonstrates that FIPA ACL itself does not support some of the advanced communication features required for agents to realise their full potential in this context. It is clear that in agent communication, as in all other forms, that the more sophisticated the reasoning and action sought from agents, the more sophisticated and knowledge-rich their communication is required to be. However, any increase in communication and reasoning complexity causes the design and implementation of such multiagent systems to become significantly more complex also. This clashes with the intention behind web services – that they are relatively simple components that can be easily constructed and used. Therefore, it would seem unlikely that a single communication formalism could, at the same time, be both simple to implement and use, and be able to support the sort of complex dialogues discussed above.

6. Conclusions

Examination of various research efforts seeking to interoperate and/or integrate (semantic) web services and agent services reveals three main integration themes:

1) **Two service spaces using uni-directional inter-operation.**
   Web services are seen as providing only relatively straight-forward services, that conform to the simple request-response model embodied in SOAP. Agents are seen as providing more complex services, where the services are embedded within interaction protocols (e.g. contract-net, auctions, etc.). This view-point often includes the usage of such basic web services by agents, where the each of the web services used contribute a part of the overall service provided by the agent – i.e. the agent is acting in the role of choreographer and orchestrator of the web services (e.g. the Nuin system described in section 5.6).
   Such situations would support the usage of gateway and translation techniques (such as the Web Services Integration Gateway described in section 5.2), that enable agents to discover and invoke web services.
   This approach has been applied in situations where the role of the agent is as an intelligent user interface that dynamically discovers and utilises appropriate web services in response to action requests directly from the user (e.g. Semantic Web Fred [SWF] and the proposal of Iqbal *et al.* described in section 5.1).
   However, current research in the semantic web services field is based on the development and usage of complex, choreographed and orchestrated services within a web services environment. This does not coincide with the view of web services as a 'simple' service layer, and removes some of the basis for the distinction between agent services and web services.

2) **Single service space using bi-directional inter-operation.**
   Agents and web services fully inter-operate and there is no significant conceptual difference between them. This is implied by single conceptual models encompassing both types of service (such as the FIPA Abstract Service Architecture described in section 5.1). This view is also partially implied by gateway architectures that provide bi-directional inter-operation via translation between service formalisms (such as the Agentcities gateway proposal and the Web Service Integration Gateway described in section 5.2). Such gateway proposals retain some distinction between the two service types, though this is largely based on the different formalisms used.
   However, this conceptualisation of little or no distinction between the services types loses part of the *agency* of the agent services. A simple example of this can be seen in the difference between web service invocation and agent service requests. In addition, web services invoking agents services presupposes that agents publish declarative service descriptions of pre-determined behaviours, which has serious implications for agent autonomy. If the service can be written to conform to the web service pattern of declarative description and pre-determined

behaviour it is difficult to see in what way it is an agent service. Agency is about more than just using FIPA - agency is about *how* the service is implemented, not the technology stack used. Furthermore, multiagent systems have an underlying conceptualization that takes into account the fact that the providers of services and users of these services may have competing goals that drive their behaviour. Agent interaction protocols require a significant degree of complexity to enable agents to successfully (in terms of their own goals) interact under these circumstances.

3) **Single integrated service space.**
In this view web services in the dominant paradigm that provides the service oriented architecture middleware for all services. Such web services conform to the vision of semantic web services, in that they are dynamic, coordinable (through choreography and orchestration), flexible, etc. These web services may utilise agent and AI techniques within their implementation, and can, therefore, exhibit degrees of autonomy, social ability, pro-activity, etc.
However, web services are limited in their ability to exhibit agency, largely due to limitations imposed by web service communication formalisms. Attempts to overcome these limitations can be seen in efforts to apply agent communication techniques to web service communication (such as that by Willmott *et al.* described in section 5.3).
Therefore, is this scenario both *conventional* web services and *agent* web services would co-exist in a single service architecture, but in order to retain a single description, invocation and coordination framework the degree to which agent techniques can be applied is severely limited. Extensions to web service communication would reduce the limitations on agency, but this may then require a distinction to be made between the service types. This then leads to the situation described above, in which different service types interoperate via translation/gateways. One possible approach to managing the different service types within a single architecture would be to use a layered communication model.

