



Data, Information and Process Integration
with Semantic Web Services

DIP

Data, Information and Process Integration with Semantic Web Services

FP6 - 507683

Deliverable

D3.1

Report on State of the Art and Requirements analysis

Version 1.0

Sinuhé Arroyo

Ioan Toma

Dumitru Roman

Christian Drumm

Marin Dimitrov

Murray Spork

Gabor Nagypal

John Domingue

Jan Henke

July 16th, 2004

EXECUTIVE SUMMARY

This deliverable delineates the state of the art of on process, business, data and epistemological ontologies that are to be used within the DIP¹ project. It presents the Semantic Web Service Usage process as a starting point to motivate the discussion, and to point out where each of the different ontologies fits. Further it introduces the different tools that currently help realizing the different parts of such usage process, and its relation with the different ontologies presented in the state-of-the-art, if any. Finally the requirements in terms of Service description and service requirements for DIP and Workpackage 3 of the DIP project² are presented.

The intended audience of the deliverable is mainly the DIP project partners, in particular those involved in Workpackage 3 and Workpackage 4. Nevertheless, anyone interested on Service description and how semantically achieve it, may find it of interest.

In regard to the impact of this deliverable for DIP, it provides a detailed state of the art on the technology available to describe Services, pointing out the lack or young state of it, which can be considered the main output of the work.

¹ DIP (“Data, Information and Processes) is an Integrated Project under the European Commission's 6th Framework Programme. See <http://dip.semanticweb.org/>

² Workpackage 3 within the DIP project is titled “Service Description and Service Ontologies”.

Document Information

IST Project Number	FP6 – 507483	Acronym	DIP
Full title	Data, Information, and Process Integration with Semantic Web Services		
Project URL	http://www.nextwebgeneration.org/projects/dip/		
Document URL			
EU Project officer	Daniele Rizzi		








Deliverable	Number	3.1	Title	Report on state of the art and requirements analysis
Work package	Number	3	Title	Service Ontologies and Service Description

Date of delivery	Contractual	30.06.04	Actual	30.06.04
Status	Version. 1.0		final <input checked="" type="checkbox"/>	
Nature	Prototype <input type="checkbox"/> Report <input checked="" type="checkbox"/> Dissemination <input type="checkbox"/>			
Dissemination Level	Public <input checked="" type="checkbox"/> Consortium <input type="checkbox"/>			

Authors (Partner)	Ioan Toma (UIBK), Dumitru Roman (UIBK), Christian Drumm (SAP), Marin Dimitrov (Sirma), Murray Spork (SAP), Gabor Nagypal (FZI), John Domingue (OU)			
Responsible Author	Sinuhé Arroyo		Email	sinuhe.arroyo@deri.at
	Partner	IF mailto:dyb.kjaer@pdc.dk	Phone	+43 (0) 512 / 507 6480

Abstract (for dissemination)	This deliverable describes the state of the art on Service Ontologies and Service Description developed within the DIP project.	
Keywords	Service Ontologies, Service Description, Epistemological Ontologies, Business Ontologies, Data Ontologies, Process Ontologies.	
Version log/Date	Change	Author
20-04-2004	First Version	Sinuhé Arroyo
18-05-2004	Version 0.6	Ioan Toma
25-05-2004	Version 0.7	Ioan Toma
07-06-2004	Version 0.8	Sinuhé Arroyo
16-07-2004	Latest Version	Ioan Toma

Project Consortium Information

Partner	Acronym	Contact
National University of Ireland Galway	NUIG 	Prof. Dr. Christoph Bussler Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland Email: chris.bussler@deri.ie Tel: +353 91 512460
British Telecommunications Plc.	BT 	Dr John Davies BT Exact (Orion Floor 5 pp12) Adastral Park Martlesham, Ipswich IP5 3RE, UK Email: john.ni.davies@bt.com Tel: +44 1473 609583
Swiss Federal Institute of Technology, Lausanne	EPFL 	Prof. Karl Aberer Distributed Information Systems Laboratory École Polytechnique Fédérale de Lausanne Bât. PSE-A 1015 Lausanne, Switzerland Email : Karl.Aberer@epfl.ch Tel: +41 21 693 4679
Institut für Informatik, Leopold-Franzens Universität Innsbruck	IFI 	Prof. Dieter Fensel Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria Email: dieter.fensel@uibk.ac.at Tel: +43 512 5076485
ILOG SA	ILOG  Changing the rules of business	Christian de Sainte Marie 9 Rue de Verdun, 94253, Gentilly, France Email: cma@ilog.fr Tel: +33 1 49082981
SAP AG	SAP 	Dr. Elmar Dörner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany Email: elmar.dorner@sap.com Tel: +49 721 6902 31
Tiscali Österreich GmbH	Tiscali 	Dieter Haacker Tiscali Österreich GmbH. Diefenbachgasse 35, A-1150 Vienna, Austria Email: Dieter.Haacker@at.tiscali.com Tel: +43 1 899 33 160

Fundacion De La Innovacion.Bankinter	<p>Bankinter</p> 	<p>Jose Luis Bas Fundacion de la Innovation. BankInter, Paseo Castellana, 29, 28046 Madrid, Spain Email: jlbas@bankinter.es Tel: 916234238</p>
Berlecon Research GmbH	<p>Berelcon</p> 	<p>Dr. Thorsten Wichmann Berlecon Research GmbH, Oranienburger Str. 32, 10117 Berlin, Germany Email: tw@berlecon.de Tel: +49 30 2852960</p>
Essex County Council	<p>Essex</p> 	<p>Mary Rowlett, Essex County Council, PO Box 11, County Hall, Duke Street, Chelmsford, Essex, CM1 1LX, United Kingdom. Email: maryr@essexcc.gov.uk Tel: +44 (0)1245 436524</p>
Forschungszentrum Informatik	<p>FZI</p> 	<p>Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany Email: abecker@fzi.de Tel: +49 721 9654 0</p>
inubit AG	<p>Inubit</p> 	<p>Torsten Schmale, inubit AG Lützowstraße 105-106 D-10785 Berlin, Germany Email: ts@inubit.com Tel: +49 30726112 0</p>
Intelligent Software Components, S.A.	<p>iSOCO</p> 	<p>Dr. V. Richard Benjamins, Director R&D Intelligent Software Components, S.A. Francisca Delgado, 11-2 28108 Alcobendas, Madrid, Spain Email: rbenjamins@isoco.com Tel. +34 913 349 797</p>
Net Dynamics Internet Technologies GmbH u. Co KG	<p>Net Dynamics</p> 	<p>Peter Smolle Net Dynamics Internet Technologies GmbH u. Co KG Prinz-Eugen-Strasse 68-70 A-1040 Wien, Austria Email: peter.smolle@netdynamics-tech.com Tel.: +43 1 503982615</p>
Sirma AI Ltd.	<p>Sirma</p> 	<p>Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria Email: atanas.kiryakov@sirma.bg Tel.: +359 2 9768 303</p>

The Open University	<p>OU</p> 	<p>Dr. John Domingue Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK Email: j.b.domingue@open.ac.uk Tel.: +44 1908 655014</p>
Unicorn Solution Ltd	<p>Unicorn</p> 	<p>Dr. Joshua Fox Senior Software Architect Malcha Technology Park 1 Jerusalem 96951 Israel Email: Tel.: +972 2 6491115</p>
Vrije Universiteit Brussel	<p>VUB</p> 	<p>Carlo Wouters, Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel ,Belgium Email: carlo.wouters@vub.ac.be Tel.: +32 (0) 2 629 3719</p>

TABLE OF CONTENTS

EXECUTIVE SUMMARY.....	I
TABLE OF CONTENTS	VI
1 INTRODUCTION.....	1
2 SEMANTIC WEB SERVICE USAGE PROCESS.....	3
2.1 Discovery.....	3
2.2 Selection.....	3
2.3 Composition.....	4
2.4 Mediation.....	4
2.5 Execution.....	5
2.6 Execution Support	6
2.6.1 Monitoring	6
2.6.2 Compensation.....	7
2.6.3 Replacement	7
3 STATE OF THE ART	9
3.1 Service description.....	9
3.1.1 Syntactic vs. Semantic service description.....	11
3.1.1.1 Syntactic description.....	11
3.1.1.2 Semantic description.....	11
3.2 Service Ontology	12
3.2.1 Epistemological Ontologies	12
3.2.1.1 WSDL	12
3.2.1.2 OWL-S.....	13
3.2.1.3 WSMO.....	14
3.2.1.4 SWSL.....	17
3.2.2 Epistemological Ontologies Comparaison	18
3.2.3 Process Ontologies.....	19
3.2.3.1 BPEL4WS.....	20
3.2.3.2 BPML/WSCI	21
3.2.4 Process Ontologies Comparison.....	23
3.2.5 Business Data Ontologies	24
3.2.5.1 XML Business Standards	25
3.2.5.2 Other Business Standards	26
3.2.5.3 Core Component Technical Specification and Core Component Library	27
3.2.6 Business Data Ontologies Comparaison	27
3.2.7 Goal Ontologies	27
3.2.8 Goal Ontologies Comparaison	29

3.2.8.1	eCl@ss	29
3.2.8.2	UNSPSC	29
3.2.8.3	PGMT	30
4	TOOLS	31
4.1	Tools state of the art	31
4.1.1	Manual tools	31
4.1.1.1	Triana	31
4.1.1.2	BPWS4J	31
4.1.1.3	Self-Serv	32
4.1.2	Semi-automatic and automatic tools.....	33
4.1.2.1	BioOpera	33
4.1.2.2	Mind Swap - Web Service Composer	33
4.1.2.3	ServiceCom.....	34
4.1.2.4	Sword	35
4.1.2.5	Pegasus	35
4.1.2.6	IRS-II	36
4.2	Relevant for DIP	37
5	REQUIREMENTS.....	38
5.1	Service Description Requirements	38
5.2	Service Ontology Requirements	39
	CONCLUSION	41
	REFERENCES	42

1 INTRODUCTION

In simple terms, Web Services are software components accessible via the Web. By Service can be understood any type of functionality that software can deliver, ranging from mere information providers (such as stock quotes, weather forecasts, or news aggregation) to more elaborate ones that may have some impact in the real world (such as book sellers, plane ticket sellers, or e-banking), basically any functionality offered by the current Web can be envisioned as a Web Service. The big issue about Web Services resides in their capabilities for integration, allowing organizations to cooperate in a more efficient way. Web Services will change the Web from a static collection of information, to a dynamic place where different software components (business logics) can be easily integrated. This is a very ambitious goal, which at the moment cannot be achieved with the current state of the art technologies around Web Services. UDDI [9], WSDL [18] and SOAP [12] facilitate the means to advertise, describe, and invoke Services, using semi-formal natural language terms, but do not say anything about what Services can do, nor how they do it in a machine understandable and processable way. ebXML (electronic business XML) [25] and RosettaNet represent alternatives to UDDI, WSDL and SOAP, allowing to exchange and transport business documents over the Web using XML. They converge to the previously cited technologies, differing from them in the degree to which they recognize business process modeling as a core feature [46]. All these initiatives lack proper support for machine processable semantics and therefore human intervention is needed to actually discover, combine, and execute Web Services. The goal is to minimize any human intervention, so the integration of business logics can be done in a task-driven way and with the least support from the user side.

The Semantic Web in general and ontologies in particular are the right means to realize a dynamic Web. They provide the machine processable semantics that added on top of current Web Services realize the idea of the semantic Web Services. Semantic Web Services combine the Semantic Web and Web Service technologies. Semantic Web Services are defined as “*Self-contained, self-describing, semantically marked-up software resources that can be published, discovered, composed and executed across the Web in a task driven automatic way*” [3].

What really makes a difference with respect to traditional Web Services is that they are semantically marked-up. Such enhancement enables to fully realize the Web Service Usage Process: discovery, selection, composition, execution, monitoring, mediation, compensation and replacement at least in a semi-automatic way. To fully realize such process different ontologies, such as process, data and goal ontologies, which further describe the different relevant aspects of web services, need to be used in order to achieve fully interoperation and. On top of these particular ontologies some others can be found that try to provide a general framework of Web Service descriptions. They present different approaches to the problem of describing Services, representing a meta-layer for the data, process and goal ontologies. WSMO [71] and OWL-S [49] are counted among the most important of such ontologies.

The goal of this deliverable is to present the state of art on Service ontologies and Service description, and to state the requirements for the DIP project in this area.

The structure of the document is as follows: Section 2 introduces the Semantic Web Service Usage Process, describing each one of the steps required. Section 3 presents the state of the art on Service description and Service ontologies, based on the discussion

presented in this section, it introduces the concepts of Service description and Service ontologies, and the different approaches used to describe Services: syntactic vs. semantic. Section 4 provides an overview of the different tools that support the usage or make use of the Service ontologies presented in Section 3. Finally, Section 4 sketches the requirements for Service Description and Service Ontologies.

