

# ACE-GIS

## Adaptable and Composable E-commerce and Geographic Information Services

IST-2002-37724



### **D6.2.0 Proposal for extending W3C and OGC standards with semantic modelling capabilities** **D6.2.1 Formal Model Extension to Handle Evolving Semantics**

#### **Project Deliverable**

Date:	2003-12-06
Author(s):	Florian Probst, Werner Kuhn, Patrick Maué
Distribution:	All partners
WP:	6
Version:	1
Keywords:	semantic annotation, ontology architecture, ontology engineering, representation languages
Description:	Work package 6, phase 2, combined deliverables 1 and 2





## Executive Summary

Current GI service composition based on syntactic descriptions such as WSDL is error-prone because the meaning of the labels used in these descriptions is unclear. We identify three types of semantic problems that can result from such semantically heterogeneous descriptions during service composition. We present a three level, two thread ontology architecture to tackle the identified problems. The problems types are illustrated with examples we encountered in the e-Emergency composite service. The tasks which a service provider has to fulfill when building a composite service are related to the problem types and subsequently solutions, the ontology architecture offers, are presented. We justify the representation language we have chosen and document the current state of our ontology engineering effort. To deal with evolving semantics, two implementation threads are introduced to the ontology architecture. Both threads offer different advantages concerning semantic modeling and handling changes.



# Table of Contents

1	Introduction	4
1.1	Work Package Integration	5
2	Semantic Problems Types in the e-Emergency Composite Service	6
2.1	The e-Emergency Composite service	6
2.2	Problem Type I (Naming Heterogeneity)	6
2.3	Problem Type II (Data Type Heterogeneity)	7
2.4	Problem Type III (Conceptual Heterogeneity)	7
3	Ontology Architecture	8
3.1	Application Level	8
3.2	Conceptual Level	9
3.3	Semantic Grounding Level	9
3.4	Formalization Threads	9
4	Semantic Problem Types Related to User Tasks during Service Composition	10
5	Ontology Engineering	12
5.1	Representation Language	<b>Fehler! Textmarke nicht definiert.</b>
5.1.1	OWL - Ontologies for the World Wide Web	12
5.1.2	OWL-S - a Web Ontology for Services	13
5.1.3	Open questions	13
5.2	Application Ontologies	14
5.3	Domain Ontology and Grounding Level	14
6	Evolving Semantics	14
6.1	Mark-up vs. Modeling	14
6.2	Stable vs. changing levels	15
7	Conclusion	16
8	Output to Other Work Packages	16
9	Plans for Phase III	16
9.1	Enable collaboration between different threads of the architecture	17
9.1.1	Haskell-UML mappings	17
9.1.2	Haskell-OWL mappings	17
9.2	Publications	17
10	References	18



## 1 Introduction

The objective of work package 6 is to develop an architecture for semantic interoperability in service composition and to supply its components for semantic modelling and mapping in the ACE-GIS project. This deliverable addresses the questions

- how existing standards for service description can be extended to include semantics (D6.2.0)
- how the algebraic modelling approach described in D6.1.1 can be extended to deal with evolving semantics (D6.2.1).

We decided to combine these two deliverables into one document, as they represent two intertwined aspects of the same architecture: its threads and its levels. The document presents a three level, two thread architecture for semantic modelling. This architecture extends and builds on the Web Service Activity of the W3C (<http://www.w3c.org/2002/ws/>). It establishes the basis for our phase III work in work package 6, where it will be applied to solve semantic interoperability problems in the ACE-GIS pilots.

The phase II goals of work package 6 were to

- enhance composite services and modelling tools with *mechanisms for semantic modelling and mapping*;
- propose extensions to model-based as well as OGC standards providing semantic modelling capabilities;
- investigate how users can be provided with means to adapt semantic models and mappings to their changing needs.

We have extended the original scope of D6.2.1, generalising from "model-based" to "W3C and OGC" standards. The approach taken here adopts the best available standards from W3C and OGC for producing and registering service descriptions.

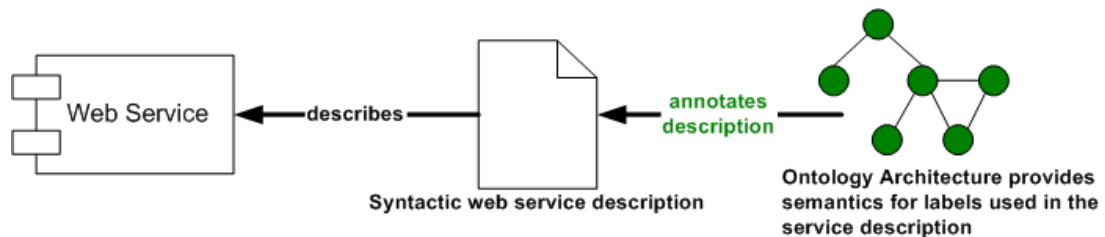
The three levels structure the problem of semantic modelling into levels for human cognition, domain modelling, and application modelling. The two threads capture two kinds of representation with different expressive powers, functional languages and mark-up languages plus UML.

This architecture responds to some fundamental needs posed by the overall target of semantic interoperability:

- a conceptualist approach to semantics (as opposed to one that has no grounding in human concepts);
- a separation of concerns between the modelling phase and that of exposing the results through mark-up languages in the semantic web;
- a separation of the dynamic application semantics from the more stable domain semantics.

Providing semantic translations is out of scope for this deliverable. Before it is addressed in phase III of the ACE-GIS project, we need to solve the problem of referencing the meaning of concepts unambiguously (within the geospatial domain). Semantic mappers or translators will be able to build on this modelling and referencing architecture, in the manner put forward through the vision of semantic reference systems (Kuhn 2003; Kuhn and Raubal 2003).

Figure 1 shows the add-on (or modular) character of the semantic annotations developed in phase II. One important development criterion was to leave the OGC and W3C standards unchanged. We do not intend to extend the existing standards since this would imply to change the generally syntactic character of these standards. Instead our approach focuses on a tiered ontology architecture which can be easily integrated into standard service descriptions thus leading to a more complete service description. This deliverable deals with the development of the ontology architecture and protocols the experiences made with this semantic add-on while integrating it in the e-Emergency pilot.



**Figure 1:** Goal of deliverable 6.2.0. is to develop an ontology architecture for annotating standard service descriptions.

