



SWWS – Semantic Web Enabled Web Services

Title:

Report on Development of Web Service Discovery Framework

Version: 1.0

Date: October 06, 2003

Pages: 30

Responsible Authors:

Boris Motik, FZI

Co-Author(s):

Andreas Abecker, FZI

Status:

- Draft
- To be reviewed
- Proposal
- Final / Released to CEC

Confidentiality:

- Public - for public use
- INT - for SWWS consortium (and Project Officer if requested)
- Restricted - for SWWS consortium and Project Officer only

Project ID: IST-2002-37134

Deliverable ID: 3.1


Workpackage No: WP3

Title:

Report on Development of Web Service Discovery Framework

Summary / Contents:

This document describes the current status of the Web Service Discovery Framework. In particular, it describes its functionality and initial architecture.

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 2 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

SWWS Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2002-37134. The partners in this project are: Leopold-Franzens Universität Innsbruck (IFI, Austria); National University of Ireland, Galway (NUI, Galway, Ireland); Forschungszentrum Informatik (FZI, Germany); Intelligent Software Components S.A. (iSOCO, Spain); OntoText Lab. - Sirma AI Ltd. (SAI, Bulgaria); Hewlett Packard (HP, UK), British Telecom (BT, UK)

Leopold-Franzens Universität Innsbruck (IFI)

Institut für Informatik
Technikerstrasse 13
A-6020 Innsbruck Austria

Tel: +43 512 507 6489
Fax: +43 512 507 9872

Contact person: Juan Miguel Gomez
E-mail: juan.miguel@uibk.ac.at

National University of Ireland, Galway (NUI)

National University of Ireland,
University Road
Galway, Ireland

Tel: +353 91 750414
Fax: +353 91 562894

Contact person: Liam Caffrey
E-mail: Liam.Caffrey@nuigalway.ie

FZI – Forschungszentrum Informatik

Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany

Tel: +49 721 9654816
Fax: +49 721 9654817

Contact person: Adreas Abecker
E-mail: abecker@fzi.de

Intelligent Software Components S.A. (iSOCO)

Francisco Delgado 11, 2nd Flor
28108 Alcobendas, Madrid, Spain

Tel: +34 913 349797
Fax: +34 913 349799

Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

OntoText Lab.- Sirma AI Ltd. (SAI)

OntoText Lab.
38A Chr. Botev Blvd.
Sofia 1000, Bulgaria

Tel: +35 92 9810018,
Fax: +35 92 9819058

Contact person: Atanas Kiryakov
E-mail: Atanas.Kiryakov@sirma.bg

Hewlett Packard (HP)

HP European Laboratories
Filton Road, Stoke Gifford
BS34 8QZ Bristol, UK

Tel: +44 117 3128631
Fax: +44 117 3129285

Contact person: Janet Bruten
E-mail: janet.bruten@hp.com

Associated Partner:

British Telecommunications plc. (BT)

Orion 5/12, Adastral Park
Ipswich ip5 3RE, UK

Tel: +44 1473 609583
Fax: +44 1473 609832

Contact person: John Davies
E-mail: john.nj.davies@bt.com



	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 3 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public


Table of Contents

1	Introduction	6
2	Usage Scenario	8
2.1	Types of actors.....	8
2.2	Locating services by human users	8
2.3	Querying details about services.....	9
2.4	Service registration	10
2.5	Automatic Service Registration.....	10
3	Service Modelling Ontology.....	12
3.1	Modelling services using ontologies	12
3.2	Describing information at different levels of abstraction.....	14
3.3	Service modelling ontology structure.....	14
3.4	Domain ontology	15
3.5	Business partner ontology	15
3.6	Service capabilities ontology.....	16
3.7	Service modelling ontology	17
3.8	Low-level description of services	18
4	Query Interface.....	20
4.1	Approaches to Service Description Querying.....	20
4.2	Navigation as a Complement for Queries.....	20
4.3	Instance querying	21
4.4	Template queries.....	23
5	Service Registry.....	25
6	Obtaining Service Description Semi-Automatically	26
6.1	Obtaining descriptions from UDDI	26
6.2	Obtaining descriptions from the Web.....	27
7	Conclusion.....	29
8	References.....	30

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 4 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public


List of Figures

Figure 1. High-Level Tasks of Service Discovery Component.....	6
Figure 2. Structure of Ontologies Involved in Service Modelling	15
Figure 3. Business Partner Ontology	16
Figure 4. Service Ontology Structure.....	18
Figure 5. Result of an Example F-Logic Query	20
Figure 6. Service Registry Architecture	25
Figure 7. Correspondences among UDDI Model and Service Modelling Ontology	27

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 5 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

List of Acronyms and Abbreviations

Acronym/abbreviation	Resolution
DAML+OIL	... a semantic markup language for Web resources http://www.daml.org/2001/03/daml+oil-index.html
KAON	Karlsruhe Ontology and Semantic Web Tool Suite http://kaon.semanticweb.org/
OWL	Web Ontology Language http://www.w3.org/TR/owl-ref/
RDF(S)	Resource Description Framework (Schema) http://www.w3.org/RDF/
RDFS-MT	RDF Model Theory Semantics http://www.w3.org/TR/rdf-mt/
UDDI	Universal Description, Discovery and Integration of Web Services http://uddi.org/
WSDL	Web Service Description Language

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 6 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

1 Introduction

Service discovery component is of paramount importance to supporting semantic Web services in practice. Adopting a parallel with the classical, non-semantic Web, the discovery component is roughly similar in role and functionality to search engines, such as Google, providing means for searching the information space. We explain the functionality of the service discovery component on a very simple, but general example.

In order to enable service consumers to reuse a service, the service provider must first describe his service using some formal language. This description should capture the semantics of the service as closely as possible – this enables easier selection of the service later by potential service users.

After the service description is ready, the service provider must store it in some public repository. In such a way the service description is made persistent and made available for the general public.

Some service consumer will query the persisted descriptions in order to locate the service that may accomplish his task at hand. In order to obtain only those services that really fulfil his task, he relies on the formal semantics in the description of the service. He issues a query that formulates as close as possible his needs formally. The query is then matched against the formal service descriptions and corresponding services are located.

This simple, yet effective example of the functionality of the service discovery component within SWWS is presented schematically in Figure 1. The figure highlights the basic tasks which are performed and are related to the discovery, from which we derive the following subcomponents of the service discovery component:

- The **Service Modelling Ontology** provides a way of describing services in a formal way. Its main role is to provide means for capturing the semantics of the service.
- The **Service Registry** is responsible for persisting and managing a large number of service descriptions.
- The **Query Interface** provides a way for querying registered services.

