



Data, Information and Process Integration  
with Semantic Web Services

**DIP**

*Data, Information and Process Integration with Semantic Web Services*

**FP6 – 507483**

Deliverable

**WP3: Service Ontologies and Service Description**

**D3.11**

**Methodology for building SWS ontologies in DIP**

Gábor Nagypál

June 24, 2005



## EXECUTIVE SUMMARY

This deliverable describes a reference ontology building methodology based on the analysis of existing state-of-the-art ontology building methodologies. It goes even further by providing a collection of practically applicable ontology design principles. Therefore it provides more help for ontology designers in DIP than usual ontology methodologies, which stop at a more general level. The ultimate goal of the deliverable is to serve as a cookbook for developing the various Semantic Web Service specific ontologies in DIP which are needed for the various DIP use cases (telecommunication, e-banking and e-government).

The results described in this deliverable are thus potentially of interest mainly for the use case workpackages of the DIP project. This deliverable therefore addresses the “Real use case implementation” golden bullet point of DIP.

The expected audience of this deliverable are domain experts and ontology engineers who have to develop domain-specific ontologies inside or outside the DIP project. Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.

## DOCUMENT INFORMATION

<b>IST Project Number</b>	FP6 – 507483	<b>Acronym</b>	DIP
<b>Full Title</b>	Data, Information, and Process Integration with Semantic Web Services		
<b>Project URL</b>	<a href="http://dip.semanticweb.org/">http://dip.semanticweb.org/</a>		
<b>Document URL</b>			
<b>EU Project Officer</b>	Kai Tullius		

<b>Deliverable</b>	<b>Number</b>	3.11	<b>Title</b>	Methodology for building SWS ontologies in DIP
<b>Work Package</b>	<b>Number</b>	3	<b>Title</b>	Service Ontologies and Service Description




<b>Date of Delivery</b>	<b>Contractual</b>	M15	<b>Actual</b>	31-Jun-05
<b>Status</b>	version 1.0		final	<input checked="" type="checkbox"/>
<b>Nature</b>	prototype <input type="checkbox"/> report <input checked="" type="checkbox"/> dissemination <input type="checkbox"/>			
<b>Dissemination Level</b>	public <input checked="" type="checkbox"/> consortium <input type="checkbox"/>			

<b>Authors (Partner)</b>	Gabor Nagypal (FZI)			
<b>Resp. Author</b>	Gábor Nagypál		<b>E-mail</b>	nagypal@fzi.de
	<b>Partner</b>	FZI	<b>Phone</b>	+49 (721) 9654-714




<b>Abstract (for dissemination)</b>	An ontology building methodology together with a collection of best-practice ontology design principles is presented based on the analysis of state-of-the-art methodologies.
<b>Keywords</b>	ontology, ontology building methodology, ontology design principles

<b>Version Log</b>			
<b>Issue Date</b>	<b>Rev No.</b>	<b>Author</b>	<b>Change</b>
18-04-05	1	Gábor Nagypál	First public revision
31-05-05	2	Gábor Nagypál	Extended introduction, added ontology reuse section. This is the version for internal DIP review
08-06-05	3	Gábor Nagypál	Incorporated comments from Jesus Contreras
20-06-05	4	Gábor Nagypál	Incorporated comments from Alistair Duke

## PROJECT CONSORTIUM INFORMATION

Partner	Acronym	Contact
National University of Galway	NUIG 	Prof. Dr. Christoph Bussler Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland E-mail: chris.bussler@deri.ie Tel: +353 91 512460
Fundacion De La Innovacion.Bankinter	Bankinter 	Monica Martinez Montes Fundacion de la Innovation. BankInter, Paseo Castellana, 29 28046 Madrid, Spain Email: mmtnez@bankinter.es Tel: 916234238
Berlecon Research GmbH	Berelcon 	Dr. Thorsten Wichmann Berlecon Research GmbH, Oranienburger Str. 32, 10117 Berlin, Germany E-mail: tw@berlecon.de Tel: +49 30 2852960
British Telecommunications Plc.	BT 	Dr. John Davies BT Exact (Orion Floor 5 pp12) Adastral Park Martlesham Ipswich IP5 3RE, United Kingdom Email: john.nj.davies@bt.com Tel: +44 1473 609583
Swiss Federal Institute of Technology, Lausanne	EPFL 	Prof. Karl Aberer Distributed Information Systems Laboratory École Polytechnique Fédérale de Lausanne Bât. PSE-A 1015 Lausanne, Switzerland E-mail : Karl.Aberer@epfl.ch Tel: +41 21 693 4679
Essex County Council	Essex 	Mary Rowlatt, Essex County Council, PO Box 11, County Hall, Duke Street, Chelmsford, Essex, CM1 1LX, United Kingdom. E-mail: maryr@essexcc.gov.uk Tel: +44 (0)1245 436524
Forschungszentrum Informatik	FZI 	Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany E-mail: abecker@fzi.de Tel: +49 721 96540

Institut für Informatik, Leopold-Franzens Universität Innsbruck	<b>UIBK</b> 	Prof. Dieter Fensel Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria Email: dieter.fensel@deri.org Tel: +43 512 5076485
ILOG SA	<b>ILOG</b> 	Christian de Sainte Marie 9 Rue de Verdun, 94253, Gentilly, France E-mail: csma@ilog.fr Tel: +33 1 49082981
inubit AG	<b>inubit</b> 	Torsten Schmale, inubit AG, Lützowstraße 105-106 D-10785 Berlin, Germany E-mail: ts@inubit.com Tel: +49 30726112 0
Intelligent Software Components, S.A.	<b>iSOCO</b> 	Dr. V. Richard Benjamins, Director R&D Intelligent Software Components, S.A. Pedro de Valdivia 10 28006 Madrid, Spain E-mail: rbenjamins@isoco.com Tel. +34 913 349 797
NIWA WEB Solutions	<b>NIWA</b> 	Alexander Wahler NIWA WEB Solutions Niederacher & Wahler OEG Kirchengasse 13/1a A-1070 Wien E-mail: wahler@niwa.at Tel. +43(0)1 3195843 11
The Open University	<b>OU</b> 	Dr. John Domingue Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK E-mail: j.b.domingue@open.ac.uk Tel.: +44 1908 655014
SAP AG	<b>SAP</b> 	Dr. Elmar Dörner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany E-mail: elmar.dorner@sap.com Tel: +49 721 6902 31
Sirma AI Ltd.	<b>Sirma</b> 	Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria E-mail: atanas.kiryakov@sirma.bg Tel.: +359 2 9768 303

Unicorn Solution Ltd.		<p>Jeff Eisenberg          Unicorn Solutions Ltd,          Malcha Technology Park 1          Jerusalem 96951,          Israel          E-mail: Jeff.Eisenberg@unicorn.com          Tel.: +972 2 6491111</p>
Vrije Universiteit Brussel	  	<p>Carlo Wouters,          Starlab- VUB          Vrije Universiteit Brussel          Pleinlaan 2, G-10          1050 Brussel, Belgium          E-mail: carlo.wouters@vub.ac.be          Tel.: +32 (0) 2 629 3719</p>

## TABLE OF CONTENTS

1	INTRODUCTION	1
2	ONTOLOGY BUILDING METHODOLOGIES	3
2.1	Development Activities . . . . .	5
2.2	Project Management Activities . . . . .	7
2.3	Integral Activities . . . . .	8
2.4	Development Life Cycle . . . . .	9
3	ONTOLOGY DESIGN PRINCIPLES	10
4	ONTOLOGY REUSE	17
4.1	Inclusion . . . . .	17
4.2	Reference . . . . .	17
4.3	Semantic reuse . . . . .	18
5	CONCLUSION	19

# 1 INTRODUCTION

The ultimate goal of the DIP project is to provide a technology infrastructure for Semantic Web Services (SWS). The DIP infrastructure builds on the WSMO conceptual model [28] where ontologies play a central role.