The following short scenario is used to explain in which context we encountered semantic problems and the ontology architecture offers solutions:

A service provider (e.g. the “user” of the ontology architecture) is about to build a composite service for the management of accidents involving toxic gas releases from a chemical plant. He builds the composite service starting with the most specialised service<sup>1</sup> and subsequently adding further services until the composite service meets the user’s requirements. The user already has the central, most specialized service of the composite service available. This is a service for calculating a toxic plume (afterwards called *CalculateGasDispersionService*). The first step is to check the input and output of the plume service. It requires information about the wind speed, the wind direction, the location of the gas emission and the emission rate as input. The output is a polygon indicating the dispersion of the gas. The user chooses as next step to find an additional service providing information on the wind direction and therefore searches for services providing weather information in a UDDI registry (Bellwood, Clément *et al.* 2003). He discovers two services: the *GlobalWeatherService* and the *AirportWeatherService* provided by CapeScience (<http://www.capescience.com>). The user now has to determine whether these services actually match (syntactically and semantically) the requirements of the *CalculateGasDispersionService*. The semantic problems he encounters are explained in chapter 2.

The document is organized as follows: Chapter 2 describes semantic problem types which can occur during web service composition and illustrates these with examples from the e-Emergency composite service. Chapter 3 introduces a three level ontology architecture implemented along two threads. Chapter 4 relates the problems types identified in chapter 2 to tasks a user will encounter in the scenario described in chapter 1. Chapter 5 documents the decision to use OWL as ontology language and describes the current state of the ontology engineering process. The idea of evolving semantics is introduced in chapter 6. Chapter 7 concludes and in chapter 9 our plans for phase III are presented.

## 1.1 Work Package Integration

WP 1b	The e-Emergency Pilot provides the context for the definition of the GasDispersion composite service. This service reduces the complexity of the e-Emergency Pilot. Its reduction to currently six services was a joint effort of the project team allowing to focus on a reduced number of semantic problems.
WP 2	The GI Web Services provided by ionic are part of the composite service. The service descriptions provided in this workpackage are used to identify semantic problems in the composite service and to build application ontologies.
WP 4	Work package 4 provides the infrastructure for the e-Emergency composite service. The workflow descriptions are used to identify semantic problems
WP 5	The UML support provided in this work package clarified the workflow and composition of the GasDispersion composite service. Based on this UML support the project team agreed on the

<sup>1</sup> To keep things simple, we assume that each web service has only one operation. We therefore use the terms web service and operation interchangeably.



	design of the composite service, which is an elementary prerequisite for further identification of semantic problems occurring between the web services used in the composite service.
--	--

## 2 Semantic Problems Types in the e-Emergency Composite Service

Currently service discovery relies on the labels of input and output descriptions and on the labels of data types given in WSDL (or similar service metadata) descriptions. It is generally assumed that if the labels are the same, the transported information is, too. However, this is not always the case. Different types of semantic heterogeneity have been identified for GIS (Bishr 1998) and GI web services in general (Lutz, Riedemann *et al.* 2003). In the following, we present three types of heterogeneity problems that play a role during GI web service composition. Each problem type is illustrated by relating it to the scenario given above and to the services employed in the e-Emergency composite service.

### 2.1 The e-Emergency Composite service

The e-Emergency pilot composite service comprises six services.

1. The getLocation operation of the PreEmergencyPlan service, provided by e-blana (<http://www.e-blana.com/>) transforms a place name into coordinate pair.
2. The GetNearestAirportCode service provided by ionic (<http://www.ionicsoft.com/>) takes coordinates as input and finds the nearest airport to the given location.
3. Alternatively the PreEmergencyPlan service can also find the nearest airport to a given location. Having two services providing the same operation allows showing the adaptability of the composite service. Adaptability in combination with services providing the same functionality ensures that the composite service will produce results even if some of the employed services are temporarily not available
4. The weather services provided by CapeClear return the weather information needed for calculating a plume. There are two services available, the AirportWeather service (<http://www.capescience.com/webservices/airportweather/index.shtml>) and the GlobalWeather service (<http://www.capescience.com/webservices/globalweather/index.shtml>). The AirportWeather service is used in the composite service of the pilot. However, it returns a string containing a weather report. The information needed as input to the Calculate Gas Dispersion Plume service needs to be parsed out of this string.  
In the following examples we additionally employed the GlobalWeather Service for explaining semantic problems.
5. The CalculateGasDispersionPlume service provided by ionic calculates the gas dispersion based on the input provided by the weather service and the leakage details provided by the user.
6. The CreateGasDispersionMap produces a map with several layers one of which is the gasDispersion layer.

### 2.2 Problem Type I (Naming Heterogeneity)

*The output of a web service and the input of a second web service are represented with the same data type and refer to the same domain concept, but have **different** labels (names).*

Example 1:

The *AirportWeatherService* and the *GlobalWeatherService* both provide information about the wind direction. However, the labels of the data types containing the required information are different. The *GlobalWeatherService* refers to the information as `prevailing_direction` while the *AirportWeatherService* labels the information as `wind`. However, both elements represent the same (domain) concept of *wind direction*.

*Example 2:*

Airport codes identify an international airport unambiguously. The PreEmergencyPlan service produces an airport code as output. The AirportWeather service requires an airport code as input. Unfortunately there are two systems of airport codes used to identify international airports, a four letter code is used by ICAO (International Civil Aviation Organization) and a three letter code is used by the IATA (International Air Transport Association).

From the WSDL file of the services, it is not to tell which type of airport code is required. The AirportWeather service merely requires an input part labelled `arg0` of type `String` for its `getWind` operation. The PreEmergencyPlan service provides an output labelled `LocateNearestAir-portResult` of type `String`. The elements of both services refer to information about an ICAO air-port code. Different labels are used to represent the same data type (string) with which information about the same domain concept (ICAO) is transported.

Problems of this type can be solved in two steps. First, each of the elements used in the WSDL descriptions are referenced to the same domain concept. Second, the reference includes additional restrictions on the properties of the domain concept, constraining its meaning. If reference and restriction are identical, the meaning of the used symbol (label) is the same.

### 2.3 Problem Type II (Data Type Heterogeneity)

*The output of a web service and the input of a second web service have the same labels (names) and refer to the same domain concept, but are represented with **different** data types.*

*Example:*

Example 1 given above is further complicated by a second source of heterogeneity. The *GlobalWeatherService* provides the wind direction information represented as a complex type labelled `Direction`. The *AirportWeatherService* provides this information contained in a `String`. However, both elements represent the same domain concept<sup>2</sup>.

This problem is not simply to be considered syntactical heterogeneity since the meaning of the information contained in a complex type is not explicit to the user. When dealing with complex data types, a semantic description of the data type can help in applying a suitable parser to transform the data types. In an intermediate step appropriate parsers can be offered to transform the information into the required data type of the preceding service.