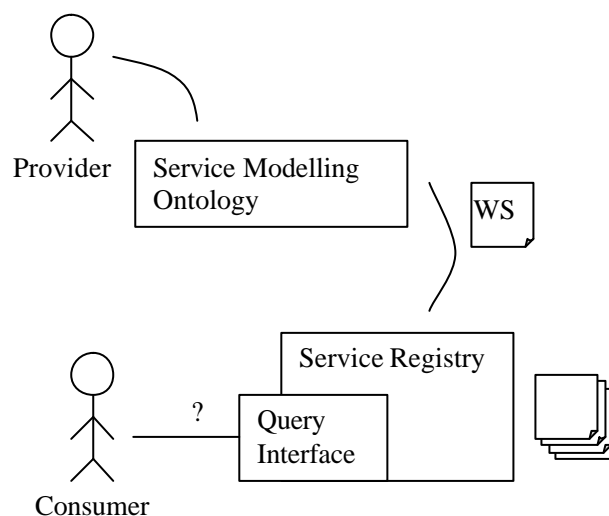




Figure 1. High-Level Tasks of Service Discovery Component

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 7 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

In order to have impact on the state-of-the-art of service description standards, the service discovery component should not start from scratch and neglect existing standards. In particular, the UDDI standard has become prominent in industry. It offers simple capabilities for tagging services with keywords and associating them with information about the business partners. Based on these observations, one goal of the service discovery component would be to extend the UDDI model with semantics, rather than replace it with a new and incompatible model.

Apart from the three main components mentioned in the above discussion, producing Web service descriptions manually is a tedious and error-prone process. Hence, it makes sense to investigate whether and how existing registries of Web service information, such as UDDI registries, may be reused. In particular, a mechanism for enriching non-semantic service descriptions with semantics is needed. In that light, we identify these two additional components:

- The ***UDDI Integration Engine*** provides a way for extracting service descriptions from existing UDDI repositories.
- The ***Profile Crawler*** provides a way for extracting service descriptions from the Web.

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 8 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

2 Usage Scenario

In this section we present a more concrete usage scenario which will drive our architecture design in the following section. The usage scenario is a rather typical B2B integration problem.

Let us assume that there is some company A, which specializes in production of sports utilities for adventure sports, such as climbing. Hence, it produces items such as 'climbing rope' or 'harness'. The goal of the company A is to participate in an electronic marketplace, in hope of increasing its productivity through the automation potential arising from that. The business activities that A participates in are typical in the manufacturing industry and include taking an order, shipping the goods and collecting payment.

Let us assume that B is a big sports utilities store. B also wants to increase its productivity and automate business processes as much as possible. Business activities that it participates in include monitoring the stock, ordering items that are out of stock, receiving the shipment, payment, and searching for new manufacturers.

In order to automate some of their business processes, both A and B create several Web services, by means of which it is possible to conduct some of the business activities. There are certainly some activities which cannot be automated in this way (e.g. shipping involves human interaction). However, many activities, such as placing an order, payment processing or locating the business partner of interest are prime candidates for automation using Web services.

Since this document describes the service discovery architecture, we shall focus on explaining the requirements related to service description and discovery. Since the basic requirement on the service discovery component is locating services, we start with use cases related to searching for services. Based on these, we derive the requirements on service description. Although description must be done before searching, it seems reasonable to expect that the requirements on locating the service will determine how the service must be described in the first place.

2.1 Types of actors

In descriptions of use cases we use three types of actors:


- **Service provider** – this actor represents the entity or person which creates a service and makes it available to others for usage. An example of a service provider is the company A which produces sports utility equipment.
- **Service consumer** – this actor has a business need which might be satisfied by using some web service. An example of a service consumer is the company B which buys sports utility equipment from the providers. A subtype of this actor is the Human service consumer.
- **Registry manager** – this actor is responsible for managing a service registry.

2.2 Locating services by human users

Actors: Human service consumer

Use case flow:

1. The actor formulates the query using the constructs of the query API.

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 9 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

2. The actor submits the query to the service registry.
3. Service registry locates the services matching the conditions of the query.
4. Service registry returns the collection of located services to the actor.

Examples queries:

*What are the services that **sell** pieces of **climbing gear**?*

*Where can I **buy** the **climbing rope** which is **produced by** the company **Mammut**?*

*Where can I **submit an order** for **100 karabiners produced by** the company **Petzl**?*

*Which service can I use to **check** the **status of my order**?*

Discussion:

A human will typically want to locate a service by specifying features that the service must fulfil. This specification is usually not formal and does not contain technical details on how the service should be invoked. Rather, the query specifies in general terms what the service should accomplish. In the example, the user is not interested in knowing the exact sequence of operations that must be performed to submit an order or to check the order status. The details may include supplying various parameters, such as details about the items ordered, or about the order the status of which is being checked. These details are not important in this use case. Rather, it is important to define the semantics of various concepts common in business, such as the concept of buying, selling or checking the order status. Further, it is important to define the semantics of the domain vocabulary. Such queries are very important to be able to broadly navigate through the space of offered services without getting lost.

2.3 Querying details about services

Actors: Service consumer

Use case flow:


1. The actor formulates the business goal that needs to be fulfilled.
2. The actor builds an execution plan which fulfils the plan.
3. The actor contacts the service registry to query for services which may fulfil certain steps in the plan.
4. The registry responds with the collection of services that may fulfil the task.
5. The actor examines the result and if needed, continues with step 3.

Examples queries:

*What information I need to **check** the order status?*

*What other things must I do to **place an order** for **100 karabiners**?*

*What is the process of **order fulfilment** by the company offering this service?*

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 10 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

Discussion:

The flow of this use case is similar to the flow of the use case “Locating services by human users” as it is also concerned with querying the service registry. However, queries and the context in which querying is performed, are radically different.

The overall context is different since the actor is trying to build a plan for fulfilling a goal. Therefore, the high-level information about the service is not sufficient. Rather, information about which steps need to be performed in an interaction with the service, as well as what parameters must be transferred in each step, must be described. Hence, this use case reveals the need for a completely different level of granularity of service description.

2.4 Service registration

Actors: Service provider

Use case flow:

1. The actor chooses appropriate vocabulary for service description.
2. The actor builds the service profile.
3. The actor submits the profile to the service registry.
4. The system registers a service.
5. The system provides a confirmation of the registration.
6. The actor makes the profile available on-line.

Discussion:


This use case reveals the need to manage the profiles of each service. Further, a need for referring to a common vocabulary is needed.

2.5 Automatic Service Registration

Actors: Registry manager

Use case flow:


1. The actor specifies several entry points in the information space by giving the addresses of pages describing interfaces of Web services.
2. The system obtains on-line profiles for the services.
3. The system enriches obtained information with semantics.
4. The system stores located profiles into the registry.
5. The system examines the available links that lead to other profiles.
6. The system repeats the step 2 starting from these links. The actual algorithm for remain to be specified.

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 11 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

Discussion:

This use case reveals a need to collect service profiles from on-line resources. These resources may include e.g. WSDL pages describing interfaces of Web services. Another type of information source may be a UDDI registry.