The development of ontologies is comparable in complexity with the design and development of complex software. As it is unrealistic to expect that someone who knows only the basics of a programming language will be able to design a good software architecture using that language, similarly it is not enough just to know the available ontology formalisms to build good ontologies.

To stay with the analogy of software development: a software analyst first of all needs lots of analysis and design experience to be able to design high quality software applications. Unfortunately experience cannot be described in books or other documents. There are, however, various software development methodologies which provide a framework for the various software development activities. There are also various analysis and design patterns which communicate some general rules about how a good analysis or design model should look.

Similarly, an ontology engineer first of all needs lots of ontology building experience which cannot be described in books or other documents. There are, however, various ontology development methodologies available. They provide guidance through the complex process of ontology development. Based on existing ontology development experience some philosophical and practical design principles were also reported in the literature. Unfortunately these principles are not yet organized in the form of patterns, as ontology engineering is still a nascent field compared to software engineering.

This deliverable describes a reference ontology building methodology based on the analysis of existing state-of-the-art ontology building methodologies. It goes even further by providing a collection of practically applicable ontology design principles. Therefore it provides more help for ontology designers in DIP than usual ontology methodologies, which stop at a more general level. The ultimate goal of the deliverable is to serve as a cookbook for developing the various domain specific ontologies in DIP which are needed for the various DIP use cases (telecommunication, e-banking and e-government).

In WSMO ontologies describe a specific problem domain and their definitions can be used in web service descriptions. WSMO itself defines “only” a framework for web service descriptions, containing items such as ontologies, goals, mediators, web service capabilities, choreography and orchestration specifications [6]. Thus, WSMO specifies *how* to describe a web service, in a domain-independent way. Better to say, WSMO as a meta-ontology, describes the domain of web services.

The ontology elements of WSMO, on the other hand, describe specific real-world problem domains. By using elements of these ontologies in WSMO goal and web service descriptions we can bind those specifications to the real world. It is important to note, however, that these ontologies are not web service dependent. They can be reused in other applications outside WSMO in the same problem domain, as well. As the goal of the deliverable is to describe the process of developing such ontologies, our methodology will not be specific to web services, either. The only exception is that in WSMO ontologies we should scope our conceptualisation based on the web services we want to semantically describe, that is, we should make sure that the domain ontologies really cover all of the possible user goals, web service inputs, outputs and states.

The structure of the deliverable is the following. Section 2 provides an overview on the available ontology development methodologies and describes a methodology in detail which is strongly based on the METHONTOLOGY ontology development methodology. Section 3 introduces some best-practice ontology design principles which can be used as a guideline in the conceptualisation phase of the ontology development. Section 4 discusses the important issue of ontology reuse and finally Section 5 concludes the deliverable.

Although most of the discussion in this deliverable is not dependent on a specific ontology formalism, we had to commit to a consistent terminology. We therefore use the usual W3C terminology in this document to name ontology entities, that is, we speak of *concepts*, *subconcepts*, *properties*, *subproperties* (as binary relations) and *instances*. This terminology is also consistent with the latest WSML specification [4] with the difference that binary relations where the domain is constrained to a specific concept are termed *attributes* there.

The DIP D3.3 deliverable (A Business Data Ontology) [24] has successfully followed an early version of the methodology described in this deliverable, thus the principles described here have been already partially validated in praxis.

## 2 ONTOLOGY BUILDING METHODOLOGIES

Ontology development is a very complex, creative process. A methodology which coordinates the various activities involved is therefore crucial for its success. Ontology development is comparable with software engineering in complexity, a field where already lots of matured methodologies exist like the Rational Unified Process [15] or Extreme Programming [1]. Unfortunately, since the field of ontologies is not as mature as the field of software engineering, presently there is no set of established, generally accepted methodologies. Numerous methodologies have been proposed, however, and some of them are quite elaborate.

Based on a recent study [8] the two most mature and detailed methodologies that presently exist are METHONTOLOGY [11] and the On-to-Knowledge methodology [32] which strongly build on the best ideas of other, older methodologies. Therefore, we will discuss only these here and recommend reading one of the detailed studies about other methodologies for the interested reader. Some good studies and overviews are the following: [8], Chapter 3 in [12], and [16].

The On-to-Knowledge methodology (OTK) concentrates on building knowledge-based systems, where ontologies form an important part of the system. This methodology defines two orthogonal processes, the *Knowledge Process* and the *Knowledge Meta Process*. The former describes the process of ontology usage, the latter guides the ontology creation. Therefore, in this document we are interested only in the Knowledge Meta Process.

OTK defines the following steps as part of the Knowledge Meta Process (see also Figure 2.1 taken from [32]):

