



NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 — “Semantic-based knowledge and content systems”

D5.5.1 NeOn methodology for the development of large-scale semantic applications

Deliverable Co-ordinator: Óscar Muñoz-García

Deliverable Co-ordinating Institution: UPM

Other Authors: Raúl García-Castro (UPM), Asunción Gómez-Pérez (UPM), Margherita Sini (FAO)

This deliverable presents the first version of the NeOn methodology for the development of large-scale semantic applications. The methodology will help application developers to build semantic applications from scratch or to include semantic components into traditional information systems. The deliverable presents a characterization of large-scale semantic application, common use cases that appear when developing this kind of applications and a set of architectural patterns that can be used for modeling the architecture of a large-scale semantic application. Details are given for the *Requirements Engineering* and *Design* processes.

Document Identifier:	NEON/2009/D5.5.1/v1.0	Date due:	January 31, 2009
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	February 28, 2009
Project start date	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

<p>Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk</p>	<p>Universität Karlsruhe – TH (UKARL) Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 11 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de</p>
<p>Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es</p>	<p>Software AG (SAG) Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com</p>
<p>Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com</p>	<p>Institut ‘Jožef Stefan’ (JSI) Jamova 39 SL-1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si</p>
<p>Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l’Europe Montbonnot Saint Martin 38334 Saint-Ismier, France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr</p>	<p>University of Sheffield (USFD) Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield, United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk</p>
<p>Universität Koblenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de</p>	<p>Consiglio Nazionale delle Ricerche (CNR) Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it</p>
<p>Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de</p>	<p>Food and Agriculture Organization of the United Nations (FAO) Viale delle Terme di Caracalla 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org</p>
<p>Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.pariantelobo@atosorigin.com</p>	<p>Laboratorios KIN, S.A. (KIN) C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es</p>

Work package participants

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

- Universidad Politécnica de Madrid (UPM)
- Food and Agriculture Organization of the United Nations (FAO)

Change Log

Version	Date	Amended by	Changes
0.00	14-05-2008	Óscar Muñoz-García	Document preparation
0.01	21-05-2008	Óscar Muñoz-García	Table of Contents (added)
0.02	22-05-2008	Óscar Muñoz-García	Table of Contents (modified), WP5 Objectives and Main Tasks (added)
0.03	04-06-2008	Óscar Muñoz-García, Asunción Gómez-Pérez	Table of Contents (modified), State of the Art (added)
0.04	10-06-2008	Óscar Muñoz-García, Asunción Gómez-Pérez	Table of Contents (modified), Executive Summary (added), Introduction (added), Deliverable Main Goals and Contributions (added), Deliverable Structure (added), Relation with the Rest of WPs within the NeOn Project (added)
0.05	2-07-2008	Óscar Muñoz-García, Asunción Gómez-Pérez	Table of Contents (modified), Traditional Knowledge-Based Systems (added), First Generation of Semantic Web applications (added), Next Generation of Semantic Web applications (added), Classification of Semantic Web applications (added)
0.06	3-07-2008	Óscar Muñoz-García, Asunción Gómez-Pérez	Traditional Knowledge-Based Systems (modified), First Generation of Semantic Web applications (modified), Next Generation of Semantic Web applications (modified), Classification of Semantic Web applications (modified)
0.07	16-07-2008	Óscar Muñoz-García, Asunción Gómez-Pérez	Classification of Semantic Web applications (modified)
0.08	17-07-2008	Óscar Muñoz-García, Asunción Gómez-Pérez	Classification of Semantic Web applications (modified)
0.09	18-07-2008	Óscar Muñoz-García, Raúl García-Castro, Asunción Gómez-Pérez	The Semantic Web Framework (added)
0.10	21-07-2008	Óscar Muñoz-García	Component-based development (added), State of the Art (modified)
0.11	22-07-2008	Óscar Muñoz-García	Component-based development (modified), Table of contents (modified), Agile Software Development (added)
0.12	31-07-2008	Óscar Muñoz-García	Classification of Semantic Web applications (modified)
0.13	31-07-2008	Óscar Muñoz-García	Classification of Semantic Web applications (modified)

0.14	9-08-2008	Raúl García-Castro	Revision
0.15	11-08-2008	Asunción Gómez-Pérez	Revision
0.16	12-08-2008	Óscar Muñoz-García	Executive Summary (modified)
0.17	13-08-2008	Óscar Muñoz-García	Introduction (modified)
0.18	14-08-2008	Óscar Muñoz-García	Definitions for Methodology, Method, Technique, Process, Activity and Task (added)
0.19	15-08-2008	Óscar Muñoz-García	Component-based development (modified)
0.20	25-08-2008	Óscar Muñoz-García	Agile software development (modified)
0.21	26-08-2008	Óscar Muñoz-García	Characteristics of Semantic Applications (modified)
0.22	27-08-2008	Óscar Muñoz-García	Executive Summary and Introduction (modified)
0.23	28-08-2008	Óscar Muñoz-García	Agile software development (modified)
0.24	29-08-2008	Óscar Muñoz-García	Scenarios for building semantic applications (added)
0.25	1-09-2008	Raúl García-Castro	Revision
0.50	1-10-2008	Óscar Muñoz-García	Changes with respect to the previous revision
0.51	2-10-2008	Raúl García-Castro	Revision
0.52	1-11-2008	Óscar Muñoz-García	Changes with respect to the previous revision
0.53	1-12-2008	Asunción Gómez-Pérez	Revision
0.60	19-1-2009	Óscar Muñoz-García	Changes with respect to the previous revision
0.61	1-2-2009	Óscar Muñoz-García	Sent to QA
0.62	28-2-2009	Óscar Muñoz-García	Changes after QA
0.63	1-3-2008	Asunción Gómez-Pérez	Revision
0.64	10-3-2009	Óscar Muñoz-García	Changes with respect to the previous revision
0.65	11-3-2009	Raúl García-Castro	Revision
0.64	20-3-2009	Óscar Muñoz-García	Changes with respect to the previous revision
0.65	21-3-2008	Asunción Gómez-Pérez	Revision
0.99	9-4-2009	Óscar Muñoz-García	Changes with respect to the previous revision
1.0	10-4-2008	Asunción Gómez-Pérez	Revision

Executive Summary

Software development methodologies are broadly used in Software Engineering (e.g. the Rational Unified Process [Kru00]) and Knowledge Engineering. For example, the CommonKADS [SAA⁺00] approach is a methodology that helps to design and implement knowledge-based systems. In the past several methodologies have been developed to support the creation and management as well as the population of single ontologies. However, the ontology engineering field lacks a methodology for developing applications that incorporate and use ontologies in what we call semantic applications.

The next generation of semantic applications will need to deal with significant problems associated with the scale and heterogeneity of the data they must manipulate as well as with the different location of the resources, the varying degrees of quality of the information contained in them, the multilinguality aspects, etc. The next generation of semantic applications will produce and consume its own and external data and will not be bound to a particular domain, selecting the appropriate knowledge from different sources according to application-dependent criteria such as the quality of the data and, therefore, the knowledge can be exploited jointly.

The scientific goal of this deliverable is to define the first version of a methodology for the rapid prototyping and development of large scale semantic applications that, which help application developers to build semantic applications from scratch or to include semantic components into traditional information systems. This methodology is based on the state of the art of rapid software development and component-based software engineering and its aim is to guide the developer through the application development process.

The principles that guide the construction of such a methodology are the following:

1. The methodology should be general enough so that it can help software developers to build large-scale semantic applications independently of the development platform used.
2. The methodology should define each process and activity precisely, stating clearly its purpose, its inputs and outputs, the actors involved, when its execution is more convenient its execution, and the set of methods and techniques to be used for performing the activity. Furthermore, the methodology should provide prescriptive guidelines for each process or activity.
3. The methodology should be presented in a manner non-oriented to researchers to facilitate a promptly assimilation by software developers and ontology practitioners. Examples of how to use the methodology in different use cases should be provided.

For building this methodology a characterization of semantic applications has been made taking into account the existing literature on semantic applications and different use cases. With these characteristics, we have defined a set of scenarios and provided a set of architectural patterns for each scenario. Such patterns will be extremely useful for developers when designing the semantic applications.

The architectural patterns have been built using the components identified in the Semantic Web Framework [GGMN08], a joint work of partners in the Knowledge Web Project (FP6-507482). The Semantic Web Framework is defined as a structure in which Semantic Web applications can be organized and developed [GGMN08].

Since the Semantic Web Framework components are defined at the conceptual level and are decoupled of the technology that implements such components, the architectural patterns included in this deliverable

are not bound to a particular implementation. So, after using this NeOn methodology, the architecture of the application will remain independent of concrete component implementations. Architecture realizations in particular settings are out of the scope of this deliverable.

We describe the methodology with a software engineering approach that will permit software developers and ontology practitioners to speed up the process; we also include examples of how to use the methodology in the NeOn use cases.

This deliverable does not treat the building of ontologies since this issue is dealt with in the other WP5 deliverables [SAB⁺07, SAB⁺08, SFG⁺08]. Therefore, the scope of this deliverable is limited to the following topics:

1. The *Requirements Engineering* process and, concretely, the *Requirements Elicitation and Analysis* activity, in which the requirements of a large-scale semantic application are discovered.
2. The *Design* process, and concretely, the *Component Identification* activity, in which the component architecture of the semantic application is designed.

The second version of this deliverable will include the description of the other of the large-scale semantic applications development processes: *Provisioning and Development*, *Integration* and *Testing*.

Contents

1	Introduction	18
1.1	WP5 Goals	18
1.2	Deliverable Goal	19
1.3	Deliverable Structure	20
2	Related Work in Software Engineering	22
2.1	Definitions for Methodology, Method, Technique, Process, Activity and Task	22
2.2	Component Based Development	22
2.2.1	Requirements	24
2.2.2	Specification	25
2.2.3	Provisioning	25
2.2.4	Assembly	25
2.2.5	Testing	25
2.2.6	Deployment	26
2.3	Agile Software Development	26
2.3.1	Extreme Programming	26
2.3.2	SCRUM	27
2.3.3	The Crystal Methods	28
2.3.4	Feature Driven Development	29
2.3.5	Comparison of Presented Methods	29
2.4	Requirements Engineering Process	30
2.4.1	Methods	30
2.4.2	Techniques	32
2.5	Design Process	33
2.5.1	Methods	33
2.5.2	Techniques	35
2.6	Conclusions	37
3	Related Work in the Characterisation and Classification of Semantic Applications	38
3.1	Characteristics of Semantic Applications	38
3.1.1	Traditional Knowledge-Based Systems Characteristics	39
3.1.2	Semantic Web Applications Characteristics	39
3.1.3	Comparison of the Characteristics Presented	40
3.2	Scenarios for building Semantic Applications	41
3.2.1	Classification of Ontology Applications	41
3.2.2	Classification of Semantic Web Applications	44

3.3	The Semantic Web Framework	45
3.3.1	Definition and Classification of Components	45
3.4	Conclusions	49
4	Research Methodology	50
4.1	General Framework for Describing the Methodology	50
4.2	Requirements for the NeOn Methodology for Building Large-Scale Semantic Applications	52
4.2.1	Generic requirements	52
4.2.2	Specific requirements	54
5	NeOn Methodology for Building Large Scale Semantic Web Applications	55
5.1	Requirements Engineering	55
5.1.1	Inputs	56
5.1.2	Outputs	58
5.2	Design	58
5.2.1	Inputs	58
5.2.2	Output	59
6	Semantic Application Characteristics	60
6.1	Ontologies Dimension	60
6.2	Data Dimension	62
6.3	Reasoning Dimension	62
6.4	Non-functional Characteristics Dimension	63
7	Use Cases Catalogue	64
7.1	Query Information Use Case	64
7.2	Search Resources Use Case	67
7.3	Browse Resources Use Case	68
7.4	Extract Information Use Case	70
7.5	Manage Knowledge Use Case	71
8	System Models Catalogue	74
8.1	Basic Symbols	74
8.1.1	Resources	74
8.1.2	Dynamic Resources	77
8.1.3	Applications and Systems	78
8.2	Relationships Between Symbols	79
8.2.1	Conforms To	79
8.2.2	Aligned With	80
8.2.3	Annotate	81
8.3	Basic Templates	83
8.3.1	Data Sources with Schema	83
8.3.2	Annotated Resources	85
8.4	System Models	88
8.4.1	Query Information System Models	89
8.4.2	Search Resources System Model	91

8.4.3	Browse Resources System Models	92
8.4.4	Extract Information System Model	93
8.4.5	Manage Knowledge System Models	93
8.5	Examples	95
8.5.1	Example 1	95
8.5.2	Example 2	95
8.5.3	Example 3	95
9	Architectural Patterns	98
9.1	Semantic Web Framework Component Interfaces Description	98
9.1.1	Data and Metadata Management	99
9.1.2	Querying and Reasoning	103
9.1.3	Ontology Engineering	105
9.1.4	Ontology Customization	109
9.1.5	Ontology Instance Generation	112
9.2	Components Not Defined in the SWF	114
9.2.1	Non-ontological Resource Discovery and Ranking	114
9.3	Components associated to Basic System Models Symbols	115
9.3.1	Components Associated to Resources	115
9.4	Components associated to Relationships Between Symbols in System Models	119
9.4.1	Conforms To	119
9.4.2	Aligned With	119
9.4.3	Annotate	125
9.5	Components Associated to the Basic Templates	132
9.5.1	Components Associated to Data Sources with Schema	133
9.5.2	Components Associated to Annotated Resources	136
9.6	Components Associated to System Models	136
9.6.1	Query Information	136
9.6.2	Search Resources	137
9.6.3	Browse Resources	139
9.6.4	Extract Information	140
9.6.5	Edit	142
9.6.6	Populate	143
9.6.7	Learn	145
10	Requirements Engineering Process	147
10.1	Proposed Guidelines for Requirements Elicitation and Analysis	147
10.1.1	Task 1. To Identify the Use Cases	148
10.1.2	Task 2. To Identify the Semantic Characteristics and Ontological Needs	149
10.1.3	Task 3. To Identify System Models	149
10.1.4	Task 4. To Document Requirements	150
10.1.5	Task 5. To Estimate Requirements	150
10.1.6	Task 6. To Prioritize Requirements	150
11	Design Process	152

11.1 Proposed Guidelines for Component Identification	152
11.1.1 Task 1. To Identify Dialogs and System Facades	152
11.1.2 Task 2. To Identify Knowledge Sources	153
11.1.3 Task 3. To Create the Initial Architecture	155
12 Fictitious Example	156
12.1 Requirements Elicitation and Analysis Activity	156
12.1.1 Business Requirements	156
12.1.2 Task 1. To Identify the Use Cases	157
12.1.3 Task 2. To Identify Application Characteristics and Ontological Needs	164
12.1.4 Task 3. To Identify System Models	165
12.2 Component Identification Activity	166
12.2.1 Task 1. To Identify Dialogs and System Facades	166
12.2.2 Task 2. To Identify Interfaces to Knowledge Sources	166
12.2.3 Task 3. To Create the Initial Architecture	167
13 Real Example. FAO case study	169
13.1 Requirements Elicitation and Analysis Activity	169
13.1.1 Business Requirements	169
13.1.2 Task 1. To Identify the Use Cases	170
13.1.3 Task 2. To Identify Application Characteristics and Ontological Needs	176
13.1.4 Task 3. To Identify System Models	177
13.2 Component Identification Activity	178
13.2.1 Task 1. To Identify Dialogs and System Facades	179
13.2.2 Task 2. To Identify Interfaces to Knowledge Sources	180
13.2.3 Task 3. To Create the Initial Architecture	180
14 Conclusions and Future Work	183
A Questionnaires for Identifying the Characteristics of Semantic Applications	185
Bibliography	195

List of Tables

2.1	Prescriptive Characteristics of Agile Methods [CLC03]	29
3.1	Properties of different kinds of semantic applications	41
4.1	Template for Process and Activity Filling Card	52
7.1	Mapping between the scenarios analysed and the use cases obtained	65
7.2	Template for describing use-cases [Lar05]	65
7.3	<i>Query Information</i> use case template	67
7.4	<i>Search Resources</i> use case template	68
7.5	<i>Browse Resources</i> use case template	70
7.6	<i>Extract Information</i> use case template	71
7.7	<i>Manage Knowledge</i> use case template	73
8.1	Symbol 1. Static Ontology	75
8.2	Symbol 2. Static Instances	75
8.3	Symbol 3. Static Non-ontological Resource Schema	76
8.4	Symbol 4. Static Non-ontological Resource Content	76
8.5	Symbol 5. Static Unstructured Document	76
8.6	Symbol 6. Dynamic Ontology	77
8.7	Symbol 7. Dynamic Instances	77
8.8	Symbol 8. Dynamic Non-ontological Resource Schema	77
8.9	Symbol 9. Dynamic Non-ontological Resource Content	78
8.10	Symbol 10. Dynamic Unstructured Document	78
8.11	Symbol 11. Application	78
8.12	Symbol 12. System Limit	79
8.13	Relationship 1. Instances that Conform To a Given Ontology	80
8.14	Relationship 2. Non-ontological Resource Content that Conforms To a Given Schema	80
8.15	Symbol 13. Non-ontological Resource Content that Conforms To a Given Schema Abbreviation	80
8.16	Relationship 3. Two Aligned Ontologies	81
8.17	Relationship 4. Non-ontological Schema Aligned With an Ontology	81
8.18	Relationship 5. Unstructured Document Annotated by a set of Instances	82
8.19	Relationship 6. Unstructured Document Annotated by Non-ontological Metadata	82
8.20	Relationship 7. Ontology Annotated by a Set of Instances	82
8.21	Relationship 8. Ontology Annotated by a Non-ontological Metadata	83
8.22	Basic Template 1. Data Sources with schema: ontological resources	84

8.23 Basic Template 2. Data Sources with Schema: Non-ontological Resources	84
8.24 Basic Template 3. Data Sources with Schema: Ontological and Non-ontological resources	85
8.25 Basic Template 4. Annotated Documents with Ontological Metadata	86
8.26 Basic Template 5. Annotated Documents with Non-ontological Metadata	87
8.27 Basic Template 6. Annotated Ontologies with Ontological Metadata	87
8.28 Basic Template 7. Annotated Ontologies with Non-ontological Metadata	88
8.29 System Model 1. Query information with a Single Ontology/Schema Approach	89
8.30 System Model 2. Query information with a Multiple Ontology Approach	90
8.31 System Model 3. Query information with a Hybrid Ontology Approach	91
8.32 System Model 4. Search Resources	92
8.33 System Model 5. Browse Annotated Resources	92
8.34 System Model 6. Browse Ontological Resources	92
8.35 System Model 7. Extract Information	93
8.36 System Model 8. Edit Ontological Data Sources	93
8.37 System Model 9. Edit Annotations	94
8.38 System Model 10. Populate	94
8.39 System Model 11. Learn	94
9.1 Stereotypes used for describing the SWF dimensions	99
9.2 Interfaces provided and required of the <i>Information Directory Manager</i> component	100
9.3 Interfaces provided and required of the <i>Ontology Repository</i> component	101
9.4 Interfaces provided and required of the <i>Data Repository</i> component	102
9.5 Interfaces provided and required of the <i>Alignment Repository</i> component	102
9.6 Interfaces provided and required of the <i>Metadata Registry</i> component	103
9.7 Interfaces provided and required of the <i>Query Answering</i> component	104
9.8 Interfaces provided and required of the <i>Semantic Query Processor</i> component	105
9.9 Interfaces provided and required of the <i>Semantic Query Editor</i> component	105
9.10 Provided and required interfaces of the <i>Ontology Editor</i> component	106
9.11 Interfaces provided and required of the <i>Ontology Browser</i> component	106
9.12 Interfaces provided and required of the <i>Ontology Matcher</i> component	108
9.13 Interfaces provided and required of the <i>Ontology Learner</i> component	108
9.14 Interfaces provided and required of the <i>Ontology Evaluator</i> component	109
9.15 Interfaces provided and required of the <i>Ontology Localization and Profiling</i> component	110
9.16 Interfaces provided and required of the <i>Ontology Discovery and Ranking</i> component	111
9.17 Provided and required interfaces of the <i>Ontology Adaptation Operators</i> component	111
9.18 Interfaces provided and required of the <i>Ontology View Customization</i> component	112
9.19 Interfaces provided and required of the <i>Manual Annotation</i> component	113
9.20 Interfaces provided and required of the <i>Automatic Annotation</i> component	113
9.21 Provided and required interfaces of the <i>Ontology Populator</i> component	114
9.22 Interfaces provided and required of the <i>Non-ontological Resource Discovery and Ranking</i> component	115
9.23 <i>Ontological Resource Access and Management</i>	116
9.24 <i>Non-ontological Resource Access and Management</i>	116
9.25 <i>Dynamic Ontological Resource Access</i>	117
9.26 <i>Dynamic Ontological Resource Access via Federated Discoverers</i>	118

9.27 <i>Dynamic Non-ontological Resource Access</i>	118
9.28 <i>Alignment between Ontologies Access and Management</i>	120
9.29 <i>Alignment between Ontology and Non-ontological Resource Schema Access and Management</i>	121
9.30 <i>Alignment between Dynamic Ontology Access and Management</i>	122
9.31 <i>Alignment between a Dynamic Ontology and a Dynamic Non-ontological Resource Access and Management</i>	123
9.32 <i>Alignment between a Dynamic Ontology and a Static Non-ontological resource Access and Management</i>	124
9.33 <i>Alignment between a Static Ontology and a Dynamic Non-ontological resource Access and Management</i>	125
9.34 <i>Annotations within documents Access and Management</i>	126
9.35 <i>Annotations outside documents Access and Management</i>	127
9.36 <i>Annotations of discovered documents Access and Management</i>	128
9.37 <i>Annotations of Documents according to a Discovered Ontology Access and Management</i> . . .	129
9.38 <i>Annotations of Discovered Documents according to a discovered Ontology Access and Man- agement</i>	130
9.39 <i>Ontologies Annotated with Ontological Metadata Access and Management</i>	131
9.40 <i>Ontologies Annotated with Non-ontological Resource Content Access and Management</i>	132
9.41 <i>Ontological Data Sources Access and Management</i>	133
9.42 <i>Non-ontological Data Sources Access and Management</i>	134
9.43 <i>Ontological and Non-ontological Data Sources Access and Management</i>	135
9.44 <i>Query Information</i>	137
9.45 <i>Search Resources</i>	139
9.46 <i>Browse Resources</i>	140
9.47 <i>Extract Information</i>	142
9.48 <i>Edit</i>	143
9.49 <i>Populate</i>	144
9.50 <i>Learn</i>	146
10.1 General description of the <i>Requirements Elicitation and Analysis</i> activity	148
10.2 The structure of the requirements document	151
11.1 Component Identification Filling Card	153
11.2 Mapping between the tasks described in the state of the art and the tasks proposed by the NeOn methodology for the <i>Component Identification</i> activity	154
12.1 <i>Obtain Optimum Route</i> use case	161
12.2 <i>Track Shipment</i> use case	164
12.3 Symbols associated to the resources used by the example application	165
12.4 Relationships between the resources used in the example application	166
12.5 Patterns associated to the repositories used by the example application	168
13.1 <i>Search Concept</i> use case	173
13.2 <i>Manage Terms</i> use case	174
13.3 <i>Corpus Analysis</i> use case	176
13.4 Characteristics of the FAO case study with respect to the dimension of the ontologies.	176

13.5 Characteristics of the FAO case study with respect to the data dimension.	176
13.6 Characteristics of the FAO case study with respect to the reasoning dimension.	177
13.7 Characteristics of the FAO case study with respect to the non-functional dimension.	177
13.8 Symbols associated to the resources used by the AGROVOC Concept Server application . . .	178
13.9 Relationships between the resources used by the example application	178
13.10 Patterns associated to the repositories used by the FAO application	180

List of Figures

2.1	Process, Activities and Tasks	23
2.2	The workflow in the overall component development process [CD01]	24
2.3	Scrum life cycle (from http://controlchaos.com/)	28
2.4	The requirements engineering process [Som07]	30
2.5	Spiral model of requirements engineering process [Som07]	31
2.6	The requirements elicitation and analysis activity [Som07]	32
2.7	The <i>Component identification</i> activity [CD01]	34
2.8	Different types of contracts [CD01]	35
3.1	Common access to information scenario. Data access via shared ontology variation [JU99]	43
3.2	Common access to information scenario. Data access via mapped ontologies variation [JU99]	43
3.3	Components of the Semantic Web Framework	46
3.4	Dependencies of the components in the <i>Ontology engineering</i> dimension	47
4.1	Inputs considered for obtaining the NeOn Methodology and building Large-scale Semantic Applications	51
5.1	Overview of the <i>Requirements Engineering and Design</i> Processes	56
6.1	Characteristics of Large-scale Semantic Applications	60
8.1	Datasources with schema	83
8.2	Annotated Resources	85
8.3	<i>Obtain Information</i> system model election criteria	91
8.4	<i>Obtain Information</i> use case with an example of the single ontology approach	95
8.5	<i>Obtain Information</i> use case with an example of the multiple ontology approach	96
8.6	<i>Obtain Information</i> use case with an example of the hybrid ontology approach	97
9.1	Semantic Web Framework component representation	99
10.1	Tasks in the <i>Requirements Elicitation and Analysis</i> activity	149
11.1	The component identification activity	154
12.1	Identified use cases of the fictitious case study	157
12.2	System model identified for the fictitious example	167
12.3	Example Application Architecture	168
13.1	Use cases identified in the FAO case study	171

13.2 Identified system model for the FAO case study	179
13.3 FAO Application Architecture (Search Concept)	181
13.4 FAO Application Architecture (Manage Terms)	181
13.5 FAO Application Architecture (Corpus Analysis)	182

Chapter 1

Introduction

Software development methodologies are broadly used in Software Engineering (e.g. the Rational Unified Process [Kru00]) and Knowledge Engineering (e.g. the CommonKADS [SAA⁺00] approach).