2 SEMANTIC WEB SERVICE USAGE PROCESS

The semantic markup of services enables the discovery, selection, composition, mediation, execution, monitoring, replacement and compensation of services to fulfill some user-defined task. The different steps required to achieve such user-defined task have been named Semantic Web Service Usage Process. It will allow to:

- Locate different Services suitable for a given task (discovery)
- Select the most appropriate Services among the available ones (selection)
- Combine Services to achieve a goal (composition)
- Solve mismatches (data, protocol, process) among the combined Services (mediation)
- Invoke Services following programmatic conventions (execution)
- Control the execution process (monitoring)
- Facilitate the replacement of Services by equivalent ones (replacement – or perhaps substitution would be a better term)
- Provide transactional support and undo or mitigate unwanted effects (compensation)

This section presents the Semantic Web Service Usage Process. It follows an example that shows the process of solving a simple mathematical equation.

2.1 Discovery

Service discovery can be defined as: *Location of Services that satisfy the service requester specification for a concrete task* [3].

Ideally, a task will be expressed using any natural language, and then translated to the ontology vocabulary suitable for the concrete application domain. Due to the fact that there will most likely be thousands of different ontologies for a concrete domain, – different service providers will express their business logics using different terms conventions, and different service requesters will express their requirements using different vocabularies– the appropriate support for domain knowledge mediation must be available. Semantically marked-up Services will be published in semantically enhanced service repositories where they can be easily located and their capabilities matched against the user’s requirements. In a nutshell these repositories are traditional UDDI registries augmented with a semantic layer on top that provides machine processable semantics for the Services registered.

As an example a service requester might say, “Locate all the Services that can solve mathematical equations.” A service discovery engine will then query/browse all available repositories to find Services that can achieve the given task. The list of available Services will probably be huge, so the user might impose some limitations, such as functional – what the service provides – and non-functional – execution time, cost, and so on – in order to get a set of Services that optimally suit the task on hand.

2.2 Selection

Service Selection can be defined as: *”Choice among discovered Services, based on functional and non-functional attributes, of those that better suit the user requirements”*

Heterogeneity is the key word in this part of the usage process of Services. Normally the number of discovered Services will be vast. Services are provided by different

vendors and with different characteristics, same from the functional and non-functional point of view. A common point of the Services that reach the selection phase is, that they can be used to provide at least a partial solution to our problem, of course, with different degree of fulfilment. From a non-functional point of view it is important to consider the following aspects [23]: performance, reliability, security, scalability, robustness, accuracy, transactional support/properties, trust, financial, network-related quality of service (QoS).

As an example after discovery two different services suitable for the subtraction task might have been found. One of them has very strong security requirements, is expensive and really reliable, while the other one is not so reliable, but is cheaper and is provided by our favourite software vendor. All these requirements might have been previously provided by the user, even before the discovery phase, allowing carrying out the web service selection task in a more or less automatic way. Also, the user could browse the whole list of selected Services and select the one that interests him the most.

Various techniques for automating the selection of services have been explored ([41],[52]). Usually some ranking algorithm is employed to order the services and then the first ranked service is automatically selected.

2.3 Composition

Service composition can be defined as: *“Assembly of Services based on their functional specifications in order to achieve a given task and to provide a higher order of functionality”* [3].

Once a list of available Web Services has been retrieved, it could happen that none of the available services completely fulfils the proposed task alone, or other – cheaper, faster, vendor-dependent – combinations of Services are preferred. If this is the case, some of the Services would need to be assembled, using programmatic conventions to accomplish the desired task. During this stage, Web Services are organized in different possible ways based on functional (pre-conditions – conditions that must hold before the service is executed –, and post-conditions – conditions that hold after the service execution), and non-functional (like processing time, cost) requirements.

As an example a service requester might say, “Compose available Services to solve the following set of mathematical operations $[(a * b) + c] - d$ ”. During the discovery phase different multiplication, addition and subtraction Services might have been found, each one of them with its particular functional and non-functional attributes. Let us suppose that some multiplication Services have been located, but they are all too slow and expensive, so we are not interested in using them. Instead we want to use additions to perform the multiplication. Such knowledge, i.e. the fact that multiple addition can realize multiplication, should be stated in some domain knowledge – characterization of relevant information for an specific area – in a way that the service composer can understand all different possibilities to solve the required set of operations. Different alternatives that lead towards the task accomplishment are then presented to the service requester who chooses among them the composition path that suits its needs the best.

2.4 Mediation

Service mediation can be defined as: *Arbitration of interacting Services in terms of domain knowledge used to describe them, data exchanged in the interaction (types used,*

and meaning of the information), business models of the different parties and protocol used in the communication [4].

Mediation is the magic maker of the usage process. It is very much interleaved in the whole usage process and relies on the use of ontologies to carry out its task.

Mediation plays a relevant role during the discovery and composition. It permits to align the different vocabularies used to describe services in regard to some domain knowledge, and therefore match the capabilities of the Service with the requirements of the (sub)task (discovery), and arrange services in a particular order also based on their capabilities (composition).

Once all the required services to solve a task are brought together following some programmatic conventions, interacting Services need to be aligned in terms of the:

- **Types and meaning of the exchanged information – Data Mediation:** Depending on the business model for which the services are deployed, different data types and domain knowledge might be used to encapsulate data and its meaning, which requires mediation to allow interoperation.
- **Protocol used – Protocol Mediation:** Different parties might use different message exchange patterns and protocols which need to be mediated in order to enable communication.
- **Business models – Process Mediation:** Services belonging to different business models require the appropriate process mediation in order to permit their cooperation.

In regard to the relation among the different types of mediation, it is interesting to point out that process mediation, in a broad sense, relies on data and protocol mediation, to successfully carry out its task. When taking a bottom-up approach firstly of all data types and its meaning are mediated, then the appropriate protocol mediation, if required is carried, and finally process mediation can be done.

As an example let us suppose that, “the addition service uses business model A, which first sends a message informing about the type and format of the result of the operation, and then a second one providing the result itself, while the subtraction Service uses business model B, which expects just one message with the type and the result of the operation. Moreover the addition service labels its output *result* while the subtraction expects *outcome*”. The proper alignment of the business models must be provided, probably by means of supplementary service that collects the messages sent by the addition service, transforms them into one message, and sends them to the subtraction one. In regard to the semantic mismatch in the name of the output, it could also be the task of this mediation service to understand both terminologies and send the expected parameter name.

2.5 Execution

Service execution can be defined as: *Invocation of a concrete set of services, arranged in a particular way following programmatic conventions that realizes a given task* [3].

When the available Services have been composed and the service requester has chosen the execution path that best suits its needs according to the non-functional requirements, the chosen set of Web Services is to be executed. Each Web Service specifies one or more signatures of its operations that allow its execution. The semantic mark-up

facilitates all the information regarding inputs required for service execution, and outputs returned once it has finished.

As an example a service requester might say, “Solve the following mathematical operations using the execution path which contains no multiplication Services: $(((3 * 3) + 4) - 1)$ ”. So what the user is actually saying is: solve $(((3 + 3 + 3) + 4) - 1)$. The addition service that realized the multiplication will be put into a loop, the result will be added to 4 and then 1 will be subtracted to obtain the result.

Prior to its execution, the service may impose some limitations on the service requester; such restrictions are expressed in terms of assumptions – conditions about the state of the world. A service provider may state that the service requester has to have a certain amount of money in the bank prior to the execution of the service, or whatever other requirements considered necessary as part of a concrete business logic. Once the composed service has been successfully executed, it might be registered in any of the available semantically enhanced repositories. By doing so, a new level of functionality is made available for reuse.

2.6 Execution Support

The execution of Services includes many different important aspects of the Web Service usage process that need to be cared for. The remaining of this section briefly comments on these aspects providing an overview of their most relevant characteristics

2.6.1 Monitoring

Service monitoring can be defined as: *“Supervision of the correct execution of services and dealing with exceptions thrown by composed services or the composition workflow itself”* [4].

It could happen that for different reasons the execution of a composed workflow of services fails at some point. Among the different causes are counted:

- **Service related issues:** The server hosting the service crashes, or the service throws an exception that does not allow continuing execution.
- **Network related issues:** Network crashes disallowing communication.
- **Composition related issues:** The composition workflow throws an exception that does not allow continuing execution, or it is not well designed so it reaches a dead end.

As an example let us suppose, “that the first of the addition services which are put on the loop to add 3 to itself three times, at some point in time of the execution throws an exception reporting that the data type given is not appropriate. Additionally during the execution of the second addition service of the loop the server that hosts it crashes”. If this is the case, it will be the task of the monitoring to detect the data type exception, allowing the required mediation. Such mediation will probably be a new service which is placed in the workflow where required, and that takes care of the data type transformation. In regard to the host crash the problem will be detected and the appropriate compensation and replacement strategy applied to allow the continuation of the execution.

2.6.2 Compensation

Service compensation can be defined as: *Reparation or undoing of the operations involved in a transaction in case of failure, so unwanted side effects are rolled back or at least mitigated if complete undo is not possible,*

When the execution of services must follow a transactional model, either it is fully completed or not at all. There might be situations where the whole transaction can not be fully finished. If this is the case, some of the effects of the previously executed services might not be undoable. In such situations compensation mechanisms must be applied in order to either mitigate the unwanted effects, usually by executing another operations whose side effect compensate the side effects of the unwanted one.

As an example let us suppose that, “the whole mathematical operation must be executed as a transaction, but with the particularity that once any of the services (addition, subtractions) is executed, the cost of the execution of the Service is taken from our bank account.” If this is the case, and the third addition fails to finish completely, some money will have been withdrawn from our bank, but the result of the operation will not have been delivered to us. The compensation strategy in this particular situation will be to return to our bank account the money taken, undoing this way the unwanted effects of a failure within a transactional operation.

2.6.3 Replacement

Service replacement can be defined as: *Substitution of Services by equivalent ones, which solely or in combination can realize the same functionality as the replaced one, in case of failure while execution.*

In case that any of the constituents of the execution workflow fails to accomplish its goal (e.g., network breaks down, service provider’s server breaks down, etc), a recovery procedure must be applied to replace the failed service with another one or set of Services capable of finishing the work.

As an example let us suppose that, “the server which hosts the subtraction service, crashes while invoking it, so the service can not be executed.” If this is the case, a new subtraction service, probably among the list retrieved during the discovery phase must be put in place to overcome the malfunctioning, and allow the continuation of the normal execution. This is not a trivial case, and different aspects should be taken under consideration. First, in case none of the retrieved services during the discovery phase fits the purpose or the non-functional attributes do not fit the requirements of the service requester, a new discovery process should be carried out; second, a common execution environment (context) should be provided where the current execution state is kept; third, the new service must be appropriately mediated in order to allow it to fit in the general execution workflow; and fourth, in case that not only one service fulfils the required functionality but instead a set of them do, the required composition process should be carried out.

The concepts of compensation and replacement are orthogonal, i.e. in case the execution of a service fails, there might be a need for compensation, in case the service produce side effects. On the other hand, if a service fails, after compensating its side effects, we normally consider the replacement of this service. But such a link between the two operations is not necessary, i.e. there can be replacement without compensation (e.g. we

are sure that there were no side effects) or compensation without replacement (we abort the whole process, if one service fails, and do not seek replacement).

3 STATE OF THE ART

This section presents the state of the art in Service Description and Service Ontologies. It is organized as follows: Section 3.1 presents the concepts behind Service description, the different approaches to describe services (syntactic vs. semantic), it explains the advantages of describing services using ontologies, the ideas that motivate this approach and finally roughly sketches the different ontology languages available. Section 3.2 introduces the state of the art on epistemological ontologies to describe Services, essentially OWL-S and WSMO.

3.1 Service description

The Semantic Web is the next generation of the WWW where information has machine-processable and machine-understandable semantics. This technology will bring structure to the meaningful content of Web pages, being not a separate Web, but an augmentation of the current one, where information is given a well-defined meaning. The Semantic Web will include millions of small and specialized reasoning services that will provide support for the achievement of tasks based on accessible information.

One of the first one to come up with the idea of the Semantic Web was the inventor of the current WWW, Tim Berners-Lee. He envisioned a Web where knowledge is stored on the meaning or content of Web resources through the use of machine-processable meta-data. The Semantic Web can be defined as “*an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in co-operation*” [11].

The core concept behind the Semantic Web is the representation of data in a machine interpretable way. Ontologies facilitate the means to realize such representation. Ontologies represent formal and consensual specifications of conceptualizations, which provide a shared and common understanding of a domain as data and information machine-processable semantics, which can be communicated among agents (organizations, individuals, and software) [28]. Many definitions of “*ontology*” have been given during the last years. One that really fits the purpose in the computer science field is the one give by [31]: “*An ontology is a formal, explicit specification of a shared conceptualization*”.

Ontologies bring together two essential aspects that help to further develop the Web. On the one hand they provide (a) *machine processability* by defining formal semantics for information making computers able to process it; on the other, they allow (b) *machine-human understanding* due to their ability to specify real-world semantics that permit to link machine processable content with human meaning using a consensual terminology as connecting element [28].