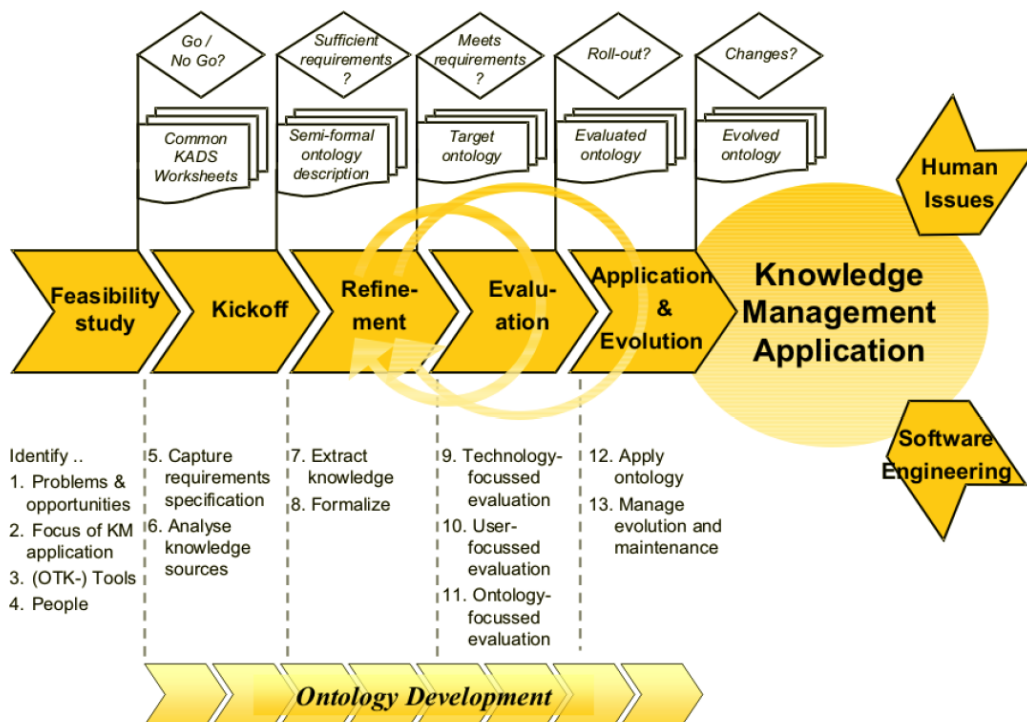


Figure 2.1: OTK steps

**Feasibility Study:** OTK follows the well-known CommonKADS methodology [29] in order to decide whether it makes sense to start the project, that is, to build the ontology.

**Kick-off:** During this phase the ontology requirements are finalised: the exact goal and the scope of the ontology is determined. According to the middle-out strategy (see Section 2.1) during a brainstorming session an initial list of important entities is collected and a list of relevant experts and knowledge sources is compiled. Design guidelines are also made which will guide the development process. Competency questions are collected which can be used later to validate the ontology.

**Refinement:** In this phase relevant knowledge is extracted from the identified knowledge sources (and from human experts) and formalised.

**Evaluation:** OTK identifies three types of evaluation: technology focused, user focused and ontology focused evaluation. These are described in more detail later as part of our methodology based on METHONTOLOGY. OTK proposes a cyclic ontology development process, that is, the Refinement and Evaluation phases are iterated until a stable, high-quality ontology version is reached.

**Application & Evolution:** An ontology is never ready, as it represents our knowledge of a specific domain which is naturally incomplete and constantly evolves. On the other hand, during the ontology development, design decisions have to be made as to which knowledge is relevant and thus should be included and which is not. Some of these decisions can be erroneous. Therefore, during ontology usage it can turn out that some important knowledge is missing from the ontology (or irrelevant things are included). Especially because our domain knowledge changes constantly, ontology maintenance is a continuous process. OTK identifies two types of ontology maintenance strategies: *centralised* and *decentralised*. In the centralised case one person (or a group of people) is responsible for ontology changes. In the decentralised case everyone can make changes to the ontology. In this case tool support for ontology change management is crucial. The centralised strategy should be followed if high quality is important. On the other hand, the decentralised strategy is cheaper, faster and more flexible.

OTK was used in various use cases during the On-to-Knowledge project<sup>1</sup>. The most important lessons learned during these use cases were the following:

- “Human Issues” can dominate other forces during a Knowledge Management (KM) project
- domain experts in an industrial context need pragmatic development guidelines (which we also provide in this document)
- collaborative ontology engineering requires physical presence and advanced tool support

---

<sup>1</sup>See <http://www.ontoknowledge.org>

- brainstorming is very helpful for early stages of ontology engineering, especially for domain experts not familiar with modelling. In OTK MindMaps were used quite successfully.

The METHONTOLOGY methodology describes a similar process to OTK, but it focuses more on the ontology development (it does not address the issue of Knowledge Process) and provides a more detailed framework. We base our process therefore on this methodology and describe it here. Where it is useful, we extend the original methodology with ideas from other methodologies, like OTK.

In the remaining part of this section we describe the various activities which form the ontology development process, as METHONTOLOGY defines them.

## 2.1 Development Activities

These are the core activities of the ontology development process which define the phases of the development.

**Specification:** This phase results in the ontology specification document. This can be informal (natural language description, informal competency questions) or formal (formal competency questions). The document should define at least the goal and the scope of the ontology clearly. It should also list the major information sources for the ontology together with the most important entities of the domain. This phase is similar to the Kick-off phase in OTK. In contrast to OTK METHONTOLOGY does not address the issue of feasibility study, although it is a good idea to conduct one before starting the ontology development.

It DIP it is especially important that as many common information sources are used in the use case ontologies as possible. If the semantic overlap among the use case ontologies are significant, it will be easier to define mappings between them as part of DIP WP5. Specifically, the use of the high-level business data ontology defined in D3.3 [24] should be considered as a common semantic core for use case ontologies.

**Conceptualisation:** This phase consists of building an intermediate conceptual model. This model is not necessarily suitable for reasoning and can be in any form which is understood and accepted by domain experts (e.g. Excel sheets, a mind map, semi-structured text). The use of intermediate models was already proved in many real-world systems (like Galen [27], VICODI [23] or in the OTK use cases [32]). METHONTOLOGY describes various artefacts which specify different aspects of the conceptual model:

- Glossary of Terms
- Concept and property classification hierarchies
- Tables of class and instance attributes
- Tables of constants
- Table of instances
- Verbs dictionary
- Table of formulas

This list is only a possible way to define a conceptual model, it can vary from project to project which is the best understood formalism by domain experts.

There are different strategies for defining a conceptualisation. Following the *top-down* strategy one starts with the most general concept (e.g. **THING**) and tries to refine the ontology structure along different distinguishing notions. This strategy is usable mostly in the case of top-level, philosophical ontologies.

The *bottom-up* strategy starts with a suitable set of information resources (databases, documents) which should be described by an ontology. In the first step interesting entities are collected from these resources which are worthy of inclusion in the ontology. This process can be supported by information extraction (IE) and ontology learning tools like Text-to-Onto<sup>2</sup> or Amilcare<sup>3</sup>. Later on the ontology engineer tries to find common superconcepts and superproperties of the identified concepts and properties, and tries to find the proper concepts for the identified instances.

The advantage of the bottom-up strategy is that the ontology will definitely describe the target document corpus or database(s) properly, that is, it will describe the “information supply” well. On the other hand, with this approach there is a danger that the ontology will be too focused on a specific information resource, thus it will not be reusable. Experience also shows that semi-automatically generated ontologies are of lower quality than manually engineered ones. It is also important to note that this strategy can be used only for very low-level ontologies, where a specific information resource should be semantically described. For example the business data ontology which is described in D3.3 [24] is on a much higher level where this approach could not be followed.

Finally, the *middle-out* strategy starts with a list of most important ontology entities (concept, properties) which can be collected during a brain-storming session. This approach can be used for both low-level, application ontologies, or medium-level ontologies where a list of most important concepts can be easily identified at the beginning.

Generally speaking we can say that both the top-down and middle-out strategies can result in high-quality ontologies which represent the “information need” in the domain quite well. There is a danger, however, that the ontology will not describe a specific collection of documents or a database adequately.