### 2.4 Problem Type III (Conceptual Heterogeneity)

*The output of a web service and the input of a second web service have the same labels (names), are represented with the same data type, but refer to **different** domain concepts.*

*Example:*

Consider now the wind information represented as a `String` provided by the *AirportWeatherService* as described above. And consider further that the *CalculateGasDispersionService* also requires wind information represented as a `String`. However, the *CalculateGasDispersionService* interprets the provided `String` not as degrees characterising the direction the wind is blowing from. Instead it interprets the `String` as characterizing the direction the wind is blowing to. This misinterpretation introduces a 180-degree mismatch.

This can lead to the creation of composite services that produce results not intended by the user. This is due to the fact that currently most composite services are built manually using WSDL-based service descriptions. If inputs and outputs of operations have the same name and data type, the user cannot tell that these operations

---

<sup>2</sup> Note that in the example which is taken from "real" services, problem types I and II occur simultaneously. For clarification, the example problem is split into two problems types.



refer to different domain concepts. The underlying assumption causing this problem is that if something is described identically, it must have the same meaning.

By referencing the application level concepts to different domain concepts, or by using different restrictions on the same domain concepts, it can be prevented that services whose inputs and outputs do not match (on the conceptual level) are combined in a service chain.

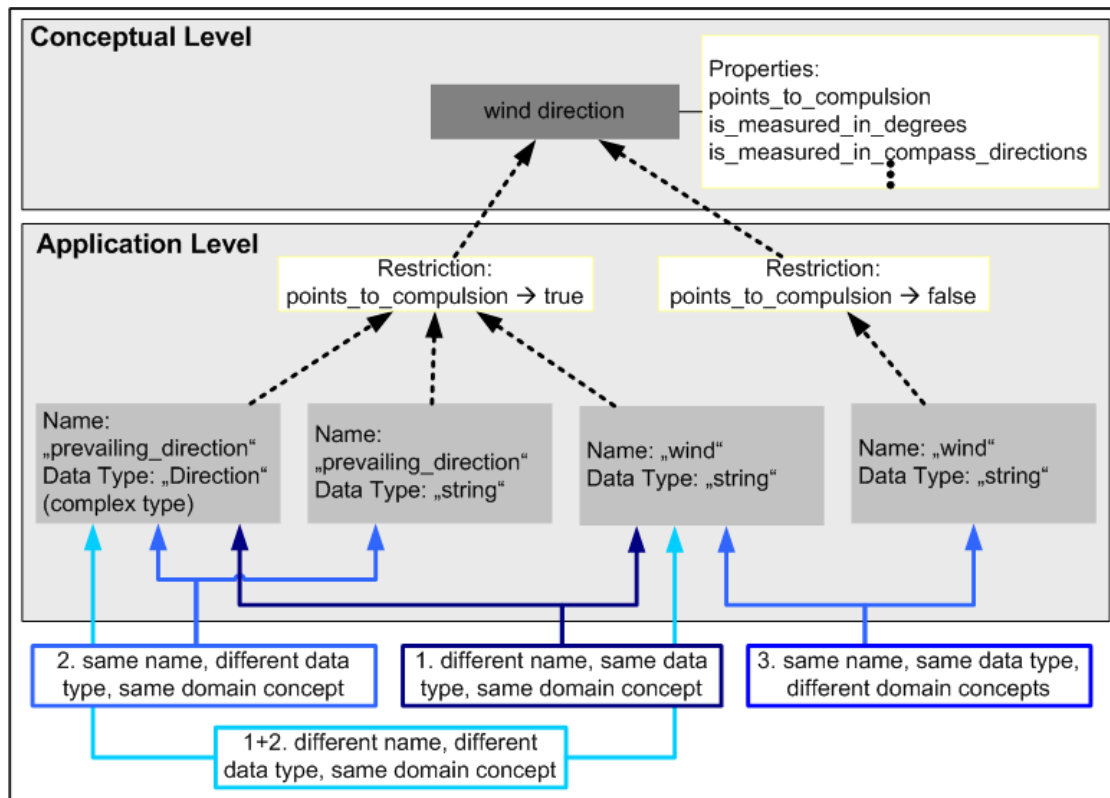


Figure 2: Types of semantic problems. Restrictions on a domain concept change its meaning.

### 3 Ontology Architecture

This chapter introduces an ontology architecture based on three levels and two threads. The levels are named application-, domain- and grounding level. The threads are named test and design-thread and visualize and expose-thread. The application and domain levels consist of ontologies, i.e. “explicit specifications of a conceptualization” (Gruber 1993). The necessity of introducing three levels is explained in chapter 4, which relates the user’s tasks during service composition to the semantic problems identified in the previous section. The necessity of introducing two formalization threads is further explained in chapter 6. In the following, the levels and how they are related to each other is described.

#### 3.1 Application Level

On this level the labels used in the WSDL-based description of the service are captured in an application ontology. The terms used in the service description are related to each other with non-taxonomic relations. The application ontology serves to capture the service’s “world view”. It represents only how the service models the information it is providing. The labels or symbols used in this application ontology e.g. *prevailing\_wind* are referenced to a domain ontology concept e.g. *wind direction*. It is important to note that these references to the domain ontology do not necessarily represent inheritance relationships as could be inferred from (Guarino 1998 figure 4). Additionally, the reference can put restrictions on the properties of the domain concept thus constraining the meaning of the domain ontology concept. An application ontology concept has only one distinct meaning. This is illustrated by the fact that in a domain ontology cardinalities may be 0 whereas in an application ontology a concept has a fixed number of properties which all are instantiated.