Regarding Web Services, machine processability enabled by ontologies represents their most valuable contribution. They offer the necessary means to describe the capabilities of a concrete Web Service in a shared vocabulary that can be understood by every service requester. Such a shared functionality description is the key element towards task-driven Web Service discovery, invocation, composition, execution, monitoring, mediation and compensation of inter-organization business logics. It will allow to (1) locate different services which solely or in combination with others will provide the means to solve given task, (2) combine services to achieve a goal, and (3) facilitate the

replacement of such services by equivalent ones, that solely or in combination can realize the same functionality, e.g. in case of failure while execution.

As ontologies are built (models of how things work) it will be possible to use them as common languages to describe Web Services and the payloads they contain in much more detail [21].

Ontology languages must be able to describe and organize knowledge in a machine understandable way. However, organizing knowledge requires the facilities of a logical formalism which can deal with temporal, spatial, epistemic, and inferential aspects of knowledge. The different available ontology languages must provide these inference services, making them much more than just simple data storage and retrieval systems. Among the most relevant ones are counted: XML(s) [26], RDF(s) ([60],[13]), DAML+OIL [4], and OWL ([20],[8]).

XML allows structuring the content Web-pages as labelled trees. It constitutes the representation syntax used by the some of the most relevant Semantic Web Languages. XML schema is built on XML. It provides the means for defining constraints on valid XML documents. They have the same purpose as DTDs.

Although XML provides much more space for users to define their own tags, it fails to define the semantics in the machine understandable and processable way. RDF comes to fill up the hole. RDF and RDF(s) provide the way to represent the semantics of information on the Web. They allow the representation of concepts, taxonomies of concepts and binary relations. It is also intended to provide mechanisms to explicitly represent services, processes and business models. They are the first web language capable to define the ontology in a limited way due to the lack of the necessary rich concept forming operators. However, richer languages can be built on top of them (such as OIL, DAML+OIL, and OWL).

DAML+OIL is an ontology language specifically designed for the Semantic Web. It exploits existing Web standards (XML and RDF), adding the ontological primitives of object oriented and frame-based systems, and the formal rigor of expressive description logic. As an ontology language, DAML+OIL is designed to describe the *structure* of a domain. DAML+OIL is based on SHOIQ(D). It describes the structure of the domain being in terms of *classes* and *properties*, and the set of *axioms* that assert characteristics of these classes and properties.

The OWL which stands for Ontology Web Language is actually the standard language for representing knowledge on the Web. This language was designed to be used by applications that need to process the content of information instead of just presenting information to humans. OWL is mainly based on DAML+OIL and therefore the main features of OWL are very similar to those of OIL. OWL follows a layered approach. There are three OWL sub-languages: OWL-Lite, OWL DL and OWL Full. Starting from the lower layer, OWL-Lite, up to the upper layer, OWL Full, each sub-language is enriched in syntax and semantic. OWL-Lite gathers the most common features of OWL. It limits expressivity with the aim to provide a minimal subset of the language features. It is intended for the users that only need to create class taxonomies and simple constraints. OWL DL extends OWL-Lite. It offers maximal expressiveness, while maintaining tractability. The last language, OWL Full contains the complete vocabulary and constructs offering very high expressiveness but without tractability.

3.1.1 Syntactic vs. Semantic service description

Web Service description is a two fold matter, since it includes a syntactic and a semantic depiction of the functionality at least in the Semantic Web field [57]. **Syntactic** information is concerned with the implementation aspects of a Web Service whereas **semantic** information is concerned with the meaning of data exchanged, with the messages exchanged and also with process model and protocols of a Web Service. In this section we provide a comparison of the two approaches to describe services. We start by describing how Web Services can be described in syntactic way, than we underline the drawbacks of this approach and the need of semantic description of Web Services.

3.1.1.1 Syntactic description

When we describe a Web Service at a syntactic level we are concerned only with the implementation aspects of it. The description is given only in terms of input/output data types and operations that the service performs but there is no meaning associated with the inputs, outputs and operations. For a syntactic description of Web Services one can use the Web Services Description Language (WSDL) [18]. WSDL is XML based and allows description of Services distributed over the Web as a set of endpoints operating on messages containing either document-oriented or procedural-oriented information. What WSDL lacks is the shared semantic which is necessary in order for machines to understand what a service does. This will permit the semi-automatic execution of the different parts of the Web Service Usage Process..

3.1.1.2 Semantic description

Specifying a service solely in terms of its input/output data types only offers a very incomplete picture. It is important to describe *what* the service *does* and *how* does it do it. Somehow we need to incorporate semantics in the description of the service. Ontologies are the solution to this problem. Semantic description, using ontologies, will enable agents to understand it and implicitly to improve the Web Service Usage Process. When we refer to the semantics associated with a Web Services we have to distinguish between different levels of description. Depending on whether the service requestor is an end-user, a developer or a machine, different kinds of service description are required. For the end-user, only semantic description is needed whereas developers or machines need both semantic and syntactic information. A complete description of Web Service, syntactically and semantically, will help solving user-defined task with minimum human user intervention. At this moment Web Services require human intervention all over their usage processes. The combination of Web Services and the Semantic Web, known as Semantic Web Services will provide real dynamic and distributed computation using heterogeneous sources [37]. The vision of Semantic Web Services will allow to fully realizing the semantic usage process of Web Services as presented in Section 2.

A service description needs to provide constructs that enable the description of functional, non-functional and behavioural information in a semantic as well as in a syntactic form. The description language should also support inferences on descriptions. Using these descriptions intelligent agents can reason and compare all available alternatives for solving a concrete task. When we require a specific service we expect our request to be matched against any Web Service that provides that functionality. Given some utility function one or more of the matched Web Services are selected to

fulfil the request. We think that this will be possible only if Web Services are described at a syntactic level as well as a semantic level.

3.2 Service Ontology

In order to provide a full description of the particular aspects of Web Services, different ontologies are required, each one of them taking care of a particular part of the problem.

Ontologies are required to describe:

- Functional and non-functional aspects of Services
- Process Model
- Business Data
- Goals

This section presents a review of the different available technologies that address each one of the issues presented.

3.2.1 Epistemological Ontologies

This section presents the state of the art of epistemological ontologies to describe services. Epistemological ontologies are meta-ontologies defining the modeling constructs used in other ontologies providing a high level model of the world. Epistemological ontologies are also referred as Upper level ontologies. In general they represent ontologies to describe services – in this context – so the term Service Ontology is also applicable.

We provide a short overview of WSDL, even though it cannot be considered an ontological approach, since it describes only the syntax, and does not contain any semantic constructs.

3.2.1.1 WSDL

Even though the Web Service Description Language (WSDL) [18] is not an ontology language it has been included here since it is based on XML, and also for the sake of completeness of the report and to support further discussion about syntactic and semantic description of services. Since it provides a description of Service, which is the final aim of the initiatives presented here, we include into this section.

WSDL is an XML-based language for describing Web services. A WSDL description is comprised of an XML document which describes a Web Service. Such a description specifies the location of the Service, the operations that it exposes, the data expected to carry the exposed operations, the results that are delivered after the execution ends, typically using XML schema data types, and the communication protocols and transport it supports.

WSDL uses the concept of network endpoints (ports) to define services. A Service description is divided in two parts: the interface definition and the end point definition. The interface comprises types, messages, port types, and operations, while the end point includes the description of the binding, service and port.

- Types: Individual data types describing messages
- Messages: Used as a communication mechanism by the described Web Service

- **Port types:** Represents the logical grouping of operation. It defines the Web Service, in terms of the operations that can be performed, and the messages that are involved to carry them out.
- **Operations:** Used to perform actions on the exchanged data. They allow the Web Service to interpret the received data, and to determine what, if any is the data to be returned.
- **Binding:** Specifies the message format and protocol details for operations and messages defined by a particular PortType.
- **Service:** It encloses one or more PortTypes
- **Endpoints:** used to expose a set of operation or PortTypes

3.2.1.2 OWL-S

OWL-S [49] is an epistemological ontology that provides the means to mark-up Web Services describing their capabilities and properties in unambiguous, computer interpretable way. The aim of OWL-S is to make Web services computer interpretable enabling their automated use. OWL-S sits on top of WSDL in order to facilitate the syntactic description of the service. OWL-S makes use of WSDL to describe knowledge about a service in terms of what the service does –represented by messages exchanged across the wire between service participants –, why does it do it, and how it does it. From version 0.9 OWL-S is built upon OWL (Web Ontology Language), while previous releases were based on DAML+OIL.

OWL-S ontology allows the definition of knowledge to state what the service requires from agents, what provides them with, and how does it do it. To answer such questions it uses three different elements:

Service Profile, which facilitates information about the service and its provider to enable its discovery. It presents a public high-level description of the service that states its intended purpose in terms of: (1) service description, information presented to the user (when browsing service registries), about the service provider and the service itself, that helps to clarify whether the service meets concrete needs and constraints such as security, and quality requirements; (2) functional behavior, description of duties of the service; and (3) functional attributes, additional service information such as time response, accuracy, cost or classification of the service.

Service Model allows a more detailed analysis of the matching among service functionalities and user needs, enabling service composition, activity coordination and execution monitoring. It permits the description of the functionalities of a service as a process, detailing control and data flow structures. The Service Model includes two main elements, namely: (1) Process Ontology, which describes the service in terms of inputs – information necessary for process execution –, outputs – information that the process provides –, preconditions – conditions that must hold prior the process execution –, effects – changes in the world as a result of the execution of the service –, and if necessary component sub-processes; and (2) Process Control Ontology that describes process in terms of its state – activation, execution and completion. Processes can have any number of inputs, outputs, preconditions and effects. Both outputs and effects can have associated conditions.

Service Grounding, which specifies how communications among participants are to be carried on and how the service will be invoked. The grounding is defined as mapping from abstract to concrete realization of service's descriptions in terms of inputs and outputs of the atomic process, realized by means of WSDL and SOAP. The service grounding provides a mapping between OWL-S atomic processes and their parameters, and WSDL operations and its messages.

Essentially OWL-S is an epistemological ontology for describing functional and non functional parameters of Web Services. It allows to semantically enriching Web Services, specifying what they do and how they do it, and so, it provides the means to automatically discover compose and execute them.

3.2.1.3 WSMO

The Web Service Modelling Ontology (WSMO) [71] is a formal ontology and language for describing the various aspects related to Semantic Web Services. It represents the backbone for the development of:

- Web Service Modelling Language (WSML)
- Web Service Modelling Execution Environment (WSMX)

WSMO takes a layered approach that starts with a basic WSMO, called WSMO-Lite, continues with a mature set of concepts called WSMO-Standard, and ends with a full ontology called WSMO-Full.

The WSMO language is suitable for modeling Web Services, aiming at providing the facilities required by its usage process namely, discovery, invocation, composition, execution, monitoring, mediation and replacement (in WSMO terms: compensation).

The objective of WSMO and its surrounding efforts is to define a coherent technology for Semantic Web Services. Means for semi-automated discovery, composition, and execution of Web Services shall be based on logical inference-mechanisms, because of the well-known competences of suchlike techniques, and its appropriateness for the purpose. In order to allow suitable logic-based reasoning on Semantic Web Service, the description language has to provide reasonable expressiveness for describing relevant aspects of the Services, together with well defined formal semantics that support effective reasoning. WSMO applies F-Logic as the underlying logical formalism, since it provides a standard model theory, it is a full first order logic language, provides second order syntax while staying in the first order logic semantics, and it has a minimal model semantics. A further benefit of F-Logic is that implemented inference engines are already available.

WSMO Principles

Every ontology design is based on principles that can be abstracted from the domain that is going to formally represent. In the following the principles of the WSMO are outlined in order to provide a guide to the ontology design decisions.

- **Decoupling and Mediation.** In an e-commerce environment applications should be as disaggregated as much as possible, hiding internal business intelligence from public access and carrying communication among processes by means of public message exchange protocols. In order to achieve this objectives the decoupling of the different components that build an e-commerce application is a must.

Scalable communication should allow anybody to speak with everybody, in order to fulfil this n:m communication style, terminologies should be aligned and interaction styles should be intervened. To better achieve these principles, a mediation approach, which allows mapping different business logics, together with the ability to establish the difference between public and private processes of e-Services, should be taken.

- **Interface vs. Implementation.** When formally describing interacting entities it is important to recognize the differences among the interface of an interacting entity, its internal implementation, as well as its externally visible and internal behaviour. The WSMO specification focuses on the external visible behaviour only.
- **Peer-to-peer vs. Client/Server.** Interacting entities, sometimes called agents, are characterized in general as one entity interacting with one or more other entities. In the simplest case, two entities are interacting. In the context of Semantic Web Services, the interaction is between equal partners, i.e., one entity sending a message and the other one receiving the message are equal in their level of control over the other entity. This is called a peer-to-peer approach in contrast to a client/server approach, where the client drives the interaction as the controlling entity. The WSMO shall recognize the fact that all entities are equal in their control structure leading in the general case to conversations amongst equal partners.
- **Execution semantics.** The WSMO is a representation of the domain of Semantic Web Services. Since it formally describes the interactions of agents, a formal execution semantics has to be defined in order to uniquely specify the execution behaviour at runtime. The concepts of WSMO shall have formal execution semantics to ensure a consistent execution model.