**Formalisation:** This phase includes choosing a suitable formalism (e.g. first order logic (FOL), F-Logic, description logic (DL)) and transforming the conceptual model into that formalism. It is possible that changes in the conceptual structure are required because of the limitations of the target formalism (e.g. DL usually supports only binary relations), and it is also possible that some elements of the full conceptual model are lost (e.g. many formalisms do not allow metaclasses, that is, entities that are instances and concepts at the same time, but those are often needed to express relevant knowledge of a domain). This formal representation is semi-computable, that is, it can be rewritten into a suitable syntax quite easily, which can serve as an input for a suitable reasoner.

---

<sup>2</sup><http://sourceforge.net/projects/texttoonto/>

<sup>3</sup><http://nlp.shef.ac.uk/amilcare/>

**Implementation:** Codifying the formal representation in a specific formal language (e.g. OWL, WSML etc.) which can be reasoned on by using a suitable reasoner.

**Maintenance:** A guideline for making future changes to the ontology should be defined, and the necessary changes should be made, as it was already described at the application and evolution phase of OTK.

Recently, the DILIGENT knowledge process for ontology maintenance was proposed [33] which defines a process which is in between the two extremes of a completely centralised and completely decentralised process. In DILIGENT, after an initial ontology development phase, users are allowed to make local modifications independently.

Uncontrolled local modifications would lead, however, to completely incompatible ontologies on the long run that would make interoperation between the various web services or applications impossible. Therefore a central board is also established which has the task of tracking local changes of the ontologies, analysing them, and revising the central core ontology accordingly. After revisions users have to make local updates to harmonize their local ontology versions with the new common ontology core.

As a matter of fact, a very similar maintenance process was proposed for the business data ontology in DIP D3.3 [24], and this is exactly the way we propose the maintenance of SWS ontologies in DIP. In a problem domain (such as telecommunication or e-banking) this process will result in a set of ontologies which contain a common semantic core and therefore the principles of semantic ontology mediation can be demonstrated on them as part of DIP WP5.

**Use and Reuse:** The ontology is used by various applications and users, and can be reused as part of other ontologies. Ontology reuse is a very important aspect, as one of the main motivations for developing ontologies is the hope that knowledge formalized in such a form is more amenable for reuse than in other forms (like relational database schemas or rule sets in expert systems). Various forms of ontology reuse are discussed in detail in Section 4.

Ontology reuse during the ontology development process happens as part of the *integration activity* which will be discussed later.

It is important to note that the conceptualisation, formalisation and implementation phases are not necessary distinct, since they can be merged if it is necessary. For example in many cases it is more comfortable to choose not only a formalism, but also the specific implementation language, and do the formalisation and implementation steps together. Further, if we use a simple metamodel to define our conceptualisation which is understood by the domain experts, a separate intermediate model may be unnecessary.

## 2.2 Project Management Activities

As ontology development is a project, project management is very important to achieve success. The following activities are identified:

**Planning:** This includes the traditional project planning tasks like deciding about the tasks, how they will be arranged, resources (time, people, SW and HW) needed by the tasks etc.

**Controlling:** Check that the plan is followed, make adjustments in the plan if needed.

**Quality Assurance:** Quality assurance is similar to but not the same as ontology evaluation, as it considers all of the artefacts produced, not only the ontology itself (ontology documentation, intermediate models, specification documents etc.)

## 2.3 Integral Activities

These activities support all phases of the ontology development process, and they are conducted more or less intensively throughout the whole ontology building project.

**Knowledge Acquisition:** Knowledge has to be extracted from the various knowledge sources. Those sources can include domain expert knowledge, existing books and existing ontologies.

**Integration:** A special kind of knowledge acquisition is the reuse of existing ontologies by integrating them into the new ontologies.

**Evaluation:** Evaluation includes *verification*, i.e judgement of ontology correctness with respect to a frame of reference (e.g. the ontology specification document). It also includes *validation*, that is, judgement of the ontology with respect to the real-world domain it is supposed to represent. From another point of view, evaluation can be categorised in three categories, as follows:

**Technology-focused evaluation:** This includes checking the syntactical correctness and semantical consistency of the ontology, its performance, modularity, maintainability etc.

**User-focused evaluation:** This includes checking whether the ontology contains all of the information which was identified in the ontology specification document. A usage pattern based evaluation is also part of this process, where it is checked that all parts of the ontology are really used, that is, there are no unnecessary parts in it.

**Ontology-focused evaluation** This checks the semantical correctness of the ontology. Both philosophical methods (like OntoClean [14]) and ontology evaluation rules can be used to find incorrect conceptualisations.

**Documentation:** Proper documentation of an ontology is crucial for later maintenance and reuse. One of the most serious hindrances in ontology reuse nowadays is that most of the ontologies available on the Web are not properly documented. Both the meaning of ontology entities and the design decisions which led to a specific ontology must be documented, otherwise the ontology cannot be judged objectively when someone wants to decide about the integration of the ontology into a new one.

**Configuration management:** Mainly in the case of big ontologies which are developed collaboratively, configuration management is at least as crucial as in the case of big software development projects. For this activity the tools of Software Engineering can be used, but also some ontology editing environments provide such features.

## 2.4 Development Life Cycle

It is important to consider in which order the defined activities are executed. A definition of a life cycle is what it makes a methodology. METHONTOLOGY defines an evolutionary type of life cycle (similarly to most of the other methodologies) which is shown on Figure 2.2. This means that the development activities are executed iteratively throughout the development process. From the project management activities, planning is done at the very beginning while control and quality assurance are continuous. All of the integral activities are done continuously, although the amount of knowledge acquisition, integration and evaluation decreases as the ontology matures and its structure stabilises.