In the past several methodologies have been developed to support the creation and management as well as the population of single ontologies.

However, the ontology engineering field lacks of a methodology for developing applications that incorporate and use ontologies in what we call semantic applications.

This deliverable defines the first version of a methodology for the rapid prototyping and development of large-scale semantic applications that can help application developers to build semantic applications from scratch or to include semantic components into traditional information systems.

1.1 WP5 Goals

The aim of the NeOn project is to create a service-oriented, open infrastructure, and associated methodology to support the overall development life-cycle of a new generation of large-scale and complex semantic applications. The NeOn infrastructure and methodology will enable the efficient implementation of semantic applications in open environments, such as the Semantic Web, in support of the automation of Business to Business relationships, and also in company intranets. Its aim is to extend the state of the art to ensure that economically viable solutions will come on the market.

In this context, the main objectives of WP5 are

1. To create the NeOn methodology that supports the collaborative aspects of ontology development and the reuse and dynamic evolution of networked ontologies in distributed environments, in which contextual information is introduced by developers (domain experts, ontology practitioners) at different stages of the ontology development process.
2. To create the NeOn methodology for the development of large-scale semantic applications that supports the reference architecture and the service oriented infrastructure developed in WP6.
3. To provide qualitative and quantitative experimental evidence of how the ontology and system developments improve by following the two NeOn methodologies.

These objectives will be achieved by investigating the following tasks:

- Task 5.3. Identification and definition of the development process and life cycle for networks of ontologies. Results of these researches were included in D5.3.1 [SAB⁺07] and 5.3.2 [SFG⁺08].
- Task 5.4. To define the NeOn methodology for building collaboratively ontology networks. The methodology will include methods, techniques and tools for carrying out the activities identified and defined

in the ontology network development process. Results of these researches were included in D5.4.1 [SAB⁺08] and will be included in D5.4.2 to be delivered in March, 2009.

- Task 5.5. To define the NeOn methodology for developing large-scale semantic applications from the initial phases (requirement analysis) of the development process to the stage prior to the implementation. A set of developer-oriented reference specifications will be defined in this document. These specifications will serve as skeleton for adapting the selected components and for developing new complex semantic-based software components and semantic applications.
- Task 5.6. Experimentation with NeOn methodologies. In this task experiments, methods, and metrics are proposed for evaluating the main outcomes produced in this WP. The goal is to provide qualitative and quantitative evidence that ontologies and systems are built faster and better with the NeOn methodologies.

1.2 Deliverable Goal

A large-scale semantic application is a semantic application that makes use of semantic technologies and that manipulates huge quantities of heterogeneous decentralised knowledge and semantic data presenting different degrees of quality. The application produces and consumes its own and external data and retrieves knowledge automatically by exploring different sources. This, consequently, creates new problems, such as the need of selecting appropriate knowledge according to a given criteria, or the quality of the data and its adequacy to the task at hand [AMS⁺08].

Since the Semantic Web is a particular domain for semantic applications, it is also a large-scale source of knowledge that requires to design applications in a different fashion than the classic KBS [AMS⁺08]. The next generation of Semantic Web applications will need to deal with significant problems associated with the scale and heterogeneity of the Semantic Web, with the varying degrees of quality of the information contained in it [AMS⁺08] and with problems such provenance and others. These problems do not only appear in Semantic Web applications but also in knowledge management systems or data interpretation systems that use Semantics.

The main scientific goal of this deliverable is to produce a methodology for the rapid prototyping and development of a new generation of practical, large-scale semantic applications by drawing on contextualised networked ontologies, heterogeneous data and other knowledge-level resources. This methodology will provide the necessary framework to organise and manage the development of semantic applications à-la NeOn, thus ensuring an early uptake of NeOn technologies. The methodology will help application developers to build semantic applications from scratch, or to include semantic components into traditional information systems. The methodology will be generalized and evaluated in a case study addressed in WP7.

Since the industry and technology move too fast, requirements change at rates that swamp traditional methods [HOC00]; on the other hand, customers have become increasingly unable to definitively state their needs in advance while, at the same time, they expect more from their software [CLC03]. Therefore, the state of the art of Agile Methods will be taken into account since it enables the rapid development of large-scale semantic applications covered by the methodology presented in this deliverable.

Recently a popular approach to software development [Obe06] has emerged; it consists in constructing applications from a collection of reusable components and frameworks. Both elements offer a number of benefits since they simplify application development and maintenance, allowing systems to be more adaptive and to respond rapidly to changing requirements [Obe06]. An approach that describes reusable components for semantic applications is the Semantic Web Framework (SWF) [GGMN08]. The SWF has been developed within the Knowledge Web Project (FP6-507482); it describes the software used to build Semantic Web applications as reusable components. Besides using the Semantic Web Framework as a starting point, the NeOn methodology for building large-scale Semantic applications will take advantage of the methods existing for building component-based software.

The principles that guide the construction of such a methodology are the following:

1. The methodology should be general enough so that it can help software developers to build large-scale semantic applications independently of the development platform used.
2. The methodology should define each process and activity precisely, stating clearly its purpose, its inputs and outputs, the actors involved, when its execution is more convenient its execution, and the set of methods and techniques to be used for performing the activity. Furthermore, the methodology should provide prescriptive guidelines for each process or activity.
3. The methodology should be presented in a manner non-oriented to researchers to facilitate a promptly assimilation by software developers and ontology practitioners. Examples of how to use the methodology in different use cases should be provided.

For building this methodology a characterization of semantic applications has been made taking into account the existing literature on semantic applications and different use cases. With these characteristics, we have defined a set of scenarios and provided a set of architectural patterns for each scenario. Such patterns will be extremely useful for developers when designing the semantic applications.

The architectural patterns have been built using the components identified in the Semantic Web Framework [GGMN08], a joint work of partners in the Knowledge Web Project (FP6-507482). The Semantic Web Framework is defined as a structure in which Semantic Web applications can be organized and developed [GGMN08].

Since the Semantic Web Framework components are defined at the conceptual level and are decoupled of the technology that implements such components, the architectural patterns included in this deliverable are not bound to a particular implementation. So, after using this NeOn methodology, the architecture of the application will remain independent of concrete component implementations. Architecture realizations in particular settings are out of the scope of this deliverable.

We describe the methodology with a software engineering approach that will permit software developers and ontology practitioners to speed up the process; we also include examples of how to use the methodology in the NeOn use cases.

This deliverable does not treat the building of ontologies since this issue is dealt with in the other WP5 deliverables [SAB⁺07, SAB⁺08, SFG⁺08]. Therefore, the scope of this deliverable is limited to the following topics:

1. The *Requirements Engineering* process and, concretely, the *Requirements Elicitation and Analysis* activity, in which the requirements of a large-scale semantic application are discovered.
2. The *Design* process, and concretely, the *Component Identification* activity, in which the component architecture of the semantic application is designed.

The second version of this deliverable will include the description of the other of the large-scale semantic applications development processes: *Provisioning and Development*, *Integration* and *Testing*.

1.3 Deliverable Structure

The deliverable is structured as follows:

- Chapter 2 presents some related work in Software Engineering and, concretely, agile methods for rapid software engineering and component-based software engineering.
- Chapter 3 analyses the state of the art of characterizations and classifications of semantic applications and different scenarios where semantic applications can be applied. This chapter also summarizes the

Semantic Web Framework, a component-based framework that describes the software components involved in the architecture of semantic applications.

- Chapter 4 presents the research methodology used to build the NeOn methodology for the development of large-scale semantic applications.
- Chapter 5 introduces the global vision of the methodology presented in this deliverable summarizing the processes here treated.
- Chapter 6 identifies a set of characteristics of large-scale semantic applications.
- Chapter 7 introduces a set of use case templates that can be adapted to describe the scenarios to be solved by the semantic application.
- Chapter 8 presents the catalogue of system model templates used for modeling the structure of a semantic application in a graphical fashion.
- Chapter 9 deals with a set of architectural patterns that can be applied to model the architecture of a large-scale semantic application.
- Chapter 10 describes the *Requirements Engineering* process and proposes a set of guidelines for carrying out the *Requirements Elicitation and Analysis* activity.
- Chapter 11 describes the *Design* process and proposes a set of guidelines for carrying out the *Component Identification* activity.
- Chapter 12 introduces an example showing how to carry out the *Requirements Elicitation and Analysis* and *Component Identification* activities, given a fictitious case study.
- Chapter 13 presents an example showing how the *Requirements Elicitation and Analysis* and *Component Identification* activities for the AGROVOC Concept Server Workbench 3.0 have been carried out. The AGROVOC Concept Server Workbench 3.0 is a web-based distributed collaborative tool for managing multilingual ontologies about agriculture.
- Chapter 14 presents the conclusions to this deliverable and proposes future work.

Chapter 2

Related Work in Software Engineering

With respect to Software Engineering, the methodology for the development of large-scale semantic applications is based on the following pillars: agile methods for rapid software development and component-based software engineering. This chapter covers the state of the art in both topics.

2.1 Definitions for Methodology, Method, Technique, Process, Activity and Task

Throughout literature, the terms *methodology*, *method*, *technique*, *process*, *activity*, etc. are used indistinctively. To make a clear use of these terms, we have adopted several IEEE ¹ definitions in this deliverable, which are described in detail in [IEE95a, Som07, IEE90, IEE97, IEE95b].

- **Methodology.** A comprehensive, integrated series of techniques or methods that create a general system theory of how a class of thought-intensive work ought to be performed [IEE95a].
- **Method.** Methods are parts of methodologies. A method is a set of “orderly processes or procedures used in the engineering of a product or in performing a service” [Som07]. Methods are composed of processes.
- **Technique.** Techniques are parts of methodologies. Techniques are “the application of accumulated technical or management skills and methods in the creation of a product or in performing a service” [IEE90]. Techniques detail methods.
- **Process.** A set of activities whose goal is the development or the evolution of software [Som07].
- **Activity.** A defined body of work to be performed, including its required input and output information [IEE97]. Activities can be divided into zero or more tasks.
- **Task.** The smallest unit of work subject to management accountability. A task is a well-defined work assignment for one or more project members. Related tasks are usually grouped to form activities [IEE95b].

Within this deliverable we use this terminology, which is also shown in Figure 2.1.

2.2 Component Based Development

Reuse-based software engineering is becoming the main development approach for business and commercial systems [Som07]. One of this reuse-based approaches is Component-Based Software Engineering

¹<http://www.ieee.org>

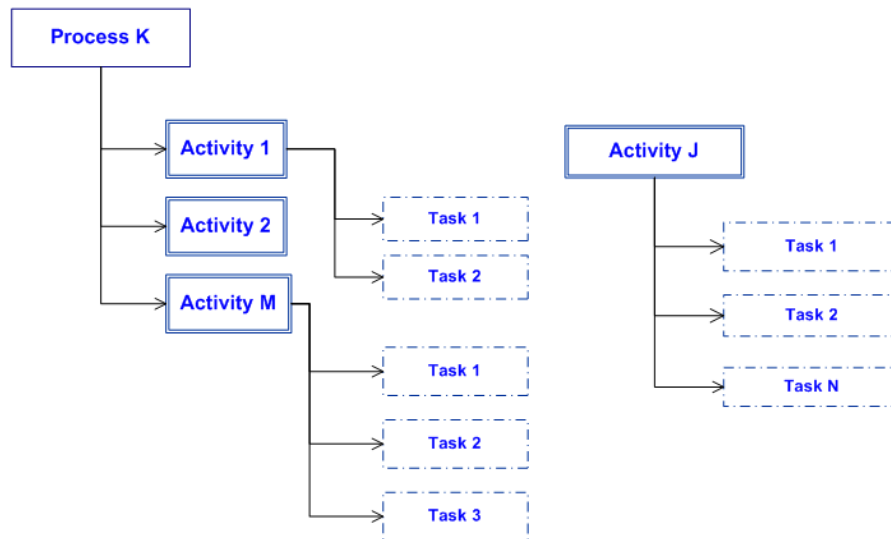


Figure 2.1: Process, Activities and Tasks

(CBSE), which is the process of defining, implementing and integrating or composing loosely coupled independent components into systems.

The CBSE approach is based on the existence of a significant number of reusable components. The application development process is focused on integrating these components into a application rather than on developing them from scratch [Som07].

CBSE is used as an attempt to reduce development costs by incorporating code and previously proven designs in new software products. The basic premise behind this model is that systems should be built with existing components, as opposed to custom-building new components. The net effect of reusing components would be shorter development schedules and more reliable software since the developer uses components that have been previously “shaken down”.

This model relies on [Som07]

1. *Independent components* that are completely specified by their interfaces.
2. *Component standards* that facilitate the integration of components.
3. *Middleware* that provides software support for component integration.
4. *A development process* that is geared to CBSE.

If it is possible to find and use a software component that fulfills the requirements at hand, this will be in general cheaper and include more functionalities than an in-house component would. Furthermore, its quality is known and it is immediately available. With this approach the amount of software to be developed is reduced and so are cost and risks. This reduction usually leads leads to faster delivery of the software.

Component-based frameworks [Joh97] provide the features that facilitate software reuse [Kru92]: *abstraction*, to reduce and factor out details; *selection*, to help developers locate, compare and select reusable software artifacts; *specialization*, to allow specializing generic artifacts; and *integration*, to combine a collection of artifacts.

Several approaches to developing component-based software have been described thoroughly in literature (e.g., see [Som07, SW99, CD01]). The process for developing component-based software described in [CD01] has been selected because of its simplicity and completeness. This process is shown in Figure 2.2. In the figure the boxes represent workflows², and the thin arrows represent the flow of *artifacts* – deliverables

²According this deliverable terminology, the term *workflow* correspond to the term *process*.

that carry information between workflows³. Next, each of the workflows and the artifacts produced during each workflow execution are described.

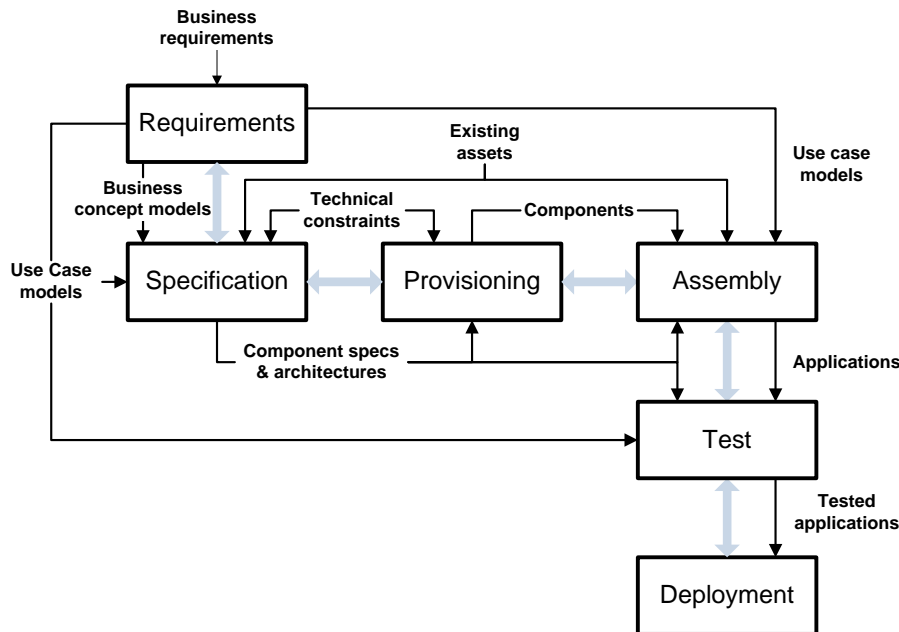


Figure 2.2: The workflow in the overall component development process [CD01]

The initial artifacts are the input to the method described in this section and, therefore, must be present before starting it. They are the following:

- *Business Requirements*. They enumerate the user requirements over what features the system is expected to provide. Requirements should be described in an understandable manner to those system users who do not have specific technical knowledge. They should only specify the external behaviour of the system and avoid, as far as possible, system design characteristics. Business requirements should be written in a natural language, with simple tables and forms and intuitive diagrams [Som07].
- *Existing assets*. Business existing and project useful assets such as legacy systems (software packages, data bases, etc.). Legacy systems are socio-technical computer-based systems that have been developed in the past, often with older or obsolete technology. Legacy systems are often business-critical systems, which are maintained because it is too risky to replace them [Som07].
- *Technical constraints*. Constraints on the system such as data representations used in system interfaces or the use of particular programming languages, frameworks, tools, etc.

2.2.1 Requirements

This workflow corresponds to the *Requirements Engineering* software engineering process. *Requirements Engineering* is the process of finding out, analyzing, documenting and checking the requirements of an application. The requirements are the descriptions of the functionalities provided by the application and its operational constraints and reflect the needs of customer for a system that helps solve some problem [Som07].

The requirements workflow provides the *Business Concept Model* and the *Use Case Model* artifacts that are described next.

³Inputs and outputs for each activity.

- *Business Concept Models*. The business concept model is a conceptual model⁴. The main purpose of this model is to capture concepts and identify the relationships between them.
- *Use Case Models*. This type of model clarifies the system scope by identifying the actors that interact within the systems and the concrete interactions.

2.2.2 Specification

This workflow corresponds to the *Design* software engineering process. *Design* is the process of describing the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes the algorithms used [Som07].

The artifacts produced by the *Specification* workflow are the *Components Specifications and Architectures*. These artifacts consist of a set of components and interfaces specifications, interrelated by a component architecture. Component specifications are composed of a set of interface specifications and a number of constraints about how the interfaces must be implemented, that is, the contracts.

2.2.3 Provisioning

This workflow ensures that all the components needed by the application are obtained by

- developing them from scratch,
- acquiring them from a provider,
- reusing, integrating or modifying an existing component or other kind of software.

The *Provisioning* workflow includes the unitary testing of every component before going to the *Assembly* workflow.

The input of this workflow is the *Component Specification and Architecture* document. The outputs of this workflow are the implementation of the *Components* that will conform the application.

2.2.4 Assembly

This workflow correspond to the *Integration* software engineering process. *Integration* is the process of assembling all the components creating an application that satisfies the requirements selected for the current release.

This workflow takes all the components and puts them together with the existing assets and user interfaces in order to build an application that satisfies the business requirements. The output coming from the *Specification* workflow is used within the *Assembly* workflow to guide the right component integration.

The artifacts produced by this workflow are the *Applications* generated by assembling components, existing assets and user interfaces.

2.2.5 Testing

Testing here consists in checking that the current release of the application meets its specification and does what the customer wants. In this process the application is verified. The verification responds to the following question: Are we building the product correctly? [Boh79].

During this workflow the applications developed are tested. The artifacts produced by this workflow are the *Tested Applications*, which are the applications verified.

⁴The business concept model is not a model of software but a model of the information that exists in the problem domain. In software engineering the business concept model is often represented by a UML class diagram (<http://www.uml.org/>).

2.2.6 Deployment

During this workflow the application is deployed in the environment where it will be executed.

2.3 Agile Software Development

Since the industry and technology move too fast, requirements change at rates that swamp traditional methods [HOC00], and customers have become increasingly unable to definitively state their needs in advance while, at the same time, they expect more from their software [CLC03]. Therefore the state of the art of Agile Methods is taken into account for enabling the rapid development of large-scale semantic applications covered by the methodology presented in this deliverable.

In the implementation of traditional software engineering methods, work begins with the elicitation and documentation of a complete set of requirements, followed by architectural and high-level design, development and inspection [CLC03]. Agile Methods are a reaction against traditional methods of developing software and an acknowledgement the need to have an alternative to documentation driven, heavyweight software development processes [BBB⁺01].

Beginning in the mid-1990s, some practitioners have found that following exhaustively traditional methods is frustrating and, perhaps, impossible [Hig02]. As a result, several consultants have independently developed methods and practices to respond to the inevitable changes they expected [CLC03]. The goal of Agile Methods is to allow an organization to be agile when producing software, that is, to be able to deliver quickly, change quickly and change often [HOC00]. These Agile Methods are actually a collection of different techniques (or practices) that share the same values and basic principles gathered in a manifesto that reads as follows [BBB⁺01]:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*

That is, while there is value in the items on the right, we value the items on the left more."