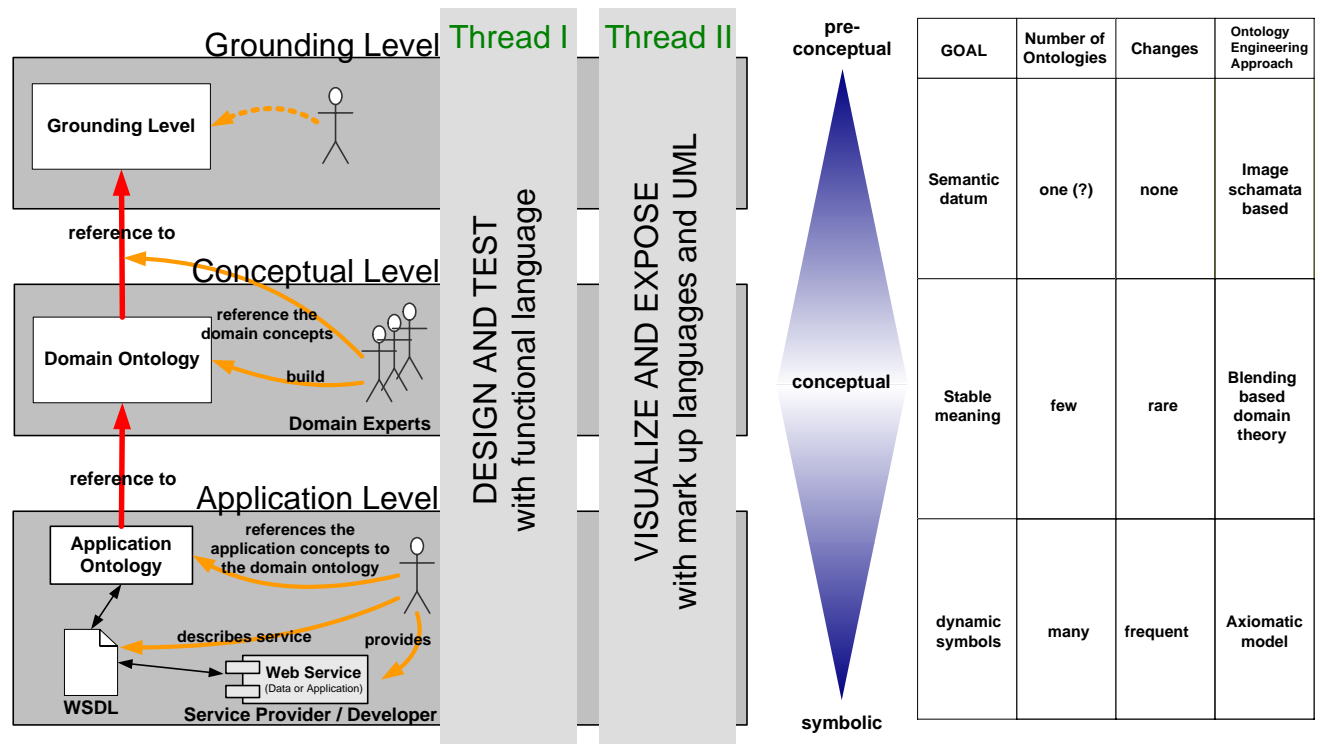


Figure 3: Three level ontology architecture; left: the levels meet different requirements.

### 3.2 Conceptual Level

Domain ontologies describe a certain part of the world (a domain) from a certain perspective. For example, a domain ontology for meteorology contains the relevant concepts for explaining meteorology. Such ontology will never be complete since the meteorological knowledge evolves and the terms used change. But it will serve to represent the vocabulary humans use to communicate about meteorology. The concepts used in this ontology will be explained with respect to meteorology although these concepts may have many more meanings in different contexts. The Domain Ontology Level is the level of human concepts. Here no data types or data models are described. Via the references and restriction posed from the application ontology, the domain ontology supplies meaning to the symbols used in the application ontology.

### 3.3 Semantic Grounding Level

The human concepts described in the domain ontology need further grounding. A domain ontology relates human concepts to each other, and thus restricts their interpretation. However, it cannot be assumed that every human user has the same understanding of a certain concept e.g. *wind direction*. To escape the vicious circle of defining concepts with other concepts on an ever higher level of abstraction a Semantic Grounding Level is required. We propose image schemata to semantically ground the concepts used on the Domain Ontology Level. Image schemas fall between abstract propositional structures (such as predicates) and concrete images (such as the spatial mental images in (Barkowsky 2001)). They are developed through bodily experiences and influence our reasoning through the recurrence of form and function. They stand in a long tradition of representing elements of knowledge into patterns or *schemas*. An image schema can be seen as a generic and abstract structure that helps people establish a connection between different experiences that have this same recurring structure. Therefore, meaning involves image-schematic structures (Johnson 1987; Gärdenfors 2000).

### 3.4 Formalization Threads

Orthogonal to the levels we introduced two formalization threads. One thread serves to design and test, the other thread serves to visualize and expose the ontologies. Thread I is based on the functional language

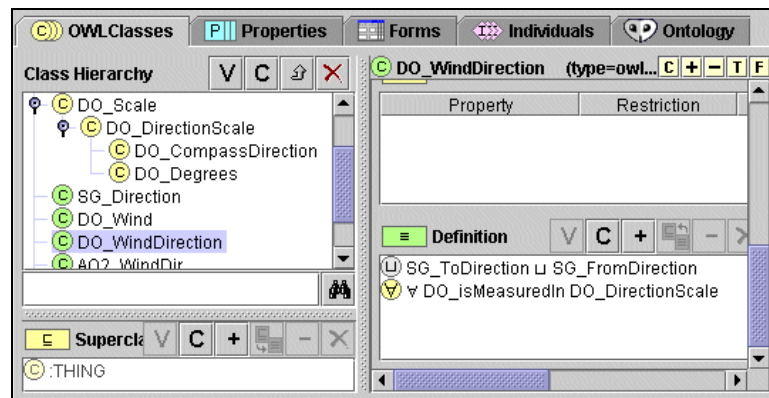


Haskell (Thompson 1996). The second thread is based on markup languages and UML. Currently, most work for the ACE-GIS pilots has been conducted in thread II (Figure 3 and chapter 5). We introduced the two threads to allow taking advantage of the different modeling capabilities they offer. We aim to model both threads exemplarily resulting in independent prototypes. However, during the development time both threads will produce results used in the other thread and vice versa. The motivation for this two threaded approach is documented in 9.1.1 and 9.1.2.

## 4 Semantic Problem Types Related to User Tasks during Service Composition

The types of semantic problems identified in the previous section will now be related to the user's tasks during service composition in order to explain the need of a three level ontology architecture. Each of the levels and the references between them will be illustrated by providing formal definitions of concepts for the example scenario. For enhanced readability these concepts are stored in *one* ontology (rather than three) and marked with a prefix according to their level (AO for the application, DO for the domain and SG for semantic grounding level). The ontology language employed is OWL (McGuinness and Van Harmelen 2003). To build the ontologies we employed the ontology editor Protégé 2.0 beta (Noy, Sintek *et al.* 2001) in combination with an plug-in supporting OWL.

1. *Defining the concept of "wind direction"*. As starting point of service discovery, the user needs the possibility to specify his concept of wind direction. This is performed with querying a domain ontology (for meteorology). In the example, he will find the concept wind direction with all properties relevant in the domain of meteorology. These are, for example, that wind direction is measured in degrees or in compass directions and that the wind direction points to the source of a compulsion or away from it<sup>3</sup>. Compulsion is an image (Johnson 1987) schema and therefore the domain ontology references this property of wind direction to the grounding level. The possibility of referencing domain ontology concepts to image schemata decreases the likelihood of ambiguously interpreted domain ontology concepts. Figure 4 shows an example for such a definition of the domain ontology concept *wind direction* that uses other domain as well as grounding level concepts (SG\_ToDirection and SG\_FromDirection).