The exact mechanisms for adding semantics to the obtained descriptions remain to be specified. The basic idea is to try to use a natural language processing and information extraction techniques for aligning service interface descriptions against a domain ontology.

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 12 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

3 Service Modelling Ontology

Based on the requirements from the previous section, in this section we elaborate the aspects of the service modelling ontology that we consider relevant for the service discovery component. It is important to understand that defining such an ontology for the SWWS project is not within the scope of the service discovery component. In particular, such an ontology will be delivered in work package 2 in the context of the ongoing work on the Web Service Modelling Framework, which will result in the deliverable D2.3. In this section we merely focus on the aspects of the service modelling ontology which we identified as important from the standpoint of service discovery. In that light, we harmonized the terminology with the one used in deliverable 2.1 as much as possible. Still, there are some differences in the terminology and conceptualization which will be resolved as the work on creating the actual service modelling ontology progresses.

3.1 Modelling services using ontologies


As may be inferred from the name, we believe that services should be modelled using a suitable ontology formalism. This has numerous advantages, from the standpoint of service discovery:

- Ontologies provide a vocabulary for modelling knowledge in a limited domain. They are created by achieving consensus within a community of interest. In such a way, the community lays the foundations for seamless knowledge interchange.
- Ontology languages are usually equipped with formal semantics, typically founded in mathematical formalisms, such as model theory or first-order logic. This in turn unambiguously specifies the meaning of various constructs of the ontology language. Based on these definitions, it is possible to infer new information from explicitly present information.
- Common vocabulary and mathematical specification of semantics open the way to automatic information processing. In such a way the information is not only understood by humans, but may be (partially) understood by machines as well.

Since ontologies have proven themselves as useful for modelling semantics, we propose modelling service descriptions as instances. This allows us to reap the benefits of the ontology technology, such as inferencing, in the context of service discovery.

Many ontology formalisms have been developed, differing often in the choice of the fundamental logical formalism. An overview of the major formalisms has been given in [1]. Many ontology languages (e.g. F-Logic) are rooted in Horn logic. This puts certain boundaries on the expressivity of the language. For example, in Horn logic it is not possible to reason about indefinite disjunctive statements (e.g. statements such as 'this service is offered by either company A or company B, but we are not really sure which' cannot be made in F-logic). Also, an important drawback of the whole class of languages is the Horn logic is known not to be decidable. If a logic is undecidable, this means that a complete algorithm for query answering in such a logic does not exist – either evaluation of some queries will not terminate, or some queries will be evaluated partially. In the latter case this means that the query evaluation algorithm in some cases does not return all correct answers to a query and that there is no general way of estimating the effects of this incompleteness.

In light of these difficulties, OWL [4] has been recently proposed as the language of choice for applications in the Semantic Web. OWL is actually a syntactical variant of the SHOIN(D+)

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 13 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

description logic. Description logics are a family of well-researched knowledge representation formalisms. A common denominator of most description logics is the existence of a sound, complete and decidable inference procedure. Hence, each query can always be answered correctly. Further, the trade-offs between expressivity and computational complexity have been studied for almost each class of the logic. The description logic that OWL is based upon offers a significant degree of expressivity, including reasoning with negative and disjunctive statements, as well as limited reasoning with equality. Further, OWL comes in three flavours:

- OWL-Lite is a simplified language which has been created with the goal of being implemented easily. In reality, the level of expressivity offered by OWL is still relatively high and requires advanced implementation techniques.
- OWL-DL includes the full SHOIN(D+) description logic. Implementing it requires advanced techniques, such as tableaux calculus.
- OWL-Full goes beyond SHOIN(D+) and blurs the distinction between classes and instances. Whereas in OWL-DL (and OWL-Lite) some symbol is either a class or an instance, in OWL-Full something can be a class and an instance at the same time. However, this comes at the expense of a non-standard formal semantics, whose consequences have not been studied in detail yet.


Since the activity in the SWWS project is closely related to the Semantic Web initiative, it is quite natural to use OWL as the ontology formalism. However, there is a practical problem related to this choice. Namely, at the time this document was written, the tool support for OWL was quite limited. Existing tools can be separated into two classes:

- *Inference engines for description logics*, of which the most prominent example is RACER. Such tools usually provide a sound and complete inference procedure for (most of the features of) OWL. However, the performance of such tools is usually quite limited, in particular in case of ontologies containing a large number of instances.
- *Tools evolved out of the RDFS initiative*, of which examples are Jena and Protégé. Such tools usually provide reasonable performance. However, they often support a very limited subset of OWL-Lite, and the support for OWL-Lite is not at the maturity level that is desired.

However, an alternative to OWL might be the language presented in [1] and implemented in KAON¹ – an ontology management system developed at FZI and AIFB at the University of Karlsruhe. The ontology language of KAON has been designed with typical business requirements in mind, such as persistence and scalability. Whereas the detailed discussion of the language is out of scope, we just outline some main features of KAON.

Briefly, the ontology language is based on RDF(S) [2], but with clean separation of modelling primitives from the ontology itself (thus avoiding the pitfalls of self-describing primitives such as subClassOf), thus departing from RDFS-MT [3] semantics. Further, several commonly used modelling primitives have been incorporated into the ontology model, such as transitive, symmetric and inverse properties. In order to preserve tractability and enable ontology

¹ <http://kaon.semanticweb.org/>

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 14 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

evolution in our approach, we treat property domain and range specifications as constraints, not as axioms. We found this view to be intuitive and desired by most users having a strong background in object-oriented and database technologies. A distinguishing feature of our model is the explicit support for modelling meta-classes and explicit modelling of lexical information. All information is organized in so-called OI-models (ontology-instance models), containing both ontology entities (concepts and properties) and their instances². This allows grouping of concepts with their well-known instances into self-contained units.

We consider the question of which ontology language to use and which tools can support the chosen language not definitely answered by the SWWS consortium. These choices have a moderate impact on the service discovery component. The most of material presented in this document tries to be agnostic of the ontology language and/or the tool platform.

3.2 Describing information at different levels of abstraction

As already contrasted in use cases “Locating services by human users” and “Querying details about services”, we anticipate that services will be queried with different goals in mind, therefore requiring different levels of abstraction:

- One should be able to describe the high-level capabilities of the service. This includes saying *what* the service does in broad terms, rather than *how* the service does it.
- One should be able to describe the low-level capabilities of the service. This includes specifying exactly the *actions* that the service may perform along with the *parameters* needed for the action.


The support for these two levels of abstraction must be reflected in the service modelling ontology. In a way, the low-level description of the service depends on the high-level description. Because of that, we consider defining the higher level of abstraction more important for the SWWS project and the service discovery component.