While Agile techniques vary in practices and emphasis, they share common characteristics, including [CLC03]:

- Iterative development and focused interaction. This allows the development team to adapt themselves quickly to the changing requirements.
- Working in close location and focusing on communication. It allows teams to make decisions and act on them immediately, rather than wait correspondence.
- Reduction of resource-intensive intermediate artifacts when they not add value to the final deliverable. It allows to devote more resources to the development of the product itself so it can be completed sooner.

Next, a selection of several Agile Methods, taken from [CLC03], is presented.

2.3.1 Extreme Programming

Extreme Programming (XP) [Bec99b] is an agile method that consists of the following 12 rules:

- **The planning game.** At the start of each iteration, customers, managers, and developers meet to flesh out, estimate, and prioritize requirements for the next release. The requirements are called user stories and are captured on story cards in a language understandable to all parties.
- **Small Releases.** An initial version of the system is put into production after the first few iterations. Subsequently, working versions are put into production anywhere at intervals of few days or few weeks.
- **Metaphor.** Customers, managers, and developers construct a metaphor, or a set of metaphors after which to model the system.
- **Simple design.** Developers are urged to keep design as simple as possible and to say everything once and only once [Bec99a].
- **Tests.** Developers work test-first; that is, they write acceptance tests for their code before they write the code itself. Customers write functional tests for each iteration, and at the end of each iteration all tests should run.
- **Refactoring.** As developers work, the design should evolve to keep it as simple as possible.
- **Pair programming.** Two developers sitting at the same machine write all code.
- **Continuous integration.** Developers integrate new code into the system as often as possible. All functional tests must still pass after the integration or after the new code is discarded.
- **Collective ownership.** The code is owned by all developers, who may make changes anywhere in the code at anytime they feel necessary.
- **On-site customer.** A customer works with the development team at all times to answer questions, perform acceptance tests, and ensure that the development is progressing as expected.
- **40-hour weeks.** Requirements should be selected for each iteration so that developers do not need to put in overtime.
- **Open workspace.** Developers work in a common workspace set up with individual workstations on the periphery and common development machines in the center.

Practitioners tend to agree that the strength of XP does not result from each of the 12 practices alone, but from the emergent properties resulting from their combination. There are five key principles of XP all of which enhanced by its practices: communication, simplicity, feedback, courage, and quality work [Hig02].

2.3.2 SCRUM

Scrum [Sch01], along with XP, is one of the most widely used Agile Methods [CLC03]. Scrum is a process that accepts that the development process is unpredictable, formalizing the do what it takes mentality. This method and has achieved great success with numerous independent software vendors [CLC03]. Figure 2.3 depicts the Scrum life cycle. Scrum projects are split into iterations (sprints) consisting of the following:

- **Pre-sprint planning.** All work to be done on the system is kept in what is called the release backlog. During the pre-sprint planning, functionality and features are selected from the release backlog and placed into the sprint backlog, or a prioritized collection of tasks to be completed during the next sprint. Since the tasks in the backlog are generally at a higher level of abstraction, pre-sprint planning also identifies a Sprint Goal that reminds developers why the tasks are being performed and at which level of detail are to be implemented [Hig02].

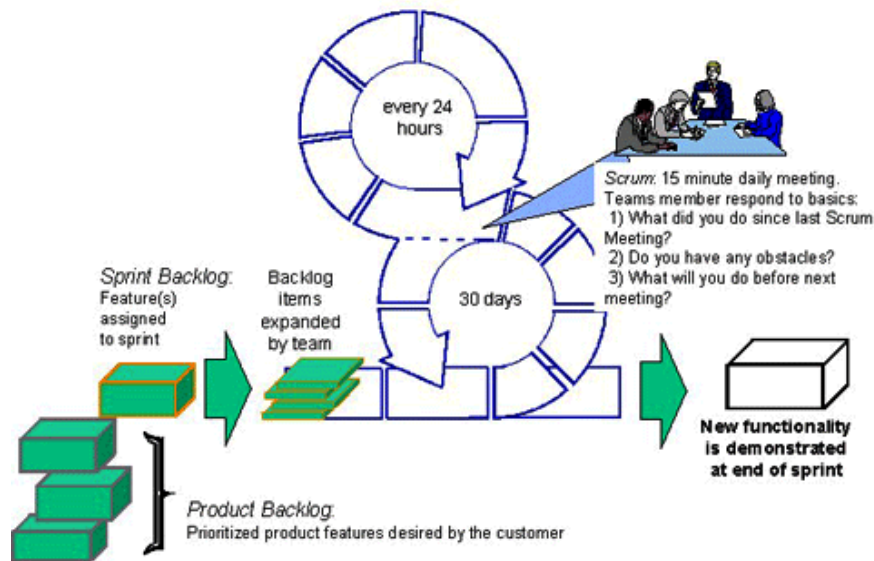


Figure 2.3: Scrum life cycle (from <http://controlchaos.com/>)

- **Sprint.** Upon completion of the pre-sprint planning, teams are handed their sprint backlog and told to sprint to achieve their objectives [Sch01]. At this point, tasks in the sprint backlog are frozen and remain unchangeable for the duration of the sprint. Team members choose the tasks they want to work on and begin development. Short daily meetings are critical to the success of Scrum. Scrum meetings should be held every morning to enhance communication and inform customers, developers, and managers on the status of the project, identify any problems encountered, and keep the entire team focused on a common goal.
- **Post-sprint meeting.** After every sprint, a post-sprint meeting should be held to analyze project progress and demonstrate the current system.

The key principles of Scrum are [Sch01]

- Small working teams that maximise communication, minimise overhead, and maximise sharing of tacit, informal knowledge.
- Adaptability to technical or marketplace (user/customer) changes to ensure the best possible product is produced.
- Frequent builds, or construction of executables, that can be inspected, adjusted, tested, documented, and built on.
- Partitioning of work and team assignments into clean, low coupling partitions or packets.
- Constant testing and documentation of a product as it is built.
- Ability to declare a product done whenever required (because the competition just shipped, because the company needs the cash, or because the user/customer needs the functions).

2.3.3 The Crystal Methods

The Crystal Methods [Coc00] were modelled to address the need of solving one of the major obstacles facing product development: poor communication. Since written documentation can be replaced with face-to-face interactions, the reliance on written work products can also be reduced and thus the likelihood of delivering

the system increases. The more frequently you can deliver running, tested slices of the system, the more you can reduce the reliance on written promissory notes and improve the likelihood of delivering the system [HOC00]. The Crystal methods focus on people, interaction, community, skills, talents, and communication as first order effects on performance. Process remains important, but in a secondary place [Hig02].

The Crystal Methods are a set of versions of the same process, arranged around an identical core [Hig02]. As such, the different methods are assigned colours arranged in ascending opacity. The most Agile version is Crystal Clear, followed by Crystal Yellow, Crystal Orange, Crystal Red, etc. The version of Crystal you use depends on the number of people involved, which translates into a different degree of emphasis on communication. As you add people to the project, you move to more opaque versions of Crystal. As the project criticality increases, the methods harden. The methods can also be altered to fit other priorities, such as productivity or legal liability.

All Crystal Methods begin with a core set of roles, work products, techniques, and notations, and this initial set is expanded as the team grows or the method hardens. As a necessary effect, more restraints leads to a less agile method.

2.3.4 Feature Driven Development

Feature Driven Development (FDD) [PF02] is a simple, well-defined process based on short, iterative feature-driven life cycle with the following stages:

- **Stage 1. Develop an overall model.** The FDD process begins with the development of a model. Team members and experts work together to create a walkthrough version of the system.
- **Stage 2. Build a features list.** Next, the team identifies a collection of features representing the system. Features are small items useful in the eyes of the client. They are similar to XP story cards written in a language understandable to all parties. Features should take up to 10 days to develop [Hig02]. Those features requiring more than 10 days are broken down into sub features.
- **Stage 3. Plan by feature.** The collected feature list is then prioritized into subsections called design packages. The design packages are assigned to a chief programmer, who in turn assigns class ownership and responsibility to the other developers.
- **Stage 4. Design by feature and build by feature.** After design packages are assigned, the iterative portion of the process begins. The chief programmer chooses a subset of features that will take 1 to 2 weeks to implement. These features are then planned in more detail, built, tested, and integrated.

2.3.5 Comparison of Presented Methods

Table 2.1 presents a set of prescriptive characteristics of Agile Methods that state under which conditions the different models works and do not work. These characteristics are team size, iteration length, support for distributed teams and system criticality. More details can be found in [CLC03].

	XP	Scrum	Crystal	FDD
Team Size	2-10	1-7	Variable	Variable
Iteration Length	2 weeks	4 weeks	< 4 months	< 2 weeks
Distributed Support	No	Adaptable	Yes	Adaptable
System Criticality	Adaptable	Adaptable	All types	Adaptable

Table 2.1: Prescriptive Characteristics of Agile Methods [CLC03]

2.4 Requirements Engineering Process

In this section we present a brief introduction to the existing methods and techniques for carrying out the *Requirements Engineering* process.

2.4.1 Methods

Sommerville [Som07] describes the following activities for obtaining the so-called requirements document and shown in Figure 2.4:

1. **Feasibility study.** The results of the feasibility study should be a report that recommends whether or not it is worth carrying on with the requirements engineering and system development processes.
2. **Requirements elicitation and analysis.** The goal of the requirements elicitation and analysis activity is to state what the system developers should implement and when it should be done. This is reflected in the software requirements document and in the release planning respectively. Agile development methods argue that requirements change so rapidly that a requirements document is out of date as soon as it is written, so the effort spent in writing a very formal and sound requirements document is largely wasted [Som07]. In this activity, software engineers work with customers and system end-users to find out about the application domain, what features the system should provide, the required performance of the system, and so on. This activity generates the *System models*. These models are graphical representations that describe business processes, the problem to be solved and the system that is to be developed. System models are an important bridge between the requirements engineering and the design processes [Som07].
3. **Requirements specification.** In this activity the *User and system requirements* are specified.
4. **Requirements validation.** This activity is concerned with showing that the requirements actually define the system that the customer wants. In this activity the final *Requirements document* is output.

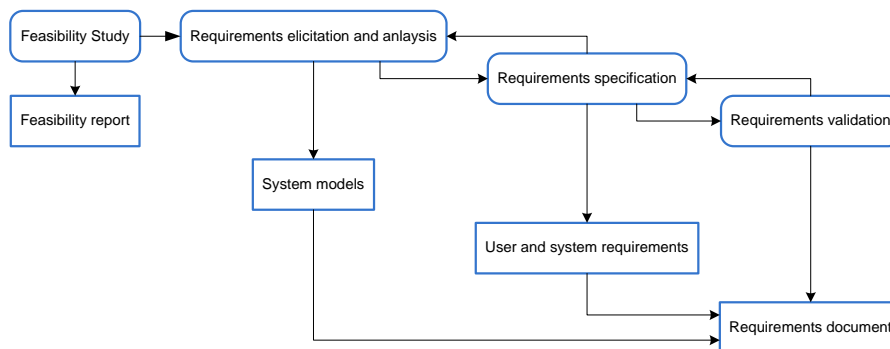


Figure 2.4: The requirements engineering process [Som07]

An alternative perspective on the requirements engineering process is presented in Figure 2.5. This presents the process as a three-stage activity where the activities are organised as an iterative process around a spiral. The spiral process accommodates approaches to development in which the requirements are developed to different levels of detail. The number of iterations around the spiral can vary, so the spiral can be exited after some or all of the user requirements have been elicited [Som07]. Agile methods, as an approach to iterative development, follow this alternative perspective of the requirements engineering process.

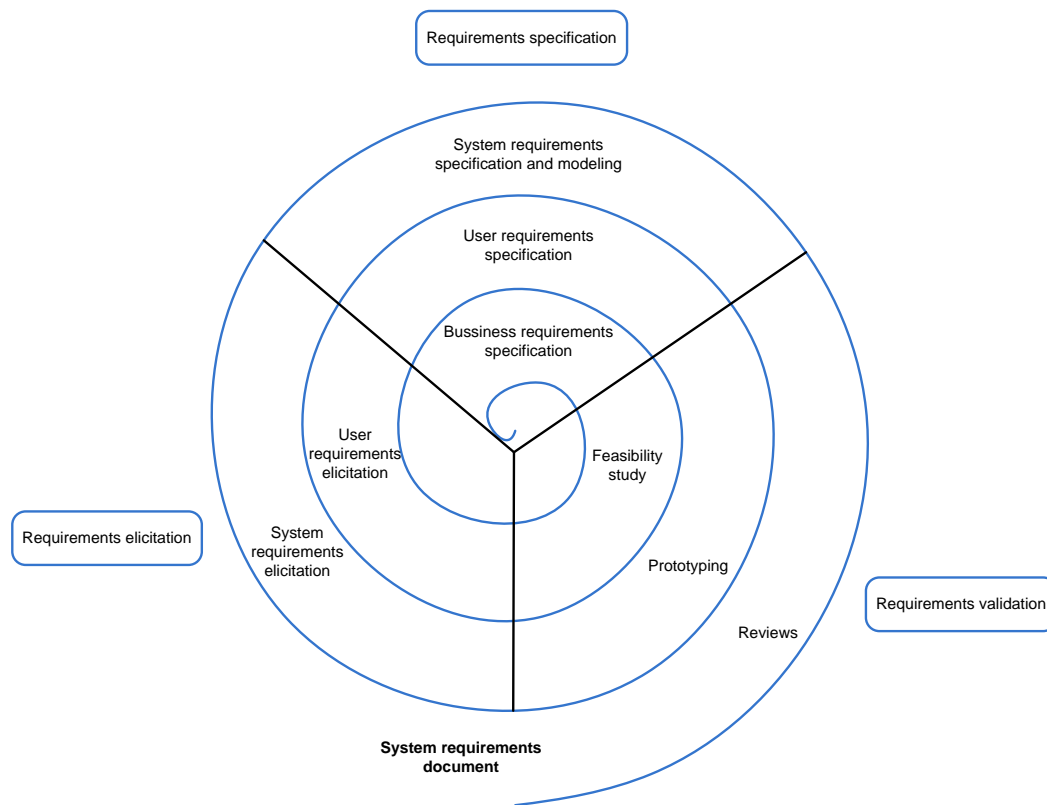


Figure 2.5: Spiral model of requirements engineering process [Som07]

Requirements Elicitation and Analysis

A very general workflow of the *Requirements elicitation and analysis* activity is shown in Figure 2.6. The tasks of this activity are organised within a spiral so that the tasks are interleaved as the process proceeds from the inner to the outer rings of the spiral [Som07].

The tasks within the requirements elicitation and analysis activity are [Som07]

1. *Requirements discovery*. This is the process of interacting with stakeholders in the system to collect their requirements.
2. *Requirements classification and organisation*. This activity takes the unstructured collection of requirements, groups related requirements and organises them into coherent clusters.
3. *Requirements prioritisation and negotiation*. This activity is concerned with prioritizing requirements, and finding and resolving requirements conflicts through negotiation.
4. *Requirements documentation*. The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced.

Agile Methods implement the requirements elicitation and analysis activity in the following ways:

- In Extreme Programming [Bec99b] all requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks in small releases. Thus during the requirements engineering process, the requirements are specified as user stories that are selected for the next release and broken down to tasks. Once the tasks are identified, the development team estimates the effort and resources required for implementing each task, and then the customer prioritizes the stories for implementation, choosing those stories that can be used immediately to deliver useful business support. Finally, when the current release is finished, the workflow starts again.

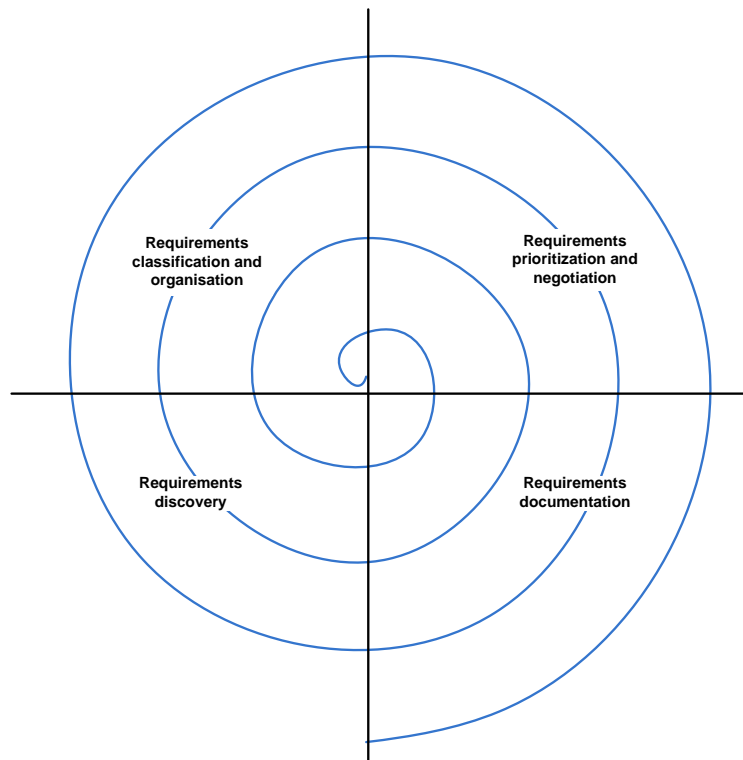


Figure 2.6: The requirements elicitation and analysis activity [Som07]

- According to Scrum [Sch01], all the application requirements are kept in the release backlog. During the pre-sprint planning, requirements are selected from the release backlog and placed into the sprint backlog in order to be completed during the next sprint. Upon completion of the pre-sprint planning, teams are handed their sprint backlog and they implement the selected requirements. Upon completion of the implementation (after the post-sprint meeting), the workflow starts again.
- According to Feature Driven Development [PF02] the team identifies requirements as a collection of features useful in the eyes of the client representing the system. These requirements are similar to XP story cards. The collected feature list is then prioritized into design packages and finally the iterative portion of the process begins assigning, implementing, testing and integrating such features.

2.4.2 Techniques

There are different techniques that can be applied for discovering requirements. Examples of these techniques are viewpoints, interviewing, scenarios and use cases.

- *Viewpoints* [FKN⁺92] can be used as a way of classifying stakeholders and other sources of requirements. The three generic types of viewpoint are interactor, indirect and domain viewpoints. Interactor viewpoints provide detailed system requirements covering the system features and interfaces. Indirect viewpoints are more likely to provide higher-level organisational requirements and constraints. Domain viewpoints normally provide domain constraints that apply to the system. A key strength of viewpoint-oriented analysis is that it recognises multiple perspectives and provides a framework for discovering conflicts in the requirements proposed by different stakeholders [Som07].
- Formal or informal *interviews* with system stakeholders are part of most requirements engineering process methods. Requirements are derived from the answer to the questions formulated in the interview. Interviews may be of two types: closed interviews where the stakeholder answers a predefined set of questions and open interviews where there is no predefined agenda [Som07].

- *Scenarios* are descriptions of example interaction sessions. Each scenario covers one or more possible interactions. Several forms of scenarios have been developed, each of which provides different types of information at different levels of detail about the system. Using scenarios to describe requirements is an integral part of agile methods, such as extreme programming. Scenario-based elicitation can be carried out informally, where the requirements engineer works with stakeholders to identify scenarios and to capture details of these scenarios. Scenarios may be written as text, supplemented by diagrams, screen shots and so on [Som07].
- *Use-cases* [Jac92] are a scenario-based technique for requirements elicitation. They have become a fundamental feature of the UML notation for describing object-oriented scenarios. Use-cases identify the individual interactions between the system and the actors that interact with it. The set of use-cases represents all of the possible interactions to be represented in the application requirements. However, because they focus on interactions, they are not as effective for eliciting constraints or high level and non-functional requirements from indirect viewpoints or for discovering domain requirements [Som07].

2.5 Design Process

In this section we present a brief introduction to the existing methods and techniques for carrying out the *Design* process.

2.5.1 Methods

The design process may involve developing several models of the system at different levels of abstraction. As a design is decomposed, errors and omissions in early stages are discovered. These feed back to allow earlier design models to be improved [Som07].

The specific design activities are [Som07]:

1. **Architectural Design.** The sub-systems making up the system and their relationships are identified and documented.
2. **Abstract specification.** For each sub-system, an abstract specification of its services and the constraints under which it must operate is produced.
3. **Interface design.** For each sub-system, its interface with other sub-systems is designed and documented. This interface specification must be unambiguous as it allows the sub-system to be used without knowledge of the sub-system operation.
4. **Component design.** Services are allocated to components and the interfaces of these components are designed.
5. **Data structure design.** The data structures used in the system implementation are designed in detail and specified.
6. **Algorithm design.** The algorithms used to provide services are designed in detail and specified.

This is a general model of the design process and real, practical processes may adapt it in different ways [Som07]. In [CD01] there is a description of a component-based *Design* process through the description of the following activities:

1. **Component Identification.** This is the first activity of the *Design* process. It makes use of the artifacts produced by the *Requirements* process. The goal of this activity is to create an initial component and architecture specification by elaborating a draft that will be refined during the next specification activities.

The emphasis at this activity is on discovery what information needs to be managed, what interfaces are needed to manage it, what components are needed to provide that functionality, and how they will fit together. During this activity several decisions are taken, as for example, the system structuration into different layers (e.g., *system services layer* and *business services layer*) and the identified interfaces within each layer. Also, during this activity we must take into account the existing assets to be integrated into the system through an interface or by means of adapting them.

The tasks proposed in [CD01] for carrying out the activity (shown in Figure 2.7) are the following [CD01]:

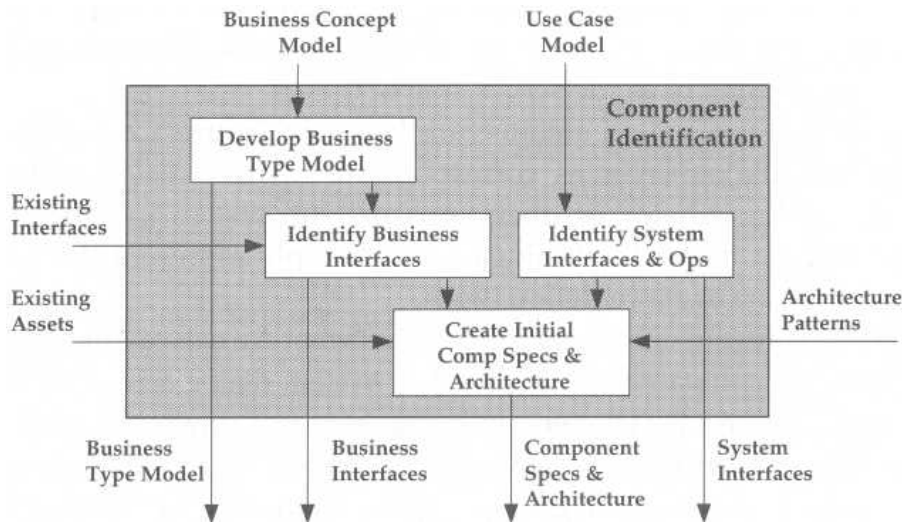


Figure 2.7: The *Component identification* activity [CD01]

Task 1. Develop Business Type Model. The business types describe the data structure that the application needs to manage.