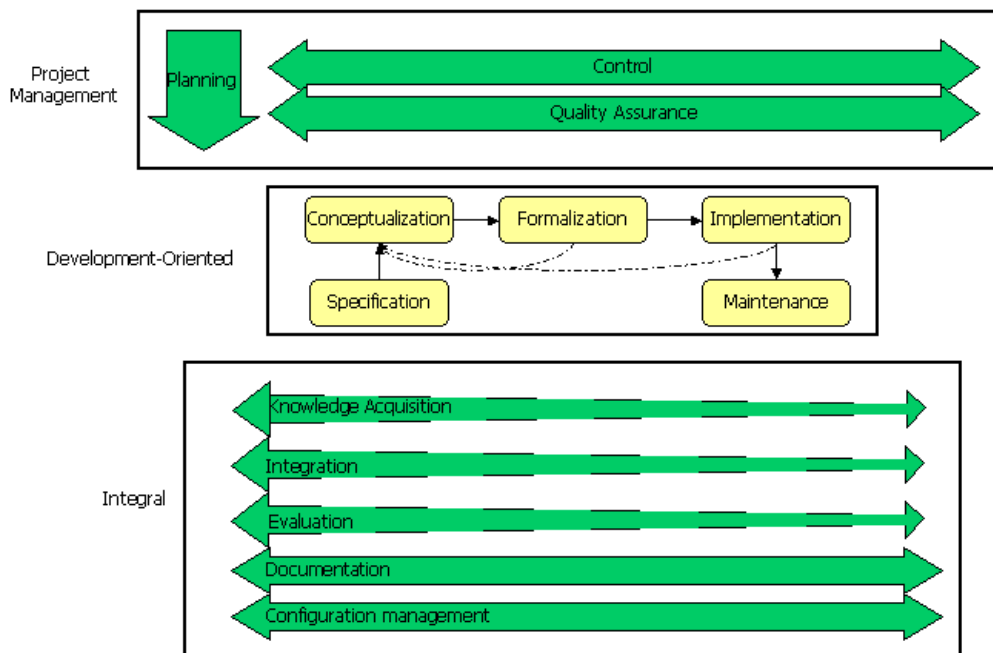


Figure 2.2: Methodology lifecycle

### 3 ONTOLOGY DESIGN PRINCIPLES

Unfortunately there is no standard catalogue of “ontology design patterns”, “knowledge patterns” or “semantic patterns”. Although there is already some preliminary work in this area (e.g. [3]), ontologies are still immature in comparison to software development, where a significant body of design and analysis patterns are available (see e.g. [9, 10]). Therefore, we think that it is a valuable contribution to list some of the best-practice principles which were collected from state-of-the-art ontology related literature (including [14, 12, 11, 26, 27, 16, 13, 19, 31, 5, 7, 8, 32, 25]) and are validated by our own ontology building experience [23].

**Define and use naming conventions:** To achieve a consistent and meaningful ontology it is extremely important to define naming conventions and to obey them throughout the development process. Probably the most important areas where clear guidelines are needed are the following:

**Capitalisation:** It is a common convention to begin concept names with capital letters and instance and property names with non-capital letters.

**Delimiters:** Common conventions are: using space or the “-” character as delimiters, or write names in CamelCase which eliminates the need for delimiters. Note that CamelCase is also suitable for ontology URIs therefore it is frequently used in web ontologies.

**Singular or plural:** It is important to decide whether to use singular or plural form of names. It is a common convention to use the singular form in concept names.

**Prefix and suffix conventions:** These conventions are mainly important for property names. A usual solution is to use “has” and “is” as prefixes for properties, with an additional “Of” suffix for the “is” form. In case of an inverse property pair, one of them should follow the “has” scheme and the other should follow the “is” scheme.

**Avoid abbreviations:** Abbreviations should be avoided in names, except from some very well established ones, like URI for Uniform Resource Identifier.

**Exclude/include superconcepts in names:** The name of superconcepts or superproperties should be consistently included or excluded in the name of subconcepts or subproperties throughout the ontology. For example both `ONEWAYTICKET`, `RETURNTICKET` or `ONEWAY`, `RETURN` can be used as names for subconcepts of the `TICKET` concept, but `ONEWAYTICKET` and `RETURN` should not be used together.

**Scope your ontology:** Scoping is crucial because an ontology should not contain all of the imaginable distinctions of the target domain, but on the other hand it should contain all of the important distinctions of the domain. Note that for high-level ontologies it is not easy to decide what is relevant and what is not. In this case scoping means rather deciding about the generality level where the ontology should stop.

Specifically in the case of SWS ontologies scoping means deciding about the problem domain (such as e-banking, telecommunication), or even about the specific task at hand (like mortgage processes in the e-banking domain). If specific

web services are already known which should be semantically described, these should be listed, and can be used to further limit the scope of the ontology. The more limited is the scope of the ontology, the easier it is to develop it.

**Introducing new entities:** Introduce a new subconcept or subproperty only if it is significant for the problem domain, that is, we can say something about that very entity which is not true for any of the other entities in the ontology. For example typically you should introduce new concepts only if they will have a different set of properties than other concepts in the ontology. On the other hand, you should introduce a new entity whenever a distinction is important in the domain.

Specifically for WSMO ontologies, the domain ontology should contain all of the entities which are required to describe capabilities and interfaces of the web services in the application domain, and typical user goals which are foreseen at design time. Of course, ontology usage should be monitored during the maintenance phase, and newly identified user goals should be added to the ontology. It is also possible that later new web services appear in the application domain which are not covered by the ontology. As it was already noted, an ontology is never ready, and it should be constantly evolved during its lifetime.

**Formal concept hierarchy:** Always define a formal concept hierarchy: if  $D$  is subconcept of  $C$ , all instances of  $D$  must be also instances of  $C$ . Do not mix concepts with topics! Web directories such as Yahoo!<sup>1</sup> or the Open Directory<sup>2</sup> use topic hierarchies to organize web pages. Topic hierarchies are structures that are intuitive for human browsing but do not fulfil the formality requirement and therefore they are not suitable for automatic reasoning.

If you are unsure about a subclass relationship, the OntoClean evaluation methodology [14] can help to validate the decision.

Examples:

- All `RETURN TICKETS` are `TICKETS`. (Good)
- All `LOCOMOTIVES` are `TRANSPORTATION`. (Bad, it is a topic hierarchy, as a locomotive is not a transportation)
- All `ADDED VALUES` are `SERVICES`. (Bad, as added value is rather a property of a good service, than a kind of service. The example was taken from [21])

It is also important to identify all of the formal hierarchical relationships and model them properly. E.g. in [21] `SERVICE CONTRACTED BY CUSTOMER IN CHANNEL` is not defined as subconcept of `SERVICE` and thus a reasoner later on cannot infer that instances of `SERVICE CONTRACTED BY CUSTOMER IN CHANNEL` are also services. As a result there is a danger that a web service which expects a service as its input cannot be invoked by an instance of `SERVICE CONTRACTED BY CUSTOMER IN CHANNEL`.

**Optimal number of subconcepts:** The optimal number of subconcepts is between 2 and 12 (according to [25]). If you have only one subconcept it is a good indication that either the subconcept is unnecessary or new siblings of that concept

---

<sup>1</sup><http://dir.yahoo.com/>

<sup>2</sup><http://dmoz.org/>

should be introduced. If you have more than twelve, introducing an intermediate classification level may be useful.

**New concept or property value:** It is a common decision during the ontology development you have to make: should I represent something by introducing a new concept, or is it enough to fill in the right value for a property at the instance level? For example the distinction between slow and fast locomotives can be represented by introducing new **SLOWLOCOMOTIVE** and **FASTLOCOMOTIVE** concepts, or by filling the proper values of a **HASSPEED** property (**Slow** or **Fast**) at the instances of **LOCOMOTIVE**. If a distinction makes entities participate in different relations, make the new entities concepts, otherwise a property value is probably enough [25]. For example if there is a concept **HIGHSPEEDLINE** and we know that only fast locomotives are allowed to travel on such lines, it is probably a good idea to define **FASTLOCOMOTIVE** as a concept.