WSMO Elements

WSMO defines the modelling elements for describing several aspects of Semantic Web Services. The conceptual grounding of WSMO is based on the Web Service Modeling Framework (WSMF) [28], wherein four main components are defined as shown in figure 1.

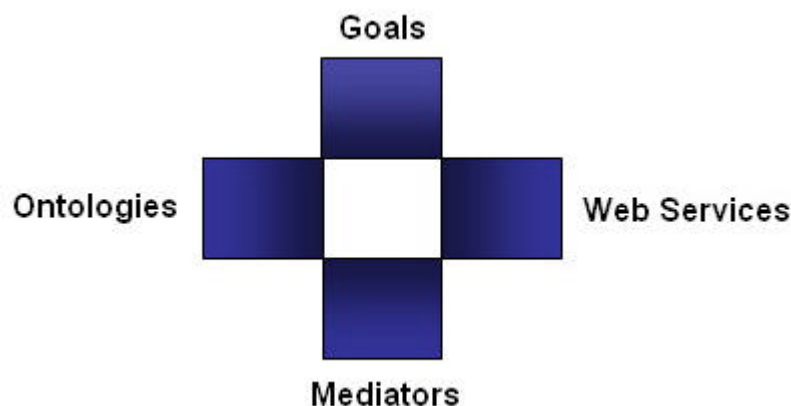


Figure 1. WSMO elements

- **Ontologies** provide the formal semantics to the information used by all other components.
- **Goals** specify objectives that a client may have when consulting a Web Service.
- **Web Services** represent the functional part which must be semantically described in order to allow their semi-automated use.
- **Mediators** used as connectors provide interoperability facilities among the rest of components.

Ontologies represent the key element in WSMO. They serve a twofold purpose, first they define the formal semantics of information, and second allow to link machine and human terminologies. They play a key role in the WSMO specification since they give meaning to the different elements, providing the terminology that can be aligned to allow their interoperability from a semantic point of view. WSMO specifies the following constituents as part of the description of an ontology: non-functional properties, used mediators, axioms, concepts, relations and instances.

Ontologies are used to: (1) express goals in a machine processable and understandable language, i.e. buy a plane ticket; (2) they permit to enhance Web Services so they can be matched against goals, i.e. Web Services provided by on-line travel agencies; and (3) interconnect the different elements with each other by means of mediators, i.e. services and goals using different terminology.

Goals represent the objectives to be fulfilled when consulting a Web Service. The WSMO definition of goal is restricted to post-condition, effects, non-functional properties and used mediators, leaving aside pre-conditions and assumptions. Goals provide the means to express high level description of a concrete task.

The Web Service element in WSMO represents the atomic piece of functionality that can be reused to build more complex ones. In this sense it is straight forward to assimilate Web Services to software components that are independently assembled, deployed and plugged in any context that requires its functionality, as it stems from component-based software engineering (CBSE) field. In the WSMO specification Web Service are described, by means of a capability, interfaces, used mediators and non functional properties. A Service is described by one and only one capability and multiple interfaces.

Mediators allow overcoming functional and non-functional mismatches among the different elements that build the WSMO specification. Currently the specification defines four different types of mediators, which are classified in two main classes: refiners and bridges. While refiners are used to define new components as a specialization of an existing one, bridges help to overcome interoperability problems by enabling components to interact with each other. Among the four types of mediators that currently build the specification, ggMediators and ooMediators are refiners, while wgMediators and wwMediators are counted as bridges. In the general case WSMO defines mediators by means of non-functional properties, source component target component and mediation service, where source and target component can be a mediator, a Web Service, and ontology or a goal.

In our particular example, mediators could be easily used to arbitrate among goals and the capabilities of a given Web Service, probably expressed using different ontology

languages and terminology. Such would be the case of ooMediators as part of a wgMediator.

3.2.1.4 SWSL

One of the important initiatives in Semantic Web Services area is Semantic Web Services Initiative (SWSI) [62]. The members of this initiative have identified two major aspects regarding Semantic Web Services: the *architecture* to support Semantic Web Services and the *language* to specify information about Semantic Web Services. Two committees have been created: SWSA Committee and SWSL Committee. Architectural aspects of Semantic Web Services are the main focus of SWSA Committee which mission is to develop architectural and protocol abstractions forming a reference architecture to support Semantic Web Service technologies. Language aspects are the concern of SWSL Committee which objectives are to develop a language for the declarative specification of this semantic information and to develop standardized ways of conceptualizing and organizing semantic information about services. The language developed by SWSL Committee is called Semantic Web Services Language (SWSL).

SWSL is a formal language that allows rich declarative specification of a wide variety of information about Web Services, which will support automation of a broad spectrum of activities related to Web Services, such as discovery, selection, composition, negotiation and contracting, invocation, monitoring of progress, and recovery from failure. At present the work done in SWSL concerns only the identification of the requirements. Lot of other things have to be done in order to make SWSL an existing language for Semantic Web services specification.

SWSL is intending to cover, all the important aspects related to Semantic Web Service specification. In the end SWSL should:

- Enable automation and dynamism in all aspects of Web Services.
- Be extensible and allow an incremental exploitation.
- Support the construction of powerful tools and methodologies.

To develop a language that provides all the above facilities is not an easy task. Anyway some major steps in SWSL development have been identified so far. These are:

1. Identify requirements for Web Service description language(s).
2. Specification of a formal service model for semantic Web Services.
3. Establish the relationship between Web Service standards and Semantic Web standards.
4. Exploring techniques for implementing and applying parts of the languages. For example, techniques for inferencing with service descriptions.
5. Dissemination.

SWSL is built upon the OWL-S and therefore the main features of SWSL are very similar to those of OWL-S. Using OWL-S as reference point, SWSL can be used to specify information about Web Services, but not in a complete manner. Domains like Concurrent Transaction Logic, AI Planning, and Process Specification could help to create a complete specification of a Web Service. Concurrent Transaction Logic [19] is a deductive database language that integrates queries, updates, and transaction composition in a simple logical framework. Concurrent Transaction Logic and also other logics can help in modelling processes. From this point of view SWSL has another

input from Process Specification Language. The Process Specification Language identifies, formally defines, and structures the semantic concepts intrinsic to the capture and exchange of discrete manufacturing process information. AI Planning is another area that could help Semantic Web Services specification. For example a request from a client can not be satisfied by a single service, but there could be some combinations of existing services that solve user's request. AI planning techniques are solutions for automate composition of Web services [73].

Currently SWSL is in the requirements phase, which represents the first step in the SWSL development. After the requirements gathering the project suffers of a lack of progress. Among the causes can be counted: problems with the conceptual model definition, problems with the integration with the Web standards and Semantic Web standards, dissemination and other important problems are waiting for a solution.

3.2.2 Epistemological Ontologies Comparaison³

Even though the purpose of OWL-S and WSM(O/L/X) is similar, both aim at providing an ontology for describing Web Services that enables users and software agents to realize the Web Service Usage Process, the grounding of both approaches slightly differs. While WSMO is based on a number of explicit principles of the domain that the ontology formally represents, OWL-S lacks of these explicit and well-established grounding principles [36]. Also the Semantic Web Services Language (SWSL) Committee aims at identifying and developing a technology that will provide a firm, long-term foundation for Web Services, currently in an early stage of development.

- **Discovery.** OWL-S provides/defines/uses the Service Profile which provides information about the service and its provider in terms of service description, functional behaviour and functional attributes to allow its discovery. WSM(O/L/X) makes use of goals to define the objectives that a client may have when consulting a Web Service allowing Service discovery. In particular WSMO restricts the definition of goals to post-conditions and effects. Post-conditions define the state of the desired information space. Effects describe the desired state of the world after the execution of the Web Service. WSMX provides the required inferencing support to discover Web Services. In SWSL discovery will be made possible primarily by a set of semantic descriptions that are expressed in a formal logical language [62].
- **Selection.** Neither WSM(O/L/X), OWL-S, nor SWSL foresee mechanisms to rank services according to functional and non-functional aspects and such enable selection.
- **Mediation.** WSM(O/L/X) supports scalable mediation based on mediators. It offers four types of mediators. OOMediators to link 2 ontologies, GGMediators to link 2 goals, WWMediators to link 2 Web Service and WGMediators to link a Web Service and a Goal, or more precisely, to link the Capability of a Web Service and a Goal. WSMX will provide an implementation to support Web Service mediation. OWL-S does not specify any particular form of mediation. SWSL does not make a clear distinction between different types of mediation, like WSM(O/L/X) does.

³ The information used in the comparison in regard to SWSL is based on the requirements of the language, since no implementation is yet available.

- **Composition.** OWL-S makes use of Service Profile to allow the matching among service's functionalities and objectives, enabling service composition. In WSM(O/L/X), the description of the Web Services in terms of capabilities and interfaces covers the same aspect as the Service Model in OWL-S, permitting the specification of the functionality of the service. The capability of a Web Service defines its functionality in terms of preconditions and post-conditions, assumptions and effects. The Interface provides further information on how the functionality can be achieved. It describes the communication pattern that allows consuming the functionality: choreography; and/or how the overall functionality is achieved by means of cooperation of different service providers: orchestration. WSM(O/L/X) supports both static and dynamic composition while OWL-S counts with limited support for dynamic composition [36]. SWSL proposes two approaches for Web services composition. The first approach is a manual one. Users can create composite processes "by hand". The second approach is an automatic approach, where Web services are composed using AI planning algorithms.
- **Execution.** While WSM(O/L/X) counts on WSMX which provides an execution platform for Semantic Web Services, OWL-S also has an execution environment called OWL-S VM. Execution in SWSL is also not clearly defined.
- **Monitoring.** OWL-S service model by means of its Process Control Ontology allows the monitoring of the execution of services describing the process in terms of its state, namely activation, execution and completion. WSM(O/L/X) by means of WSMX, its execution environment, should provide the required support for monitoring the execution of Web Services. SWSL lacks such feature.
- **Replacement:** None of the presented initiatives currently supports replacement. WSMO will support this feature in its execution environment WSMX.
- **Compensation.** OWL-S currently does not specify any particular form of compensation. Even though it is not formally stated, WSMX should support Web Service compensation. SWSL currently lacks compensation.

In a nutshell the WSM(O/L/X) seems to try to cover a wider spectrum of the usage process of Semantic Web Services, providing a clearer direction in discovery, composition, mediation and execution. In regard to monitoring, OWL-S offers a better defined path, while none of the efforts seem to have put special interest in compensation. SWSL is in early specification stage compared to the other initiatives.

3.2.3 Process Ontologies

At present, apart from OWL-S Process model detailed in Section 3.2.1.2, there are no process ontologies available, i.e. there are no process standards available that are modeled using an ontology language like OWL or RDF. Nevertheless there are a number of so called standards available, which allow to model processes. Among them are counted XLANG [66] and WSFL [40], that merged into BPEL4WS [14] or BPML/WSCI ([2],[55]) no longer maintained. These initiatives allow generating XML-based documents at design time that contain the workflow specification of the Services

taking part in a concrete interaction. Once the workflow has been generated it is executed by an engine to actually realize the interaction.

In the following some of two of the most relevant so called standards will be discussed in order to give an overview of what is available.

3.2.3.1 BPEL4WS

The Business Process Execution Language for Web Services (BPEL4WS) [14] has its origins in Microsoft XLANG [66] and IBM WSFL [40]. It represents an important initiative towards the generation of workflow specifications that try to model the behavior of Web Services taking part in a business process interaction. It represents compositions where the flow of the process and the bindings between services can be dynamically determined. BPEL4WS uses an XML-based grammar for describing the coordination logics of Web Services participating in a process flow.

BPEL4WS presents a process model layered on top of WSDL, where both the process model and its partners are represented as a services using WSDL. It includes transactions support based on WS-Transaction [16], and exceptions handling mechanisms built on top of WS-Coordination [15]. BPEL4WS differentiates among two different kinds of business processes:

- **Executable processes.** Permit to model the behavior of a participant in an interaction, without making any separation among public and private details of the business process; they provide orchestration facilities;
- **Abstract process.** Model the visible behavior of the parties based on its message exchange, without revealing its internal details. They facilitate choreography facilities, meant to couple Web Service interface definitions with behavioral specifications.

The BPEL4WS specification differentiates between two types of activities: **basic** and **structured**. Basic activities represent the simplest form of interaction with a service, they describe individual operations that manipulate passing data, or handle exceptions. In contrast structured activities specify the order in which a collection of basic activities take place. Structured activities use the usual primitives (while, switch, sequence, etc) to specify the execution flow for basic activities. It is possible to abstract structure activities as the underlying programming logic for BPEL4WS [56]. Activities also represent the basic unit that can be grouped within a transaction by means of scopes. Scopes have a one-to-one mapping with database transactions in the sense that all activities enclosed within a concrete scope should either all complete or none of them complete, in case of failure the necessary rollback mechanisms are put to work. Scopes also allow the definition of compensation mechanisms.

In a nutshell, BPEL4WS represents an initiative towards the generation of workflow specifications using an XML-based grammar in a Web Service environment, where the flow of the process and the bindings between services are known a priori. It includes transaction and exception handling support, and uses the concept of scope as the grouping unit of activities that take part in the same transaction.