Task 2. Identify Business Interfaces. The business interfaces are abstractions of the information that must be managed by the system.

Task 3. Identify System Interfaces and Operations. In this task two components are defined for each identified use case: one *Dialog* component and one *System Facade*. The *Dialog* component will be in charge of managing the dialog with the application user and the *System Facade* will be in charge of providing operations for every step within a use case.

Task 4. Create Initial Component Specifications and Architecture. Within this task the architecture of the application is drawn by taking into account the components defined in the previous tasks as well as architectural patterns.

2. **Component Interaction.** Within this activity it is decided how will the components work together to provide the desired functionality. This is done by defining the interactions that will take place between the system components as well as identifying how will be used the component interfaces by other components.

During this activity several decisions such patterns adaptation and operations identification are taken, and the dependencies between the component interfaces are fully understood. Thus, at this activity the defined interfaces identified in the previous activity (*Component Identification*) are refined because new interfaces and operations are discovered.

3. **Component Specification.** This is the final activity of the *Design* process. During this activity contracts are specified. A contract is a formal agreement between two or more parties. It describes (or

specifies) the detail of the agreement in an unambiguous form. This involves stating the responsibilities or obligations of each party (what each component will provide the other components). It does not state how the components will do it, it simply states that they will. We distinguish two different types of contracts (see Figure 2.8):

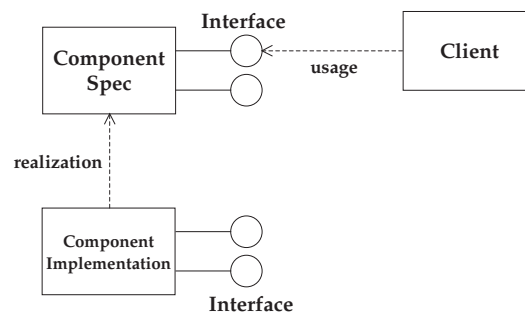


Figure 2.8: Different types of contracts [CD01]

- **Usage Contract.** A *Usage Contract* describes the relationship between a component interface and a client, and is specified in the form of an interface.
- **Realisation Contract.** The contract between a component specification and its implementation(s), and must be adhered to by the person who is creating the implementation.

2.5.2 Techniques

There are different techniques that can be applied for designing an application. Different structures that can be used when designing an application are explained next [Som07].

Architectural design

A system architecture model is a compact, manageable description of how a system is organised and how the components interoperate. It identifies the sub-systems that provide some related set of services and compose the whole system. Three advantages of explicitly designing and documenting a software architectures are [BCKB03]

- *Stakeholder communication.* The architecture is a high-level presentation of the system that may be used as a focus for discussion by a range of different stake-holders.
- *System analysis.* Making the system architecture explicit at an early stage in the system development requires some analysis.
- *Large-scale reuse.* The system architecture is often the same for systems with similar requirements and so it can support large-scale software reuse.

Distributed systems architectures

A distributed system is a system where the information processing is distributed over several computers rather than confined to a single machine. Obviously, the engineering of distributed systems has a great deal in common with the engineering of any other software, but there are specific issues that have to be taken into account when designing this type of systems, such as complexity, security, manageability and unpredictability [Som07].

The architecture of distributed systems is characterized by the inclusion of *middleware*, that is, software that can manage diverse parts and ensure that they can communicate and exchange data [Som07].

Two generic types of distributed system architecture are [Som07]

- *Client-server architectures.* In this approach, the system may be thought of a set of services that are provided to clients that make use of these services. Servers and clients are treated differently in these systems.
- *Distributed object architectures.* In this case, there is no distinction between servers and clients, and the system may be thought of as a set of interacting objects whose location is irrelevant. There is no distinction between a service provider and a user of these services.

Generic application architectures

Generic application architectures are generic structural models of a given kind of application. Usually, systems of the same type have similar architectures, and the differences between these systems are in the detailed functionality that is provided. This can be illustrated by the growth of Enterprise Resource Planning (ERP) systems and vertical software packages for particular applications [Som07]. Generic application architectures can be used in a number of ways [Som07]:

1. *As a starting point for the architectural design process.* If someone is unfamiliar with a given type of application, he can base his initial designs on the generic architectures.
2. *As a design checklist.* A given architectural design can be checked against the generic application architecture to see whether any important design component is missed .
3. *As a way of organising the work of the development team.* The generic application architecture identifies stable structural features of the system architectures and, in many cases, it is possible to develop these in parallel.
4. *As a means of assessing components for reuse.* If there exists components then they might be able to be reused and compared to the generic structures to see whether reuse is likely in the application that is being developed.
5. *As a vocabulary for talking about types of applications.* Generic application architectures facilitates the comparison between applications of the same time and the discussion about them. The concepts identified in the generic architectures can be used to talk about the applications.

Object oriented design

Object oriented design is concerned with developing an object-oriented model of a software system to implement the identified requirements. The objects in an object-oriented design are related to the solution to the problem. [Som07].

Object oriented design is part of object-oriented development where an object-oriented strategy is used throughout the development process.

Real-time software design

Real-time software design concerns the design of real-time systems. A real-time system is a software where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced [Som07].

User interface design

Careful user interface design is an essential part of the overall software design process. If a software system is to achieve its full potential, it is essential that its user interface should be designed to match the skills, experience and expectations of its anticipated users. Good user interface design is critical for system dependability.

2.6 Conclusions

Component Based Software Engineering (CBSE) is used as an attempt to reduce development costs by incorporating previously proven designs and code in new applications. A net effect of component reuse would be shorter development schedules and more reliable applications since the developer uses components that have been previously “shaken down”. In addition, agile methods enable the rapid development of applications.

During the whole application life-cycle requirements change continuously because customers are unable to definitively state their needs in advance while and because the industry and technology move too fast [HOC00]. In order to deal with this situation the requirements engineering process has to be carried out according to an iterative life-cycle approach. Agile methods provide the aforementioned iterative life-cycle approach.

Scenario-based elicitation and, specifically, use-cases are an appropriate approach to the requirements engineering process when implementing an agile method. In addition, use-cases are increasingly used for requirements elicitation and are a fundamental feature of the UML standard [Som07]. However use-cases are not as effective for eliciting constraints or high-level and non-functional requirements as, for example, those related to the characteristics presented in Subsection 3.1.

User requirements should be written in natural language because they have to be understood by people who are not technical experts. These statements in natural language can be complemented with diagrams when graphical representations help to understand better the application needs. As a graphical representation, system models are often more understandable than detailed natural language descriptions of the system requirements [Som07].

The essence of software design is to make decisions about the logical organisation of the software. There is no right or wrong way to design software and nor is there a universal recipe for software designs [Som07].

During the *Component Identification* activity an initial architecture design is created. However, a design rarely starts from scratch when making decisions about the software organization since it is based on previous design experiences [Som07].

The same benefits provided by CBSE and agile methods to software development are supposed to be obtained when applying these approaches to the development of semantic applications.

However, for applying CBSE to large-scale semantic applications, we need to define the following elements

1. *Independent components* commonly used in semantic applications that should be completely specified by their interfaces.
2. *Component standards* related to semantic technologies that facilitate the integration of those components.
3. *Middleware* that provides software support for integrating components related to semantics.
4. *A development process* that is geared to CBSE and adapted for the development of large-scale semantic applications.

Finally, agile methods should be adapted within the processes and activities that the methodology covers, in order to enable the rapid development of large-scale semantic applications.

Chapter 3

Related Work in the Characterisation and Classification of Semantic Applications

This chapter presents the state of the art of the characterization and classification of semantic applications as well as different scenarios where semantic applications can be applied. Finally, the Semantic Web Framework, a component-based framework that describes the software components involved in the architecture of semantic applications, is summarized.

Although semantic Web applications is a subset of the semantic applications, the features of the former are a superset of the characteristics of the latter. It is important to point out that most of the characteristics and classifications analyzed were obtained from different studies of semantic Web applications and not from the study of semantic applications in general. However we assume that the characteristics and classifications of semantic Web applications can be used to characterize and classify large-scale semantic applications in general due to the following assumptions:

- Semantic Web applications are large-scale semantic applications that run on the Internet, whereas other large-scale semantic applications not using ontologies and data from the Web usually run in intranets. Thus the amount of information contained in the semantic Web will always be bigger than the information contained in any intranet.
- Due to the previous reason, a large-scale semantic application non oriented to the Semantic Web will require less degree of scalability than a Semantic Web application.
- The open world, which characterizes the semantic Web, is a superset of the closed world, typical of other kinds of semantic applications that do not run in the Web.
- Most of semantic technologies have been developed in the context of the Semantic Web research field, but they can be applied to any kind of large-scale semantic application.
- Because of the reasons abovementioned, it could be considered that semantic Web applications technologically subsume other kinds of large-scale semantic applications.

3.1 Characteristics of Semantic Applications

This section analyzes the state of the art of the characterisation of semantic applications. In the literature there are a number of characteristics of semantic applications that permit analyzing the evolution of the semantic applications from traditional knowledge-based systems to next generation semantic Web applications [AMS⁺08].

3.1.1 Traditional Knowledge-Based Systems Characteristics

A “knowledge-based system” (KBS) is a computer system that relies, on the one hand, on knowledge formalized in a knowledge representation language and, on the other hand, on reasoning mechanisms for problem solving. Traditional knowledge-based applications present the following characteristics [AMS⁺08]:

- **Closeness.** Traditional knowledge-based applications produce and consume their own data. They operate in a closed domain.
- **Centralisation.** Classic knowledge bases were built in a a centralised fashion, normally by a small team of knowledge engineers. Their data is centrally managed, so trust is not a key issue.
- **Lack of Scalability.** The size of the knowledge these systems contain is small in comparison to the size of the Semantic Web.
- **Heavyweight Reasoning.** They tend to use reusable models (ontologies) as knowledge bases [Gru93] and reasoning components (problem solving methods) [Mot99, SAA⁺00]. The sophisticated reasoning mechanisms are enabled by combining high quality knowledge bases with powerful models of *generic tasks*, such planing, diagnosis, scheduling, etc.
- **Homogeneity.** The ontologies these systems use are selected, designed and integrated manually and carefully [AMS⁺08]. They present homogeneity in dimensions such the encoding of the ontologies, their expressivity, their quality, etc., usually exhibiting a high degree of quality.
- **Use of a small quantity of ontologies.** They use a small quantity of ontologies, very often just one.

3.1.2 Semantic Web Applications Characteristics

In [MS06] there is a distinction between the first and the next generation of Semantic Web Applications. First generation of Semantic Web applications present the following characteristics:

- **Closeness.** Most Semantic Web applications built to date tend to produce and consume their own data much like traditional knowledge-based applications [MS06].
- **Decentralisation.** They make use of distributed knowledge, using data from multiple sources [AMS⁺08].
- **Staticity.** Knowledge is gathered and engineered at design-time. The selection of the data resources used is made also at design time and the integrated data is typically scraped together using ad hoc mechanisms [MS06].
- **Scalability.** Semantic Web applications exploit semantic information on a large scale. The Semantic Web already contains millions of documents and billions of triples. Even though the first generation of Semantic Web Applications normally focuses on specific subsets of the Semantic Web, the applications must locate and process relevant information more efficiently than usual KBS [AMS⁺08].
- **Lightweight Reasoning.** Logical reasoning becomes less important than in KBS since scale and data integration become here key issues. They integrate different reasoning techniques such as machine learning and linguistic or statistical techniques [AMS⁺08].
- **Heterogeneity.** Semantic Web applications are characterized by their heterogeneity in several dimensions such as the encoding of the ontologies, quality, complexity, modeling, views, etc. When they use data from multiple sources the integration effort is non-trivial. They may make use of information stored-held in the Semantic Web and originated from a large variety of sources that exhibit differences in quality, so trust becomes a key issue. Semantic Web applications usually integrate non-semantic data [AMS⁺08].

- **Use of a small quantity of ontologies.** Semantic Web applications are usually based on a single ontology whose role is to support the retrieval, integration and processing of the available data [MS06].

The characteristics of the first generation of Semantic Web applications have enabled the development of Corporate Semantic Webs¹ that in many cases provide perfectly adequate solutions to a company's need [AMS⁺08].

In the next generation, Semantic Web applications will dynamically exploit the potential of the Semantic Web as a large scale source of distributed knowledge [AMS⁺08]. According to [AMS⁺08, DF08, KSF08] for the next generation of Semantic Web applications, the following characteristics can be extracted:

- **Openness.** They exploit the Semantic Web as a large-scale source of information producing and consuming its own and foreign data. The application is not bound to a particular domain [AMS⁺08]. In principle, anybody can contribute as a provider or consumer of information [DF08].
- **Interoperability.** Interoperability should be provided through the integration of heterogeneous proprietary and legacy solutions through a common interface [DF08].
- **Decentralisation.** The provision and management of information and services is under the control of distributed entities, rather than of a central authority [KSF08]. The next generation of Semantic Web applications will retrieve knowledge automatically by exploring the web [AMS⁺08].
- **Dynamicity.** Services and content are constantly moving, changing, appearing or disappearing in highly ad hoc manners [KSF08]. After knowledge retrieval, the appropriate knowledge has to be selected according to application-dependent criteria, for example, the quality of the data and its adequacy to the task at hand. Therefore, the relevance of a particular resource to a problem-solving need cannot be judged at design time. [AMS⁺08].
- **Scalability.** Like it happened to the first generation of Semantic Web applications, second generation applications will have to deal with the scale of the Semantic Web [AMS⁺08].
- **Hybrid Reasoning.** Selected Heterogeneous knowledge sources are exploited in a way that no assumption can be made on the ontological nature of the elements manipulated. Hence the process that exploit knowledge needs to be generic enough so that it can make use of any semantic resource online [AMS⁺08].
- **Heterogeneity.** As happened to the first generation of Semantic Web applications, second generation applications will have to deal with the heterogeneity of the Semantic Web [AMS⁺08].
- **Use of a big quantity of ontologies.** Second generation applications use a big quantity of ontologies combined from different sources, so that they can be exploited jointly [AMS⁺08].

3.1.3 Comparison of the Characteristics Presented

A comparison can be made of the characteristics of the aforementioned different kinds of applications aforementioned: knowledge based systems, the first generation of Semantic Web applications and the second generation of Semantic Web Applications. Table 3.1 shows the result of the comparison.

This comparison has been made according to the following application properties:

- **Domain Openness.** This property specifies whether the information used by the application (ontologies and instances) is bound to a particular domain or, on the contrary, the application permits the execution in multiple domains.

¹<http://www.gartner.com/it/page.jsp?id=495475>

- **Interoperability.** This property specifies if the application integrates heterogeneous proprietary and legacy solutions.
- **Knowledge Distribution** This property determines whether the knowledge (ontologies and instances) is centralized in a single resource or distributed using data from multiple sources.
- **Dynamicsity.** This property specifies if services and content are constantly moving, changing, appearing or disappearing in highly ad hoc manners, or, on the contrary, knowledge is gathered and engineered at design-time.
- **Scale.** This property specifies if the application can or cannot operate at scale in a context such as the Semantic Web, which is estimated to contain a few millions of documents describing millions of entities through billions of statements.
- **Reasoning.** This property determines if the application will use heavyweight, lightweight or hybrid reasoning mechanisms.
- **Heterogeneity.** This property specifies whether the application is characterised by the heterogeneity along several dimensions (encoding of the ontologies, quality, complexity, modeling, views, etc.) or, on the contrary, it is characterised by the use of ontologies that are selected, designed and integrated manually and carefully (homogeneity).
- **Number of Ontologies.** This property refers to whether the application is based on a single ontology or it makes use of several ontologies organised in a network.

	KBS	1st generation SWA	2nd generation SWA
Domain Openness	No	No	Yes
Interoperability	Not common	Not common	Yes
Knowledge Distribution	Centralisation	Decentralisation	Decentralisation
Dinamicity	No	No	Yes
Scale	Lack of Scalability	Lack of Scalability	Scalability
Reasoning	Heavyweight	Lightweight	Hybrid
Heterogeneity	No	Yes	Yes
Number of Ontologies	Single ontology	Small networks of ontologies	Big networks of ontologies

Table 3.1: Properties of different kinds of semantic applications

3.2 Scenarios for building Semantic Applications

This subsection explains the related work that provides understanding and a classification of Semantic Applications through the description of different scenarios where semantics can be applied.

3.2.1 Classification of Ontology Applications

In [JU99] a set of scenarios for applying ontologies to achieve one or more purposes is presented. These scenarios are abstractions of specific applications of ontologies and have been taken from industry or-and research. Each scenario involves a set of actors. The following list describes the actors involved in the aforementioned scenarios:

- **Ontology Author (OA).** The role of the author of an ontology. This role is usually played by a person (or a team).

- **(Operational) Data Author (DA).** The role of the author of operational data in the language used or defined in terms of the vocabulary of the ontology.
- **Application Developer (AD).** The role of the developer of the application.
- **Application User (AU).** The role of the user of the application.
- **Knowledge Worker (KW).** The role of the the person who makes use of the knowledge.

In addition, information can play three different roles in each scenario:

- **Operational Data.** A role that information plays whereby the information is consumed and produced by applications during runtime.
- **Ontology.** A role that information plays, whereby the information specifies terms and definitions for important concepts in some domain.
- **Ontology Representation Language.** A role that information plays whereby the information is used by ontology authors or application developers, during the development process, to write ontologies.

Next the scenarios [JU99] and the list of possible variations that each scenario can present are described.

1. Neutral Authoring

In this scenario, an information artifact is authored in a single language and converted into a different form for use in multiple target systems. The benefits of this approach include knowledge reuse, improved maintainability and long term knowledge retention. An important distinction to be made is whether the authored artifact requiring translation is an ontology or operational data. This distinction is specified through the specification of two variations of the *Neutral authoring* scenario:

- **Authoring Ontologies.** The motivation behind authoring neutral ontologies is to decrease cost of reuse and maintenance of knowledge. To accomplish this, the actors should develop an ontology, which is then translated and used in multiple operational target systems. The principle actors are the ontology author and the application user.
- **Authoring Operational Data.** The motivation behind authoring neutral operational data is to improve the maintainability and transportability of the operational data. To accomplish this, an ontology author (secondary actor) must develop an ontology that defines the neutral format used by the primary actor when authoring the operational data. Tools can then translate this into operational data used by an application.

2. Ontology as Specification

In this scenario, an ontology of a given domain is created and used as a basis for the specification and development of some software. The benefits of this approach include documentation, maintenance, reliability and knowledge (re)use.

3. Common access to Information

In this scenario, information can be required by one or more persons or by computer applications, but it is expressed using unfamiliar vocabulary or in an inaccessible format. The ontology helps render the information intelligible by providing a shared understanding of the terms, or by mapping between sets of terms. The benefits of this approach include inter-operability, and a more effective use and reuse of knowledge resources. This scenario has four possible variations:

- **Human Communication.** In this scenario, the ontology authors create an ontology which knowledge workers reference in their work.

- Data Access via Shared Ontology.** The motivation behind data access via a shared ontology is to reduce the cost of multiple applications that have common access to data. This kind of access may, in turn, facilitate inter-operability. Inter-operability is accomplished through the agreement of developers on a shared ontology that defines a common language for exchanging or sharing operational data. The principle actors are ontology authors and application developers. Figure 3.1 illustrates this scenario.
- Data Access via Mapped Ontologies.** The motivation behind this scenario is the same as the previous one. The key difference is that here, there is no explicit shared ontology; instead, mapping rules are used to define what a term in one ontology means in another ontology. A mediator uses these rules at runtime so that applications can access each other's data. The principle actors here are ontology authors and application developers. Figure 3.2 illustrates this scenario.
- Shared Services.** The motivation behind shared services is neutrality (i.e., language, machine, operating system, location). Developers achieve this by agreeing on a shared ontology, which defines interfaces in multiple target languages. This is very similar to data access via shared ontologies, except for the focus of what is being shared. The principal actors here are ontology authors and application developers.

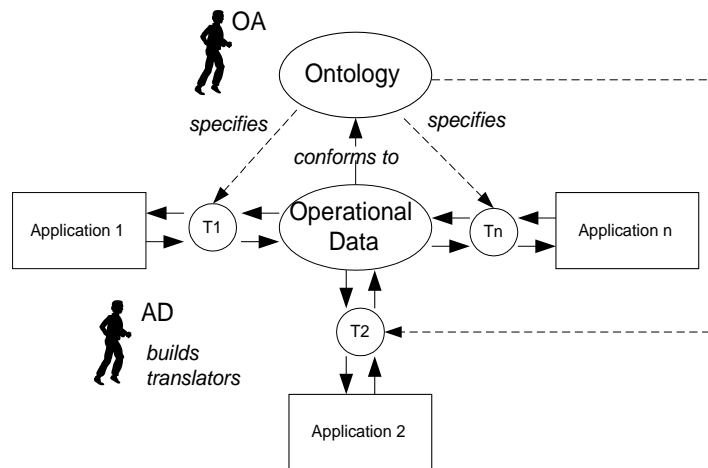


Figure 3.1: Common access to information scenario. Data access via shared ontology variation [JU99]

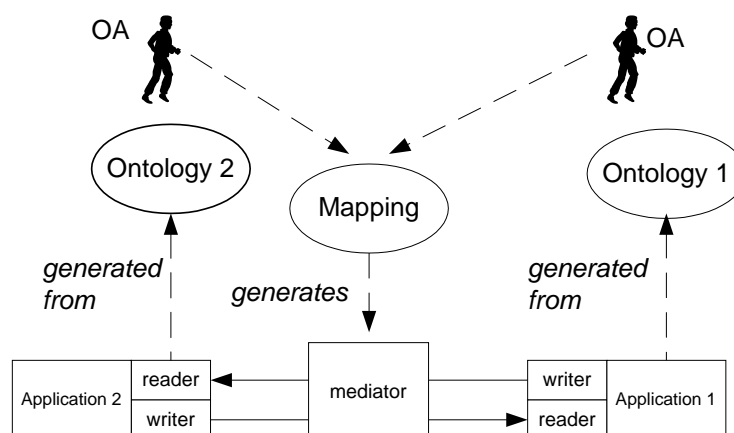


Figure 3.2: Common access to information scenario. Data access via mapped ontologies variation [JU99]

4. **Ontology-based search.**

In this scenario, an ontology is used to search an information repository for wanted resources (e.g., documents, web pages, names of experts). The chief benefit of this approach is faster access to important information resources, which leads to a more effective use and reuse of knowledge resources.