In light of these identified research themes, we can see that there are two intertwined avenues that can be pursued to bring web services and agent-based services closer together:

- **Interoperation**
Web services and agent-based service retain a separate identity and focus, creating two heterogeneous service spaces. Interoperation is mainly based on agents using web services, using gateways and translation between formalisms (plus some implicit assumptions regarding service use). However, the use of agent-based services by web services (or any other client using web service formalisms) remains problematic.

- **Integration**
  Current developments in semantic web services have lead to an increasing overlap with agent-based services, both in the aims and abilities of the paradigms. Therefore, there is now some scope for providers of web services to utilise some agent / artificial intelligence techniques in their implementation, leading to a degree of integration of agent services into web services. This is particularly true for those agent services whose claim to agency was weak, and largely based upon the development/implementation environment rather than the characteristics of the service itself.
  However, there is a limit to this integration due to fundamental differences in system conceptualisation:

  - The key feature of agent systems is that of autonomy – agents are designed as independent entities that act on the basis of their own goals and knowledge. 'High-level' agent systems are based on modelling processes in terms of 'mentalistic notions', such as in Belief, Desire, Intention (BDI) models. In such systems, interaction between agents is based on conversations, negotiation and argumentation dialogues (see section 5.8), in which the agents autonomously pursue individual (or group) goals. These systems produce 'societies' of independent agents, which may exhibit emergent behaviour at this societal level.

  - Web services are intended to provide an underlying architecture for distributed functionality, based on declarative advertisement of services followed by dynamic discovery and invocation of the services. Web services are intended to be relatively simple, i.e. easy to use and implement, robust, reliable, etc.

  Therefore, increasing use of agent techniques in web service implementations moves them away from their intended task, i.e. makes them complex to implement and use. In a web service architecture the ability to exhibit autonomy must be limited in order to respect the requirements service advertisement and invocation. This implies that a distinction must remain between 'high-level' agent societies and web services, and that interoperation between them is limited to the use of web services by agents as external functionalities. One of the goals of continuing research into semantic web services must be to determine the degree to which agent techniques can be applied to web services, and the degree to which web services can and should exhibit agency. It is important to recognize when it is appropriate to use agent-based solutions – if a component does not exhibit any features of agency then there is no necessity to implement it as an agent.

Finally, in addition to the possibilities of interoperation and integration between web services and agent-based services, it is clear that there are a number of areas in which the two paradigms address similar issues and could usefully inform each other:

- Dynamic service coordination – agent interaction protocols and web service choreography bear significant similarity. In conjunction with other related approaches (such as the coordination ontology described in section 5.4), these .
- Negotiation and agreement – as applied in WS-Agreement (see section 5.5).
- Representation of user goals and intentions – widespread research in this within the agent research field. Connecting abstract user goals to concrete services.
- Service description and matchmaking – service description and consequently service matchmaking stronger in web services.
- Intelligent decision-making – agent research can contribute significant experience in enabling components to reach intelligent decisions and take intelligent actions based on their own knowledge. For example the Nuin system, that uses semantic web languages to represent knowledge within a Belief, Desire, Intention model that enables the agents to use this knowledge to act in pursuit of their own goals (see section 5.6).

Bibliography

[Andrieux04] – Andrieux, A., Czajkowski, K., Dan, A., *et al*. "Web Services Agreement Specification (WS-Agreement)". World-Wide-Web Consortium (W3C), 2004.

[AOSG] – Agent Oriented Software Group: Company web site, 2004.

[Aristotle28] – Aristotle. *Topics*. Clarendon Press, Oxford, UK, 1928. W. D. Ross, Editor.

[Ben-Ari90] – Ben-Ari, M. *Principles of Concurrent and Distributed Programming*. Prentice Hall, 1990.

[Bohman97] – Bohman, J. and Rehg, W. (eds.) *Deliberative Democracy: Essays on Reason and Politics*. MIT Press, Cambridge, MA, USA, 1997.