BPEL4WS Engines

Several BPEL engines have been developed, among them:

- IBM BPWS4J (IBM Business Process Execution Language for Web Services Java™ Run Time)
- Collaxa BPEL Orchestration server
- OpenStorm ChoreoServer

IBM Business Process Execution Language for Web Services Java™ Run Time (*IBM BPWS4J*) [14] includes: a platform upon which business processes written using the Business Process Execution Language for Web Services (BPEL4WS) can be executed; a set of samples demonstrating the use of BPEL4WS; and a tool that validates BPEL4WS documents. The BPWS4J platform also includes an Eclipse plug-in that provides a simple editor for creating and modifying BPEL4WS files.

Collaxa BPEL Orchestration server [17] is a complete BPEL engine running on top of a J2EE application server. It helps Java developers to publish synchronous and asynchronous Web Services and compose them into reliable and transactional business flows. It helps in building composite Web Services, integrate asynchronous services, coordinate multi-step business processes, publish BPEL processes as Web services, catch exceptions, coordinate compensating business transactions, define manual activities, track execution (audit trail), test/debug BPEL processes, side-by-side versioning and monitoring.

OpenStorm ChoreoServer [48] is a BPEL engine running on top of the .NET platform. The Service Orchestrator is a full-featured web service orchestration platform. It includes development environment (Orchestrator Studio) as well as a runtime environment (Orchestrator Server):

- The Studio: A GUI for creating Web Service orchestrations.
- The Server: A fully compliant BPEL 1.1 engine for executing service compositions.

3.2.3.2 BPML/WSCI

The Business Process Management Language (BPML) [1], is a meta-language for modeling business processes in terms of control flow, data flow and event flow, to execute them later using Web Services.

The Web Services Choreography Interface (WSCI) specification defines an XML-based language for web services collaboration. It describes the overall choreography in terms of messages exchanged between web services that participate in a business process. WSCI defines the public (observable) interaction in terms of messages. It is not in the scope of WSCI the definition of executable business processes (control flow, data flow and event flow).

BPML and WSCI can work in combination and actually are complementary efforts. While WSCI describes public interactions and choreographies between services, BPML focuses on describing the private part. Both efforts share the same execution model and their syntaxes are similar. Roughly speaking BPML takes care of the definition of the

business processes that will be executed using Web Services, while WSCI specifies the interoperation between such services.

In the following paragraphs a more detailed look is taken into the features of both standards. The BPML description is based on [1] while WSCI one is based on [2] and [55].

BPML is a meta-language for modeling collaborative and transactional business processes, based on the concept of transactional finite-state machine, which allows the definition of abstract and executable processes. It relies on an XML Schema based grammar that enables the persistence and interchange of definitions across heterogeneous systems. BPML represents business processes based on control flow, data flow, and event flow, adding orthogonal design capabilities for business rules, security roles, and transaction contexts independently of the synchronous or asynchronous nature of the transaction.

In order to define business processes BPML uses five different elements, namely: (1) *activities*, components that perform specific functions, which can be either simple or complex; (2) *processes*, particular types of complex activities that define their own context; (3) *contexts*, which define the environment for the execution of related activities; (4) *properties*, which enable the information exchange within a context; (5) *signals*, which allow the coordination of activities that are being executed in the same context.

BPML uses a modeling language to define business processes in a language independent fashion, placing such description on top of the BPML XML Schema layer. It makes use of primitives for sending, receiving, and invoking services, together with the typical programmatic conventions to handle conditional choices, sequential and parallel activities, joins, and loops. On top of this constructs for task scheduling and XML exchanges between various participants are added. The language was designed to manage long-running processes, supporting persistence in a transparent manner. Finally the language supports recursive composition in order to facilitate the means to compose sub-processes into larger business process.

Regarding transactional support and exception handling mechanisms, it is worth to mention that BPML sustains both characteristics, same for short and long running transactions, using scoping based compensation techniques to compensate processes within a defined scope, in the case of complex transactions. BPML also supports process and transaction nesting, exception handling, and timeout constraints for activities defined within the process.

In a nutshell, BPML provides means for defining business processes involving different parties and Web Services, with support for persistence, compensation or exception handling. This description covers the definition of distributed business processes, including how the different partners and services collaborate. Nevertheless, BPML does not cover automation support, as the business process and the partners and services involved have to be specified at design time and cannot change dynamically, mainly due to the lack of explicit semantics.

WSCI is an XML-based interface description language that defines, using messages, the overall choreography, – temporal and/or logical dependencies –, of a Web Service taking part in an interaction. WSCI defines the public (observable) interaction in terms of messages, not paying any attention to the definition of executable business processes.

There is a direct correspondence among WSCI and WSDL. Each WSCI unit of work corresponds with a specific WSDL operation which allows WSCI to specify the choreography among operations. Essentially WSDL describes the entry points of each service while, WSCI describes interactions among WSDL operations [55].

In order to provide its functionality, WSCI uses the following concepts: (1) **Interface**, models the externally observable behavior of a Web Service in a choreographed, long-lasting and stateful message exchange with one or more other services; (2) **activities**, basic unit of behavior of choreographed Web Services, they can be either atomic or complex, composed of other activities; (3) **processes**, identifiable portions of behavior; WSCI differentiates two types of processes, *top-level processes*, defined at the interface level, and *nested processes* defined within complex activities; (4) **properties**, which are used to reference a value within an interface definition; (5) **context**, which defines the environment in which a set of activities is executed; there exists a 1 to 1 mapping among set of activities and contexts; (6) **message correlation**, which allows defining how the service should manage multiple conversations with the same or different partners; it describes how conversations are structured and which properties must be exchanged to retain the semantic consistency of the conversation; (7) **exceptional behavior**, alternative patterns of behavior exhibited by a Web Service at a given point in a choreography, it associates exception and activities that the Web Service will perform in response to those exceptions; (8) **transactional behavior**, definition of operations that should be performed in a transactional way within a particular context; (9) **global model**, allows describing a multi-participant view of the overall message exchange in contrast to the interface that allows the same purpose but just for one participant.

WSCI provides an answer for different issues related with the interoperation of services such as message sequencing, relation among incoming and/or outgoing messages or start and end of sequences among others.

Essentially WSCI permits to describe the dynamic interface of the Web Service participating in a given message exchange and models the observable behavior of a Web service, or set of interacting Web Services in terms of temporal and logical dependencies among the exchanged messages. WSCI does not address the definition and implementation of the internal processes that actually drive the message exchange.

3.2.4 Process Ontologies Comparison

Even though none of the standards presented here are underlined by ontological concepts, i.e. all of them lack semantics, we provide a comparison of the efforts presented in the report for sake of completeness.

In the following some more detailed remarks are provided:

- BPML and WSCI are not maintained and supported any more by their communities, meaning that they have no future in their current forms. However, WSCI, and especially its global model, served as a valuable input for the Web Services-Choreography Description Language (WS-CDL) [70], currently developed by W3C Web Services Choreography Working Group.
- There is no implementation to support BPML and WSCI, while several tools have been built for supporting BPEL4WS (see section 3.2.3.1).

- There is no formal semantic provided neither for BPEL4WS nor for BPML/WSCI, which would help to make interesting proofs about them (e.g. deadlock-free, safety, liveness, etc.).

It is very common to compare this type of languages based on their expressivity power in terms of the workflow patterns [69] that these languages support. A comparison in this respect between BPEL4WS, XLANG, WSFL, BPML and WSCI is provided in [72]. A short summary of the comparison between BPEL4WS, BPML and WSCI, based on the workflow patterns they directly or partially support is given below (for a detailed description of the patterns refers to [69]).

- The basic routing constructs (Sequence, Parallel Split, Synchronization, Exclusive Choice and Simple Merge patterns) and Implicit Termination, Multiple Instances without Synchronization, Multiple Instances With a Priori Design Time Knowledge, Deferred, Cancel Activity and Cancel Case patterns are directly supported by BPEL4WS, BPML and WSCI.
- Multiple choice and Synchronizing Merge is directly supported by BPEL4WS (this is a consequence of the “dead-path elimination” principle inherited from WSFL; the idea of death-path elimination is that both positive and negative values can be propagated through control links, determining whether the activities in a path should be executed or not), but not supported by BPML and WSCI.
- Multi-merge is not supported by BPEL4WS, but partially supported by BPML (due to the fact that BPML supports invocation of sub-processes) and WSCI.
- Discriminator, Arbitrary Cycles, Multiple Instances with a Priori Runtime Knowledge, Multiple Instances without a Priori Runtime Knowledge and Milestone are not directly supported by BPEL4WS, BPML and WSCI.
- Interleaved Parallel Routing is partially supported by BPEL4WS (due to the concept of serializable scopes), but not directly supported by BPML and WSCI.

From a workflow patterns perspective both BPEL and WSCI basically offer the same functionality. Although BPEL4WS is a complex language, in the sense that it offers many overlapping constructs (i.e. many possible solutions for some of the patterns), it is a viable alternative to the combination of BPML and WSCI and seems to impose itself on the market due to the tool support for it and current efforts by its community to push it forward. BPEL4WS represents an alternative to the combination of BPML and WSCI, and just like them suffers of the lack of semantics, which refrains it from providing full support for the Web Service Usage Process. BPEL4WS counts with more support in terms of engines and documentation available. Also partners behind BPEL4WS seem to be more active in pushing it forward.

3.2.5 Business Data Ontologies

Currently no business data ontologies exist, i.e. there are no business standards available that are modeled using an ontology language like OWL or RDF. Nevertheless there are a large number of business standards available. These standards model the common concepts (e.g. Customer, Purchase-Order) naturally present in business transactions. Most of these standards, especially the recent ones are build upon XML.

Although these standards don't capture all the semantic information that would be contained in a business ontology, they can at least be used as a basis for building such an ontology. In the following we will discuss some of the more popular standards in order to give an overview of what is available.

Apart from that the IEEE [64] tries to standardize a so called 'Upper Ontology'. This ontology will contain all the generic concepts necessary to define domain specific ontologies.

3.2.5.1 XML Business Standards

The available XML business standards can be divided into two categories, the so called *vertical* standards and the so called *horizontal* standards. In this context horizontal standards are the one that cover the needs of many different industries (e.g. xCBL, UBL) whereas the vertical standards are those that are specifically tailored for one industry like for example the automotive industry.

The following section will give an overview of some of the available standards. To give a good overview of the existing work we tried to select the most important horizontal standards (i.e. backed by large companies or user groups) and some vertical standards from different industrial areas. A good overview over all existing XML Business Standards is given in the XML Cover Pages⁴.

The selection of standards presented in the following chapter only contains freely available standards. These standards can therefore easily be used as starting points or building blocks for the development of business data ontologies in DIP.

3.2.5.1.1 UBL

The Universal Business Language (UBL) [68] belongs to the category of horizontal standards.

The UBL project was started by the OASIS group⁵. Its goal is to develop a generic XML based interchange format for business documents that can be extended in order to meet the requirements of specific industries.

The current status of the project is that the release version 1.0 of UBL has been approved as an OASIS draft. This version contains eight basic documents that support the generic order-to-invoice process. In addition to that the release contains a large number of reusable XML Schemas (e.g. reusable schemas of Business Information Entities (BIE) and reusable data type schemas).

Note that the BIEs, on which UBL is based on, are implementations of the Core Component Technical Specification developed also by the OASIS group in the ebXML project [25].

It is also important that many of the other existing standards will support UBL by either making their new version UBL compliant or by providing input to the development of UBL. The groups/standards that announced support for UBL include among others HL7, UIG, RossettaNet, xCBL and OAGIS.

⁴ <http://xml.coverpages.org/>

⁵ <http://www.open-oasis.org>

xCBL

The XML Common Business Language (xCBL) [75] is like UBL a horizontal standard. It was developed by Commerce One. The goal of the xCBL project was to provide a schema document framework that could be used for robust XML data exchange.

Currently version 4.0 of xCBL is available.

OAGIS

The Open Applications Group Integration Specifications is a standard developed by the Open Application Group [50]. It's a horizontal standard that has the goal to provide specifications for intra- and inter-enterprise integration.

Currently version 8.0 of the specification is available. This version contains a large number of XML schemas for common business objects.

RosettaNet

RosettaNet [61] is like all of the following standards a vertical one. It is developed to support e-business in the Electronics, IT and Telecommunications industries.

RosettaNet defines the business processes between partners, called RosettaNet Partner Interface Processes (PIPs). Each PIP consists of a business process and its vocabulary and a business process with its choreography and message interfaces.

HL7

Health Level Seven [32] aims to provide standards for the exchange, management and integration of data in the Healthcare domain. Currently version 3.0 of the standard exists as a draft.

IFX

The Interactive Financial Exchange (IFX) Forum [33] develops a specification for an open and interoperable financial market place. The latest version of the IFX Specification is version 1.5. It provides a framework for the robust and scalable exchange of financial data.

3.2.5.2 Other Business Standards

Besides the standards mentioned so far there also exist business standards which are not based on XML. The most prominent of these standards are those belonging to the EDI family. EDI is a data interchange format based on plain text.