3.2.2 **Classification of Semantic Web Applications**

From the classification of types of ontology usage for Semantic Web applications in [KHS⁺08], several scenarios for building semantic applications can be derived. The categories there found and presented in this work are the following:

1. **Usage as a Common Vocabulary.** This type of application uses ontologies as a common vocabulary to enhance inter-operability of knowledge content. Applications in which ontologies play mainly a role to unify the vocabulary are categorised here. This is the most fundamental and common usage for the 2 to 9 categories.
2. **Usage for Search.** Any application whose main goal is to search will be categorised here. This kind of search systems use semantic information for searching.
3. **Usage as an Index.** An application that uses ontologies as indexes for knowledge resources belongs to this category. The difference between this category and categories 1 and 2 is that the applications of this category utilize not only the index vocabulary, but also its structural information explicitly when accessing the knowledge resources.
4. **Usage as a Data Schema.** Applications of this category use ontologies as a data schema to specify data structures and values for target databases.
5. **Usage as a Media for Knowledge Sharing.** Applications of this category aim to share knowledge among systems, between people and systems, or among people using ontologies and instance models of the target knowledge. This category includes such applications for knowledge alignment, systems for knowledge mapping, communication systems among agents, support systems for communication, and others.
6. **Usage for a Semantic Analysis.** Applications of this category analyze contents annotated by meta-data. One of the most typical methods for such an analysis is the automatic classification of concepts definitions using inference engines.
7. **Usage for Information Extraction.** Applications which aim to extract meaningful information from the search result are categorized here. In comparison with other categories, applications in category 2 just output search results without modifications. Applications of category 6 add some analysis to the output of 2, while those of category 7 extract meaningful information before outputting it for users.
8. **Usage as a Rule Set for Knowledge Models.** Applications in this category use instance models built upon definitions of classes in ontologies as knowledge models of the target world. In other words, they use ontologies as meta-models that rule the knowledge (instance) models. Compared to category 4, knowledge models in category 8 need more flexible descriptions in terms of meaning of the contents. A heavy-weight ontology that models the world appropriately with deep semantics helps both the knowledge modeling and the reasoning at a deeper level. On the other hand, when the target world is large, light-weight ontologies are useful enough for reasoning and efficient processing.
9. **Usage for Systematising Knowledge.** Applications in this category organise concepts of the target world by putting ontologies as the core conceptual structure. Typical applications of this category include integrated knowledge systems of categories 1 to 8, such as knowledge management systems and content management systems.

3.3 The Semantic Web Framework

As it has been expounded in Subsection 2.2, the essentials of the component-based software engineering are *independent components* that are completely specified by their interfaces, *component standards* that facilitate the integration of components, *middleware* that provides software support for component integration, and a *development process* that is geared to CBSE.

The Semantic Web Framework (SWF) [GGMN08] provides the skeleton for a specification of the independent components needed for the CBSE of semantic Web applications. The SWF is a component-based framework from which semantic Web applications can be organized and developed describing the functionalities that the components that semantic Web applications provides and those that they use. The SWF also classifies its components and identifies the main dependences between them. The current version of the SWF [GMG⁺07] does not define the concrete specification of the component interfaces and contracts, which should be defined in future work, nor does it define the different patterns that can be used when developing Semantic Web applications, such patterns are dealt in with the methodology presented in this deliverable.

3.3.1 Definition and Classification of Components

The SWF follows the definition of component given by Szyperski [Szy98]. A Semantic Web Framework component is an autonomous and modular unit with well defined interfaces² that describes a service that performs a specific functionality. Such components can be used either independently or grouped together to develop applications for the Semantic Web. This kind of components can be divided into three types: services, program libraries and applications.

The components of the SWF have been classified into an architecture according to several dimensions as the major properties of the problem space that have significant variation over the systems of concern. The dimensions are in other words, the groups of components that provide some specific support to the architecture. These dimensions are subjective; in the Semantic Web Framework the different components have been classified according to the main functionalities they provide. Furthermore, these dimensions are not exhaustive. In Figure 3.3, each dimension of the architecture is represented as a column. The order of the components or of the dimensions does not imply any precedence or relation between them. The basic dependencies of the components in an example dimension are shown in Figure 3.4.

Figure 3.3 presents the components of the Semantic Web Framework that have been identified from software currently available or under construction. The enumeration of components is neither exhaustive nor complete, and is open to improvements and extensions. The complete description of the components can be found in [GMG⁺07]. A list of the implementations of the components have been identified in the same deliverable.

Next, we provide a description of the dimensions of the Semantic Web Framework and of the components included in each dimension.

Data and metadata management

This dimension includes those components that manage knowledge and data sources.

- *Information directory manager component.* This component provides functionalities to handle query distribution, to manage a content provider directory, to identify information providers from a query, and to handle the storage and access to distributed ontologies and data.
- *Ontology repository component.* This component provides functionalities to locally store and access ontologies and ontology instances.
- *Data repository component.* This component provides functionalities to locally store and access data and ontology annotated data.

²Note that different component implementations can have different interfaces (API, service description, etc.) because no successful standardization efforts have been made for providing common interfaces to the same component.

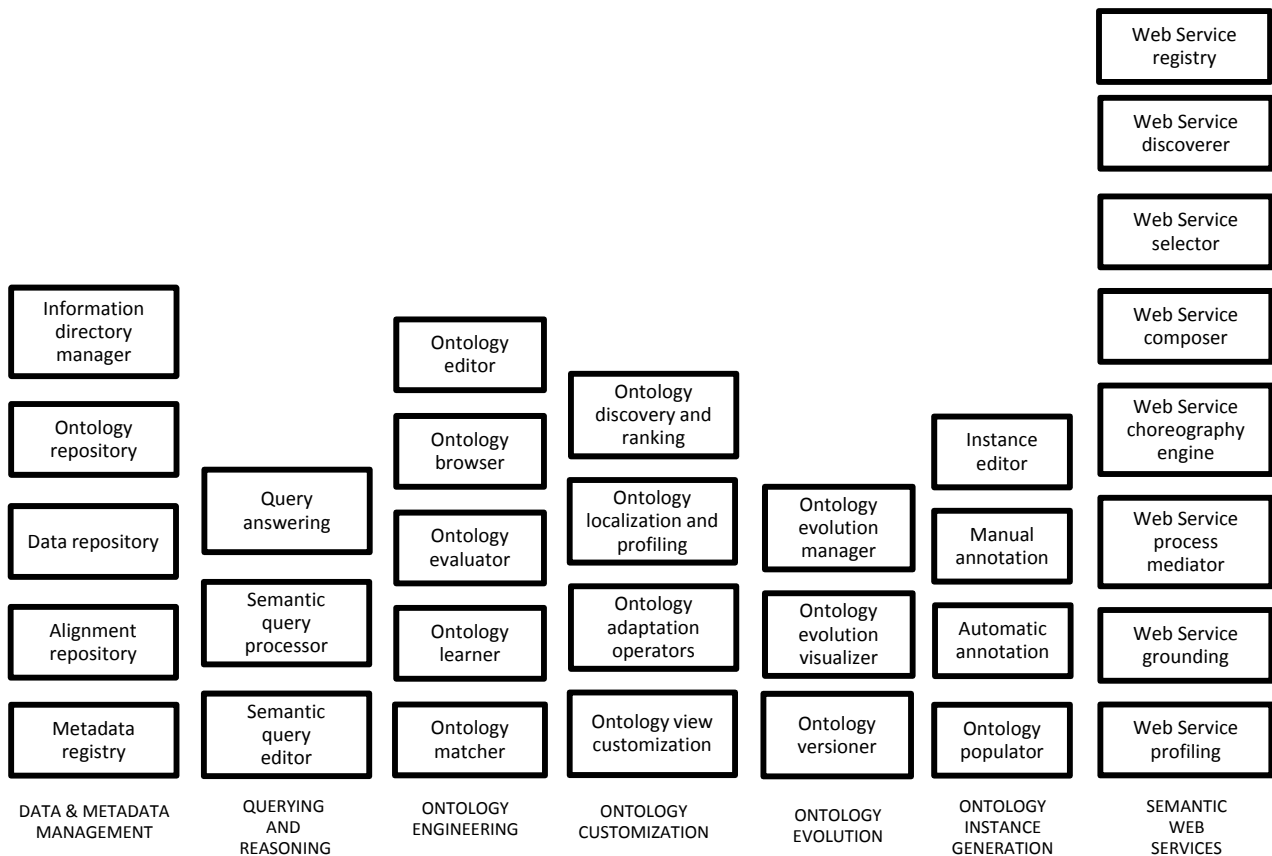


Figure 3.3: Components of the Semantic Web Framework

- *Alignment repository component.* This component provides functionalities to handle the storage and access to distributed alignments.
- *Metadata registry component.* This component provides functionalities to locally store and access metadata information.

Querying and reasoning

This dimension includes those components that generate and process queries.

- *Query answering component.* This component takes care of all the issues related to the logical processing of a query by providing reasoning functionalities to search results from a knowledge base.
- *Semantic query processor component.* This component takes care of all issues related to the physical processing of a query by providing functionalities to manage query answering over ontologies in distributed sources.
- *Semantic query editor component.* This component takes care of all the issues related to the user interface.

Ontology engineering

This dimension includes those components that provide functionalities to develop and manage ontologies.

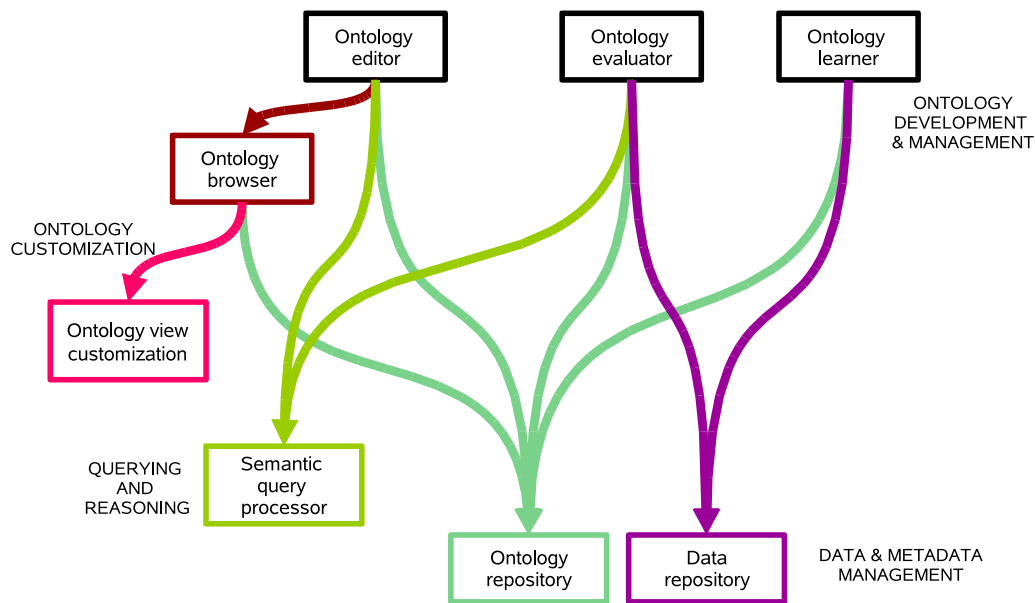


Figure 3.4: Dependencies of the components in the *Ontology engineering* dimension

- *Ontology editor component*. This component provides functionalities to create and modify ontologies, ontology elements, and ontology documentation. These functionalities include a single element edition or a more advanced edition, such as ontology pruning, extension or specialization.
- *Ontology browser component*. This component provides functionalities to visually browse an ontology.
- *Ontology evaluator component*. This component provides functionalities to evaluate ontologies, either their formal model or their content, in the different phases of the ontology life cycle.
- *Ontology learner component*. This component provides functionalities to acquire knowledge and generate ontologies of a given domain through some kind of (semi)-automatic process.
- *Ontology matcher component*. This component provides functionalities to match two ontologies and output some alignments. We can distinguish two types of such systems: those that generate matchings and those that use matchings for other tasks (merging, mediating, etc.).

Ontology customization

This dimension includes the components that customize and tailor ontologies.

- *Ontology localization and profiling component*. This component provides functionalities to adapt an ontology according to some context or some user profile.
- *Ontology discovery and ranking component*. This component provides functionalities to find appropriate views, versions or sub-sets of ontologies, and then to rank them according to some criterion.
- *Ontology adaptation operators component*. This component is in charge of applying appropriate operators to the ontology in question, the result of which is an ontology customized according to some criterion.
- *Ontology view customisation component*. This component is responsible for enabling the user to change or amend a view on a particular ontology that fits a particular purpose.

Ontology evolution

This dimension includes those components that manage the ontology evolution.

- *Ontology versioner component.* It maintains, stores and manages different versions of an ontology.
- *Ontology evolution visualizer component.* This component allows visualizing different versions of an ontology.
- *Ontology evolution manager component.* It is in charge of the timely adaptation of an ontology to the changes undergone and of the propagation of such changes to dependent artifacts.

Ontology instance generation

This dimension includes those components that generate ontology instances.

- *Instance editor component.* This component provides functionalities to manually create and modify instances of concepts and of the relations between them in existing ontologies.
- *Manual annotation component.* This component is in charge of the manual and semi-automatic annotation of digital content documents (e.g., web pages) with concepts in the ontology. This annotation process may be assisted or guided by a machine (semi-automatic annotation).
- *Automatic annotation component.* This component is in charge of the automatic annotation of digital content (e.g., web pages) with concepts in the ontology. Occurrences in the considered content of instances of concepts in the ontology are automatically detected and subsequently annotated.
- *Ontology populator component.* This component provides functionalities to automatically generate new instances in a given ontology from a data source.

Semantic web services

This dimension includes those components that discover, adapt/select, mediate, compose, choreograph, ground, and profile semantic web services.

- *Web service discoverer component.* This component provides functionalities to publish and search service registries, to control access to registries, and to distribute and delegate requests to other registries.
- *Web service selector component.* After discovering a set of potentially useful services, this component needs to check whether the services can actually fulfill the user's concrete goal and under what conditions.
- *Web service composer component.* This component is in charge of the automatic composition of the web services in order to provide new value-added web services.
- *Web service choreography engine component.* This component provides functionalities to use the choreography descriptions of both the service requester and the service provider to drive the conversation between them.
- *Web service process mediator component.* This component provides functionalities to reconcile the public process heterogeneity that can appear during the invocation of web services.
- *Web service grounding component.* This component is responsible for the communication between web services.

- *Web service profiling component.* This component provides functionalities to create web service profiles based on their execution history.
- *Web service registry component.* This component provides functionalities to register semantic web services.

3.4 Conclusions

The analysis made in [MS06] and [AMS⁺08] provides a set of characteristics of semantic Web applications. These characteristics are useful to determine the behavior of a given application according to factors such as the nature of the resources that the application will deal with, or the kind of reasoning that the application will use. Analyzing the values for these characteristics during the *Requirements Engineering* process will help to understand better the semantic application beforehand. Therefore, it will be helpful to provide a catalogue of common characteristics of semantic application in order to facilitate the identification of its semantic requirements.

A set of scenarios for applying semantic applications have been identified in the state of the art [JU99, KHS⁺08]. The work presented in [KHS⁺08] synthesizes the scenarios by exhaustively studying a big number of case studies (those published in the ISWC, ESWC and ASWC conferences). However, these works describe the scenarios in a researcher-oriented fashion that could be difficult to understand for a software engineer non-expert in semantic technologies, or for the customer that orders the development of an application. Thus, an effort should be made to explain these scenarios from a user-goal perspective using techniques that software engineers are accustomed to, avoiding, when possible, to use the complex terminology related to semantic technologies.

As it has been treated in Section 2.6, in order to apply Component Based Software Engineering to large-scale semantic applications it is necessary to define, among others, a set of independent components commonly used in semantic applications that should be completely specified by their interfaces. The Semantic Web Framework identifies and describes those components. However, the interfaces that the components of the SWF provide and require are not identified. In consequence, the interface identification should be made. Also, it should be helpful to provide architectural patterns that describe typical organizations of these components within the architecture of a large-scale semantic application. Out of the scope is the identification of particular technologies that support such component implementations.

Chapter 4

Research Methodology

In this chapter, we present the research methodology used for building the NeOn methodology for the development of large-scale semantic applications and of the main requirements that guide its development.

4.1 General Framework for Describing the Methodology

For building the methodology we used a “*divide and conquer*” strategy, that is, we decomposed the general problem to be solved in different subproblems.

For each subproblem, we provided different strategies and alternatives to find the solution. To obtain the solution to the general problem, that is, the development of a large-scale semantic application, the solutions to the different subproblems were combined.

In our case, the subproblems were the *Requirements Engineering* and *Design* processes. For the *Requirements Engineering* process we provided the application characteristics, use cases, system models described in chapters 6, 7 and 8 respectively as solutions for this subproblem. For the *Design Process* we provided the patterns described in 9 as a solution for this subproblem.

The *Requirements Engineering* and *Design* processes and their associated activities are described in chapters 10 and 11.

To obtain the methodological guidelines for building large-scale semantic applications, we grounded in the following approaches, as presented graphically in Figure 4.1.

- *Existing methodologies and methods.* In this case, we used Component-based Software Engineering and several agile methods that provide guidelines for carrying out a process or an activity. This is the case of the *Requirements Elicitation and Analysis* activity that had as a starting point the idea of iterative development proposed by Agile Methods.
- *Existing practices and previous experiences.* NeOn consortium members had built a lot of semantic applications in different domains across several European and national funded projects. Therefore, we made a retrospective analysis of the processes or activities performed within such projects to get a preliminary set of informal steps. As an example, we can mention the different use cases being described in the Knowledge Web project (FP6-507482) [GMG⁺07], whose application architectures have inspired the architectural patterns proposed in this deliverable.
- *The Semantic Web Framework.* The SWF has been elaborated in the Knowledge Web Project and describes the software used to build Semantic Web applications as reusable components. The components contained in the architectural patterns and gathered in this deliverable were extracted from this input.
- *Existing categorizations and scenarios of semantic applications.* In this case, we have extracted from [AMS⁺08, Mot99, DF08, KSF08] different characteristics of large-scale semantic applications; and

from [JU99, KHS⁺08] different use cases applied to semantic applications.

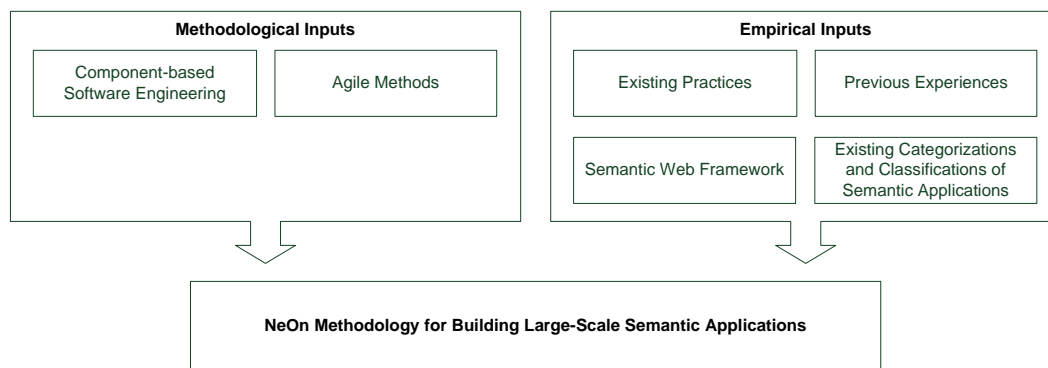


Figure 4.1: Inputs considered for obtaining the NeOn Methodology and building Large-scale Semantic Applications

Thus, the current version of NeOn methodology for building large-scale semantic applications describes the *Requirements Engineering* and *Design* processes, a set of 32 characteristics, 11 system models, 5 use cases and 28 architectural patterns which can be combined between them. Each process is decomposed in a set of activities, which, in turn, are provided with detailed methodological guidelines. In this sense, the deliverable is written with a process- or an activity- centric approach, and in a prescriptive way rather than a descriptive one.

For describing each process and activity included in the NeOn methodology here presented, we use the following template:

- Introduction.
- State of the art, with methods, techniques, and conclusion about related works.
- Detailed guidelines proposed for carrying out the process or the activity, including:
 - Definition.
 - Goal, explaining the main objective intended to achieve by the process or the activity.
 - Input, which includes the resources needed for carrying out the process or the activity.
 - Output, which includes the results obtained after carrying out the process or the activity.
 - Who, for identifying people or teams involved in the process or the activity.
 - When, for explaining in which moment the process or the activity had to be carried out.
 - How, which included details for carrying out the process or the activity in a prescriptive manner. A graphical workflow on how the process or the activity should be carried out is also included, with inputs, outputs and actors involved. The workflow will be represented using UML 2.0¹ activity diagrams. Additionally, methods and techniques supporting the process or the activity were proposed.
 - Why, for explaining the motivation behind performing the process or activity.
 - Where, for identifying the context in which the process or activity had to be carried out.

For each process and activity included in this deliverable, we provide a filling card including all the aforementioned information, except for the “how”. The use of such filling cards has allowed us to explain the information of NeOn methodology in a practical and easy way. Each filling card follows the filling card template shown in Table 4.1.

¹<http://www.uml.org>

- Examples explaining the proposed guidelines using previous experiences and/or NeOn use cases, whenever it has been possible.

Table 4.1: Template for Process and Activity Filling Card

Process or Activity Name	
<i>Definition</i>	
<i>Goal</i>	
<i>Input</i>	<i>Output</i>
<i>Who</i>	
<i>When</i>	
<i>Why</i>	
<i>Where</i>	

4.2 Requirements for the NeOn Methodology for Building Large-Scale Semantic Applications

The methodology presented in this deliverable must fulfill a set of requirements that can be grouped into two main types, namely, the generic requirements and the specific requirements (those that depend on the specific goals of the methodology). The generic requirements are those that any methodology must fulfill. The specific requirements of a given methodology are determined by factors such as the domain where the methodology is applied, cases, situations or problems it deals with, characteristics of the material (economic, technological, etc.), human or temporal resources, etc.

In this section we present the generic and specific requirements considered in the development of the NeOn methodology for building large-scale semantic applications. We have based our requirements on Paradela's conditions [Par01], having adapted them whenever it was necessary.

4.2.1 Generic requirements

- **Generality.** A methodology should be general enough and should not be driven to solve ad-hoc cases or problems.

In our case, the methodology here presented tackles the development of large-scale semantic applications in general, by means of proposing common semantic application characteristics; general use cases that appear in large-scale semantic applications; and different architectural patterns expressed with components of semantic applications and described in a conceptual level, which means that specific implementations of those components are not considered.

- **Completeness.** A methodology must consider all the cases presented and propose solutions to all of them.

In our case, the first version of the NeOn methodology deals with the *Requirements Engineering and Design Process* as subproblems of the general problem of building a large-scale semantic application

and provides a number of 32 characteristics, 5 use cases, 11 system models, and 28 architectural patterns as solutions of these subproblems. The second version will include the definition of the *Provisioning*, *Integration* and *Testing* processes and solutions for these subproblems.

- **Effectiveness.** A methodology should solve adequately the proposed cases that have a solution, with independence of the person that uses it. So, the methodology should be more prescriptive than descriptive.

In our case, we describe the methodology in a simple way, and any software developer with knowledge on semantic technologies will be able to understand and use it with no special effort.