[Booth03] – Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (eds.) *Web Services Architecture,* W3C Working Draft, 8th August 2003. `http://www.w3.org/TR/2003/WD-ws-arch-20030808/`

[Booth04] – Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (eds.) *Web Services Architecture,* W3C Working Group Note, 11th February 2004.
`http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`

[Breese98] – Breese, J.S., Heckerman, D., and Kadie, C. "Empirical Analysis of Predictive Algorithms for Collaborative Filtering". In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998.

[Buhler04] – Buhler P.A. and Vidal J.M. "Integrating Agent Services into BPEL4WS Defined Workflows". In *Proceedings of the Fourth International Workshop on Web-Oriented Software Technologies*, Munich, Germany, 2004.

[Burdett04] – Burdett, D. and Kavantzas, N. *WS Choreography Model Overview (working draft)*. 2004.
`http://www.w3.org/TR/2004/WD-ws-chor-model-20040324/`

[Christensen01] – Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. *Web Services Description Language (WSDL) 1.1*. 2001.
`http://www.w3.org/TR/wsdl`

[Dan04] – Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M., and Youssef, A. "Web services on demand: WSLA-driven automated management". In *IBM Systems Journal*, vol. 43, num. 1, 2004.

[Dale03a] – Dale, J., Hajnal, A., Kernland, M., and Varga, L.Z. "Integrating Web Services into Agentcities Recommendation". *Agentcities Technical Recommendation* [actf-rec-00006], 2003.

[Dale03b] – Dale, J. and Lyell, M. "Towards an Abstract Services Architecture for Multi-agent Systems", 2003.
www.agentcities.org/Challenge03/ Proc/Papers/ch03_dalelyell.pdf

[Decker95] – Decker, K. and Lesser, V. "Designing a family of coordination algorithms". In *Proceedings of the First International Conference on Multi-Agent Systems*, San Francisco, CA, 1995.

[Dickinson03] – Dickinson, I. and Wooldridge, M. "Towards Practical Reasoning Agents for the Semantic Web". In *Proceedings of Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia, 2003.

[Dickinson04] – Dickinson, I. *Nuin: the Jena Agent Framework*. 2004.
http://www.nuin.org

[Dickinson05] – Dickinson, I. and Wooldridge, M. "Agents are not (just) we services: considering BDI agents and web services". To appear in *Proceedings of SOCABE'05*, 2005.

[Durfee88] – Durfee, E.H. *Coordination of Distributed Problem Solvers*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1988.

[Dustdar04] – Dustdar, S. and Treiber, M. "A View Based Survey on Web Services Registries". 2004.
http://www.infosys.tuwien.ac.at/Staff/sd/papers/TUV-1841-2004-19.pdf

[Eemeren92] – van Eemeren, F.H. and Grootendorst, R. *Argumentation, Communication and Fallacies: A Pragma-Dialectical Perspective*. LEA, Mahwah, NJ, USA, 1992.

[Eriksson05] – Eriksson, H. *JessTab*, 2005.
http://www.ida.liu.se/~her/JessTab/

[Fikes03] – Fikes, R., Hayes, P., and Horrocks, I. "OWL-QL – A Language for Deductive Query Answering on the Semantic Web." *Technical Report: Knowledge Systems Laboratory (KSL-03-14)*, Stanford University, CA, 2003

[FIPA] – *FIPA Specification,* Foundation for Intelligent Physical Agents, 2002.
http://www.fipa.org/

[Fiorino89] – Fiorino, D.J. "Environmental risk and democratic process: a critical review". In *Columbia J. Environmental Law*, vol. 14, pp. 501–547, 1989.

[Forester99] – Forester, J. *The Deliberative Practitioner: Encouraging Participatory Planning Processes*. MIT Press, Cambridge, MA, USA, 1999.

[Friedman-Hill05] – Friedman-Hill, E. *Jess*, 2005.
`http://herzberg.ca.sandia.gov/jess/`

[Georgeff90] – Georgeff, M. and Ingrand, F. *Research on Procedural Reasoning Systems (Final Report – Phase 2)*. SRI International, 1990.

[Ghallab04] – Ghallab, M., Nau, D., and Traverso, P. *Automated Planning : Theory & Practice.* Morgan-Kaufmann, 2004.