These standards have been in use for much longer time than the XML standards and therefore have a large number of users (e.g. automotive industry).

As they have been developed much earlier than the XML standards the EDI based standards lack some important features of the XML standards like:

- Modularity (e.g. xCBL, UBL)
- Clear semantics of the elements (e.g. UBL).

Because of that the non XML standards are not very suitable to be the basis of the development of a business data ontology.

3.2.5.3 Core Component Technical Specification and Core Component Library

The Core Component Technical Specification (CCTS) was part of the ebXML project in the beginning, but was eventually set up as an own project by the OASIS group.

The goal of the CCTS project is to provide a methodology for developing a common set of semantic building blocks that represent the general types of business data used. These building blocks, called Core Components (CC), are meant to be a syntax-neutral representation of the semantics of business data. This means CC can be implemented using any available technology (like e.g. XML).

The collection of all the available CC is called the Core Component Library (CCL). This library is very close to a business ontology as it semantically describes business objects. However, in contrast to an ontology, the CCL doesn't define any relations between the CCs.

3.2.6 Business Data Ontologies Comparison

It doesn't make much sense to compare the existing standards to one another, basically since they are not underlined by ontological concepts. In regard to the vertical standards, it is important to notice that they can only be compared to other standards focused on the same industry.

Besides that it can be said, that UBL has lately received a lot of recognition, both in industry and publications. This might suggest that together with its strong industrial backing, UBL will become one of the most important standards in future. With respect to DIP and the necessary development of one or different Business Data Ontologies a nice feature of UBL is that it is based on CCTS. In contrast to other standards UBL therefore is not only a collection of schemas, but these schemas also have a clear semantic. Furthermore a standard approach for extending and modifying the existing elements exist.

3.2.7 Goal Ontologies

The usage of Web Services implies the specification of goals to be achieved. These goals are often complex, which means they actually consist of sub goals / tasks to be fulfilled. The decomposition of a goal together with the serial handling of each single step can be a difficult and especially time-consuming thing.

The goal related work within DIP is founded on WSMF [29] and further in UPML framework [47], form which borrows some of the main concepts. One of the four components of an UPML specification is the so called Problem Solving Methods (PSM). The basic principle of PSMs is to make knowledge about the dynamics of the process solving process explicit.

UPML distinguishes between primitive and complex PSMs. Complex PSMs decompose tasks into subtasks. Their description is made up of the elements Pragmatics, Costs, Communication policy, Ontology, Competence description and Operational specification.

- The pragmatics provides a human readable description including author and creation date.
- The costs subsume those for interaction (e.g. with the user) and those for computation (e.g. average / worst cases).

- The communication policy describes the communication style of the method and its components.
- The ontology provides a signature used for describing the method.
- The competence description introduces pre- and post-conditions restricting in- / outputs respectively.
- The operational specification defines the data and control flow between the reasoning steps to achieve the functionality of the PSM.

Comparably to PSMs the concepts behind Semantic Web Services aim at allowing people to reach their goals without even having to think about the tasks their goal may consist of. This shall be accomplished by the decomposition of goals followed by the run of the Web Service Usage Process (see Section 2).

One step in reaching this really big vision is to create a certain base of knowledge about which goals a user actually might have. Semantic Web Services will combine Web Services and Semantic Web technologies in order to enable maximal automation and dynamism in all aspects of Web Services. As part of this picture Semantic Web technologies [3] are mainly based on ontologies and we believe that the goals of a user shall be modeled in an ontology, too.

In [22] goal ontologies are mentioned as a part of a goal description which furthermore consists of a name, input and output roles, a pre-condition, a post-condition and assumptions. The goal ontology shall provide the vocabulary for specifying pre-conditions and post-conditions. Furthermore the authors want to use concepts from this ontology to specify the structure of input and output data.

Also [38] are describing their web services using ontologies. User goals are modeled using abstract concepts like alter, copy, create, erase and move. This way they want to give their users the opportunity to query services by means of common sense without the requirement of any domain specific knowledge.

In [38] ontologies are subdivided into domain ontologies and system ontologies. The goal ontology is assigned to system ontologies which also include the so called service ontology. They further delve into what they call the “cooperative goal” but goal ontologies are not discussed any further.

At the time of writing of this deliverable not a single goal ontology could be detected. So this can be assumed to be quite an open field of research. Nevertheless, WSMO [71] includes the concept of a goal in its conceptualization, to allow the specification of the objectives a user might have, providing the means to express high level description of a concrete task. In WSMO goals are described by means of:

- Non functional properties.
- Used mediators: Goals can import ontologies using ontology mediators. Also a goal may be defined by reusing an already existing goal. This is achieved by using goal mediators.
- Post-conditions: They describe the state of the information space that is desired.
- Effects: Describe the desired state of the world after the execution of the Web Service.

3.2.8 Goal Ontologies Comparison

As mentioned above, there could not be detected any goal ontologies. For this reason we will present work that could serve as a basis for the creation of those. More precisely we will present two term classification schemes and a goal management tool.

3.2.8.1 eCl@ss

The eCl@ss standard for material classification and ware groups [24] is developed by the same called association of eleven leading German companies. It is offered as an information exchange standard for suppliers and customers. It includes 12000 keywords and has a hierarchical structure consisting of 4 classification levels. The current release is spread over six CSV-formatted files. Their content is listed below:

1. Classification table
2. Keyword table
3. Property table
4. Value table
5. Relations between (1) and (3)
6. Relations between (3) and (4)

Because of the involvement of quite a number of leading German companies eCl@ss has the potential to become a good basis for a goal ontology. At least in Germany it should be possible to reach an agreement over such a vocabulary. Besides this approach also single parts of the classification scheme could be reused and included into some goal ontology.

3.2.8.2 UNSPSC

The United Nations Standard Products and Services Code® (UNSPSC®) [67] provides an open, global multi-sector standard for efficient, accurate classification of products and services.

The hierarchical structure of the standard is made up of the following 5 levels

1. Segment
2. Family
3. Class
4. Commodity
5. Business Function

The current version is offered as a single Excel file in which these levels are defined on separate table sheets.

Although translations are planned the current version supports English language only. Nevertheless because of the involvement of the United Nations a wide international acceptance of the standard can be expected. For this be reason the UNSPSC vocabulary should be better suited than eCl@ss for instance to formulate goals of users potentially coming from any country all over the world.

3.2.8.3 PGMT

In [35] goals are defined to be objectives the system under consideration should achieve. They are distinguished as either policy goals (strategic goals) or scenario goals (tactical goals) in each policy.

The presented Privacy Goal Management Tool (PGMT) is a web-based tool that assists analysts in the goal mining (extracting), reconciliation (comparing) and management (organizing) processes. The PGMT maintains a goal repository. Each goal in the repository is associated with a

- Unique ID
- Description
- Responsible actor
- Sources of the goal
- Privacy taxonomy classification

The authors formulate functional requirements for goal management which are listed below:

1. Add a new goal
2. Update an existing goal
3. Delete a goal
4. Trace a goal
5. Display the total occurrences of a goal in the policy
6. Provide goal definitions
7. Add a template for each goal
8. Goal classification as observable goals and unobservable goals
9. Dynamically add definitions of goals into the PGMT
10. Merge goals in the policies

As can be seen the notion of a goal introduced in PGMT is less user-oriented than in DIP. Beyond this the repository is described rather less concretely.

4 TOOLS

This section provides an overview of the different tools that realize at least partly the concepts developed in Section 2, making use up to some extent of epistemological, data and process ontologies.

4.1 Tools state of the art

Service composition and the tools existing for completing this task can be classified based on different criteria. One possible criterion could be: how much work the human user has to put in solving web service composition task. According to [42] service composition can be broadly classified into three categories: *manual*, *semi-automated* and *automated* composition. Another criterion could be whether the tool is a visual one or is a non-visual one. In the next section we overview the available tools for Web Service composition having in mind the classification provided in [42], but we also underline which of the tools provide Graphical User Interface.

4.1.1 Manual tools

Most of the tools available for Web Service composition are *manual* tools. The most relevant tools that fit into this category are: Triana, BPWS4J and Self-Serv.

4.1.1.1 Triana

Triana⁶ [43] is a problem-solving based environment that allows discovery, invocation, composition and publishing of Web Services. Regarding the composition aspect, Triana allows multiple Web Services to be composed in a visual manner. Users can construct complex workflows graphically. The resulting composed graph, built manually, can then be executed or can be saved in a Triana format or other formats, like BPEL4WS format.

Besides allowing the application composition in a graphical way, Triana also counts with monitoring facilities. Users can log in, or log off during execution of the process and can visualize the progress of the process without having any affect on it. Providing a visual interface for Web Service composition can be strong point for a tool like Triana. It can be also a weak point because of visual scalability problem. In this sense dealing with large workflows, in the terms of composition and management, is a problem in Triana.

4.1.1.2 BPWS4J

BPWS4J⁷ [14] is the IBM implementation of BPEL4WS and consists of two parts: an editor and an engine. It provides a simple user interface for creating BPEL4WS documents. Users can compose a graph at the XML level. This composed graph, along with a WSDL document for the composite service, is submitted to the execution engine. An important drawback of BPWS4J is that it expects the user to specify a workflow at the XML level.

⁶ <http://www.triana.co.uk/>

⁷ <http://www.alphaworks.ibm.com/tech/bpws4j>

4.1.1.3 Self-Serv

It represents a middleware infrastructure for the composition of Web services. It is based on three foundational principles namely: (1) declarative composition of services from existing ones; (2) dynamic selection of services based on particular attributes; and (3) peer-to-peer approach to the orchestration of the composite service executions [6]. Self-Serv⁸ uses two different elements in its approach to composition:

- **Composite Services.** Composite structure that contains simple and composite services, necessary to realize a concrete functionality. It uses state charts – specifications of the different states, basic or compound, of concrete business logic – to represent the business logic of a composite service operation.
- **Service Container.** Service that aggregates other services, which fulfil the same capability. Service containers help to cope with the highly dynamic environment (providers remove, relocate or modify services) in which Web services operate. It relies on a membership mode – different modes in which a member can be part of a container (explicit, query or registration) – and a scoring service – policy that enables the container to choose a member to execute a concrete operation at invocation time – to realize attribute based dynamic selection, and to maintain its list of members.

Self-Serv uses a Peer-to-Peer approach, based on *State coordinators* and *Routing tables*, to allow the self orchestration of services over the Internet without relying on a central scheduler. State coordinators are responsible for receiving completion notifications from other state coordinators, deciding based on these notifications when to enter a particular state, invoking a particular service by means of message exchange, waiting for a reply, notification the service coordinator about the next states it needs to enter once service execution is completed, and failure handling. Routing tables represent the knowledge a coordinator must use. They are composed of pre-conditions – conditions that hold before entering a state –, and post-conditions – actions that indicate which coordinators need to be notified when a state is exited.

With the aim of providing support to discover, compose and execute Web services, Self-Serv follows a layered architecture, with five main layers; (1) *Service layer*, pool of services and containers together with methods to invoke the operation provided by containers and collect the invocation outputs; (2) *Conversation layer*, allows partners to interact with each other using a concrete B2B standard (RossetaNet, EDI or cXML); (3) *Directory layer*, set of directories that store relevant information about containers and services; (4) *Communication layer*, provides transport facilities based on SOAP messages; and (5) *User layer*, facilitates user access to the service composition environment.

Roughly speaking Self-Serv addresses the declarative composition of services out of existing ones using a Peer-to-Peer self-orchestrated approach to control the execution of the composite service. It uses the attributes of services to dynamically select the most appropriate ones and utilizes composite services and service containers to realize its approach to composition. Self-Serv built on a layered architecture made up of 5

⁸ <http://cgi.cse.unsw.edu.au/~selfserv/index.php?go=compositionprj>

different layers in order to allow discovery, composition and execution of the composed services.

4.1.2 Semi-automatic and automatic tools

Composing Web Services in a manual way present an scalability disadvantage. For example, when a vast amount of services need to be combined in order to obtain the desired functionality. One possible solution is to use *semi-automated* and *automated* tools for composition to solve this problem. In the rest of this section we will describe some of the tools for semi-automatic and automatic composition. BioOpera, Mind Swap's Web Service Composer and ServiceCom are tools that allow *semi-automated* composition.

4.1.2.1 BioOpera

BioOpera⁹ [7] provides an intuitive way to specify a parallel computation based on composition of different services. The main reason of developing BioOpera was to coordinate distributed processes within bioinformatics applications. The BioOpera system uses an underling language called BioOpera Flow Language [53] in order to model process control and data flows. This language is a visual programming language allowing users to specify in a very intuitive way: processes, related tasks, relationships between them, etc. In BioOpera a process is modelled by drawing a set of directed graphs. The nodes of the graph represent tasks and their data parameters. The edges of the graph represent control flow or data flow dependencies.