3.3 Service modelling ontology structure

In this section we present the next level of details related to the structure of the service modelling ontology, as required by the service discovery component. We have identified the need for the following type of information:

- **Domain ontology** – A domain ontology provides the general vocabulary about things from the business domain in which services are used. In the scenario from section 2, the domain ontology would define terms such as ‘climbing gear’, ‘rope’ and ‘harness’.
- **Business partner information** – Obviously, information about who is offering a service is very important for the service discovery. It is easy to imagine a use case where an important criterion for locating a service is the name of the business partner offering a service. This information is generally canonical; however, it might also contain links to the domain ontology. For example, description of the company A may reflect the fact that this company produces ‘climbing gear’.
- **Service capabilities** – A central criterion for locating services is what the service really does, i.e., what the capabilities of the service are. In order to express this, a

² In the rest of this paper we use the terms OI-model and ontology interchangeably.

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 15 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

taxonomy of standard capabilities is needed. From the point of service discovery, it should define the meaning of general concepts such as ‘buying’, ‘selling’ etc. This taxonomy may capabilities at different levels of granularity. In fact, it should get specific enough and define the meaning of ancillary information needed to really perform the action and (e.g., to sell a book, the service will need a payment method) and specifies what is obtained by performing an action (e.g., after the book is sold, the service sends the buyer an invoice).

In order to keep the service modelling ontology modular and maintainable, it is necessary to build it modularly. In particular, we propose to build a separate ontology for each area of concern (domain ontology, business partner information and service capabilities) and to integrate them in one service modelling ontology. As discussed in [5], a good approach is to keep each ontology autonomous and rely on inclusion facilities of the underlying ontology management platform to link the ontologies at run-time. Hence, Figure 2 shows a way how these ontologies might be assembled, where the lines represent the inclusion dependencies.

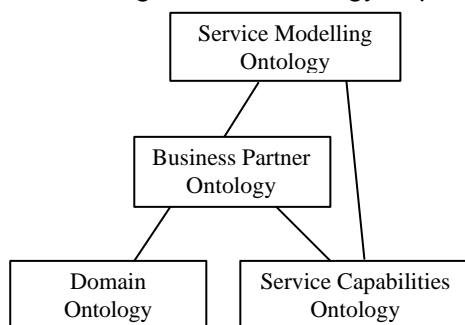
3.4 Domain ontology

As already explained, the role of the domain ontology is to provide the vocabulary for a business domain. A lot of work has been devoted to creating and managing ontologies, and this work is immediately applicable to service discovery.

For example, in [5] an infrastructure for managing domain ontologies in a distributed setting was discussed. In particular, the work addresses the problem of reusing ontologies in situations when they are developed by multiple institutions which are distributed on the web. The proposed approach solves the problem of reusing distributed ontologies by replicating the included ontology to the node where the ontology is included. Replication introduces the problem of keeping the replicated ontology in synchrony with the original ontology – if the original ontology changes, these changes must be propagated to all replicas on the Web. The proposed approach deals with this situation by keeping an evolution log for each ontology, containing information about each change performed. This log may be used to replay changes to each replica on demand. We see this approach being immediately applicable to service discovery component of the SWWS project.


3.5 Business partner ontology

The role of the business ontology is to capture information about business partners actually providing services. The possible design of this ontology is presented in Figure 3. In the rest



Legend: A — B ontology A includes ontology B

Figure 2. Structure of Ontologies Involved in Service Modelling

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 16 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

of this subsection we briefly discuss the various parts of this ontology.

The business partner ontology follows the ‘Party’ analysis pattern presented in [6]. This pattern is based on the observation that entities involved in business may be individuals, but are often also groups of individuals. Hence, a general ‘Party’ term is brought into the object model which is then partitioned into groups and individuals. Groups are represented as consisting of several ‘Parties’. In such way hierarchical organisational structure may be represented. In our case, the ‘Party’ analysis pattern is applied as follows:

- The concept ‘Business Partner’ represents the general notion of a business partner, regardless of whether it is a single entity or a group.
- The concept ‘Group’ represents a group of business partners. Members of a group are represented using has-members transitive property.
- Various types of individual business partners are represented as subconcepts of the ‘Business Partner’ concept. We did not include an intermediate subconcept capturing individual partners, since from our requirements for the time being this need is not clear and we wanted to keep the ontology as simple as possible.

Other information is canonical:

- Each business partner has some name.
- Each business partner has a contact address.

In case of a ‘Group’, one may specify a separate address for the group as a whole. In later stages of the project we plan to include a mechanism for designating the address of one group member as the main address of the group.

3.6 Service capabilities ontology

The role of the service capabilities ontology is to define the taxonomy and the structure of common capabilities that recur in business scenario, such as buying and selling. It is important to understand that the role of this ontology is not to define the semantics of each individual operation of the Web service. Rather, the goal to define the functionality offered by the Web service in an abstract way using the traditional business vocabulary.

In order to be interoperable in the context of the semantic Web, the taxonomy of capabilities is obviously needed. Only by reusing capabilities from a catalogue of common capabilities,

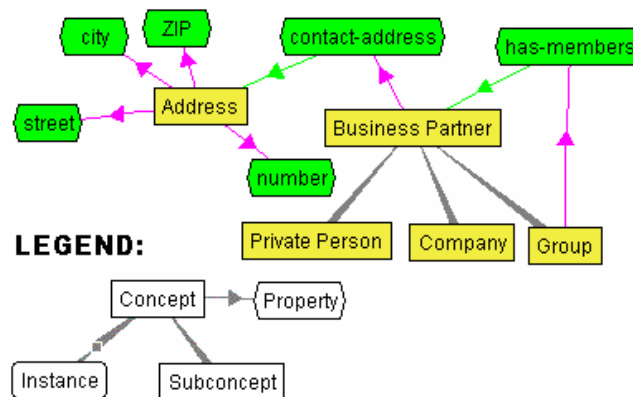



Figure 3. Business Partner Ontology

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 17 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

we see it possible to locate services in a semantically meaningful way. Hence, it makes sense to investigate existing similar taxonomies/catalogues and to evaluate the reuse potential.

One such existing catalogue of common business activities is the MIT Process Handbook³, which is being built within the Center for Coordination Science⁴ by the MIT Sloan School of Economics. The goal of the Process Handbook is to provide a repository of the structure of typical business processes. For example, the Process Handbook defines the following five top-level business activities:

- Buy,
- Make,
- Sell,
- Design and
- Manage a business.

Almost all business processes involve one or many of these fundamental activities.

Further, the Handbook contains a repository of more coarse business models which are organized around these activities. For example, 'Produce as a business' is the most high-level business model, under which many other models are defined. An example of a concrete business model is 'Produce as a Creator'. In this model a certain business partner obtains 'Raw Materials' from 'Suppliers', transforms them into a 'Product' and sells them to the 'Buyer'.