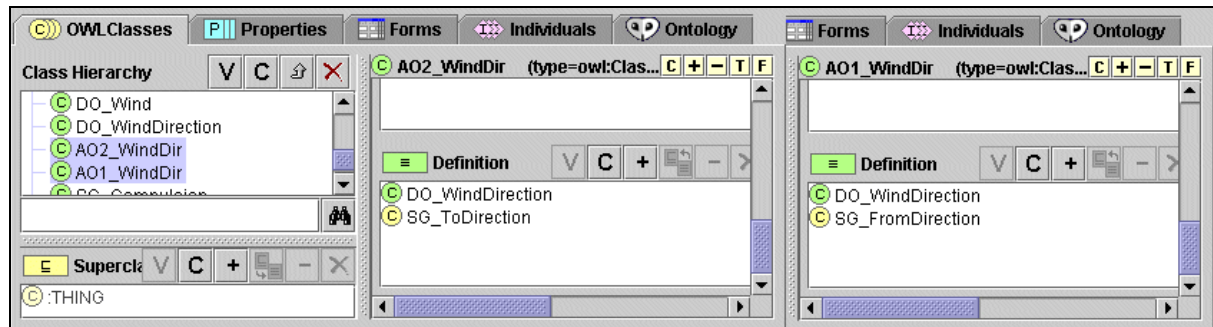
**Figure 4:** Definition of the domain ontology concept wind direction (DO\_WindDirection) in the ontology editor Protégé using the OWL plugin.

The set of image schemata, their internal structure and how they can serve as semantic grounding level need further investigation. However, the possibility to break the vicious circle of defining concepts with other (undefined) concepts is appealing.

<sup>3</sup> For simplicity reasons, this difference is currently modeled using two concepts on the semantic grounding level, namely SG\_ToDirection and SG\_FromDirection. These will be replaced by image-schematic concepts in a future version.



2. *Finding services dealing with the identified concept.* With the chosen concept “wind direction” the user discovers all three services in an application ontology registry since their labels *wind*, *wind* and *prevailing\_direction* refer to application ontology concepts (AO1\_WindDir and AO2\_WindDir in Figure 5) that are defined using this domain ontology concept. However, the restrictions the application ontologies put on their references reveal that the two concepts are different because they represent the direction *in which* respectively *from which* the wind is blowing.



**Figure 5:** Definition of two application ontology concepts representing wind direction (AO1\_ WindDir and AO2\_WindDir).

Discovering whether two concept definitions are equivalent or similar and whether concepts are related in a subsumption hierarchy is easy in the simple example we use for illustration. However, when using more complex definitions, it becomes much more difficult. In such cases a reasoning engine such as RACER (Haarslev and Möller 2003) can be used to compute a subsumption hierarchy and to identify equivalent concepts. The computed subsumption hierarchy for our example is shown in Figure 6.

With the currently existing WSDL descriptions the user has to judge whether the application concepts differ in such a way that chaining the services will produce wrong results. In this example, chaining the *AirportWeatherService* or the *GlobalWeatherService* to the *CalculateGasDispersionService* both would result in a 180-degree mismatch.

The domain ontology concept plus the restrictions on its interpretation provide *meaning* to the labels used in the WSDL service descriptions. Here it becomes obvious why domain ontology and application ontology need to form separate levels. The domain ontology provides a stable source of broadly defined (human) concepts. The application ontology provides the service developer with a flexible means to restrict or constrain the meaning of the domain ontology concepts. Finding services using the same restrictions on domain concepts solves naming problems (problem type I). The possibility for the user to be aware of different restrictions solves conceptual problems (problem type III).

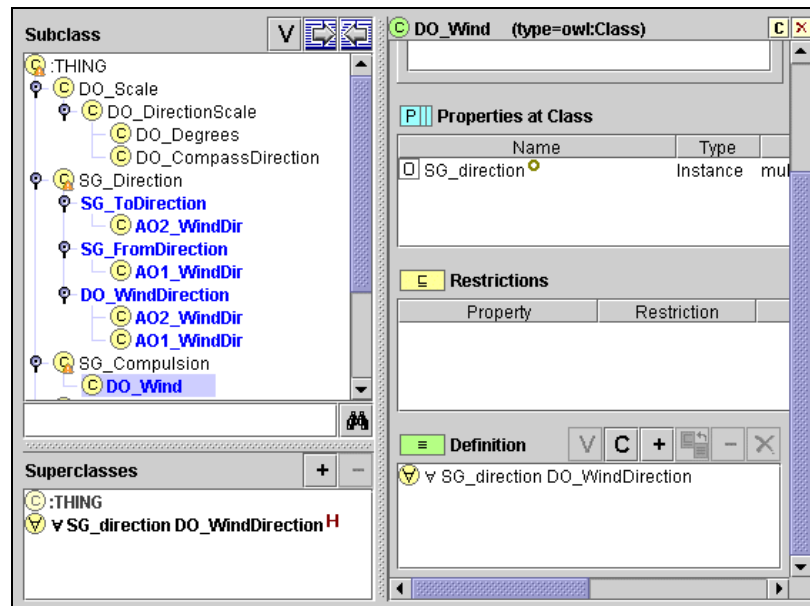


Figure 6: Subsumption hierarchy computed by RACER.

- Learn how the service represents the needed information. Consider the user searches for services dealing with wind direction that refers to the *SG\_ToDirection* concept. This results in finding the *AirportWeatherService* and the *GlobalWeatherService*. Now the user needs to know which data types are used to represent the information required. For this purpose the application ontologies of the services are queried. The resulting human-readable description of the data types employed in the service can be used to deal with problems of type II.

## 5 Ontology Engineering

The following chapter documents the evaluation of representation languages for thread II and gives a brief overview of the ontologies in progress. The general ontology engineering approach follows (Sure and Studer 2002). The ontology engineering is understood as an iterative process involving requirements specification, formalization, application and evaluation.

### 5.1 Representation Language

In phase II ontology representation languages were evaluated in order to find the representation which suits the goal of ACE-GIS best.

#### 5.1.1 OWL – Ontologies for the World Wide Web

The OWL Web Ontology Language is a widely accepted attempt to carry the progress of the semantic web. The language is used for the publication of ontologies on the World Wide Web in a semantic markup language. OWL allows the web service provider to describe the semantics in a machine accessible way. The OWL language is as recommendation for a W3C Candidate on the way to be the standard for semantic descriptions of resources available in the World Wide Web. OWL is based upon less sophisticated languages like DAML+OIL or RDF-Schema (RDFS). RDFS does not allow the disjointness of classes or boolean properties. DAML+OIL is the direct and revised precursor of OWL with less expressiveness, but altogether insignificant changes.