BioOpera assists users not only in the development process but also during the execution and monitoring process of distributed applications composed out of a library of Web Services. During the development process users can import existing Web Services available as reusable components. Then users can select a set of services and drag them into a process. Next step is to specify data and control flows. These operations are partially automatic, because the system knows how to perform name matching, human intervention is required only to resolve ambiguities. User can revise any time the data and control flows. The editor automatically keeps the two graphs synchronized. After all services are composed, the resulting process can be compiled and executed. The user has also the possibility to monitor process execution and to intercede during execution. BioOpera allows also concurrent execution of processes. One major problem with visual based tools for Web Services composition is their visual scalability. BioOpera's solution to this problem is to collapse parts of the graphs into single nodes, increasing user overall view of the all process. This way, the user may navigate through complex data and controls flows. BioOpera, due to its visual based approach, supports the user in rapidly building complex processes from a library of existing component Services.

4.1.2.2 Mind Swap - Web Service Composer

Mind Swap Web Service Composer¹⁰ is a prototype that guides the user in the dynamic composition of semantically enhanced Web Services by presenting different matching

⁹ <http://www.iks.inf.ethz.ch/projects/projects/bioopera/>

¹⁰ <http://www.mindswap.org/~evren/composer/>

services, filtering the suitable options based on semantic descriptions, and executing the resulting composition by invoking the WSDL grounding of the services. The prototype focuses on composition, which means that services had been previously discovered. Web Service Composer is built of two main elements namely:

- **Inference engine.** An OWL reasoner that is based on information about known services (specified by a DAML-S service profile) which is able to find matching services.
- **Composer.** An interface to manage the communication between the user and the inference engine, presenting only those services that can be fed to a selected service as input, and allowing the user to choose between the matching services, in order to define the execution workflow.

The operation mode is as follows, once the user chooses a service, a query is sent to the inference engine which provides a list of the different inputs of the service, for each of the inputs a new query is defined and sent to the inference engine in order to retrieve the services that can provide the data for each one of the particular inputs.

The Web Service Composer allows two different types of matching, (a) *exact match*, if the service profiles of two parameters belong to the same OWL class and (b) *generic match*, if the output is an specialized version of the input which still allows the connection of the services.

In a nutshell the Web Service Composer represents an initiative that tries to solve the composition of Web services taking a semi-automatic approach that guides the user in the dynamic composition of Web Services that are semantically enhanced. It makes use of an inference engine and a composer in order to realize its functionality and allows two types of matching: exact and generic.

4.1.2.3 ServiceCom

It presents another interesting initiative for the semi-automated Web Service composition. ServiceCom [51] is a Java based tool for modular and reusable Web Service composition. Using ServiceCom one can specify, construct and execute a complex process comprised of many Web Services. The main concept used in ServiceCom tool is the *Service component*. Service components are a packing mechanism for developing web-based distributed applications in terms of combining existing Web Services. These components, once defined, can be reused, specialized and extended. The ServiceCom tool uses a language called Service Composition Specification Language (SCSL) [76] in order to support the service component mechanism. This language defines the basics constructs used to define a Web Service composition.

In ServiceCom a service composition process has four phases. These are: *Description*, *Planning*, *Building* and *Invocation*. The first phase of the process is the *Description* phase. ServiceCom offers a visual-oriented development environment allowing the user to describe in a graphical manner the complex graph which represents the process. Components like: activities, bindings and conditions can be dragged and dropped on the designer window. In this phase of Web Services composition ServiceCom offers automatic support. Next phase is the *Planning* phase. This phase is concerned with binding the activities in the service composition to the concrete services. The provider library facility is intensively used in this phase in order to locate Web Services

providers. The *Building* phase is the most important in Web Service composition model. In this phase the Web Service composition is used as an input to generate the source files for the invocation of the composition. Finally the *Invocation* phase deals with the execution of a Web Service composition and its corresponding result handling. Basically ServiceCom is a visual tool that has an integrated approach to service composition, promoting reusability and specialization in the specification of compositions.

There are some tools for Web Service composition that offer a certain degree of automation. Some of the relevant ones that allow automated Web Service composition are: Sword and AI Planning based tools.

4.1.2.4 Sword

It is a toolset that helps service developers to quickly compose *basic* Web services to realize new composite Web Services [59]. To represent services Sword¹¹ follows a rule-based approach. Every service is represented by a rule that states the relation between inputs provided to the service and the output of the service.

The first phase of the service composition in Sword consists of defining the inputs and the outputs of the composite service. Sword determines next, in an automatic way, if the composite service can be realized using the existing services. It does this using a rule engine. The rule engine used by Sword is a Java based implementation called Jess [54]. If the composite service can be realized using the existing services, the next phase is to generate a *composition plan* for the composite Web Service. In the last phase the user can view the generated plan. A composite plan and its associated representation consist of a sequence of services that need to be invoked to obtain the composite outputs from its inputs. What Sword lacks is an automated service discovery mechanism. The user has to specify the services that will take part in the composition process. Composition can fail if the service required cannot be found. Also in Sword, composition is based on the specific implementation of the services.

Another way of dealing with the Web Service composition process is to reuse the work done in Artificial Intelligence (AI), more precisely in AI Planning [54]. The process of composing Web Services is quite similar with plan generation process for a given problem and in essence using AI Planning for service composition could increase the automation of this process. There are already some frameworks and tools developed for Web Service composition that are based on AI Planning technologies. We briefly describe the most important ones, pointing out their strong and weak points.

4.1.2.5 Pegasus

Pegasus¹² is a configurable system that can map and execute complex workflows on the Grid [58]. It stands for Planning for Execution in Grids and proposes two workflow generation systems. The first system: Concrete Workflow Generator (CWG) maps an abstract workflow onto an executable workflow. The second system, Abstract and Concrete Workflow Generator (ACWG) used AI planning technologies to generate

¹¹ <http://www2002.org/CDROM/alternate/786/>

¹² <http://pegasus.isi.edu/>

workflows in Grid environments. Pegasus has some drawbacks. First it does not have a workflow repository to store and reuse already constructed workflows. Pegasus does not provide a fault handling mechanism and also there is not a clear distinction between abstract and concrete workflows.

The proposed system translates the process models of Web Services into sets of SHOP2 methods and operators, so that SHOP2 can be used with DAML-S Web Service descriptions to automatically compose Web Services. One major drawback of this system, that is also present in other AI Planning based systems for Web Services composition, is the assumption that the information received during the planning will not be changed. In a real world domain this assumption is not valid. There are also other AI technologies that can be used in Web Service composition, like using agents to solve the composition task. An approach like this is described in [44]. The proposed system is based on Golog, a high-level logic programming language, for specification and execution of complex actions in dynamic domains. As an alternative to planning, this approach does not change the computational complexity of the task of generating the composition.

4.1.2.6 IRS-II

The Internet Reasoning Service¹³[34] is a Semantic Web Services framework, which allows applications to semantically describe and execute web services. The IRS supports the provision of semantic reasoning services within the context of the Semantic Web.

IRS-II is based on the UPML (Unified Problem Solving Method Development Language) framework [47], which distinguishes between the following categories of components specified by means of an appropriate ontology:

- *Domain models*. These describe the domain of an application (e.g. vehicles, a medical disease).
- *Task models*. These provide a generic description of the task to be solved, specifying the input and output types, the goal to be achieved and applicable preconditions.
- *Problem Solving Methods (PSMs)*. These provide abstract, implementation independent descriptions of reasoning processes which can be applied to solve tasks in a specific domain.
- *Bridges*. These specify mappings between the different model components within an application.

The main components of the IRS-II architecture are the IRS-II Server, the IRS-II Publisher and the IRS-II Client, which communicate through the SOAP protocol:

- *The IRS-II server* holds descriptions of Semantic Web Services at two different levels. A knowledge level description is stored using the UPML framework of tasks, PSMs and domain models. These are currently represented internally in OCML [45], an Ontolingua-derived language which provides both the

¹³ <http://kmi.open.ac.uk/projects/irs/>

expressive power to express task specifications and service competencies, as well as the operational support to reason about these. In addition, IRS-II has a special-purpose mapping mechanism to ground competence specifications to specific Web services.

- *The IRS-II Publisher* plays two roles in the IRS-II architecture. Firstly, it links Web services to their semantic descriptions within the IRS-II server. Secondly, the publisher automatically generates a wrapper which turns the code into a Web service. Once this code is published within the IRS-II it appears as a standard message-based Web service, that is, a Web service endpoint is automatically generated.
- *The IRS-II Client* - A key feature of IRS-II is that Web service invocation is capability driven. The IRS-II supports this by providing a task centric invocation mechanism. An IRS-II user simply asks for a task to be achieved and the IRS-II broker locates an appropriate PSM and then invokes the corresponding Web service.

IRS-II was designed for ease of use. Developers can interact with IRS-II through the IRS-II browser, which facilitates navigation of knowledge models registered in IRS-II as well as the editing of service descriptions, the publishing and the invocation of individual services. Application programs can be integrated with IRS-II by using the Java API. These programs can then combine tasks that can be achieved within an application scenario.

Concluding shortly, IRS-II, a knowledge-based approach to Semantic Web Services, which evolved from research on reusable knowledge components, is one of the main approaches that have been driving the development of Semantic Web Service frameworks.

4.2 Relevant for DIP

Out of all the tools presented in this section might be of relevant interest for DIP the use of BPWS4J [14] the IBM implementation of BPEL4WS which represents one of the best execution environments for workflows specified using BPEL. Also IRS-II might be relevant, since the conceptual background of the tool is the same as in WSMO and consequently as DIP. The tool for composition that DIP will provide should offer (semi)automatic support. A full automatic support is quite hardly to have. As stated before, BioOpera offers semi-automatic support during composition process and we think it might be relevant for DIP.

5 REQUIREMENTS

This section details the requirements for Service description and Service ontology in the context of DIP. A sketch of the main issues to be addressed is provided taking into account that some other deliverables like 3.2 Framework for Semantic Description, are tackling the problem in more depth.

5.1 Service Description Requirements

When describing Services both functional and non-functional properties must be taken into account.

Some of the non-functional aspects that need to detailed are [71]:

- **Performance:** Represents how fast a service request can be completed
- **Reliability:** Represents the ability of a web service to perform its functions (to maintain its service quality). It can be measured by the number of failures of the service in a certain time interval.
- **Security:** Represents the ability of a service to provide authentication, authorization, confidentiality, traceability/auditability, data encryption, and non-repudiation.
- **Scalability:** Represents the ability of the service to process more requests in a certain time interval. It can be measured by the number of solved requests in a certain time interval.
- **Robustness:** Represents the ability of the service to function correctly in the presence of incomplete or invalid inputs. It can be measured by the number of incomplete or invalid inputs for which the service still function correctly
- **Accuracy:** Represents the error rate generated by the web service. It can be measured by the numbers of errors generated in a certain time interval
- **Transactional:** represents the transactional properties of the web service
- **Trust:** represents the trust worthiness of the service
- **Financial:** represents the cost-related properties of a web service.
- **Network-related QoS:** Represent the QoS mechanisms operating in the transport network which are independent of the web services. They can be measured by network delay, delay variation and/or message loss.

In regard to the functional aspects they should essentially allow one to fully realize the Semantic Usage of Services as explained in Section 2. To do so, the description of the services needs to account for:

- **Specification of what the service is able to do/offer.** Such description should be independent of the protocols, bindings, business models or data represent internally used, by the Web Service.
- **User's objectives** should be also defined in a way that allows to match them against the Web Service functionality.

- The description of a Service should be done in a measurable way in order to allow rating, to choose the most appropriate one
- Taking as a starting point the Computer Based Software Engineering (CBSE) approach, Web Services should be reused independently of the domain for which they were developed, this requirement poses a strong constraint, in terms of the underlying mechanism used to describe services, which should have some formal model underneath.
- Another point that derives from the previous one is that the mediation support needs to connect Services coming from different domains. Such mediation should allow terminologies, data representation, business models and objectives in an efficient way.
- During the execution of Services errors might occur. If this is the case, Services should be defined in a way that allows the supervision of the whole process, its replacement with equivalent ones, and the roll-back to avoid unwanted effects if required.

In general the most wanted thing is an upper level ontology that allows for the description of all these different aspects.

5.2 Service Ontology Requirements

Service Ontology requirements well cared for and elaborated in Deliverable D3.2, Service Description Framework is also available per June 30, 2004, in the next lines the main ideas shown there are sketched.

Essentially four different types of ontologies are required as already presented in this deliverable:

- **Epistemological Ontologies:** They should account for all the different aspects specified in section 5.1
- **Business Data Ontologies:** Web Services need to be able to understand the data transported within the messages in terms of data types, and the meaning of the information carried. Depending on the business model for which the services are deployed different data types and domain knowledge might be used to encapsulate data and its meaning. The interoperation of Services will require the data to be semantically described in order to allow the mediation of data names, types and domain knowledge.
- **Goal Ontologies:** They should allow the description of objectives that user wants to satisfy. It will later be matched against Web Services capabilities. It was not possible to identify any initiatives that try to develop Goal ontologies, even though they are cared for in WSMO and in some extent also in OWL-S, no specific Goal ontology could be found.
- **Process ontologies:** Web Services need to be able to work with others that do not share the same business model. Services belonging to different business models might need to interoperate with each other, and if this is the case, the appropriate process description should be ready in order to allow the mediation of the different models. Also when it comes to

communication the exchange of messages must adhere to some convention. Different services might use different message exchange patterns which also need to be semantically described in order to enable communication.