- **Efficiency.** A methodology must be efficient, that is, be able to achieve its objective/goal. This means that the methodology should allow the construction of large-scale semantic applications with fewer resources (time, money, etc.) and better quality than before.

In our case, whenever it is possible, we use the methodology to describe and carry out experiments that confirm its efficiency.

- **Consistency.** A methodology must produce the same set of results for the same problem, independently of who carries it out.

In our case, the presented methodology identifies the outputs of the different activities involved in the development of large-scale semantic applications. The same set of outputs will be obtained after applying the methodology for a given case.

- **Finiteness.** Both the number of the elements composing a methodology and the number of activities must be finite, i.e., consuming a reasonable period of time.

In our case, the processes and activities presented in this deliverable include the initial and finite set of processes and the activities involved in the methodology. The number of elements used to describe the processes or the activities is also finite.

- **Discernment.** A methodology must be composed of a small set of structural, functional and representational components.

In our case

- With respect to structural components, our methodology provides a set of characteristics, use cases, system models and architectural patterns for building large-scale semantic applications.
- With respect to functional components, our methodology includes processes, activities, tasks, inputs, outputs and restrictions.
- With respect to representational components, the methodology provides textual and graphical representations for describing each process or activity, for the characteristics of large-scale semantic applications; and for the templates and patterns provided in this deliverable.

- **Environment.** Methodologies can be classified into scientific and technological. In scientific methodologies ideas are validated, whereas in technological ones artifacts are built. A technological methodology must consider the life cycle of the product guiding its development.

The NeOn methodology for building large-scale semantic applications can be considered as a technical methodology, because the main result after applying it should be a technological product, that is, a large-scale semantic application. Thus, it is needed to establish the life cycle for the semantic application. We will base the life cycle on the state of the art of Agile Methods (see Section 2.3).

- **Transparency.** A methodology must be like a white box, allowing to know in every moment the active processes or activities that are being performed, who is performing them, etc.

In our case, we explicitly define the actors, inputs, and outputs of each activity covered by the methodology.

- **Essential Questions.** The following six questions: “what”, “who”, “why”, “when”, “where”, and “how” must be considered for each activity included in the methodology.

In our case, the processes and activities described in this deliverable explains “what” each activity or process involved in the methodology refers to. The methodological guidelines for the processes or activities included in this deliverable answer the rest of the covered questions.

4.2.2 Specific requirements

- **Domain or Scope.** The presented methodology is for developing large-scale semantic applications.
- **Perspectives.** A methodology must facilitate its application following different approaches.

In our case, the methodology provides

- A set of use cases that let analyze the types of large-scale semantic applications from a functional external perspective, reflecting the interactions between the application and its users, and between the application and other external systems,
- A set of system models that let analyze the applications from an structural external perspective, reflecting the relationships between the semantic application and the resources that the application will deal with and the systems in which these resources are bound,
- A set of architectural patterns that let analyze the application from an structural internal perspective, reflecting the structure of the application as the components involved in the semantic application architecture.

- **Understanding.** A methodology must be ease to understand and learn in order to facilitate its generalized use and to achieve success.

The NeOn methodology for building large-scale semantic applications is being explained with simple descriptions and graphical representations to be easily understood by software developers in general.

- **Usability.** The degree of difficulty in using the methodology must be minimal.

The methodology is presented using a software engineering approach with different levels of complexity to facilitate a promptly assimilation.

- **Grounded in existing practices.** The methodology is grounded in existing methodologies of the Software Engineering field such Component Based Software Engineering and Agile Methods, as well as the previous experience in the development of semantic applications.

- **Flexibility.** The methodology can be adapted to concrete needs and users and permits the inclusion of new characteristics, use cases, system models and architectural patterns involved in the development of large-scale semantic applications.

- **Tool independence.** A methodology must be independent of the existing technology.

The methodology is being developed with the aim of being technology independent.

- **Reuse orientation.** The methodology will be highly oriented to software reuse.

- **Agility.** The methodology will enable the rapid development of large-scale semantic applications and permit the development team to adapt quickly to changing requirements.

- **Team size independence.** The methodology will allow different sizes of development team.

- **Distributed support.** The methodology will allow the development team to be distributed.

Chapter 5

NeOn Methodology for Building Large Scale Semantic Web Applications

A large-scale semantic application can be pure semantic or it can consist on a set of semantic components included in traditional IT systems.

In pure semantic applications the whole system is oriented to provide semantic functionalities and most of the components that conform its architecture are built upon semantic technologies and standards.

In IT systems that include semantic components not all the functionalities provided by the system are semantic. Within this kind of applications there exists a number of semantic modules oriented to provide a set of semantic functionalities among other non-semantic components.

The first version of the NeOn methodology for building large-scale semantic applications has as the main objective to guide the application developers to describe the architecture of pure large-scale semantic applications, or the semantic part of the IT systems that include semantic components starting from the application requirements.

In order to do so, according to the phases identified in Section 2.2 and taking into account the agile methods presented in Section 2.3, in this deliverable we only describe the *Requirements Engineering* and *Design* processes.

Figure 5.1 shows an overview of the *Requirements Engineering* and *Design* processes. Next, each of the processes is explained.

5.1 Requirements Engineering

During the *Requirements Engineering* process the requirements of the application must be found out, analyzed and documented. In order to facilitate the requirement analysis, the NeOn methodology for building large-scale semantic applications proposes to divide the requirements in three different categories:

- **Non-semantic Requirements.** This group gathers the application requirements that are no related to semantic functionalities. Any software engineering methodology supports the discovery of these requirements in many different ways (e.g. the Rational Unified Process [Kru00] or the agile methods presented in Section 2.3). The identification of non-semantic functionalities is out of the scope of this deliverable.
- **Semantic Application Requirements.** This group brings together the software requirements that gather the semantic functionalities of the application.
- **Set of Ontological Needs.** This group reflects the set of requirements to be taken into account in the development of the ontologies required by the semantic application. The ontologies required can be

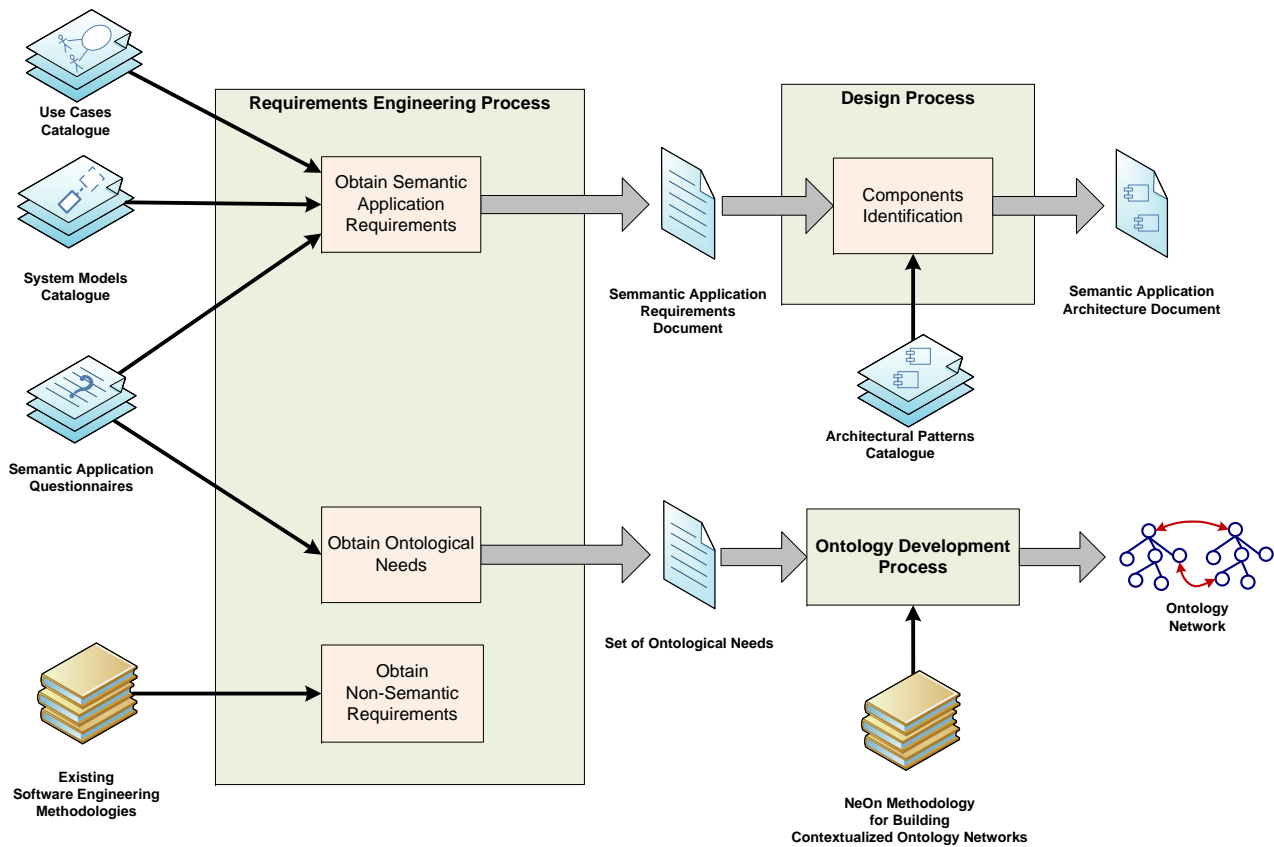


Figure 5.1: Overview of the *Requirements Engineering* and *Design* Processes

constructed under many different scenarios and following the guidelines given by the NeOn Methodology for Building Contextualized Ontology Networks [GS08]. The construction of the ontologies is out of scope of this deliverable.

5.1.1 Inputs

In this methodology, we provide the following inputs to facilitate the *Requirements Engineering* process: *Semantic Application Questionnaires* to help detecting the semantic application characteristics; and the *Use Cases Catalogue* and *System Models Catalogue* to facilitate the description of the different scenarios in which the application will operate.

Next, each of the inputs is presented.

Semantic Application Questionnaires

The NeOn methodology for building large-scale semantic applications provides a set of questions that can be used by the application developers for identifying the semantic characteristics of a given semantic application with respect to: the ontologies and data that the application will use, the reasoning features that the application will need, and other non-functional characteristics such interoperability with other applications. Additionally, the questions are also valid for identifying the set of ontological needs, and the data sets that the application will deal with.

The questions are grouped into the following questionnaires:

- *Questionnaire about Ontologies*. This questionnaire will help the application developers to determine the characteristics of the ontologies that the application will make use of. It will also help the appli-

cation developers to promptly advance possible ontological needs such as the need of performing an ontological resource reengineering, or of creating an ontology from scratch.

- *Questionnaire about Data.* Data can take two possible forms: semantic (e.g., in form of RDF instances) or non-semantic (e.g., data stored in a data base). This questionnaire will help the application developers to determine the characteristics of the data that the application will use and its relation with the ontologies which the data conform to. It will also help application developers to detect several ontological needs such, as the need of performing a non-ontological resource reengineering with the purpose of converting the non-ontological data to an ontological format, or if on the contrary, non-ontological data will be kept in the original sources and used at run-time by the application.
- *Questionnaire about Reasoning.* It will help application developers to determine the characteristics of the reasoning that the application will apply to ontology and data resources.
- *Questionnaire about Non-functional Characteristics.* This questionnaire will determine other non-functional characteristics, for example, the characteristics of interoperability of the application being built with other applications.

These questionnaires will be used by the application developers during the interviews with the application customers. The questionnaires are provided in Appendix 14.

Use Cases Catalogue

The methodology provides the application developers with a catalogue of use cases templates that describe the scenarios commonly appearing in semantic applications, such *Ontology-based Search* or *Semantic Browsing*. The catalogue have been obtained by analyzing the scenarios presented in Section 3.2.

These use cases templates can be selected, adapted and appended to the semantic application requirements by application developers.

The use cases catalogue is described in Chapter 7. The catalogue will be used during the *Requirements Engineering* process.

System Models Catalogue

As another methodological input, the methodology also provides a catalogue of system models. These system models are graphical representations that let application developers to preliminarily specify the system from the following perspectives:

- An external perspective, where the context or environment of the application is modeled by showing the limits of the application and the external systems or applications that will interoperate with the large-scale semantic application.
- A structural perspective, where the structure of the ontologies and the data processed by the application is modeled. The system models catalogue provides a set of symbols (e.g. ontological and non ontological resources, applications, etc.) and the relationships between these symbols that reflect the aforementioned structural perspective of the system.

The system models will reflect the scenarios identified during the use-case identification task constrained by the application characteristics, so the templates in the system model catalogue can be selected by taking into account the responses to the *Semantic Application Questionnaires* as well as the use cases selected from the *Use cases Catalogue*.

The system model catalogue is detailed and described in Chapter 8. This catalogue will be used during the *Requirements Engineering* process.

5.1.2 Outputs

Semantic Application Requirements Document

This output describes the semantic requirements of the application. The document contains the following:

- The semantic characteristics of the large-scale semantic application obtained after applying the *Semantic Application Questionnaires*.
- Identification of the data sets that the application will use at run-time and their associated characteristics. These data sets will be obtained from the answers to the *Semantic Application Questionnaires*.
- The applications or systems that the semantic application will interact with. These systems or applications will be obtained from the answers to the *Semantic Application Questionnaires*.
- The scenarios to be implemented by the application expressed in the form of use cases. These use cases will be obtained by adapting the use case templates included within the *Use Cases Catalogue*.
- A graphical representation of the system structure in the form of system models. These system models will be obtained by adapting the system models templates included within the *System Models Catalogue*.

The content of this document will be detailed in Chapter 10.

Set of Ontological Needs

This document describes the ontology development needs¹ in the form of a list of ontological needs. These needs identify a set of preliminary ontologies that should be developed for being integrated later within the application. The list contains the following entries:

- Ontologies that should be developed from scratch, if they are known at this stage.
- Ontologies that should be reused as they are or after a reengineering process.
- Ontologies that should be developed after applying a non-ontological resource reengineering process. In this case the non-ontological resources from which the reuse is performed should be specified.

These preliminary ontological needs will be obtained after applying the *Semantic Application Questionnaires*.

5.2 Design

Design is the process of describing the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes the algorithms used [Som07].

This deliverable is focused on the activity of obtaining the architecture of the large-scale semantic application.

5.2.1 Inputs

The inputs of the *Design* process are the following:

- Semantic Application Requirements Document.

¹The set of ontological needs are the input to the *Ontology Specification* activity described in the NeOn Methodology for Building Contextualized Ontology Networks [SAB⁺08]. Thus the *Set of Ontological Needs* input is different from the Ontology Requirements Specification Document, which is the output of the *Ontology Specification* activity.

- A catalogue of architectural Patterns.

The *Semantic Application Requirements Document* is an output of the *Requirements Engineering Process* and has been described in the previous section.

The *Architectural Patterns* are a methodological input to the *Design* process. These patterns reflect common organizations of semantic-related software components in large-scale semantic applications. They will be give support to the application developers during the *Design* process.

The components in the architectural patterns are those described in the Semantic Web Framework [GGMN08] (see Section 3.3).

5.2.2 Output

The output of the *Design* process is the *Semantic Application Architecture* document.

The architecture will be obtained with the support of the *Architectural Patterns*.

The *Semantic Application Architecture* document defines the architecture of the large-scale semantic application. The document contains the components, interfaces and relationships between them that will conform the application architecture.

Chapter 6

Semantic Application Characteristics

This chapter describes the main features that characterize large-scale semantic applications.

Appendix 14 describes a set of questionnaires that will help the ontology developer to identify the characteristics during the *Requirements Elicitation and Analysis* activity. These questionnaires can be used by the application developers during the interviews with the application customer.

We cluster the features according to the dimensions shown in Figure 6.1.

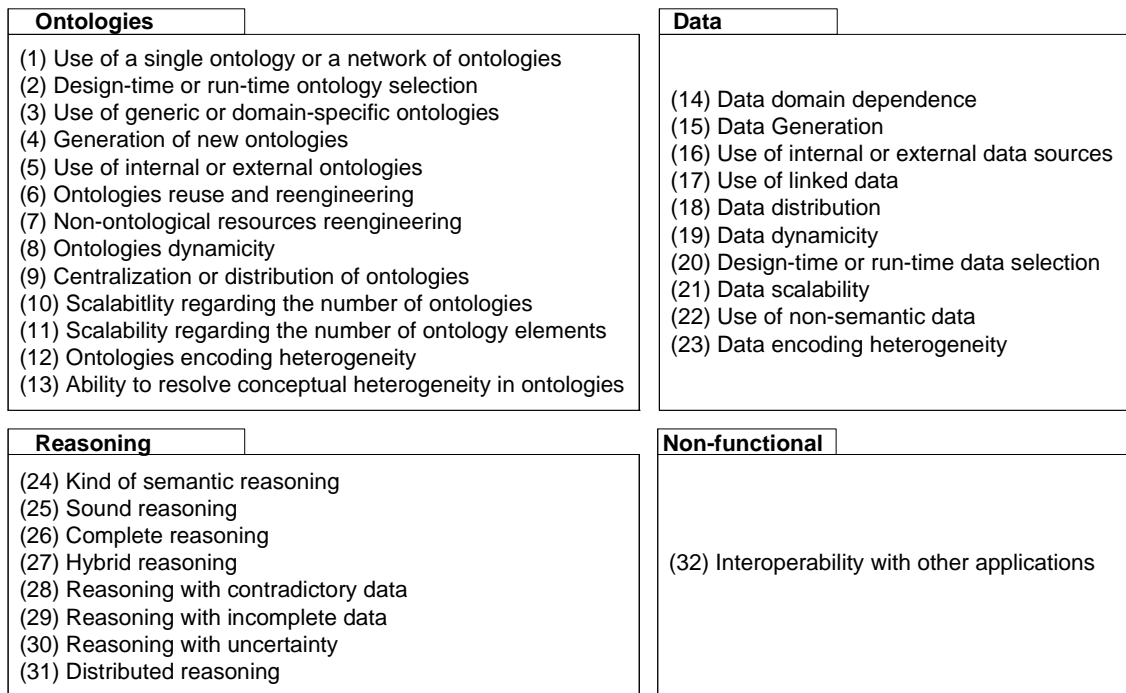


Figure 6.1: Characteristics of Large-scale Semantic Applications

Next, the characteristics in each dimension are explained.

6.1 Ontologies Dimension

The following characteristics describe the nature of the ontologies that the application will use.

Characteristic 1 – Use of a single ontology or a network of ontologies

It determines if the application will use a single ontology or a network of ontologies.

Characteristic 2 – Design-time or run-time ontology selection

It states if the ontologies will be selected at design-time by the ontology developer or at run-time by the application using some kind of criteria such as ontology quality, ontology complexity, ontology richness, user's evaluation, etc.

Characteristic 3 – Use of generic or domain-specific ontologies

It specifies if the ontologies used by the application will be bound to a particular domain or not.

Characteristic 4 – Generation of new ontologies

It determines if the application will produce ontologies.

Characteristic 5 – Use of internal or external ontologies

It determines if the application will use its own ontologies or, if the application will take ontologies from external sources, or a combination of both.

Characteristic 6 – Ontologies reuse and reengineering

It determines if the ontologies that the application uses will have been obtained by reusing or reengineering other existing ontologies.

Characteristic 7 – Non-ontological resources reengineering

It determines whether the ontologies that the application uses will have been obtained by reengineering other existing non-ontological resources.

Characteristic 8 – Ontologies dynamicity

It specifies the degree on which a part or the whole set of ontologies will constantly change during the application execution time or if they will not experiment any change during the execution time.

Characteristic 9 – Centralization or distribution of ontologies

It determines whether the ontologies used will be available in a centralized resource or distributed among different resources.

Characteristic 10 – Scalability with respect to the number of ontologies

It specifies if the application will operate at scale with a huge number of ontologies.

Characteristic 11 – Scalability with respect to the number of ontology elements

It specifies if the application can operate at scale with ontologies containing a huge number of ontology elements.

Characteristic 12 – Ontologies encoding heterogeneity

It states if the application will make use of ontologies formalized in different ontology languages (e.g. RDF(S)¹, OWL², etc.).

Characteristic 13 – Ability to resolve conceptual heterogeneity in ontologies

It defines whether the application will make use of mappings between ontologies. In that case it is important to know if the alignments can be discovered at design-time or at run-time.

¹<http://www.w3.org/TR/rdf-schema/>

²<http://www.w3.org/TR/owl-features/>

6.2 Data Dimension

The following characteristics describe the nature of the data that the application will use. Data could be expressed in RDF, XML, structured according to a relational schema or in other means.

Characteristic 14 – Data domain dependence

It specifies whether the data used by the application will be bound to a particular domain or not.

Characteristic 15 – Data generation

It determines whether the application will generate new data.

Characteristic 16 – Use of internal or external data sources

It determines whether the application will use only its own data, or if the application will take data from external sources.

Characteristic 17 – Use of linked data

It determines whether the application will use and/or generate linked data, e.g. resources from across the Open Linked Data community³. It is a particular case of Characteristic 16 when using external data sources.

Characteristic 18 – Data distribution

It determines whether data would be centralized in a single resource or distributed in multiple resources.

Characteristic 19 – Data dynamicity

It specifies if data would be changing during the application execution time, or if, on the contrary, they will not experiment any change during execution time.

Characteristic 20 – Design-time or run-time data selection

It states when the data sources will be selected. If the selection is done at run-time, then they has to be selected according to one criterion for giving preferences to the gathered data. The criteria for selecting data are the following: data quality, data richness, user's evaluation, etc.

Characteristic 21 – Data scalability

It specifies if the application can operate at scale with a huge size of data.

Characteristic 22 – Use of non-semantic data

It defines if the application will use non-semantic data, ranging from unstructured data (e.g. a corpora), to non-semantic structured resources (e.g. relational data bases, XML resources, etc.).

Characteristic 23 – Data encoding heterogeneity

It defines if the application will use data encoded in different formats (e.g. triples in RDF/XML⁴, or in N3⁵, etc.).

6.3 Reasoning Dimension

The application can apply different reasoning mechanisms to the ontologies and the data. Next some different characteristics regarding to reasoning are explained.

³<http://linkeddata.org/>

⁴<http://www.w3.org/TR/rdf-syntax-grammar/>

⁵<http://www.w3.org/DesignIssues/Notation3>

Characteristic 24 – Kind of semantic reasoning

It defines the kinds of semantic reasoning that the application will apply. These kinds of semantic reasoning can be: consistency checking, inference of new data, automatic classification, instances classification and subsumption reasoning.

Characteristic 25 – Sound reasoning

It defines if the application will apply a sound reasoning.

Characteristic 26 – Complete reasoning

It defines if the application will apply a complete reasoning.

Characteristic 27 – Hybrid reasoning

It specifies if the application will have to reason simultaneously with knowledge sources with non-ontological nature and with ontological sources. It is also important to identify if the application will need to use techniques such as:

- *Machine learning techniques like instance classification algorithms, approximate inference techniques, etc.*
- *Linguistic techniques for processing unstructured text resources, for localising ontologies, for learning ontologies or for populating ontologies.*
- *Statistical techniques.*
- *Graph matching.*

Characteristic 28 – Reasoning with contradictory data

It defines the capability of the application to deal with contradictory data.