Another indicator for using a property value instead of defining new concepts is if the value would change often [25]. For example if locomotives are painted newly each year with different colours, probably it is not the best idea to define the concepts **BLUELOCOMOTIVE** and **REDLOCOMOTIVE** as locomotive instances would change their concept frequently. Of course, if our target ontology formalism supports axiomatic concept definitions, and thus automatic categorization of instances (like description logic-based formalisms), this is not a real problem, and such concepts can be defined even if concept membership often changes.

From a more philosophical point of view, we can also check whether an entity can exist alone, or it is always dependent on other entities. For dependent entities it is better to define a new property instead of a new concept [2]. For example it makes sense to speak of “arrival time” only if we know also the entity whose arrival time we talk about (e.g. a specific train). In this case it is probably a good decision to have a **HASARRIVALTIME** property instead of an **ARRIVALTIME** concept. Of course in many cases the distinction is not so clear. For example the color “red” exists also without considering other entities, but it is also clear that in many cases we are interested in the connection between this color and other objects. This issue is very much related with the question of representing enumerations, which will be discussed below.

As an example from the DIP use cases let us consider the **CUSTOMER** concept [21]. It has three subconcepts **COMPANY**, **PERSON** and **SOHO**. This decision indicates that the type of customer does not change, i.e. a natural person normally will never be a company..

**Concept or instance:** If it is meaningful to speak of a “kind of X” in the target domain, that is, the entity represents a set of something, make X a concept. Otherwise X should be an instance. If you are unsure, make X a concept, that is the safer strategy.

For example consider the case when you have a **LOCOMOTIVE** concept. The question is whether **STEAMLOCOMOTIVE** and **DIESELLOCOMOTIVE** should be subclasses or instances of this concept. From a conceptual point of view both options are possible, the choice depends on whether we want to model concrete instances of steam or diesel locomotives (e.g. **SteamLoc234543534**). If locomotive instances are not of interest for us, we can have **SteamLocomotive** and **DieselLocomotive**

as instances of the **LOCOMOTIVE**, in which case this concept represents the set of locomotive types.

If we are unsure whether we will ever need to represent a specific locomotive instance like `Loc234543534`, it is better to have **STEAMLOCOMOTIVE** and **DIESELLOCOMOTIVE** as concepts, as we will always be able to define new sub-concepts and instances of a concept, but for an instance it would not be possible any more. In this case the **LOCOMOTIVE** will represent a set of locomotive instances, that is, it will have different semantics than in the previous case.

So, as we have seen instances close the ontology, that is, instances should appear only at the lowest level in the ontology. Unfortunately it is hard to see in advance which is the lowest level, therefore in many cases it is safer to use concepts. But with some ontology formalisms it can cause problems again, as it is not possible to point directly to concepts. Some possibilities to solve the problem in this case are described in [7].

The concept vs. instance question appears very characteristic in the case of enumerations, that is, disjoint partitions of a concept. Consider for example a concept **SPEED** and its possible partition **Slow**, **Medium** and **Fast**. Should those be instances or concepts?

Practitioners using description logic-based ontology formalisms argue that those should be concepts [5], as in this case it is possible later to refine this partition for example by defining new subconcepts of **FAST** like **LIGHTNINGFAST** and **VERYFAST**. On the other hand, by using this technique if we want to encode the fact that `SteamLoc234543534` is slow, we have to always create a new instance of the **SLow** concept representing the fact that `SteamLoc234543534` is slow (like `SteamLoc234543534hasSpeedSlow`), and connect this instance with the original locomotive instance.

This approach is normally considered as cumbersome by practitioners coming from the database field or using frame-based ontology formalisms. Those ontology engineers prefer to use instances for enumerations which results in a simpler structure (it is enough to connect the locomotive instance with the speed instance) but we lose the possibility to define hierarchical enumerations.

As an example from the e-banking use case consider the **PRODUCTRATEAPPLICATION** concept which defines the possible types of interest rates that can be applied to a financial product. Thus, it defines an enumeration. In the e-banking ontology the designers chose to represent the enumeration as subconcepts: **PRODUCTRATEAPPLICATIONFIXED**, **PRODUCTRATEAPPLICATIONMIXED**, **PRODUCTRATEAPPLICATIONVARIABLE**. This makes the ontology flexible towards future changes, if e.g. subtypes of fixed interest rates are have to be defined. Further, this technique allows users of the ontology to attach interest rate value information to the relation between a **PRODUCT** and **PRODUCTRATEAPPLICATION** by specifying the value at the newly created instance of **PRODUCTRATEAPPLICATION**. This technique is called *reification*, and it is the probably most common way to represent n-ary relations in ontology formalisms supporting only binary ones.

In this example, if the ontology language supports n-ary relations (like WSMML), and if no subtypes of the enumeration are foreseen, probably it is sim-

pler and more intuitive to represent the interest rate types as instances of **PRODUCTRATEAPPLICATION**.

Generally we can say that the concept vs. instance distinction is unnatural for humans, and in many cases the answer is that an entity is both a concept (a set of something) and an instance (a member of a set). Consider the situation described in [22].

There is a concept **SPECIES** (representing the set of all species), with instances such as **Ape**. However, **APE** may be also viewed as a set of all apes. It may be argued that **APE** may be modelled as a subconcept of **SPECIES**. However, if this is done, other irregularities arise. Since **APE** is a set of all apes, **SPECIES**, being a superconcept of **APE**, must contain all apes as their members, which is clearly wrong conceptually. Further, when talking about the **APE** species, there are many properties that may be attached to it, such as habitat, type of food etc. This is impossible to do if **APE** is a subconcept of **SPECIES**, since concepts cannot have properties (in most ontology formalisms).

This shows the need for ontology formalisms supporting metaclasses (i.e where classes can be viewed as instances at the same time) like KAON [22], OWL-Full [20] or F-Logic [17].

**Document your ontologies:** Always document ontology entities also in natural language. While logical axioms are very useful for logic reasoners, usually humans (especially domain experts with no or minimal logics background) have problems interpreting and understanding them. On the other hand, simple labels are usually ambiguous (e.g consider “wing” in an airplane or in a bird ontology). It is also possible to express design considerations in natural language, which are not expressible in the target ontology formalism, but are important when someone tries to reuse the ontology. If an ontology should be reused at an international level, like in the DIP project, all ontology elements should be documented in English. As a negative example the **SIGNALDATECONTRACT** property can be mentioned from the e-banking ontology [21], which is documented only in Spanish, or the **TERMRATEFIXED** property which is not documented at all. Positive examples from the same ontology are concepts **PRODUCT** and **SERVICE** which are documented in both English and Spanish.

**Modularise your ontologies:** It is useful especially in the case of big ontologies which are developed collaboratively to split the ontology into independent modules. A possible technique for that is described in [26] by Alan Rector, based on his more than fifteen years of ontology building experience. We overview this technique here informally and recommend the interested reader to read the referred paper for more details. The main ideas of this modularisation technique are the following:

- Primitive concepts should be defined in independent modules. A primitive concept is a concept which is not defined by logical axioms, so it is transparent for the reasoner (the reasoner does not “understand” the meaning of the concept). Primitive concepts can form a hierarchy, but merely organizing them into hierarchies still does not define them. For example we can state that **LOCOMOTIVE** is a subconcept of **VEHICLE**, but the reasoner still

does not have any information about what is the exact difference between locomotives and other vehicle types.