CONCLUSION

In this deliverable the Semantic Web Service Usage Process has been introduced as a guideline to further discuss the state of the art on Service description and the different ontologies used to describe such Services. Essentially ontologies are required to enrich the description of processes, to enhance the data exchanged among services so the types and meaning of the exchanged information can be understood; to describe the goals a user might have when consulting a Web Services and finally to wrap all these different descriptions into a full Service description by means of epistemological ontologies.

Later on the deliverable the different tools that currently exist to realize the usage of service have been presented sketching their approach Finally the requirements in terms of Service description and Service ontologies have been introduced.

As a conclusion of the deliverable, it must be pointed that the state of the art on ontologies for Service description is not very advanced, especially in process, and goal ontologies, where no standards have been found. As far as Business data ontologies are concerned, UBL seems to be a nice starting point to borrow concepts in the development of the DIP business data ontology. In regard to epistemological ontologies, there are two competing initiatives, which tackle the problem following a different approach and with different development stage. Our recommendation for DIP based on the comparison presented in section 3.2.2 is WSMO.

REFERENCES

- [1] Arkin, A., (2002). Business Process Modelling Language. <http://www.bpmi.org/>.
- [2] Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacs-Nagy, P., Trickovic, I., Zimek, S. (2002). Web Service Choreography Interface 1.0, <http://www.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf>.
- [3] Arroyo, S., Lara R., Gomez, J. M., Berka, D., Ding, Y. and Fensel, D., "Semantic Aspects of Web Services" in Practical Handbook of Internet Computing. Munindar P. Singh, editor. Chapman Hall and CRC Press, Baton Rouge. 2004.
- [4] Arroyo, S. Fensel, D.: The Semantic Web Service Usage Process. International Semantic Web Conference (ISWC04), Hiroshima (Japan), 7-11 November. Submitted.
- [5] Arroyo, S., Ying, D., Lara, R., Stollberg M., and Fensel, D.: Semantic Web Languages. Strengths and Weakness, International Conference in Applied computing (IADIS04), Lisbon (Portugal), 23-26 March 2004
- [6] Banatallah, B., Sheng, Q.Z., and Dumas, M.: The Self-Serv Environment for Web Services Composition, Jan/Feb, 2003, IEEE Internet Computing. Vol 7 No 1. pp 40-48.
- [7] Bausch, W., Pautasso, C., Schaeppi, R., and Alonso, G. BioOpera: Cluster-aware Computing on Proceedings of IEEE International Conference on Cluster Computing (CLUSTER'02), Chicago, Illinois, September 2002.
- [8] Bechhofer, S. et al, 2001. DAML+OIL is not enough. Proceedings of the First Semantic Web Working Symposium (SWWS'01), 151–159.
- [9] Bellwood, T., Clément, T., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y. L. Januszewski, K., Lee, S., McKee, B., Munter, J., von Riegen, C. (2002) UDDI Version 3.0. Published Specification, <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>, 2002.
- [10] Berners-Lee, T. (1999). Weaving the Web. San Francisco: Harper.
- [11] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. Scientific American, 284(5):34-43, 2001.
- [12] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F. Thatte, S., Winer, D. (200) Simple Object Access Protocol (SOAP) 1.1, <http://www.w3.org/TR/SOAP/>, 2000.
- [13] Brickley, D. and Guha, R.V. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation. <http://www.w3.org/TR/rdf-schema/>
- [14] Business Process Execution Language for Web Services Java (BPWS4J), <http://alphaworks.ibm.com/aw.nsf/download/bpws4j>
- [15] Cabrera, F., Copeland, G., Freund, T., Klein, J., Langworthy, D., Orchard, D., Shewchuk, J., Storey, T. (2002) Web Services Coordination (WS-Coordination)', <http://www-106.ibm.com/developerworks/library/ws-coor/>, 2002.

-
- [16] Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T., Thatte, S. (2002). Web Services Transaction (WS-Transaction), <http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/>.
- [17] Collaxa, <http://www.collaxa.com/>
- [18] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. (2001) WSDL Web Services Description Language (WSDL) 1.1, <http://www.w3.org/TR/wsdl>.
- [19] Concurrent Transaction Logic Prototype, <http://www.cs.toronto.edu/~bonner/ctr/>
- [20] Dean, M. and Schreiber, G. (2003) OWL Web Ontology Language Reference. W3C Recommendation. <http://www.w3.org/TR/owl-ref/>
- [21] Daconta, M. C., Obrst, L. J. and Smith, K. T. (2003). The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management”, Wiley Publishing, Indianapolis, Indiana.
- [22] de Bruijn, Jos et al. (2003). A Unified Semantic Web Services Architecture based on WSMF and UPML, pp. 11-13.
- [23] Dumitru Roman, Uwe Keller and Holger Lausen (eds.), D2v01. Web Service Modeling Ontology (WSMO), DERI Working Draft 14 February 2004,. <http://www.wsmo.org/2004/d2/v0.1/20040214/#L3907> accessed on 25 May 2004.
- [24] eCl@ss e.V. (2003): eCl@ss Release 5.0, <http://www.eclass.de/>
- [25] Electronic business XML (ebXML), www.ebxml.org/specs.
- [26] EXtensible Markup Language (XML), <http://www.w3.org/XML>
- [27] Evren Sirin, James Hendler, Bijan Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions", Proceedings of the 1st Workshop on Web Services: "Modeling, Architecture and Infrastructure" (WSMAI-2003), In conjunction with ICEIS 2003, Angers, France, pp. 17-24, April 2003.
- [28] Fensel, D. (2001) Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, Springer-Verlag, Berlin.
- [29] Fensel, D., and Bussler, C. (2002), The Web Service Modeling Framework WSMF, Electronic Commerce Research and Applications, 1(2).
- [30] Fensel, D.; Bussler, C.; Ding, Y.; Omelayenko, B. (2002): The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications, 1(2), 2002.
- [31] Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, 5:199-220.
- [32] Health Level Seven, <http://www.hl7.org/>
- [33] Interactive Financial eXchange , <http://www.ifxforum.org/>
- [34] IRS-II Semantic Web Reasoning Framework. <http://kmi.open.ac.uk/projects/irs/>
- [35] Jain, Neha et al. (2004): Privacy Goal Management Tool (PGMT) Software Requirements Specification, <http://william.stufflebeam.cc/papers/TR-2004-07.pdf>
- [36] Lara, R., Roman, R., Polleres, A.: D4.1v02 Conceptual Comparison WSMO/OWL-S, <http://www.wsmo.org/2004/d4/d4.1/v0.2/>, DERI Working Draft March 2004.
-

-
- [37] Lara, R. Semantic Web Services description. Requirements for automatic location, composition, invocation and interoperation Technical Report, Universidad Autónoma de Madrid TEA, 2003.
- [38] Laukkanen, M. et al. (2004): Towards Ontology-Based Yellow Page Services , WWW2004 Workshop Application Design, Development and Implementation, May 2004
- [39] Lemahieu, W. Web service description, advertising and discovery: WSDL and beyond. In J. Vandenbulcke and M. Snoeck (eds.), *New Directions in Software Engineering*, Leuven University Press, 2001.
- [40] Leymann, F. (2001). Web Services Flow Language (WSFL 1.0), <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>
- [41] Li, Lei and Horrocks, I.: A software framework for matchmaking based on semantic web technology. In Proc. of the Twelfth International World Wide Web Conference (WWW 2003), pages 331-339. ACM, 2003
- [42] Majithia, S., Waker, D. W., and Gray, W. A.: A Framework for Automated Service Composition in Service-Oriented Architectures on Proceedings of the First European Semantic Web Symposium, Heraklion, Crete, Greece, May 2004.
- [43] Majithia, S., Taylor, I., Shields, M., and Wang, I.: Triana as a Graphical Web Services Composition Toolkit on Proceeding of the UK e-Science Programme All Hands Meeting 2003, Nottingham, UK, September 2003.
- [44] McIlraith, S., and Son, T.C.: Adapting Golog for Composition of Semantic Web Services, In Proceeding of the International Conference on the Principles of Knowledge Representation and Reasoning (KRR'02), 2002, 482-496.
- [45] Motta E. Reusable Components for Knowledge Modelling. IOS Press, Amsterdam, The Netherlands. (1999)
- [46] Newcomer, E. (2002). Understanding Web Services: XML, WSDL, SOAP UDDI. Addison Wesley, 2002.
- [47] Omelayenko, B., Crubezy, M., Fensel, D., Benjamins, R., Wielinga, B., Motta, E., Musen, M., Ding, Y.: UPML: The language and Tool Support for Making the Semantic Web Alive. In: Fensel, D. et al. (eds.): *Spinning the Semantic Web: Bringing the WWW to its Full Potential*. MIT Press (2003) 141–170
- [48] Openstorm, <http://www.openstorm.com>
- [49] OWL-S. OWL for Services. <http://www.daml.org/services/owl-s/1.0/>, 2003
- [50] Open Applications Group, <http://www.openapplications.org/>
- [51] Orrienes, B., Yang, J., and Papazoglou, P.: *ServiceCom: A tool for Service Composition Reuse and Specialization*
- [52] Paolucci, M. Kawamura, T., Payne, T., and Sycara., K: Semantic matching of web services capabilities. In Proc. Of the 1st International Semantic Web Conference (ISWC), 2002.
- [53] Pautasso, C., and Alonso, G. Visual Composition of Web Service on Proceeding of the 2003 IEEE Symposia on Human Centric Computing Languages and Environments (HCC 2003), Auckland, New Zealand, October 2003.
- [54] Peer, J.: Towards Automatic Web Service Composition using AI Planning Techniques, <http://sws.mcm.unisg.ch/docs/wsplanning.pdf>
-

-
- [55] Peltz, C. (2003). Web service Orchestration and Choreography. A look at WSCI and BPEL4WS, Web Services. Journal Vol. 3(7).
- [56] Peltz, C. (2003). Web Service Orchestration: a review of emerging technologies, tools, and standards. Technical White Paper http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf
- [57] Pilioura, T., Tsalgatidou, A., Batsakis, A. Using WSDL/UDDI and DAML-S in Web Service, , Workshop on E-Services and the Semantic Web (ESSW' 03), Budapest (Hungary), May 2003.
- [58] Planning for Execution in Grid (PEGASUS) <http://pegasus.isi.edu/>
- [59] Ponnekanti, S.R., and Fox, A.: SWORD: A developer toolkit for Web Service Composition, available at <http://www2002.org/CDROM/alternate/786/>
- [60] Resource Description Framework (RDF), www.w3.org/RDF/
- [61] RosettaNet, <http://www.rosettanet.org/RosettaNet>
- [62] Semantic Web Services Language (SWSL), <http://www.daml.org/services/swsl/>
- [63] Stollberg, Michael et al. (2004). SWF Framework Semantic Web Fred, May 2004, available at <http://www.deri.at/research/projects/swf/papers/SWF-D1-SWFFramework-final.pdf>
- [64] Standard Upper Ontology Working Group (SOU WG), <http://sou.ieee.org>
- [65] Sycara, K., Klusch, M., Widoff, S. (1999) Dynamic Service Matchmaking among Agents in Open Information Environments, ACM SIGMOD Record, vol. 28, no. 1, pp. 47-53.
- [66] Thatte, S. (2001). XLANG, Web Services for Business Process design, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/.
- [67] United Nations Development Programme (UNDP), Dun & Bradstreet Corporation (1998): The United Nations Standard Products and Services Code, <http://www.unspsc.org/>
- [68] Universal Business Language (UBL), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl
- [69] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., and Barros, A.P.: Workflow Patterns. Technical Report FIT-TR-2002-02, Faculty of IT, Queensland University of Technology, Brisbane, Australia, 2002.
- [70] Web Services Choreography Description Language (WS-CDL), <http://www.w3.org/TR/ws-cdl-10/>
- [71] Web Service Modeling Ontology (WSMO), <http://www.wsmo.org/2004/d3/d3.1/v0.1>, 2003
- [72] 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 I.Y. Song, S.W. Liddle, T.W. Ling, and P. Scheuermann, editors, 22nd International Conference on Conceptual Modeling (ER 2003), volume 2813 of Lecture Notes in Computer Science, pages 200-215. Springer-Verlag, Berlin, 2003.
- [73] Wu, D., Parsia, B., Sirin, E., Hendeler, J., Nau, D. H Automating DAML-S Web Services Composition Using SHOP2 on Proceedings of Planning for Web Services Workshop in ICAPS 2003, Trento, Italy, June 2003

- [74] Wu, D., Sirin, E., Parsia, B., Hendler, J., and Nau, D.: Automatic Web Services Composition using SHOP2, In Proceedings of Planning for Web Services Workshop in ICAPS 2003, Trento, Italy, June 2003.
- [75] XML Common Business Library (xCBL), <http://www.xcbl.org/>
- [76] Yang, J., and Papazoglou, M.P.: Service Components for Managing the Life-Cycle of Service Compositions., *Information Systems*, 29 (2004) pp. 97-125