Characteristic 29 – Reasoning with incomplete data

It defines the capability of the application to deal with incomplete data. If the semantic application deals with incomplete data, then the Open World Assumption is chosen; otherwise, the semantic application will reason with the Closed World Assumption.

Characteristic 30 – Reasoning with uncertainty

It defines if the application will take into account uncertainty for reasoning.

Characteristic 31 – Distributed reasoning

It defines if the application will distribute the reasoning among different nodes.

6.4 Non-functional Characteristics Dimension

Additionally to the characteristics presented before, the application can present different non-functional ones.

Characteristic 32 – Interoperability with other applications

This specifies if the application will interoperate with other external applications, as for example, heterogeneous proprietary and legacy solutions. This interoperability in a programatically level can be achieved by consuming and/or providing different kind of interfaces such Application Programming Interfaces, (Web) services, etc. If the application interoperates by consuming and/or providing Web Services, these can have associated semantics by applying technologies such as Semantic Web Services [FM01] (e.g., WSMO⁶).

⁶<http://www.wsmo.org>

Chapter 7

Use Cases Catalogue

This chapter presents a set of general purpose use cases that commonly appear in semantic applications obtained by analyzing the scenarios presented in Section 3.2 from the view point of the system user's goals. The use cases identified are the following:

- *Query Information*, where the user's goal is to obtain integrated information from several resources given a query.
- *Search Resources*, where the user's goal is to find resources (annotated with the corresponding meta-data) related to a given query. The difference with the previous use case is that while in the previous approach the results consist in information integrated from different resources, in this use case, the results consist in a set of documents located by using semantic information for searching.
- *Browse Resources*, where the user's goal is to navigate through categorised resources. Within this use case users navigate through the resources utilizing ontologies as indexes.
- *Extract Information*, where the user's goal is to extract meaningful information from a set of resources obtained after performing a search.
- *Manage Knowledge* where the user's goal is to collaboratively construct and evolve shared knowledge (e.g. knowledge management systems and contents management systems).

The scenarios for building semantic applications analyzed in the state of the art have been adapted by explaining them from a user-goal perspective (more likely as it is done in Software Engineering) in order to be clearly understood by software engineers and final application customers. Table 7.1 shows the mapping between the use cases obtained and the scenarios analyzed in the state of the art.

Next, all the use cases are explained and described. The use-cases will be graphically represented according to UML 2.0 notation and detailed with some of the information expressed in Table 7.2 (obtained from [Lar05]).

7.1 Query Information Use Case

In the *Query Information* use case a person or system requests the application to deliver some integrated information. Then the system returns the information to the actor that has requested it.

As happens with the category presented in Section 3.2 (*Usage as a Media for Knowledge Sharing*), this use case aims at knowledge sharing among systems, between people and systems, or among people using ontologies and instance models about the target knowledge [KHS⁺08]. This use case also corresponds to the scenario *Common Access to Information* and to the categories *Usage as a Data Schema* and *Usage as a Media for Knowledge Sharing* (see Section 3.2).

Use case	State of the art scenarios	
	Scenario in [JU99]	Scenario in [KHS⁺08]
1. Query Information	3. Common access to Information	4. Usage as a Data Schema 5. Usage as a Media for Knowledge Sharing
2. Search Resources	4. Ontology Based Search	2. Usage for Search
3. Browse Resources		3. Usage as an Index
4. Extract Information		7. Usage for Information Extraction 6. Usage for a Semantic Analysis
5. Manage Knowledge	1. Neutral Authoring 2. Ontology as Specification	1. Usage as a Common Vocabulary 8. Usage as a Rule Set for Knowledge Models 9. Usage for Systematising Knowledge

Table 7.1: Mapping between the scenarios analysed and the use cases obtained

Use-case Section	Comment
<i>Use-case name</i>	The name of the use-case. It should start with a verb.
<i>Scope</i>	The system under design.
<i>Level</i>	This section states whether the use case is an “user-goal” or a “sub-function”, which is a part that is repeated in some use cases and thus its description can be reused.
<i>Primary actor</i>	The primary actor that calls on the system to deliver its services.
<i>Stakeholders and interests</i>	Who care about this use case, and what they want.
<i>Preconditions</i>	What must be true from the start, and worth telling the reader.
<i>Success guarantee</i>	What must be true on successful completion, and worth telling the reader.
<i>Main success scenario</i>	A typical, unconditional happy path scenario of success.
<i>Extensions</i>	Alternate scenarios of success or failure.
<i>Special requirements</i>	Related non-functional requirements.
<i>Technology and data variations list</i>	Varying I/O methods and data formats.
<i>Frequency of occurrence</i>	Influences investigation, testing, and timing of implementation.
<i>Miscellaneous</i>	Miscellaneous information such as open issues.

Table 7.2: Template for describing use-cases [Lar05]

In order to achieve the use case goal, ontologies can be used as common vocabularies or data schemas, and often the application will use alignments between ontologies or between ontologies and non-ontological resources schemas. This issue is treated in Chapter 8.

Table 7.3 shows the template of this use case.

Use Case Template UCT1: Query Information	
<i>UML Diagram</i>	
<p>The diagram shows a rectangular box labeled 'System' with a tab at the top left. Inside the box is an oval labeled 'Query Information'. To the left of the box is a stick figure labeled 'Primary Actor' with a line connecting it to the 'Query Information' oval. To the right of the box are two stick figures, 'Information Provider 1' and 'Information Provider N', with vertical dots between them. Lines connect the 'Query Information' oval to each of these information providers.</p>	
<i>Primary Actor</i>	
The person or system who requests the System under development to deliver some information.	
<i>Stakeholders and Interests</i>	
<ul style="list-style-type: none"> • The <i>Primary Actor</i> requires some information that the System must gather and integrate from other Information Providers. • The <i>Information Providers</i> provide the System with the information to be integrated. 	
<i>Preconditions</i>	
<ul style="list-style-type: none"> • The <i>Primary Actor</i> can access the System. 	
<i>Success Guarantee (or Postconditions)</i>	
The System returns to the required information to the <i>Primary Actor</i> after integrating correctly the information obtained from the <i>Information Providers</i> if these exist.	
<i>Main Success Scenario (or Basic Flow)</i>	
<ol style="list-style-type: none"> 1. The <i>Primary Actor</i> requests the System to deliver some integrated information. 2. The System requests an <i>Information Provider</i> the information that requires from it. 3. The <i>Information Provider</i> responds to the System with the requested information. <p><i>Steps 2-3 are repeated until there are no more Information Providers to be requested.</i></p> <ol style="list-style-type: none"> 4. The System integrates the gathered information and returns the <i>Primary Actor</i> the required information. 	

(continues on next page)

(comes from previous page)

<i>Extensions (or Alternative Flows)</i>
<p>*a. At any time, System fails:</p> <ol style="list-style-type: none"> 1. The System informs the <i>Primary Actor</i> about the failure. <p>2-3a. No <i>Information Providers</i> are required to provide the requested information.</p> <ol style="list-style-type: none"> 1. System obtains the requested information from its own resources continuing in step 4. <p>2-3b. At any time, the communication with the <i>Information Provider</i> fails.</p> <ol style="list-style-type: none"> 1. System keeps requesting other <i>Information Providers</i> continuing in step 2. <ol style="list-style-type: none"> 1a. The information requested to the <i>Information Provider</i> is mandatory; so the <i>Information Provider</i> should give a correct response to the <i>Primary Actor</i>. 1. The System informs the <i>Primary Actor</i> about the failure.

Table 7.3: Query Information use case template

7.2 Search Resources Use Case

Within the *Search Resources* use case a person or system request the application to make a search by using semantic information for it.

This use case corresponds to with the category *Usage for Search* and to the scenario *Ontology-based Search*, presented in Section 3.2.

Table 7.4 shows the template of this use case.

Use Case Template UCT2: Search Resources
<i>UML Diagram</i>
<i>Primary Actor</i>
The person or system who requests the System under development to deliver some resources according to a given search.
<i>Stakeholders and Interests</i>
<ul style="list-style-type: none"> • The <i>Primary Actor</i> requires some resources that the System must return after searching them in external providers. • The <i>Resources Providers</i> provide the System with the resources found.

(continues on next page)

(comes from previous page)

<i>Preconditions</i>
<ul style="list-style-type: none"> • The <i>Primary Actor</i> can access the System.
<i>Success Guarantee (or Postconditions)</i>
The System returns the <i>Primary Actor</i> the resources found after searching for them in the <i>External Providers</i> if they exist.
<i>Main Success Scenario (or Basic Flow)</i>
<ol style="list-style-type: none"> 1. The <i>Primary Actor</i> requests the System to deliver some resources according to a given search. 2. The System requests a <i>Resources Provider</i> the resources that fulfil the search. 3. The <i>Resources Provider</i> responds to the System with the requested resources. <p><i>Steps 2-3 are repeated until there are no more Resources Providers to be requested.</i></p> <ol style="list-style-type: none"> 4. The System returns to the <i>Primary Actor</i> the resources found among all the Resources Providers.
<i>Extensions (or Alternative Flows)</i>
<p>*a. At any time, System fails:</p> <ol style="list-style-type: none"> 1. The System informs the <i>Primary Actor</i> about the failure. <p>2-3a. No <i>Resources Providers</i> are required to provide the resources to be found.</p> <ol style="list-style-type: none"> 1. The System obtains the requested resources by itself and continues in step 4. <p>2-3b. At any time the communication with the <i>Resources Providers</i> fails.</p> <ol style="list-style-type: none"> 1. The System keeps requesting other <i>Resources Providers</i> and continues in step 2.

Table 7.4: *Search Resources* use case template

7.3 Browse Resources Use Case

Within the *Browse Resources* use case a person uses the application to navigate through a set of resources, distributed among a set of resource providers, for accessing their contents or descriptions.

This use case corresponds to the category *Usage as an Index* presented in Section 3.2. This is so because this use case utilizes the structure of an ontology for browsing the resources.

Table 7.5 shows the template of this use case.

Use Case Template UCT3: Browse Resources	
<i>UML Diagram</i>	
<p>The diagram shows a system boundary labeled 'System' containing an oval use case labeled 'Browse Resources'. To the left of the system is a stick figure actor labeled 'Primary Actor' connected to the use case by a line. To the right of the system are two stick figure actors labeled 'Resources Provider 1' and 'Resources Provider N', with vertical dots between them. Lines connect the 'Browse Resources' use case to each of these provider actors.</p>	
<i>Primary Actor</i>	
The person who navigates through a set of resources for accessing their contents or descriptions.	
<i>Stakeholders and Interests</i>	
<ul style="list-style-type: none"> • The Primary Actor, who navigates through a set of resources until selecting the desired one for accessing its content or description. • The <i>Resources Providers</i> provide the System with the selected resource. 	
<i>Preconditions</i>	
<ul style="list-style-type: none"> • The <i>Primary Actor</i> can access the System. 	
<i>Success Guarantee (or Postconditions)</i>	
The System returns the selected resource to the <i>Primary Actor</i> after obtaining it from the correspondent <i>Resources Provider</i> if this exists.	
<i>Main Success Scenario (or Basic Flow)</i>	
<ol style="list-style-type: none"> 1. The <i>Primary Actor</i> navigates through a resource and requests the System to return it. 2. The System requests the resource to the <i>Resources Provider</i>, who owns the requested resource. 3. The <i>Resources Provider</i> responds to the System with the requested resource. 4. The System returns to the requested resource to the <i>Primary Actor</i>. 	

(continues on next page)

(comes from previous page)

<i>Extensions (or Alternative Flows)</i>
<p>*a. At any time, System fails:</p> <ol style="list-style-type: none"> 1. The System informs the <i>Primary Actor</i> about the failure. <p>2-3a. The resource is located at the System itself.</p> <ol style="list-style-type: none"> 1. The System continues in step 4. <p>2-3b. At any time the communication with the <i>Resources Provider</i> fails.</p> <ol style="list-style-type: none"> 1. The System informs the <i>Primary Actor</i> about the failure.

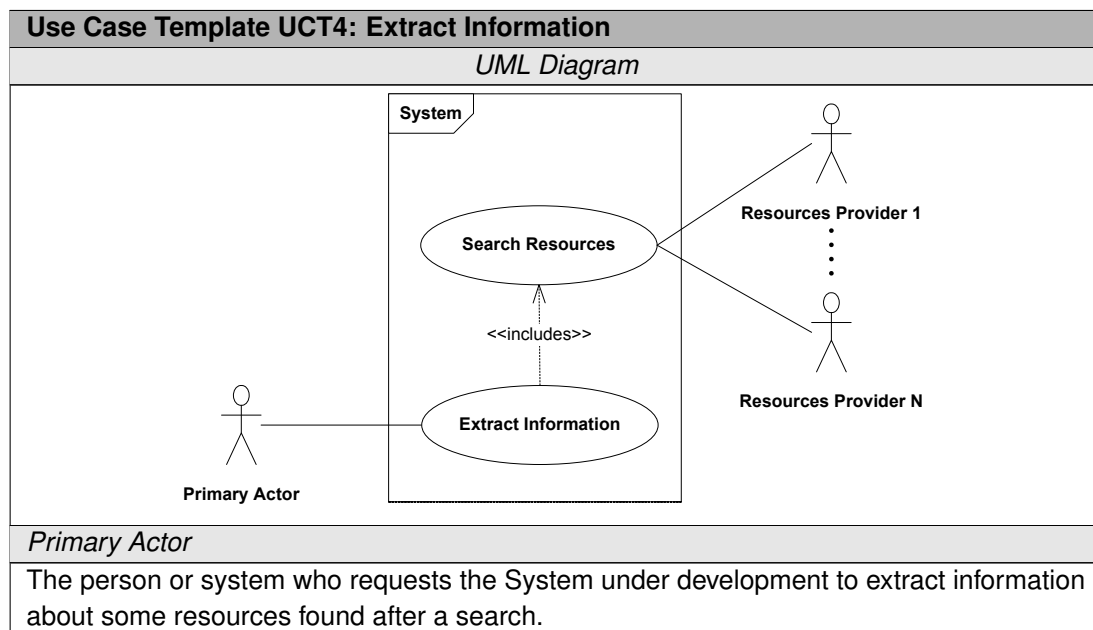
Table 7.5: *Browse Resources* use case template

7.4 Extract Information Use Case

Within the *Extract Information* use case a person or system requests the application to make a search using semantic information for it. After the search is performed, the system returns meaningful information about the resources found. The information returned can be obtained after performing some analysis to the search results.

This use case correspond to the category *Usage for Information Extraction* presented in Section 3.2. The use case includes the *Search Resources* use case in the sense that the applications implementing the *Extract Information* use case extract semantic information from the resources found.

Table 7.6 shows the template of this use case.



(continues on next page)

(comes from previous page)

<i>Stakeholders and Interests</i>
<ul style="list-style-type: none"> • The Primary Actor requires information about some resources that the System must return. • The <i>Resources Providers</i> provide the System with the resources from where extract the information.
<i>Preconditions</i>
<ul style="list-style-type: none"> • The <i>Primary Actor</i> can access the System.
<i>Success Guarantee (or Postconditions)</i>
The System returns the <i>Primary Actor</i> the information about the resources found after searching those resources in the <i>External Providers</i> if these exist.
<i>Main Success Scenario (or Basic Flow)</i>
<ol style="list-style-type: none"> 1. Find resources: Includes <i>Search Resources in External Providers (UCT2)</i>. 2. The System obtains the information associated to the documents found and returns it to the <i>Primary Actor</i>.
<i>Extensions (or Alternative Flows)</i>
*a. At any time, System fails:
<ol style="list-style-type: none"> 1. The System informs the <i>Primary Actor</i> about the failure.

Table 7.6: *Extract Information* use case template

7.5 Manage Knowledge Use Case

Within the *Manage Knowledge* use case a person or system manages a network of ontologies. The goal of this use case can be achieved in the following ways:

- By manually managing elements within an ontology (classes, properties, etc.).
- By manually managing instances or annotations.
- By learning an ontology, that is, creating or enriching an ontology by processing a given document corpora.
- By managing mappings between elements inside different ontologies, or between an element of a given ontology and a non-ontological resource schema. Any mapping can be either created in a totally manual way or in a semi-automatic way.
- By populating an ontology after processing any kind of non-ontological resource content.

This use case is related to the scenarios *Neutral Authoring*, *Ontology as Specification* and to the categories *Usage as a Common Vocabulary*, *Usage as a Rule Set for Knowledge Sharing* and *Usage for Systematizing Knowledge* presented in Section 3.2.

Table 7.7 shows the template of this use case.

Use Case Template UCT5: Manage Knowledge	
<i>UML Diagram</i>	
<p>The diagram shows a stick figure labeled 'Primary Actor' connected by a line to a rectangular box labeled 'System'. Inside the 'System' box is an oval labeled 'Manage Knowledge'.</p>	
<i>Primary Actor</i>	
The person who manages a network of ontologies (and associated instances).	
<i>Stakeholders and Interests</i>	
<ul style="list-style-type: none"> • The <i>Primary Actor</i> manages a network of ontologies (and associated instances). 	
<i>Preconditions</i>	
<ul style="list-style-type: none"> • The <i>Primary Actor</i> can access the System. 	
<i>Success Guarantee (or Postconditions)</i>	
The System commit the changes made in the ontology network by the <i>Primary Actor</i> .	
<i>Main Success Scenario (or Basic Flow)</i>	
<ol style="list-style-type: none"> 1. The <i>Primary Actor</i> request the System to add a given ontology element (class, instance, property, etc.). 2. The System commits the changes sent by the <i>Primary actor</i>. 	
<i>Extensions (or Alternative Flows)</i>	

(continues on next page)

(comes from previous page)

- *a. At any time, System fails:
1. The System informs the *Primary Actor* about the failure.
- 1a. The *Primary Actor* wants to modify a given ontology element (class, instance, property, etc.).
1. The *Primary Actor* sends the System the modifications.
 2. The System continues in Step 2.
- 1b. The *Primary Actor* wants to remove a given ontology element (class, instance, property, etc.)
1. The *Primary Actor* sends the System the ontology element to be deleted.
 2. The System continues in Step 2.
- 1c. The *Primary Actor* wants to learn an ontology from a given document corpora.
1. The *Primary Actor* sends the system the location of the corpora.
 2. The System generates the ontology and continues in Step 2.
- 1d. The *Primary Actor* wants to create manually a mapping .
1. The *Primary Actor* sends the system the two elements to be mapped.
 2. The System generates the mapping and continues in Step 2.
- 1e. The *Primary Actor* wants to modify a mapping.
1. The *Primary Actor* sends the system the mapping to be modified.
 2. The System modifies the mapping and continues in Step 2.
- 1f. The *Primary Actor* wants to delete a mapping.
1. The *Primary Actor* sends the system the mapping to be deleted.
 2. The System removes the mapping and continues in Step 2.
- 1g. The *Primary Actor* wants to automatically generate an alignment.
1. The *Primary Actor* sends the system the ontologies and/or structured non-ontological resources schemas to be mapped.
 2. The System creates the alignment and continues in Step 2.
- 1h. The *Primary Actor* wants to populate an ontology.
1. The *Primary Actor* sends the system the ontology to be populated and the non-ontological resource from where to generate the instances.
 2. The System creates the instances and continues in Step 2.

Table 7.7: *Manage Knowledge* use case template

Chapter 8

System Models Catalogue

System models provide the following information about resources:

- The nature of the knowledge resource, that is ontological or non-ontological.
- The system where the knowledge resource is located. The resource is bound within a system limit.
- The dynamic nature of the repository, that is, if resources are engineered at design-time or, on the contrary, they are gathered and selected at run-time.

This chapter presents the catalogue of system models associated to the use cases provided in Chapter 7 and constrained by the semantic characteristics presented in Chapter 6.

The chapter is structured as follows.

- First, a set of basic symbols that will appear in the system models is provided.
- Then, the possible relationships between the basic symbols are described.
- Afterwards, a set of basic templates that include various symbols and relationships are specified.
- Finally, the catalogue of system models associated to the use cases are provided.

8.1 Basic Symbols

8.1.1 Resources

The methodology proposes to draw resources on system models with an identifier, as well as with a symbol that represent aspects such as the ontological or non-ontological nature of the resource (e.g., ontology, instances, schema, data, etc.). Next, the symbols used for representing ontological and non-ontological resources are introduced.

Static Ontological Resources

Tables 8.1 and 8.2 describe the symbols used when representing either static ontologies or static instances (ontologies or instances selected at design-time) within system models.

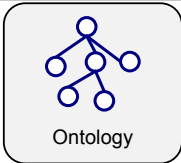
Symbol 1	
Name	Static Ontology
<i>Representation</i>	
	
<i>Associated Characteristics</i>	
2	Ontology selected at design-time.

Table 8.1: Symbol 1. Static Ontology

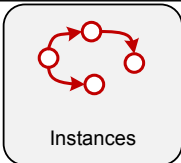
Symbol 2	
Name	Static Instances
<i>Representation</i>	
	
<i>Associated Characteristics</i>	
20	Instances selected at design-time.

Table 8.2: Symbol 2. Static Instances

Static Non-ontological Resources

Non Ontological Resources are existing knowledge-aware resources whose semantics have not been formalised yet by an ontology [SAB⁺08]. These resources present the form of free texts, textual corpora, web pages, standards, catalogues, web directories, classifications, thesauri, lexicons and folksonomies, among others [SAB⁺08].

Sabou et al. [SAA⁺07] classify non ontological resources into unstructured (e.g., corpora), semi-structured (e.g., folksonomies, html pages, etc.), and structured (e.g., databases, XML, etc.) resources. With the purpose of drawing system models, this deliverable makes a distinction between structured and unstructured non-ontological resources. Unstructured resources includes semi-structures resources.

Non-ontological resources symbols are described in Tables 8.3, 8.4 and 8.5. The first symbol is used for representing the schema of the structured non-ontological resource, while the second symbol represents the data that contains the structured non-ontological resource. Finally the last symbol represents a collection of unstructured resources.

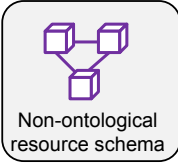
Symbol 3	
<i>Name</i>	Static Non-ontological Resource Schema
<i>Representation</i>	
 <p>Non-ontological resource schema</p>	
<i>Associated Characteristics</i>	
20	Data selected at design-time.
22	Use of non-semantic data.

Table 8.3: Symbol 3. Static Non-ontological Resource Schema


Symbol 4	
<i>Name</i>	Static Non-ontological Resource Content
<i>Representation</i>	
 <p>Non-ontological resource content</p>	
<i>Associated Characteristics</i>	
20	Data selected at design-time.
22	Use of non-semantic data.

Table 8.4: Symbol 4. Static Non-ontological Resource Content


Symbol 5	
<i>Name</i>	Static Unstructured Document
<i>Representation</i>	
 <p>Unstructured document</p>	
<i>Associated Characteristics</i>	
20	Data selected at design-time.
22	Use of non-semantic data.