Since the Process Handbook defines the basic business activities, it makes sense to reuse as much as possible of it for defining a common vocabulary of capabilities. The main hurdle in accomplishing this task is the fact that the Process Handbook is not presented in a format compatible with any of the ontology language standards. Therefore, it cannot be readily reused for service discovery. Rather, an effort must be invested in selecting the relevant part and then transforming it into the ontology format.


3.7 Service modelling ontology

The service modelling ontology is the point for where all information about a service is merged. It provides the vocabulary needed for describing services and linking them to capabilities, business partners and objects in the domain ontology.

Our proposed structure of the ontology is presented in Figure 4. The central concept in the ontology is the 'Service' concept, representing one Web service. Each service is named by means of the 'service-name' attribute. Services are associated with a business partner by means of the 'provided-by' property. Normally, the service modelling ontology should provide a way to link services with the respective groundings. However, since this information is not crucial to service discovery, we do not further elaborate this topic.

³ <http://ccs.mit.edu/ph/>

⁴ <http://ccs.mit.edu/>

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 18 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

As already explained, instead of focusing on the operations that services provide, each 'Service' may be associated with a capability through the 'has-capability' property. Thus the semantics of the service may be described in terms of a well-known vocabulary.

3.8 Low-level description of services

As already mentioned, our current focus was to provide means for high-level description of the semantics of services. However, we are aware that the specification of semantics of low-level service operations is also very important in practice. In particular, it is indispensable for (semi-)automatic service composition – in order to join several services with the goal of solving a task at hand, the exact information required and provided by the service to perform its task.

WSDL standard provides a starting-point for this task. In its current version 1.2, it provides means of defining abstract service interfaces as a set of operations. Each operation may involve different types of abstract messages, representing input and output parameters of the operation. These abstract operations are linked to a concrete protocol through the notion of protocol binding. This is very similar to the separation between the ASN.1 (Abstract Syntax Notation One) and transfer syntax developed within the OSI reference network model.

Before discussing what in our opinion is needed in WSDL to describe semantics in more detail, we consider an analogy between service composition and a well-known monkey and banana planning problem (presented e.g. in [8]). By examining this analogy, we hope to elicit additional requirements on the specification of semantics of Web services.

The problem presents a monkey which is closed in a room with a banana hanging from the ceiling. The room is filled with various objects. The monkey can move objects around or climb on top of them. The problem is to devise a sequence of monkey's actions which will enable him to reach the banana.

Since search for the solution is the fundamental technique for solving complex problems in artificial intelligence, it makes sense to formulate the presented problem as a search problem. In particular, the search space consists of all possible states of the world, with different objects being in different positions. Transitions among elements of the search space

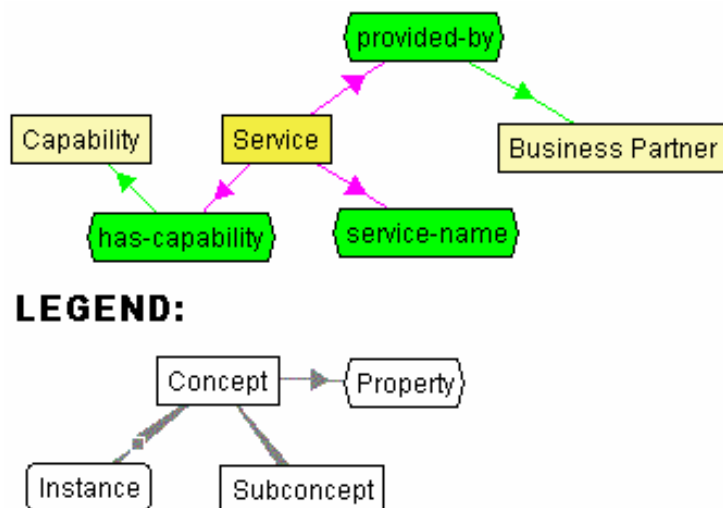



Figure 4. Service Ontology Structure

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 19 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

are done by actions which move different objects around. The sequence of transitions represents an action plan. The goal of the search is to discover an action plan leading to the state of the world where the monkey has gotten hold of the banana. From this solution we may identify two important components of all planning problems:

- A description of inputs and outputs of service operations are needed.
- A description of the state of the world is needed.
- A description of how this state is changed by individual actions is needed.

Automatic service composition is very closely related to planning: one wants to discover a sequence of actions (a plan) leading to the completion of some goal. In fact, service composition is closely related to automatic program construction which has been investigated e.g. in [7]. Automatic program construction is in fact an instance of a planning problem: the goal is to discover the sequence of program instructions that will for all input problems modify the state of variables in the correct way to generate desired output.


Even for simple problems, it has turned out that describing the state of the world is a very difficult task. There are simply too many variables that need to be taken into account. Also, the search space grows usually very large. We conjecture that this will also be the case for Web services, where the state of the world will need to include numerous business variables of all different partners.

From this analysis we conclude that much more effort is needed in providing a flexible framework for representing the effects of operations of Web services. One possible direction which we are investigating is extending WSDL operation specification by preconditions and postconditions.

A *precondition* is a logical formula which must evaluate to true before the service is being called. For example, before an order may be placed, each item in the order must be in stock.

A *postcondition* is a logical formula which must evaluate to true after the service has been called. For example, after an order has been placed, the order confirmation is created and sent to the customer.

In particular, the postconditions often describe the effects of an operation or a service in terms of the change between the state of the world before and after the service call.

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 20 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

4 Query Interface

The role of the query interface is to provide primitives for answering queries over service descriptions. It is functionally implemented by the service registry. It is beneficial to conceptualize it separately – the primitives and the expressivity of the query interface may and should be analysed independently of the implementation.

The discovery query interface is largely defined by the query facility of the underlying ontology management infrastructure. In this section we only briefly outline its structure.

4.1 Approaches to Service Description Querying

We base our approach for service description querying on the observation that a query can be seen as a function which, when applied to a data model, instantiates some results. It is beneficial if the source and target models of queries are the same. In such way, queries can be composed functionally, giving the user a considerable degree of freedom.

In the case of relational data model and relational algebra, this condition is fully satisfied, as the source and target models of algebra queries are both relational models, thus allowing the user to combine queries arbitrarily (e.g. through nesting). On the other hand, the relational model does not directly represent the semantics of modelled information and is thus not applicable to ontology modelling.

In other cases, e.g. in F-Logic [9], the source model of queries is the ontology model, but the target model is a list of variable bindings for which the query is satisfied in the model. We see several drawbacks to this approach. For one, F-Logic queries cannot be composed functionally, resulting in a more complex query language. Further, F-Logic queries destroy the semantics of the information. For example, the query selecting all services and their capabilities, formulated as `FORALL S,C <- S:Service[has-capability ->> C]`, will produce a result as given in Figure 5.