[Gibbins03] – Gibbins, N., Harris, S., and Shadbolt, N. "Agent-Based Semantic Web Services". In *Proceedings of the Twelfth International World Wide Web Conference*, ACM, 2003.
`http://eprints.ecs.soton.ac.uk/7278/`

[Good99] – Good, N., Schafer, J.B., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J., and Riedl, J. "Combining Collaborative Filtering With Personal Agents for Better Recommendations". In *Proceedings of the 16th National Conference on AI (AAAI-99)*, pp. 439–446, AAAI Press, 1999.

[Greenwood04] – Greenwood, D. and Calisti, M. "An Automatic, Bi-Directional Service Integration Gateway". In *Proceedings of AAMAS Workshop on Web Services and Agent-based Engineering*, New York, USA, 2004.

[Gurevich93] – Gurevich, Y. "Evolving Algebras 1993: Lipari Guide". In Börger, E. (ed.) *Specification and Validation Methods*, pp. 9-36, Oxford University Press, 1995.

[Hamblin70] – Hamblin, C.L. *Fallacies*. Methuen, London, UK, 1970.

[Hitchcock91] – Hitchcock, D. "Some principles of rational mutual inquiry". In van Eemeren, F., *et al.* (eds.) *Proceedings of the 2nd International Conference on Argumentation*, pp. 236–243, Amsterdam, 1991.

[Horrocks02] – Horrocks, I. "Reasoning With Expressive Description Logics: Theory and Practice". In Voronkov, A. (ed.) *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, pp. 1–15, Springer Verlag, 2002.

[Huhns02] – Huhns, M.N. "Agents as Web Services". In *IEEE Internet Computing*, July, 2002.

[Iqbal04] – Iqbal, K., Farooq Ahmad, H., Ali, A., Suguri, H., and Jamshed, M. "Autonomous Distributed Services Implementation". In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*, Tokyo, Japan, 2004.

[JADE] – Telecom Italia Lab. *JADE: Java Agent DEvelopment Framework.* `http://sharon.cselt.it/projects/jade/`

[JENA] – HP Labs Semantic Web Research. *Jena: Semantic Web Framework for JAVA*. `www.jena.sourceforge.net/`

[Krabbe01] – Krabbe, E.C.W. "The problem of retraction in critical discussion". In *Synthese*, 127(1-2), pp. 141–159, 2001.

[Kuter04] – Kuter, U., Sirin, E., Nau, D., Parsia, B., and Hendler, J. "Information Gathering During Planning for Web Service Composition". In *Proceedings of the Third International Semantic Web Conference (ISWC2004)*, 2004.
`http://www.mindswap.org/papers/ISWC04-Enquirer.pdf`

[Labrou99] – Labrou, Y., Finin, T., and Peng, Y. "Agent communication languages: The current landscape". In *IEEE Intelligent Systems*, 14(2), pp. 45–52, 1999.

[Lara04] – Lara, R., Roman, D., Polleres, A., and Fensel, D. "A Conceptual Comparison of WSMO and OWL-S". In *Proceedings of the European Conference on Web Services (ECOWS 2004)*, 2004.
`http://www.uibk.ac.at/~c703225/papers/conceptualcomparison.pdf`

[Lashkari94] – Lashkari, Y., Metral, M., and Maes P. "Collaborative Interface Agents". In *Proceedings of the 12th National Conference on Artificial Intelligence*, Seattle, Washington, 1994.

[McBurney02a] – McBurney, P. and Parsons, S. "Dialogue Games in Multi-Agent Systems". In *Informal Logic. Special Issue on Applications of Argumentation in Computer Science*, 22 (3), pp. 257-274, 2002.

[McBurney02b] – McBurney, P., Parsons, S., and Wooldridge, M. "Desiderata for Agent Argumentation Protocols". In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS '02)*, Bologna, Italy, 2002.

[McBurney02c] – McBurney, P. and Parsons, S. "Games that agents play: a formal framework for dialogues between autonomous agents". In *Journal of Logic, Language and Information*, 11(3), pp. 315-334, 2002.

[McIlraith02] – McIlraith, S.A. and Cao Son, T. "Adapting Golog for composition of semantic web services". In Fensel, D., Giunchiglia, F., McGuinness, D., and Williams,

M. (eds.) *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pp. 482–496, 2002.

[Malone94] – Malone, T.W. and Crowston, K. "The interdisciplinary study of coordination". In *ACM Computing surveys*, vol. 26**,** 1994.