- In a module a tree taxonomy of primitive concepts should be defined where
  - each concept has max. one parent concept (i.e. the taxonomy forms a tree)
  - children of a concept are always pairwise disjoint
  - the whole taxonomy is based on only one differentiating notion (e.g. speed, colour, functionality, structure). That is, **RED** and **FAST** should not be part of the same module, **RED** and **BLUE** should be.
- The modules should be connected by defined concepts which are defined by axioms (supported by a given ontology formalism) using primitive concepts from the various modules. These defined concepts should be always subconcepts of one (!) primitive concept, and connected by properties with other primitive concepts. For example we can define a **FastRedLocomotive** as a **Locomotive** which has speed “fast” and has colour “red”.

The consequences of this procedure are the following:

- The place for new instances is easily identified even for huge ontologies, as
  - all leaf concepts in a module’s tree hierarchy are pairwise disjoint (i.e. the instance should belong only to one of them)
  - the modules describe independent aspects and therefore it is easy to choose the right module for the instance
- Updates (new axioms and instances) have minimal or absolutely no impact on already existing parts of the ontology.
- The modules can be developed independently from each other, possibly by different expert groups.

As an example demonstrating the advantage of the approach consider the following. We would like to define the **FASTREDSMALLLOCOMOTIVE** concept in a big railway ontology. Using a naive conceptualisation this concept could be defined as a subconcept of **FASTLOCOMOTIVE**, **REDLOCOMOTIVE** and **SMALLLOCOMOTIVE**, which are again subconcepts of **LOCOMOTIVE**.

What happens if we later also want to make a distinction in the ontology based on the propulsion type of locomotives? How to add the **FASTREDSMALLSTEAMLOCOMOTIVE** concept? Of course we can add this new concept as the subconcept of **FASTREDSMALLLOCOMOTIVE**, but this causes two problems.

First, as we did not define clearly what the difference is between **FASTREDSMALLLOCOMOTIVE** and **FASTREDSMALLSTEAMLOCOMOTIVE** (i.e. those concepts are primitive ones for the reasoner), it is not possible to classify existing (and possibly quite numerous) locomotive instances automatically, but we have to examine all (!) of them manually, and declare them as **FASTREDSMALLSTEAMLOCOMOTIVE** when it is appropriate.

Second, it is an open question who is responsible for adding this new concept and doing the job of manual classification. The experts dealing with colors? With speed? Or with vehicle types? Most likely the result will be a deadlock, where

nobody will do anything. One should never underestimate human issues in a big ontology development project. . .

Using the modularisation approach expert groups can develop independently modules of vehicle types (containing **LOCOMOTIVE**), speed (containing **FAST**), colour (containing **RED**), and the ontology engineer can define the new concept **FASTREDSMALLLOCOMOTIVE** as a **LOCOMOTIVE** which is red, fast and small. Responsibilities for adding new instances are clear. The vehicle expert group adds new locomotives, the colour expert group can define their colour, the speed expert group their speed etc. The instances are categorised automatically and properly as **FASTREDSMALLLOCOMOTIVE** when it is appropriate.

What happens now if we want to use also propulsion types? We simply create a new group which defines a new “propulsion types” module containing the **STEAM** concept. They are also responsible for adding statements about the propulsion type used by the existing (and new) locomotive instances. The ontology engineer defines the **FASTREDSMALLSTEAMLOCOMOTIVE** concept as the set of all Locomotives which are fast, small, red and have steam propulsion. After defining that concept all of the existing (and new) instances of locomotives are categorised automatically and properly.

As an example from the DIP project the e-banking ontology [21] is modularized into independent modules representing services, products, channels, users and currencies. In this ontology no defined concepts are specified, as it was probably not needed.

## 4 ONTOLOGY REUSE

The main motivation for ontologies was, from the very beginning, the need to encode knowledge in a reusable form. This should allow for the effective reuse of existing ontologies across many applications and also in other ontologies. As building a new ontology is always extremely expensive both in terms of human effort and time, integration of existing ontologies into new ones should always be considered (see also Section 2.3). Ontology reuse can also make mediation between systems using different ontologies easier [18], as at least parts of the ontologies will follow an identical semantic structure.

It must be stressed that ontology reuse is only possible if ontologies are properly documented in natural language. Not only the actual structure of the ontology should be explained, but also the design decisions that led to the structure. With other words, it is not enough to explain what is there, but it is also necessary to explain why, and what was left out. Based on this information the developer of the new ontology can make an informed decision whether and how to reuse the ontology.

This section will review the types of ontology reuse, namely inclusion, reference and semantic reuse.

### 4.1 Inclusion

Probably the most preferable way to reuse an existing ontology is to include it in our ontology. This means that we accept all of the axioms expressed in the original ontology. Most modern ontology languages, such as OWL or WSMML support the inclusion of external ontologies.

There are two types of inclusion: embedding and extension. *Embedding* means that a specific ontology, like an ontology of numbers, money or time is reused in a more complex ontology. As an example consider [30] where ontologies of time and location are reused in an ontology of international train tickets.

*Extension* means that a high-level ontology is extended by domain-specific concepts. For example this is the proposed way to extend the high-level business data ontology developed in D3.3 [24] in DIP use cases. Extension is also frequently used as a modularisation technique. E.g. the AKT Portal ontology<sup>1</sup> or the Proton ontology<sup>2</sup> extract their high-level, domain-independent part into an external mini-ontology, which is then extended by lower-level, domain specific parts.

### 4.2 Reference

Probably the most common ontology reuse in present Web ontologies is referencing specific elements of external ontologies. Modern ontology languages denote their elements by globally valid URIs. This makes it possible to reference specific elements of an ontology without including (and thus accepting) all axioms of an ontology. For example the FOAF ontology<sup>3</sup> references specific elements of WordNet<sup>4</sup> in such a way.

---

<sup>1</sup><http://www.aktors.org/ontology/portal>

<sup>2</sup><http://proton.semanticweb.org/>

<sup>3</sup><http://xmlns.com/foaf/0.1/>

<sup>4</sup><http://wordnet.princeton.edu/>

## 4.3 Semantic reuse

Although syntactic reuse through inclusion or reference is preferable, in many cases it is not possible to fully accept the semantics of an existing ontology. The existing ontology, on the other hand, could “almost” describe the entity that we want to model. It is also possible that the legacy ontology exactly describes our conceptualisation, but it was implemented using an ontology formalism that is not suitable for us, and therefore syntactic reuse is impossible. In this case we can build our ontology by freely reusing some of the ideas, design decisions of the original ontology without syntactically reusing any parts of it.