OWL is divided into three sublanguages, in which OWL Full describes the full set of available language constructs. It provides all possible options in RDFS and is therefore suitable for knowledge systems. It is necessary for the description of a web service to use logical constructs which can be inferred and interpreted by reasoning software or software agents in our case. Spoken language is not explicit and context-dependent; the language for software agents has to be decidable.



The sublanguage OWL DL (Description Logic) provides these reasoning capabilities but has also some constraints for the development of ontologies. The third subset OWL Lite could be seen as an introduction to ontologies and should be used for the first steps in describing semantics on the web, but it has not enough language constructs to describe the actions, processes and properties used in web service architectures. It is recommended to use OWL DL for the approach in ACE GIS. The aim is the development of web service descriptions which are fully interpretable by automatic software agents. The data in OWL DL must be expressed in a well formed structure following certain rules. The axioms must have a tree-like structure with no missing or extra components. This enforcement of a well formed structure of the code support an efficient computation and therefore easier reasoning.

The major construct in OWL is the class. The Class is described through its properties, and the restrictions defined for these properties. Classes can be created with an explicit name or by constraints placed on class extensions (e.g. enumeration of property constraints). Properties can be defined in three ways: as links between two individuals, as link between an individual and a data value or as a subproperty of a property. Classes can have at least one (*minCardinality*), no more than one (*maxCardinality*), or exactly one (*cardinality*) instance of a property. Annotations are allowed for classes, properties and individuals (instances of classes), but only for a fixed set including *versionInfo*, *label*, *comment*, *seeAlso*, and *isDefinedBy*.

### 5.1.2 OWL-S, a Web Ontology for Services

Based on OWL, OWL-S provides a core set of OWL ontologies for the formal description of properties and capabilities of a web service in an unambiguous and decidable form (Martin, Burstein *et al.* 2003). The Ontology Web Language for Services is a major application of OWL with the objective of the automation of service use by software agents. This includes the discovery, selection and composition, invocation, and monitoring of the execution. OWL-S enables planning of web services. This includes the on-the-fly composition of web service configurations (like service chains) or the integration of web services. Finding the web service for a certain task is solved by a sophisticated service planning process, which includes the decomposition of the open task into subtasks and the detection of appropriate web services for each subtask. The Constraint Satisfaction Problem (CSP) is one approach for this case.

OWL-S provides three constructs for an Upper Ontology for Services, the *ServiceProfile* (*What does the service require?*), the *ServiceModel* (*How does the service work?*) and the *ServiceGrounding* (*How can we access the service?*). The *ServiceProfile* supplies resources for automated service discovery and service selection (matchmaking). The description of the functionality, effects, and inputs and outputs belong also to the *ServiceProfile* of a web service. The *ServiceModel* assumes a web service to be a process and provides a description of the behaviour of the web service. These processes can be atomic and directly invokeable or simple, which are not invokeable and not associated with a grounding. They can be assembled to composite processes. The *ServiceModel* can be used for the invocation, planning, interoperation, and the monitoring of web services.

Details of methods to access the web service are specified in the *ServiceGrounding*. Typical grounding information includes message formatting, transport mechanisms, the used protocol, and the serialization of types. In our approach the grounding describes the inputs and outputs of the atomic processes provided by the *ServiceModel*. The *ServiceGrounding* is associated with a WSDL file, which contains for example the specific details of the used protocols. WSDL provides technical information about ways to invoke the web service, OWL-S describes what the service does and what to expect as service output. The grounding relies on the used technique and is therefore implementation-specific.

### 5.1.3 Open questions

Several questions currently remain open. The Ontology Web Language is a recommended standard of the W3C Group. OWL-S is still in the beta period and the development group expected the end of this period through the beginning of November 2003. The actual discussion about semantics of web services does also include questions about rule-based constructs for web ontologies. There is a trade-off between efficient reasoning support and sufficient expressive power in ontologies. Rules can provide an improved expressiveness not offered by OWL or other ontology languages. But the proposed standard SWRL, which suggests the integration of the Rule Markup Language (RuleML) into Owl DL/Lite, currently exists only in a



draft version. Rules are in general easier to understand (and to express) for people not familiar with ontologies. The capabilities of ontology languages are rather low in the aspect of query support, languages based on rules are more suitable for this task. Hence, Rules allow users to not just describe an ontology, but also specify non-class relationships between the elements.

## 5.2 Application Ontologies

Currently application ontologies for the six services employed in the e-Emergency composite service are being built. From the WSDL descriptions provided by ionic and e-blana and CapeClear the relevant terms are extracted and re-structured with Protégé and the OWL plugin. We introduced a slot `represents WSDL label`. Each Application ontology concepts uses this slot to point to the WSDL label it represents by providing the pertaining namespace. This slot provides a unique identification of the represented WSDL file. A second slot which all application ontology concepts inherit is the `has_reference_to_domain_concept` slot. This slot is used to reference an application ontology concept to a domain ontology concept. Note that this slot is introducing a non-taxonomic relation between two concepts of distinct ontologies and should not be confused with inheritance.

## 5.3 Domain Ontology and Grounding Level

The formalization of the domain ontology and is part of phase III the grounding level will be described only conceptually in this project. Currently an example ontology is used, combining concepts of all three levels. For getting started with the prototypical implementation such praxis might be tolerable. It allows for a classification of all introduced concepts by applying a reasoning engine like RACER (Haarslev and Möller 2003). The example ontology is introduced in chapter 4.

## 6 Evolving Semantics

The meaning of terms in information system is subject to change. Terms get adopted by different information communities or sub-communities, which often changes their overall meaning. For example, the term “polygon” has taken on a peculiar meaning in the GIS community (namely, of being closed) that deviates from its mathematical definition. New terms are introduced through legislation and practice, leading to refinements of information models in some areas and sometimes replacing existing terminology. An example is the ecological vocabulary spreading through Europe as a result of the Water Framework Directive.

In addition to these changes of terminology, the models of it that we produce change as well. Existing semantic models expressed with ontologies can be found to be incomplete, inconsistent, or plain wrong. For example, the detailed semantic description of landscape features contained in the German ATKIS object catalogue (ATKIS OK) has evolved over more than a decade through many revisions, and its users as well as its producers still find room for improvement.