We may observe that WS2 has written business actions it can perform, but the query result reflects this fact by repeating the value WS2 multiple times for S. The explicit information that there is an n:m relationship between services and capabilities has been lost in the query process, and is now available only by analyzing the intent of the query. This relationship may be reconstructed from the query result by collecting all values of C for which S is the same. In our view this makes the interpretation of the query external to the query itself, which does not follow the general desire to represent the semantics of information directly.

4.2 Navigation as a Complement for Queries

To remedy the deficiencies mentioned above, we propose a dual solution. The capability of querying for collections of objects is complemented with the possibility of navigating within

S	C
WS1	Buying
WS2	Buying
WS2	Selling

Figure 5. Result of an Example F-Logic Query

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 21 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

the model. Hence, our query language offers two fundamental primitives:

- **Instance querying** – This primitive provides the means of selecting a set of instances from the ontology matching some specified condition.
- **Ontology navigation** – This primitive provides the means of following links between ontology entities in order to retrieve necessary information.

Hence, the query for all services and their capabilities would be executed in two stages:

- Using the instance querying primitive, the set of all services would be retrieved.
- For each element of the query answer, the navigation primitive would be used to traverse the “has-capability” link.

One may think of it in this way: instance querying primitive provides entry points, while retrieval of all ancillary information to each entry point is obtained through the navigational primitive.

However, there is a significant performance drawback in the specified approach. If there are n documents, then there will be $2*n+1$ query requests issued (one for retrieving all documents and the two queries per document), which will clearly be a major cause of performance problems. Further, the order in which the information is retrieved is predefined by the order in which the navigational primitives are issued. This prevents the usage of various query optimization strategies that are considered to be the key factor for the success of relational database systems. There seems to be a mismatch between the desire to retrieve as much information as needed on one side (to reduce communication overhead and to allow for optimizations) and to retrieve information using navigation on the underlying model (in order not to destroy the model’s semantics).

To avoid such problems, we apply the following approach. Our queries can still be treated as concepts and return individuals or individual pairs at most. However, with each main query we allow specifying several additional queries. These queries are not part of the query result, but they are used as hints specifying which information is needed as well. Then a query unnesting technique is applied to transform the set of these queries into one large query retrieving all necessary information. Such a query can be efficiently executed. After the results are fetched, the query nesting is applied, which transforms the result into the original nested structure. For example, the query from the subsection 4.1 might be executed with the following pseudo-code, where query primitives are marked in bold:


```

services := get all services
for each s from services
  caps := get values of has-capability for s
  for each c from caps
    // so something with c
  end
end
end

```

4.3 Instance querying

In this section we present in more detail our approach for instance querying. An important aspect of a query language for service description is ease of use. Our design of the query language is therefore guided mainly by that requirement.

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 22 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

Most existing query languages allow navigation in the data model by using variables for expressing complex conditions over the data model. Although not immediately apparent, this feature underlies even the most prominent database query language SQL. For example, in a schema with service descriptions stored in the `Services` table and their capabilities stored in the `Capabilities` table, one might select all services and their capabilities in the following way:

```
SELECT S.Name,C.Name
      FROM Services S,Capabilities C
      WHERE S.Name=C.ServiceName
```

Although not immediately apparent, names `S.Name`, `C.Name` and alike are actually variables. In a Prolog notation, this query would be written as

```
Q(S,C) :- Services(S),Capabilities(S,C)
```

Whereas such approaches to querying are quite flexible and provide a significant degree of expressivity, they are also quite complicated to understand. Even experts often have problems with tracking the conditions on various variables.

On the other hand, the description logics provide a more limited, but still quite powerful approach to information querying that is based on easily understood set operations. Queries in description logics are built using well-known operations of union, intersection and complement. Apart from that, there are numerous set constructors that allow asking for sets of instances fulfilling certain conditions. Let us demonstrate this on an example. The business partner ontology might contain concepts `Buyer` and `Seller` denoting all business partners who are buyers or sellers, respectively. However, some business partners may be buyers and sellers at the same time, so it is reasonable to pose a query for both of them. This might be done in the following way:

```
[#Buyer] AND [#Seller]
```


The above query should be read in the following way: *“The result of the query is the set of things which is obtained as intersection of all buyers and sellers”*. The Prolog version of the query is

```
Q(X) :- Buyer(X),Seller(X)
```

Whereas this does not seem to be significantly more complicated than the previous query, the query is usually interpreted in a more complicated way: one needs to deal conceptually with the fact that both predicates use the same variable `x` and to understand that the result of the query are those values of `x` for which, when the query is instantiated (that is, when `x` is replaced with the value), the query evaluates to true. In our opinion it is much simpler to conceptually comprehend the notion of set intersection than the notion of instantiation of the query for some values of variables.

Another, more complicated example is a query selecting all services whose capability is related to selling. In this example we assume that there is a concept `Service` linked to a capability through the property `has-capability` to a concept `Capability`. Further, we assume that `Selling` is a subcapability of the `Capability` concept. Such a query might easily be expressed as follows:

```
[#Service] AND SOME(<#has-capability>,[#Selling])
```


	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 23 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

The query should be interpreted like this: *“The result of the query is the set of things obtained by intersecting the set of all Services and the set of things that are linked through has-capability property to an instance of the Selling concept”*. In the interpretation of the query the only additional thing that must be conceptualized is the notion of a set of things which are linked to an instance of some other concept.

However, the Prolog query looks like this:

```
Q(X) :- Service(X),has-capability(X,Y),Selling(Y)
```

Again, to interpret this query correctly, it is necessary to conceptually follow the links of variables among various predicates.

An even more dramatic example is selecting services whose only capabilities are related to selling. This might be done in the following way:

```
[#Service] AND ALL(<#has-capability>,[#Selling])
```

The only difference to the previous example lies in the fact that the set `ALL(<#has-capability>,[#Selling])` should be interpreted as the set of things for which `has-capability` property points always to an instance of the `Selling` concept. However, expressing such a query in the Prolog style is significantly more involved and requires using negation-as-failure:

```
NotOnlySelling(X) :- has-capability(X,Y),not Selling(Y)
Q(X) :- Service(X),not NotOnlySelling(X)
```

In this case one must use double negation: the `NotOnlySelling` predicate contains all things linked through to `has-capability` property to an instance which is not an instance of the `Selling` concept. Then one selects all services which are not in the `NotOnlySelling` predicate. This is obviously much more involved than the simple set notation that we propose.


4.4 Template queries

The structure of the queries will significantly depend on the structure of the service description ontology. Hence, querying at the ontology layer directly might be too complicated. Rather, plain ontology queries might be lifted to a higher level of abstraction. Further, we anticipate that many service queries will share some common structure. Building such queries from scratch will typically be very tedious, so a way for modularizing common queries is needed.

In order to address these deficiencies, we propose to include a special template mechanism over the ontology query language into the query interface of the service discovery component. This template mechanism would allow defining new query primitives which would express commonly used queries. The mechanism should be modular and extensible, so that it can be adapted to the needs of the application at hand. In the rest of this subsection we describe what such a template mechanism will look like.