Although this makes automatic mapping between the resulting ontologies very hard or even impossible, this technique is still preferable over developing a completely new ontology. After semantic reuse the structures of the ontologies will be at least similar which makes manually defining a mapping between the ontologies easier. If two ontologies covering the same domain were developed completely independently, even manually defining a mapping could be a very challenging task.

Semantic reuse happens for example in the Proton ontology, where ideas of many other high-level ontologies, such as DOLCE<sup>5</sup> are reused. The same technique was used during the development of the DIP business data ontology (described in D3.3 [24]), where some ideas from the UBL<sup>6</sup> and the Shared Information/Data (SID) Model of the TeleManagement Forum<sup>7</sup> were reused.

---

<sup>5</sup><http://www.loa-cnr.it/DOLCE.html>

<sup>6</sup> Accessible from <http://docs.oasis-open.org/ubl/cd-UBL-1.0/>

<sup>7</sup> Accessible at <http://www.tmforum.org/browse.asp?catID=860&linkID=28357> status 31 May 2005

## 5 CONCLUSION

In this deliverable we presented an ontology development methodology based on the well-known METHONTOLOGY methodology which was extended with ideas from some other methodologies. The described methodology together with a list of best-practice ontology design principles will hopefully serve as a useful guideline for the development of future ontologies in and outside the DIP project.

---

## REFERENCES

- [1] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.
- [2] R. J. Brachman, D. L. McGuiness, P. F. Patel-Schneider, and L. A. Resnick. Living with CLASSIC: when and how to use a KL-ONE-like language. In J. Sowa, editor, *Principles of semantic networks*. Morgan Kaufmann, San Mateo, US, 1990.
- [3] P. Clark, J. Thompson, and B. Porter. Knowledge patterns. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 591–600, San Francisco, 2000. Morgan Kaufmann.
- [4] J. de Bruijn (editor). The web service modeling language WSML. Deliverable D16.1 version 0.2, WSMO, Mar. 2005. available from <http://www.wsmo.org/2004/d16/d16.1/v0.2/>.
- [5] A. R. (editor). Representing specified values in owl: “value partitions” and “value sets”. Working draft, W3C, 3 Aug. 2004.
- [6] C. F. (editor). WSMO primer. Deliverable D3.1 version 0.2, WSMO, Apr. 2005. available from <http://www.wsmo.org/TR/d3/d3.1/v0.2/>.
- [7] N. N. (editor). Representing classes as property values on the semantic web. Working draft, W3C, July 21 2004.
- [8] M. Fernández-López and A. Gómez-Pérez. Deliverable 1.4: A survey on methodologies for developing, maintaining, evaluating and reengineering ontologies. Technical report, EU IST Project IST-2000-29243 OntoWeb, 2002.
- [9] M. Fowler. *Analysis Patterns: Reusable Objects Models*. Addison Wesley, 1997.
- [10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Massachusetts, 1994.
- [11] A. Gómez-Pérez. *Handbook of Applied Expert Systems*, chapter Knowledge Sharing and Reuse. CRC Press, 1997.
- [12] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer, 1st edition, 2004.
- [13] T. R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.
- [14] N. Guarino and C. Welty. Evaluating ontological decisions with OntoClean. *Communications of the ACM*, 45(2):61–65, Feb. 2002.
- [15] J. Hunt. *Guide to the Unified Process featuring UML, Java and Design Patterns*. Springer Professional Computing. Springer Verlag, Sept. 2003.

- 
- [16] D. Jones, T. Bench-Capon, and P. Visser. Methodologies for ontology development. In *Proc. IT&KNOWS Conference, XV IFIP World Computer Congress*, Budapest, Hungary, Aug. 1998.
- [17] M. Kifer, G. Lausen, and W. James. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, May 1995.
- [18] A. Kiryakov, C. Drumm, L. Al-Jadir, A. Boukottaya, E. Kilgarriff, and J. Quantz. Mediation module specification: Business data-level mediation. Deliverable D5.2, DIP Consortium, Jan. 2005.
- [19] K. Mahesh. Ontology development for machine translation: Ideology and methodology, 9 June 1997.
- [20] D. L. McGuinness and F. van Harmelen (editors). Owl web ontology language overview. Recommendation, W3C, february 2004. available from <http://www.w3.org/TR/owl-features/>.
- [21] M. M. Montes, J. L. Bas, S. Bellido, O. Corcho, S. Losada, R. Benjamins, and J. Contreras. Financial ontology. Deliverable D10.3, DIP Consortium, Apr. 2005.
- [22] B. Motik, A. Maedche, and R. Volz. A conceptual modeling approach for semantics-driven enterprise applications. In *Proc. 1st Int'l Conf. on Ontologies, Databases and Application of Semantics (ODBASE-2002)*, Oct. 2002.
- [23] G. Nagypál. Creating an application-level ontology for the complex domain of history: Mission impossible? In *Proceedings of Lernen - Wissensentdeckung - Adaptivität (LWA 2004), FGWM 2004 Workshop*, pages 287–294, Berlin, Germany, 4–6 Oct. 2004.
- [24] G. Nagypál and J. Lemcke. A business data ontology. Deliverable D3.3, DIP Consortium, Jan. 2005.
- [25] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05 and SMI-2001-0880, Stanford Knowledge Systems Laboratory and Stanford Medical Informatics, 2001.
- [26] A. L. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In *Proceedings of the international conference on Knowledge capture*, pages 121–128. ACM Press, 2003.
- [27] A. L. Rector, C. Wroe, J. Rogers, and A. Roberts. Untangling taxonomies and relationships: personal and practical problems in loosely coupled development of large ontologies. In *Proceedings of the international conference on Knowledge capture*, pages 139–146. ACM Press, 2001.
- [28] D. Roman, H. Lausen, and U. K. (editors). Web service modeling ontology (WSMO). Deliverable D2 version 1.1, WSMO, Feb. 2005. available from <http://www.wsmo.org/TR/d2/v1.1>.
- [29] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. R. Shadbolt, and B. Wielinga. *Knowledge Engineering and Management – The CommonKADS Methodology*. MIT Press, 1 Jan. 2000.
-

- [30] M. Stollberg and R. L. (editors). WSMO use case modeling and testing. Deliverable D3.2 version 0.2, WSMO, Apr. 2005. available from <http://www.wsmo.org/TR/d3/d3.2/>.
- [31] Y. Sure. *Methodology, tools & case studies for ontology based knowledge management*. PhD thesis, University of Karlsruhe, May 2003.
- [32] Y. Sure and R. Studer. On-to-knowledge methodology. On-To-Knowledge Deliverable 18, AIFB, University of Karlsruhe, 2002. Available at [http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OTK-D18\\_v1-0.pdf](http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OTK-D18_v1-0.pdf).
- [33] D. Vrandecic, H. S. Pinto, Y. Sure, and C. Tempich. The DILIGENT knowledge processes. *Journal of Knowledge Management*, Apr. 2005. to appear.