As a result of such changes in how we express and understand geospatial information, providers and users of GI services need means to adapt semantic models and mappings. While it is difficult to plan for changes before even the stable models are established, it is equally important to take this dynamic nature of information models into account from the very beginning. In Work Package 6 of the ACE-GIS project, we are striving to create an architecture for semantic interoperability in service chains and composite services that can cope with change. This section of the current report explains the main ideas to satisfy this requirement.

### 6.1 Mark-up vs. Modeling

Today’s emphasis in semantic web discussions is on exposing semantic models, rather than on producing, testing, and adapting them. A semantic description of a service in OWL-S, for example, should make it easy to query the intended meaning. However, it hardly provides the possibility to explain the rationale for a particular model, let alone to trace effects of changes on other parts of the model. OWL-S and similar languages are made for marking up, not for modeling (they are all based in some form on XML, i.e. they are extensions of a basic mark-up language). Changing semantic descriptions in a mark-up language is comparable to changing database views – dependencies are broken, consistency is violated, and maintenance becomes impossible.



The two-threaded architecture proposed here addresses this problem by introducing a “meta level” behind the mark-up thread, a formalization thread for *modeling and adapting* semantics. It provides the reasoning power and the necessary internal coherence to propagate changes from one part of a semantic model to another (within thread I).

*Thus, strategy one for dealing with evolving semantics is to separate the semantic model (in thread I) from its view (in thread II, see section 3.4).*

The functional language adopted for this purpose has the unique feature (among ontology languages) to allow for testing of ontological specifications. Thereby, errors can be caught early in the design process, making later changes due to erroneous modeling much less likely. This is a standard software engineering principle brought to the ontology modeling area.

*Thus, strategy two for dealing with evolving semantics is to make specifications executable and thereby testable.*

Any such architecture with multiple representations has to provide means to keep the multiple threads consistent. Since we put the burden of absorbing changes on thread I, we need means to propagate changes to thread II. Our developments of automated mappings between Haskell and UML/OCL are a first step in this direction.

*Thus, strategy three for dealing with evolving semantics is to automate the propagation of changes from thread I to thread II.*

The three strategies demonstrate that the two-threaded architecture applies the typical software engineering approaches to deal with changing requirements: confining changes, reducing them, and ensuring consistency. The next sub-section shows how the *levels* of the architecture establish stability under changes.

## 6.2 Stable vs. changing levels

Which parts of a semantic model are subject to changes and which are not? Any architecture, physical or virtual, has as its primary purpose to identify stable levels to provide the foundation to absorb instability. Our framework identifies three levels at which semantics is modeled: the application, conceptual (domain), domain, and semantic grounding level. These three levels of the architecture fulfill that role of stabilization.

*Thus, strategy four for dealing with evolving semantics is to fix semantics at the grounding level, provide relative stability at the conceptual level, and offer flexibility at the application level.*

Clearly, semantics evolves through applications and within them. The application level is therefore the most unstable one, subject to frequent changes. For example, the meaning attached to a weather service can change for a variety of reasons: changes in its implementation, interpretation, and designation. These are the three types of heterogeneities again, now caused by changes within one and the same service, rather than across services in a chain. It is expected that these changes are frequent and that they will occur in the ACE-GIS pilots. Our phase III work will identify them and show how the architecture and modeling tools cope with them.

The next level in our architecture, the conceptual level which models domain semantics, is subject to changes, but of a much less frequent nature. For example, the concept of wind direction contained in a domain ontology may only capture the “from” sense, not the “to” sense (see 3.2). If a plume calculation service requires wind information in the form of a vector field, the “to” concept has to be introduced at the conceptual level and a semantic translation has to occur between these two concepts. Such conceptual level changes are expected to be much rarer than those at the application level.

Finally, the grounding level, as its name implies, anchors the building of semantic theories. The kinds of notions in which we seek grounding, image schemas, are not subject to change. They are, of course, still somewhat vaguely defined in the literature, which makes for regular changes at this level as part of our research. But the architectural framework that we propose as a result of phase II is to be seen as a “frozen”



state of this research, where a stable set of schemas serves as foundation for constructing domain and application ontologies and referring service terms to them.

## 7 Conclusion

We have identified three types of semantic problems that may occur during web service composition: Naming heterogeneity, conceptual heterogeneity and data type heterogeneity. We have illustrated these types by relating them to a real world examples based on services employed in the e-Emergency composite service. Meaningful composability of web services needs semantic interoperability between web services. We claim that semantic interoperability needs a sound theory of semantic grounding.

Therefore we have introduced a three level Ontology architecture for tackling the identified problem types. It allows the user to

- discover appropriate services, even when they are named differently than expected,
- identify data type heterogeneity and take further syntactical integration steps, and
- discover conceptual heterogeneity problems and thus avoid the construction of composite services producing unintended results.

We have shown in an example ontology how to give meaning to the symbols (labels of data types) used in WSDL descriptions. This is done by referencing the symbols to an application ontology, which in turn derives meaning for its concepts from a domain ontology, which in turn grounds its concepts on an image schemata-based grounding level.

We argue for a three level Ontology architecture to allow for the different degrees of flexibility, unambiguousness and stability such a system has to provide (Figure 3). Apart from the grounding of meaning with image schemata on the grounding level, the innovation in this approach lies in the flexibility of the application ontology level. On this level, a service developer can model virtually anything (including totally fictive worlds) using restrictions on domain ontology concepts. Likewise, a user can find a certain concept based on a domain ontology vocabulary and learn how its meaning is restricted on the application ontology level. While this avoids that statements that are valid in a certain application ontology need to be valid in another one, they can still be traced back to their common domain ontology concept.

## 8 Output to Other Work Packages

The results of phase II provide all project partners with information about the problems which will be encountered when service composition is based on syntactical descriptions only. WP1 and WP2 can use the results in two ways. First, the semantic problems described can be avoided resulting in applicable composite service. Second, WP 1 and WP 2 can be adapted to produce the semantic problems described intentionally. This results in a test environment for showing how methods to solve semantic problems can be applied.

## 9 Plans for Phase III

In phase III we aim to extend the shown example. This involves the implementation of an environment for testing the semantics of inputs and outputs of composite services. Finally, sound application ontologies for the services employed in the composite service will be developed. In combination with a domain ontology containing the concepts used in the application ontology we will develop a prototypical implementation of a Semantic Reference System based on the architecture introduced in this deliverable. The grounding level of the architecture will be further investigated by exploring the possibilities to reference domain ontology concepts to image schemata and other means of grounding.