It is realistic to expect that many queries will refer to services with the `Selling` capability. To simplify the process of querying, a template for selling services might be defined as

```
SELLING-SERVICE() =
  [#Service] AND SOME(<#has-capability>,[#Selling])
```

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 24 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

Further, some templates might be parameterized. For example, querying for a service offered by a specific business partner may be simplified through the following template:


```
SERVICE-BY-PARTNER(bp) =
  [#Service] AND SOME(<#provided-by>=bp)
```

In this case, now querying for selling services provided by FZI may be done by this simple query:

```
SELLING-SERVICE() AND SERVICE-BY-PARTNER(!#FZI!)
```

The query above hides the complexity of navigating in the service description ontology and is therefore much more suitable for usage by the end users. The service discovery component will take care of expanding this abbreviation into

```
[#Service]
  AND SOME(<#has-capability>,[#Selling])
  AND SOME(<#provided-by>=!#FZI!)
```


	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 25 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

5 Service Registry

The fundamental function of the service discovery component is to locate services through appropriate query interface. It is obvious that, to accomplish this task, a registry of service descriptions is needed. However, querying for services is not the only task of such a registry – for example, persistence and description management are other activities that must be taken care of by such a registry.

In the SWWS project such a registry will be delivered as deliverable 5.1, which will take care of all of the functionality required. However, since service discovery is invariably linked with the functionality of such a registry, in this section we explain the requirements on the registry from the service discovery point of view. In order to keep the discussion complete, we also present a possible architecture of such a registry.

A one possible such architecture is presented in Figure 6. The important component of the registry is to provide a higher-level API for management of service descriptions. In this way, the users of the registry do not operate on the ontology level (and manipulate individual instances), but work on a higher level (and manipulate service descriptions). Internally, though, the service registry is realized on top of some ontology server, which provides features such as persistence or ontology querying and inference.

The service modelling ontology, whose requirements have been stated in Section 3 and which is a significant part of the service discovery is stored in the ontology server subsystem. Further, all individual service descriptions are stored as instances of the service description ontology and are also managed by the ontology server.

The query interface of the service discovery component, as described in the Section 4, is obviously an integral part of the service registry. Hence, a significant amount of coordination related to technical details of how the query interface is physically embedded into the registry will be needed among the partners.

Other two service discovery components, namely the Service Crawler and the UDDI Integration Engine will be realized on top of the registry and will rely on the registry API for management of service descriptions. The actual role of these two components is explained in subsequent sections.

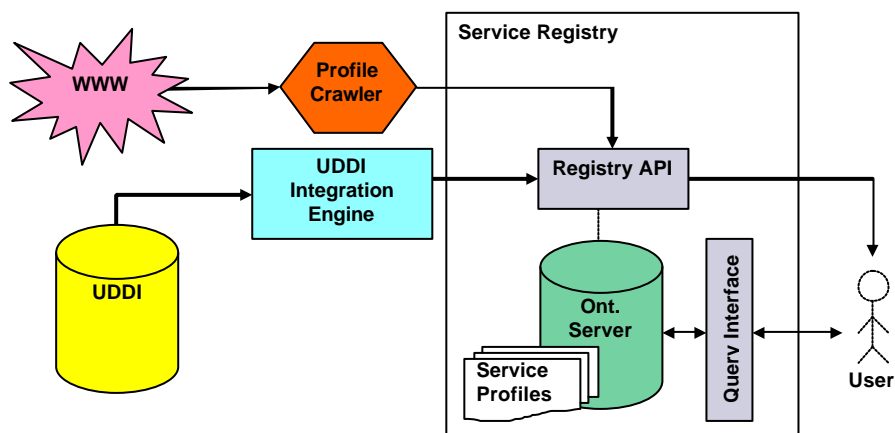



Figure 6. Service Registry Architecture

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 26 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

6 Obtaining Service Description Semi-Automatically

In this section we explain in more detail two subcomponents of service discovery, namely, the UDDI Integration Engine and the Profile Crawler. These two components have the task of filling the service registry with service descriptions (semi-)automatically.

6.1 Obtaining descriptions from UDDI

In this subsection we investigate in more detail the integration of the discovery component with existing UDDI systems. In particular, we explain the role and the function of the UDDI Integration Engine from the Figure 6.

Since the way of describing services as proposed by the SWWS project will be non-standard, it is unrealistic to expect that many services will actually be described using the presented framework any time soon. Hence, the project faces the risk of not having enough data to realistically validate the approach. Irrespective of that, we believe it makes sense to examine whether existing repositories of service descriptions can be reused in some way as starting points for providing semantic descriptions according to the project goals. Therefore, we identified a need for a subcomponent called UDDI Integration Engine.

The main task of the UDDI integration engine is to reuse descriptions from existing UDDI repositories and to enrich them with semantic information. It is clear that this goal cannot be achieved in a way that will be hundred percent satisfactory, as it would require artificial intelligence techniques which are beyond current technology state-of-the-art. In the sequel we analyse the possibilities for integration at different levels.


As explained in Section 3, the design of the service modelling ontology should be compatible by existing standard such as UDDI. Hence, some information from the UDDI registry might be relatively easily mapped to the service modelling ontology. Figure 7 highlights which structural parts of the UDDI model might correspond to different parts of the service modelling ontology. The left side of the figure presents the UDDI data model. It basically consists of three major components:

- White Pages – contains information about business partners,
- Yellow Pages – contains information about services offered by various partners,
- Green Pages – contains technical information about services.

White and Green pages can be matched to the service modelling ontology in a canonical way – White Pages information may easily be converted to the format represented by the business partner ontology, whereas Green Pages can be matched to service grounding information (not currently the scope of the service discovery component). Also, White, Yellow and Green pages are all related to the domain ontology.

The most difficult part of the integration is the Yellow Pages information, which is in UDDI based on the UNSPC classification of possible business activities. These actions correspond to some extent to service capabilities in the service modelling ontology. However, the semantic discrepancies are significant.

Further, the White Pages do not contain information related to preconditions and postconditions, describing the capabilities at the higher level. Hence, we expect that discovery of such services will be impaired.

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 27 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

Whereas the exact ways of attacking this problem are unclear as of yet, we will investigate the following options:

- It might be possible to develop translation schemes for certain classes of UNSPC classifications. These translation schemes will, of course, be highly domain-specific. However, it seems that developing some translation patterns is possible. These patterns would provide typical parts of service descriptions then could then be combined heuristically to enrich UDDI registrations with semantics.
- One might try to analyze the semantics of the WSDL specification of the service's interface. This analysis is obviously a very hard problem. However, some heuristics, e.g. based on analyzing the lexical properties of elements in the specification, complemented by matching them against a background domain ontology, might provide a partial solution to the problem.
- The process of enriching UDDI service descriptions might be semi-automatic. That is, after initial descriptions are imported into the service registry, the user might manually provide the missing elements of the description, possibly by interpreting the WSDL service definition.