[Martin04] – Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., Mcllraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. *OWL-S: Semantic Mark-up for Web Services (v. 1.1)*, 2004.
`http://www.daml.org/services/owl-s/1.1/overview/`

[Miller90] – Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. "Introduction to WordNet: An On-Line Lexical Database". In *International Journal of Lexicography,* vol. 3, pp. 235-244, 1990.

[Negroponte97] – Negroponte, N. "Agents: From Direct Manipulation to Delegation". In Bradshaw, J. (ed.) *Software Agents*, pp. 57-66, AAAI Press, 1997.

[OntoGrid] – Ontogrid Project. *Paving the way for Knowledgeable Grid Services and Systems*.
`http://ontogrid.net/`

[OWL] – W.W.W. Consortium. *Website for the specification of OWL*, 2004.
`http://www.w3.org/2004/OWL/`

[Paolucci03] – Paolucci, M., Sycara, K., and Kawamura, T. "Delivering Semantic Web Services". In *Proceedings of the 12th International Conference on the World Wide Web*, ACM Press, 2003.

[Paolucci04] – Paolucci, M., Soudry, J., Srinivasan, N., and Sycara, K. "A Broker for OWL-S Web services". In *Proceedings of the AAAI Spring Symposium Series – Semantic Web Services*, Southampton, UK, 2004.

[Parsons98] – Parsons, S., Sierra, C., and Jennings, N.R. "Agents that reason and negotiate by arguing". In *Journal of Logic and Computation*, 8(3), pp. 261-292, 1998.

[Paurobally05a] – Paurobally, S. and Jennings, N. "Protocol engineering for web service conversations". In *Engineering Applications of Artificial Intelligence - Special Issue on Agent-oriented Software Development*, 18(2), pp. 237–254, 2005.

[Paroubally05b] – Paroubally, S., Tamma V., and Wooldridge, M. "Coorperation and Agreement between Semantic Web Services". To appear in *Proceedings of the W3C Workshop on Frameworks for Semantics in Web Services*, Innsbruck, Austria, 2005.

[Pistore04] – Pistore, M., Barbon, F., Bertoli, P., Shaparau, D., and Traverso, P. "Planning and Monitoring Web Service Composition". In *Proceedings of Workshop on*

Planning and Scheduling for Web and Grid Services, 2004.
http://www.isi.edu/ikcap/icaps04-workshop/final/pistore.pdf

[Pretschner99] – Pretschner, A. and Gauch, S. "Ontology Based Personalized Search". In *Proceedings 11th International Conference on Tools with Artificial Intelligence (TAI 99)*, pp. 391-398, IEEE Computer Society, 1999.

[Protégé] – Stanford Medical Informatics. *Protégé*, 2005.
http://protege.stanford.edu/

[PSL] – PSL: process specification language.
http://www.mel.nist.gov/psl/

[Rao95] – Rao, A. and Georgeff, M. "BDI Agents: From Theory to Practice". In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.

[Rao96] – Rao, A. "AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language". In *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '96)*, pp. 42–55, Springer-Verlag, 1996.

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

[Roman04] – Roman, D., Lausen, H., and Keller, U. "D2v02. Web Service Modelling Ontology - Standard". *WSMO Working Draft*, 2004.
http://www.wsmo.org/2004/d2/v02/20040306/

[Rubinstein82] – Rubinstein, A. "Perfect equilibrium in a bargaining model". In *Econometrica*, 50(1), pp. 97–109, 1982.

[Sandholm99] – Sandholm, T.W. "Distributed rational decision making". In Weiss, G. (ed.) *Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence*, pp. 201-258, MIT Press, Cambridge, MA, USA, 1999.

[Searle69] – Searle, J.R. *Speech acts: An essay in the philosophy of language*. Cambridge University Press, 1969.

[Singh94] – Singh, M. *Multiagent Systems: a Theoretical Framework for Intentions, Know-How, and Communications,* Springer-Verlag, 1994.

[Singh05] – Singh, M. and Huhns, M. *Service-Oriented Computing*, Wiley, 2005.