The work of the consortium in phase III will supply the necessary test cases for actually dealing with evolving semantics. At the end of phase II, a good syntactic description of at least one pilot chain has been achieved. This is offering the experimental material for our phase III work. At the same time, phase II has produced ideas for adaptations of the pilot composite services. These adaptations and their realization through the work of our consortium partners, will additionally supply the experimental material for testing our approach to



modeling change. This approach will have to tackle to problem of semantic translation involving the development of a prototypical translator. We will restrict the scope of this translator to on one of the semantic problem types identified in chapter 2.

## 9.1 Enable collaboration between different threads of the architecture

### 9.1.1 Haskell-UML mappings

UML is a commonly used modelling language. It is easy to use, to learn and capable for modelling representations. However, some software developers suppose that it is not suitable for the exact specification of non-trivial application objects and the development of ontologies. It is not sufficient to introduce semantically enriched data models which can be formulated in UML (Wakeling 2001). Additionally, there is also no adequate possibility to test the behaviour of the developed model. For building an ontology it is important to have a language for describing rules available.

To overcome these limitations we intend to use Haskell. Haskell (Thompsen 1996) as a “high-level” specification-language is highly qualified for dealing with such application objects. Their operations are specified by type signatures and equations for defining their meanings. This approach is equivalent to the development of algebraic specifications. Within Haskell it is even possible to test the behaviour of the specifications directly.

Therefore we plan to enable the integration of Haskell-Specifications into an UML-Model automatically by preventing any lost of information as far as possible. This enables the use of accurately specified objects within UML-Models.

On this thread, work in phase III will concentrate on the hypothesis, that a tool for automated ontology-mapping from Haskell to UML can be specified in Haskell itself.

Therefore mapping rules have to be specified in Haskell. To proof that the rules satisfy the requirements they will be implemented in a prototypical application.

### 9.1.2 Haskell-OWL mappings

In phase 3 we plan to compare the approach of using OWL for describing services with the approach of using the Hugs dialect of the functional language Haskell (threads I and II). Functional languages support the expression of an operation (acting on something and yielding a result) as a mathematical function. This idea is close to the needs of interface specifications. It allows for a functional view of the services specified, without involving procedural notions (Frank and Kuhn 1999). We aim to develop a mapping between OWL and Haskell (considered the differences between them turn not out to bridgeable.)

In order to combine web services they need to be discovered. Service discovery consists of a matchmaking process based on service descriptions. We will start with a brief analysis of which possibilities are offered by each of the approaches to describe a service. We plan to carefully investigate different kinds of matchmaking processes, which will lead to a classification of matchmaking. As a concluding step of this investigation we will implement some scenarios out of the given ACE-GIS pilots to show the differences between the two approaches leading to an evaluation of the mapping between OWL and Haskell.

The advantages of OWL, described chapter 5, could be combined with the advantages of Haskell, which might be a better expressiveness and usability in ontology design and testing. The combination of these advantages will provide an improved way for designing composite web services.

## 9.2 Publications

A conference article describing the ontology architecture introduced in this deliverable is submitted for review. We intend to publish an article about the prototypical implementation of the ontology architecture described in this deliverable. We intend to publish an article dealing with methods for formalizing the proposed grounding level. The Haskell to OWL mapping as well as the Haskell to UML mapping may result in publications.



## 10 References

- Barkowsky, T. (2001). Mental Processing of Geographic Knowledge. *Spatial Information Theory - Foundations of Geographic Information Science, Proceedings of COSIT 2001, Morro Bay, CA, USA, September 2001*. D. Montello. Berlin, Heidelberg, New York, Springer. 2205: 371-386.
- Bellwood, T., L. Clément, et al. (2003): UDDI Version 3.0.1: <http://uddi.org/pubs/uddi-v3.0.1-20031014.htm>, last access: 2003-11-28.
- Bishr, Y. (1998). "Overcoming the semantic and other barriers to GIS interoperability." *International Journal of Geographical Information Science* 12 (4): 299-314.
- Frank, A. U. and W. Kuhn (1999). A Specification Language for Interoperable GIS. *Interoperating Geographic Information Systems*. M. F. Goodchild, Egenhofer, M., Fegeas, R., Kottman, C. Norwell, MA (USA), Kluwer: 123-132.
- Gärdenfors, P. (2000). *Conceptual Spaces - The Geometry of Thought*. Cambridge, MA, Bradford Books, MIT Press.
- Gruber, T. (1993). "A Translation Approach to Portable Ontology Specifications." *Knowledge Acquisition* 5 (2): 199-220.
- Guarino, N. (1998). *Formal Ontology and Information Systems*. Formal Ontology in Information Systems, Trento, Italy, IOS Press.
- Haarslev, V. and R. Möller (2003): RACER User's Guide and Reference Manual Version 1.7.7: <http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-7.pdf>, last access: 2003-11-15.
- Johnson, M. (1987). *The Body in the Mind: The Bodily Basis of Meaning, Imagination, and Reason*. Chicago, The University of Chicago Press.
- Kuhn, W. (2003). "Semantic Reference Systems." *International Journal of Geographical Information Science* (accepted for publication).
- Kuhn, W. and M. Raubal (2003). *Implementing Semantic Reference Systems*. AGILE 2003 - 6th AGILE Conference on Geographic Information Science, Lyon, France, Presses Polytechniques et Universitaires Romandes.
- Lutz, M., C. Riedemann, et al. (2003). A Classification Framework for Approaches to Achieving Semantic Interoperability. *COSIT 2003 (Conference on Spatial Information Theory)*. W. Kuhn, M. Worboys and S. Timpf. Ittingen, Switzerland, Springer. 2825: 200-217.
- Martin, D., M. Burstein, et al. (2003): OWL-S 1.0 (Beta) Draft Release: <http://www.daml.org/services/owl-s/1.0/>, last access: 2003-12-03.
- McGuinness, D. L. and F. Van Harmelen (2003): OWL Web Ontology Language, W3C Candidate Recommendation: <http://www.w3.org/TR/2003/CR-owl-features-20030818/>, last access: 2003-10-15.
- Noy, N. F., M. Sintek, et al. (2001). "Creating Semantic Web Contents with Protege-2000." *IEEE Intelligent Systems* 16 (2): 60-70.
- Sure, Y. and R. Studer (2002): On-To-Knowledge Methodology- Final Version: [www.ontoknowledge.org/download/del18.pdf](http://www.ontoknowledge.org/download/del18.pdf), last access: 02-09-26.
- Thompson, S. (1996). *Haskell - The Craft of Functional Programming*, Addison -Wesley.
- Wakeling, D. (2001): "A Design Methodology for Functional Programs". School of Engineering and Computer Science, University of Exeter: <http://www.springerlink.com/app/home/content.asp?wasp=4hmqwtuvnna9alwup4u&referrer=contribution&format=2&page=1&pagecount=0>, last access: 2003-12-04.