6.2 Obtaining descriptions from the Web

Apart from existing UDDI repositories, the Web may serve as the source for Web service descriptions. It is the task of the Profile Crawler subcomponent to collect these profiles from the Web and store them into the registry.

One mode of operation for the Profile Crawler is to collect existing service descriptions by collecting descriptions published on the Web. In this case, the descriptions are already in the adequate format. Hence, the task of the Profile Crawler is relatively simple, as it simply needs to follow the links among Web pages and extract existing descriptions from them.

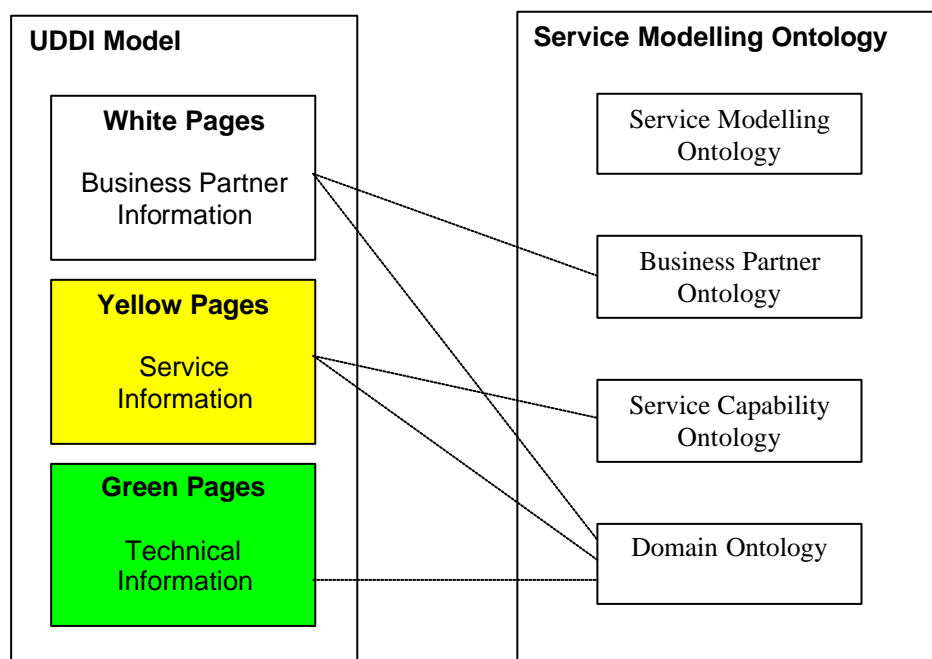




Figure 7. Correspondences among UDDI Model and Service Modelling Ontology

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 28 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

A more advanced mode of operation is obtaining service descriptions by introspecting the interface of the service. In this case the crawler is not provided with an existing, valid description, but tries to extract the description from available information. In particular, we consider the WSDL interface definition as the possible source of information.

For the second case, one may observe that the task of the Profile Crawler is very similar to the task of the UDDI Integration Engine. Hence, the mechanisms for extracting service descriptions will be shared among these two components. Because of that, technically, both components might be realized as one common software module. Still, we introduced both components separately in Figure 6 in order to stipulate that there is some inherent difference among these two components.

	<p align="center">Report on Development of Web Service Discovery Framework</p> <p align="center">Deliverable ID: 3.1</p>	Page : 29 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

7 Conclusion

This document presents use cases for the service discovery component within the SWWS project. On that basis, the document presents the current state of component's architecture. The presentation concentrates on discussing three important architectural elements: service modelling ontology, query interface, the service registry.

This document presented the requirements on the Web service modelling ontology, which provide the formal framework for describing services. A factor we consider important for the service discovery is some degree of compatibility with existing standards, notably UDDI and WSDL. The proposal for the service modelling ontology design makes reasonable efforts to maintain compatibility with either of them.


Further, this document presented the query interface by means of which discovery is actually performed. This interface provides a powerful language for querying service descriptions and is based on an existing ontology query language.

The query interface will be realized within the service registry, which is the focus of the deliverable 5.1. This document presents the requirements and a possible architecture for that registry. In particular, it tries to highlight the requirements that discovery poses for the registry.

The Web service composition provides an approach for (semi-)automatic composition of Web services into agents solving complex problems. The relationship with composition has been briefly analysed in section 3.8. The focus of our current work was on high-level description of Web services, which is not so relevant to composition. However, as mentioned in section 3.8, specifying services at the level of detail which will enable service composition should be the major focus of our future work.

Next steps for accomplishing the SWWS discovery component can be characterized as follows:

- Finalize the service modelling ontology by taking into account the principles outlined in this document.
- Realize the prototype for service discovery.
- Provide a pre- and postcondition language, sufficiently expressive to meet case study requirements, yet being processable for service discovery in a tractable manner.

	Report on Development of Web Service Discovery Framework Deliverable ID: 3.1	Page : 30 of 30
		Version: 1.0 Date: October 06 2003
		Status: Final Confid.: Public

8 References

1. B. Motik, A. Maedche and R. Volz: **A Conceptual Modeling Approach for building semantics-driven enterprise applications**. Proceedings of the First International Conference on Ontologies, Databases and Application of Semantics (ODBASE-2002), Springer, LNAI, California, USA, 2002
2. O. Lassila, R. R. Swick: **Resource Description Framework (RDF) Model and Syntax Specification**. <http://www.w3.org/TR/REC-rdf-syntax/>
3. P. Hayes: **RDF Model Theory**. <http://www.w3.org/TR/rdf-mt/>
4. P. F. Patel-Schneider and P. Hayes and I. Horrocks and F. van Harmelen: **Web Ontology Language (OWL) Abstract Syntax and Semantics**. <http://www.w3.org/TR/owl-semantics/>
5. A. Maedche and B. Motik and L. Stojanovic and R. Studer and R. Volz: **An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies**, WWW 2003, May 20-24, 2003, Budapest, Hungary
6. M. Fowler: **Analysis Patterns: Reusable Object Models**, Addison-Wesley, 1996, ISBN: 0201895420
7. C.L. Chang and R.C. Lee: **Symbolic Logic and Mechanical Theorem Proving**, Academic Press Inc., New York, 1973, ISBN: 0121703509
8. G. F. Luger and W. A. Stubblefield: **Artificial Intelligence: Structures and Strategies for Complex Problem Solving**, 3rd edition, Addison-Wesley, 1997, ISBN: 0805311963
9. M. Kifer and G. Lausen and J. Wu: **Logical Foundations of Object-Oriented and Frame-Based Languages**, Journal of the ACM, vol. 42, pp. 741–843, July, 1995