[Sirin04] – Sirin, E. and Parsia, B. "Planning for Semantic Web Services". In *Proceedings of the 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

[Smith81] – Smith, R.G. "The contract net protocol: High-level communication and control in a distributed problem solver". In *IEEE Transactions on Computers*, C-29(12), pp. 1104–1113, 1981.

[SOAP] – SOAP version 1.2.
http://www.w3.org/TR/soap/

[SWF] – Semantic Web FRED.
http://swf.deri.at

[Sycara88] – Sycara, K., Lewis, M., Lenox, T., and Roberts, L. "Calibrating Trust to Integrate Intelligent Agents into Human Teams". In *Proceedings of the 31st Annual Hawaii International Conference on System Sciences*, pp. 263 – 268, IEEE, 1998.

[Sycara02] – Sycara, K.P., Widoff, S., Klusch, M., and Lu, J. "Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace". In *Autonomous Agents and Multi-Agent Systems*, vol.5, num.2, pp. 173–203, 2002.

[Thatte03] – Thatte, S. (ed.) *Business Process Execution Language for Web Services. Specification v1.1*, 2003.
ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

[UDDI] – UDDI Specification. *Technical Committee Draft*, 19 October 2004.
http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf

[vMartial90] – von Martial, F. "Interactions among autonomous planning agents". In Demazeau, Y. and Muller, J.P. (eds.) *Decentralized AI - Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.

[vMartial92] – von Martial, F. "Coordinating Plans of Autonomous Agents". In *LNAI* vol. 610, Springer-Verlag, Berlin, Germany, 1992.

[Verma04] – Verma, K., Sivashanmugam, K., Sheth, A., and Patil, A. "METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services". In *Journal of Information Technology and Management*, 2004.

[W3CSW] – W3C. *W3C Semantic Web Activity*, 2001.
http://www.w3.org/2001/sw/

[Walton95] – Walton, D.N. and Krabbe, E.C.W. *Commitment in Dialogue: Basic*

Concepts of Interpersonal Reasoning, SUNY Press, Albany, NY, USA, 1995.

[Willmott05] – Willmott S., Pena F.O.F., Merida-Campos C., and Constantinescu I. "Adapting Agent Communication Languages for Web Service to Web Service Communication". To appear in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, Compiegne, France, 2005.

[Wohed03] – Wohed, P., van der Aalst, W.M.P., Dumas, M., and ter Hofstede, A.H.M. "Analysis of Web Services Composition Languages: The Case of BPEL4WS". In *Proceedings 29th EUROMICRO Conference - Track on Software Process and Product Improvement*, pp. 298-307, Belek-Antalya, Turkey, 2003.

[Wooldridge95] – Wooldridge, M. and Jennings, N. "Intelligent Agents: Theory and Practice". In *Knowledge Engineering Review*, vol. 10:2, pp. 115-152, 1995.

[Wooldridge00a] – Wooldridge, M. *Reasoning About Rational Agents,* MIT Press, 2000.

[Wooldridge00b] – Wooldridge, M. "Semantic issues in the verification of agent communication languages". In *Autonomous Agents and Multi-Agent Systems*, 3(1), pp. 9–31, 2000.

[Wooldridge02] – Wooldridge, M. *An Introduction to Multiagent Systems*, John Wiley & Sons, 2002.

[WSDL] – Web Services Description Language (version 2.0) Part 1: Core Language. *W3C Working Draft*, 2004.
`http://www.w3.org/TR/wsdl20/`

[WSMO-CO] – "D14v0.2: Ontology-based Choreography and Orchestration of WSMO Services". *WSMO Working Draft*, 2005.
`http://www.wsmo.org/TR/d14/v0.2/20050702/`

[Wu03] – Wu, D., Parsia, B., Sirin, E., Hendler, J., and Nau, D. "Automating DAML-S Web Services Composition Using SHOP2". In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, 2003.

[Zhao04] – Zhao, L., Mehandjiev, N., and Macaulay, L. "Agent Roles and Patterns for Supporting Dynamic Behaviour of Web Services Applications". In *Proceedings of Autonomous Agents and Multiagent Systems (AAMAS'04)*, New York, USA, 2004.