Table 8.5: Symbol 5. Static Unstructured Document

8.1.2 Dynamic Resources

As it have be observed in the previous subsection, resources must be identified with a name. This is useful when the resources are gathered or engineered at design-time. However in certain applications, any kind of resource can be obtained at run-time as it reflects characteristics 2 (*Design-time or run-time ontology selection*) and 20 (*Design-time or run-time data selection*). The symbols used for representing one or more of these dynamic run-time identified resources can be seen in Tables 8.6, 8.7, 8.8, 8.9 and 8.10.


Symbol 6	
Name	Dynamic Ontology
<i>Representation</i>	
	
<i>Associated Characteristics</i>	
2	Ontology discovered at run-time.

Table 8.6: Symbol 6. Dynamic Ontology

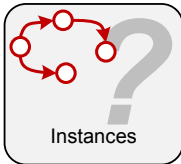
Symbol 7	
Name	Dynamic Instances
<i>Representation</i>	
	
<i>Associated Characteristics</i>	
20	Instances discovered at run-time.

Table 8.7: Symbol 7. Dynamic Instances

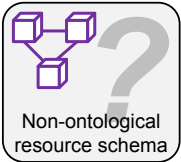
Symbol 8	
Name	Dynamic Non-ontological Resource Schema
<i>Representation</i>	
	
<i>Associated Characteristics</i>	
20	Data discovered at run-time.
22	Use of non-semantic data.

Table 8.8: Symbol 8. Dynamic Non-ontological Resource Schema

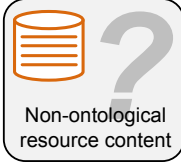
Symbol 9	
<i>Name</i>	Dynamic Non-ontological Resource Content
<i>Representation</i>	
 Non-ontological resource content	
<i>Associated Characteristics</i>	
20	Data discovered at run-time.
22	Use of non-semantic data.

Table 8.9: Symbol 9. Dynamic Non-ontological Resource Content


Symbol 10	
<i>Name</i>	Dynamic Unstructured Document
<i>Representation</i>	
 Unstructured document	
<i>Associated Characteristics</i>	
20	Data discovered at run-time.
22	Use of non-semantic data.

Table 8.10: Symbol 10. Dynamic Unstructured Document

8.1.3 Applications and Systems

The system models will reflect the applications that manage the aforementioned resources. Application will be represented with the symbol shown in Table 8.11. At least the application being developed must appear in a system model. However other existing or future applications can appear if they are somehow related to the application being developed.

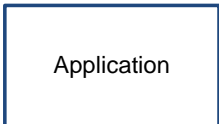
Symbol 11	
<i>Name</i>	Application
<i>Representation</i>	
 Application	

Table 8.11: Symbol 11. Application

Resources and applications will be always bounded within a system. The symbol presented in Table 8.12 will be used to surround resources and applications. At least one system limit must appear within a system model surrounding the application being developed.

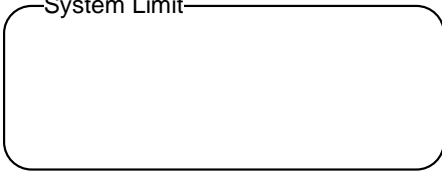
Symbol 12	
<i>Name</i>	System Limit
<i>Representation</i>	
	
<i>Associated Characteristics</i>	
32	If there exists interoperability with external systems then more than one system limit will appear in the system model.
5	If external ontologies are present then there will appear ontologies bound within one or more system limit different than the system limit of the application itself.
16	If external ontologies are present then there will appear data sources bound within one or more system limit different than the system limit of the application itself.

Table 8.12: Symbol 12. System Limit

System limits will determine the responsibilities of the development team in the development and management of the resources and applications comprised in the system to develop. Resources and applications bound by other systems should be managed and developed by other stakeholders (e.g. the customer).

8.2 Relationships Between Symbols

This section enumerates the possible relationships between the basic symbols presented that can appear in the proposed system models.

8.2.1 Conforms To

The relationship *Conforms To* represents either instances that conforms to an specific ontology (see Table 8.13) or a structured non-ontological content that is expressed according to a given schema (see Table 8.14).

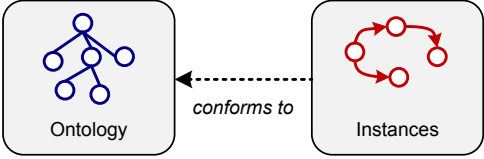
Relationship 1	
<i>Name</i>	Instances that Conform To a Given Ontology
<i>Representation</i>	
	

Table 8.13: Relationship 1. Instances that Conform To a Given Ontology

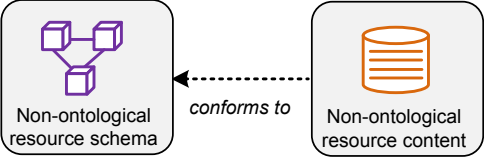
Relationship 2	
<i>Name</i>	Non-ontological Resource Content that Conforms To a Given Schema
<i>Representation</i>	
	

Table 8.14: Relationship 2. Non-ontological Resource Content that Conforms To a Given Schema

The dynamic resources shown in Subsection 8.1.2 can also participate in this relationship.

Optionally, a structured non-ontological content together with the schema it conforms to can be abbreviated with the symbol shown in Table 8.15.


Symbol 13	
<i>Name</i>	Non-ontological Resource Content that Conforms To a Given Schema Abbreviation
<i>Representation</i>	
	

Table 8.15: Symbol 13. Non-ontological Resource Content that Conforms To a Given Schema Abbreviation

8.2.2 Aligned With

The relationship *Aligned with* represents both an ontology aligned with other ontology (see Table 8.16) and an ontology aligned with a structured non-ontological schema (see Table 8.17).

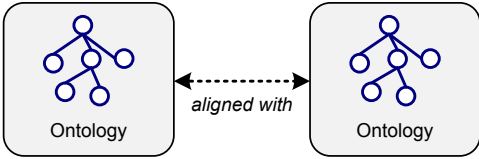
Relationship 3	
Name	Two Aligned Ontologies
<i>Representation</i>	
	
<i>Associated Characteristics</i>	
13	Ability to resolve conceptual heterogeneity in ontologies.

Table 8.16: Relationship 3. Two Aligned Ontologies

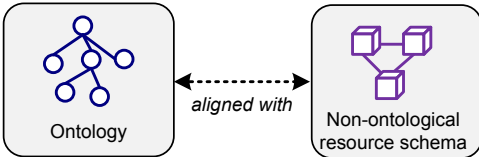
Relationship 4	
Name	Non-ontological Schema Aligned With an Ontology
<i>Representation</i>	
	
22	Use of non-semantic data.

Table 8.17: Relationship 4. Non-ontological Schema Aligned With an Ontology

The dynamic (multiple) resources shown in Subsection 8.1.2 can also participate in this relationship.

8.2.3 Annotate

The relationship *Annotate* can represent

- An unstructured document annotated by a set of instances.
- An unstructured document annotated by the content of a non-ontological resource.
- An ontology annotated by a set of instances.
- An ontology annotated by the content of a non-ontological resource.

Table 8.18 represents an unstructured document that is annotated by a set of instances. The instances that annotate the document generally conform to a given ontology.

The dynamic (multiple) resources shown in Subsection 8.1.2 can also participate in this relationship, which is

- documents can be discovered at run-time and annotated with dynamically generated instances, or
- a suitable ontology for annotating a set of given documents can be discovered at run-time, or
- both ontology and documents can be discovered at run-time.

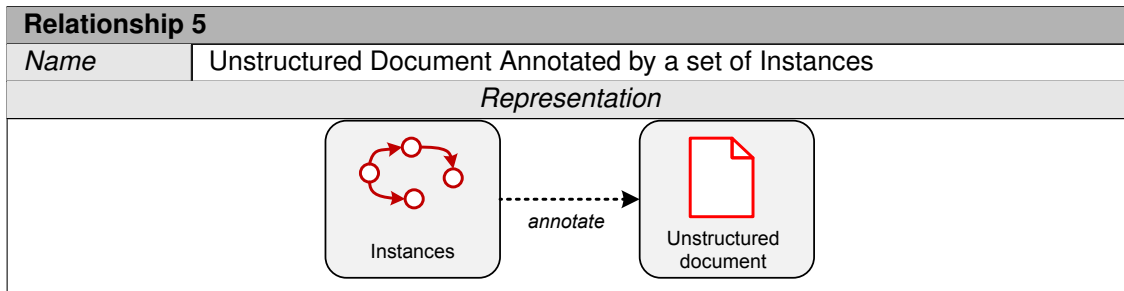


Table 8.18: Relationship 5. Unstructured Document Annotated by a set of Instances

Table 8.19 represents an unstructured document annotated by the content of a non-ontological resource. The metadata that annotates the document generally conforms to a non-ontological resource schema.

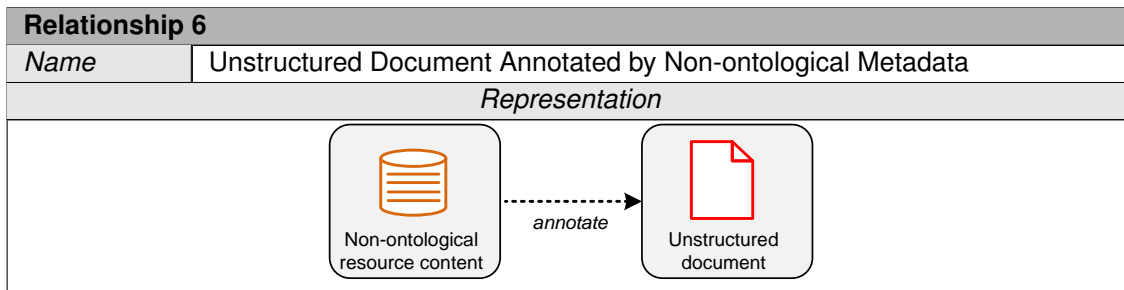


Table 8.19: Relationship 6. Unstructured Document Annotated by Non-ontological Metadata

Table 8.20 represents an ontology annotated by a set of instances. The metadata that annotates the ontology generally conforms to another ontology.

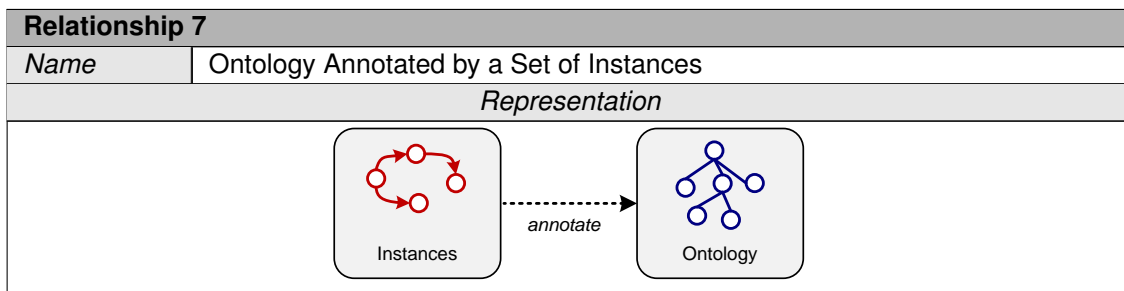


Table 8.20: Relationship 7. Ontology Annotated by a Set of Instances

Table 8.21 represents an ontology annotated by a non-ontological resource. The metadata that annotates the ontology generally conforms to a given non-ontological resource schema.

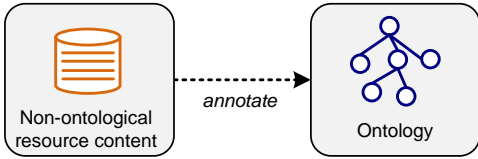
Relationship 8	
Name	Ontology Annotated by a Non-ontological Metadata
Representation	
	

Table 8.21: Relationship 8. Ontology Annotated by a Non-ontological Metadata

8.3 Basic Templates

This section shows several basic templates that will be used in the catalogue of system models presented in Section 8.4. These templates represent common organizations of symbols and relationships.

8.3.1 Data Sources with Schema

Ontological and structured non-ontological data sources (or a combination of them) that conform to a given schema can be represented under three different basic templates. Whenever the symbol shown in Figure 8.1 appears in a system model template it can be expanded by using one of the following basic templates.

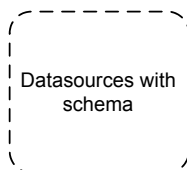


Figure 8.1: Datasources with schema

Data Sources with Schema: Ontological Resources

This template represents several ontological data sources (instances) that conform to a given ontology (see Table 8.22).

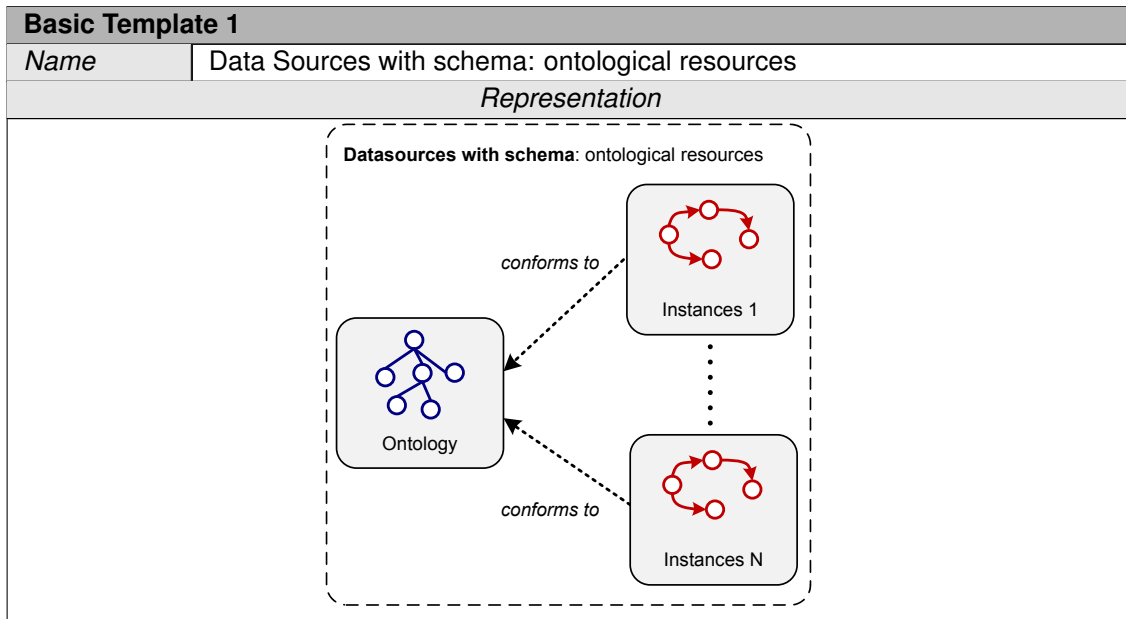


Table 8.22: Basic Template 1. Data Sources with schema: ontological resources

Data Sources with Schema: Non-ontological Resources

This template represents several structured non-ontological data sources that conform to a given non-ontological resource schema (see Table 8.23).

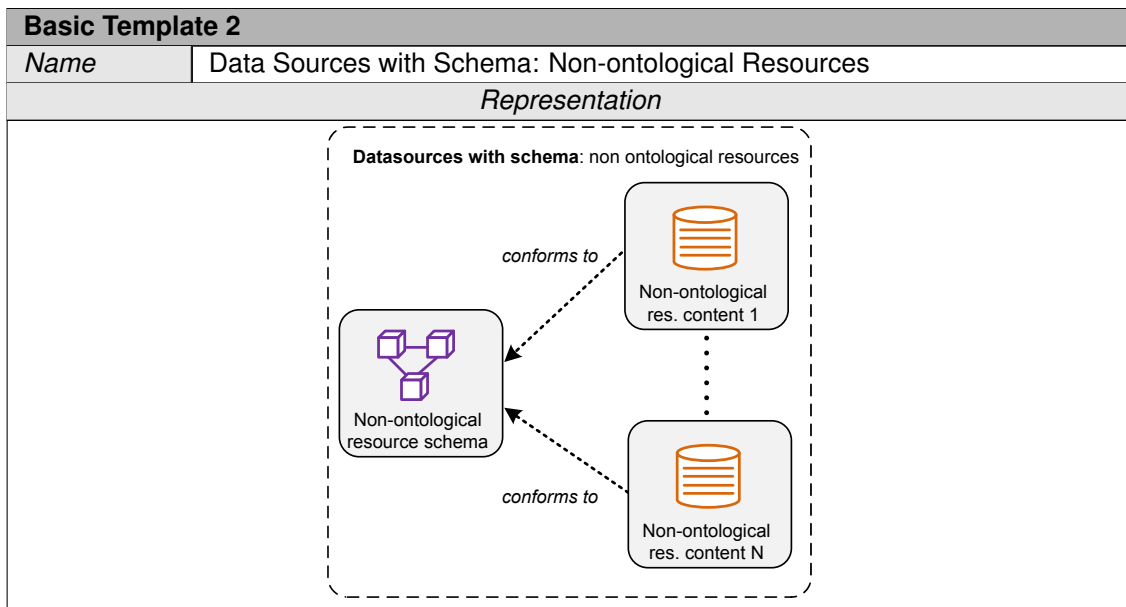


Table 8.23: Basic Template 2. Data Sources with Schema: Non-ontological Resources

Data Sources with Schema: Ontological and Non-ontological resources

This template represents several structured ontological and non-ontological data sources that conform to a given ontology.

As can be seen in Table 8.24, the schemas of the non-ontological data sources are aligned with an ontology.

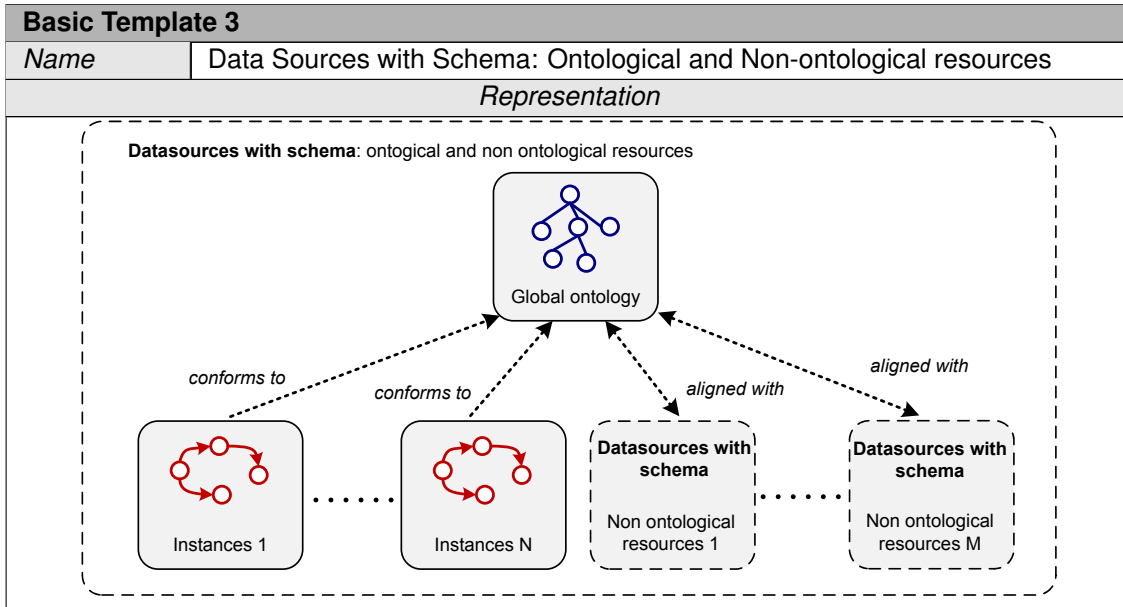


Table 8.24: Basic Template 3. Data Sources with Schema: Ontological and Non-ontological resources

In this case the main schema is considered to be the ontology because the other schemas will not be used for accessing the data directly by the application.

8.3.2 Annotated Resources

Annotated ontological or non-ontological resources whose annotations conform to a given ontological or non-ontological schema can be represented by four different templates. Whenever the symbol shown in Figure 8.2 appears in a system model template it can be expanded by using one of the following basic templates.

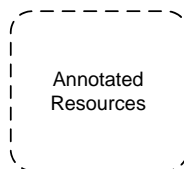


Figure 8.2: Annotated Resources

Annotated Documents with Ontological Metadata

This template represents one or more unstructured document collections annotated by one or more instances resources that conform to a given ontology. Table 8.25 shows the representation of this template.

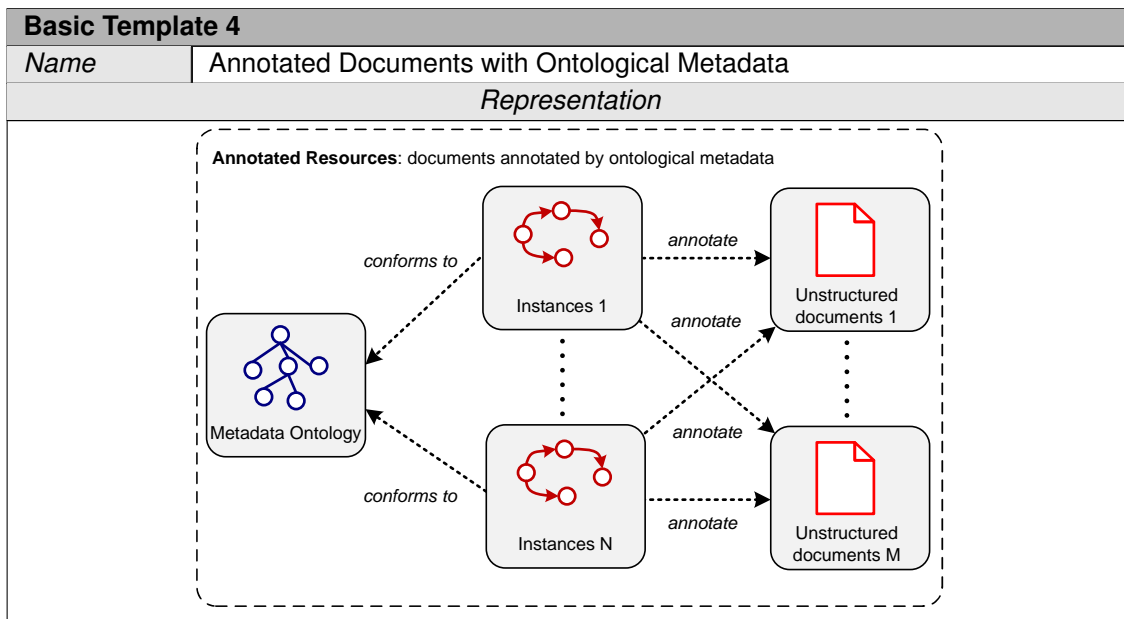


Table 8.25: Basic Template 4. Annotated Documents with Ontological Metadata

The ontology, according to which the instances (or annotations) it will conform to, can be selected at design-time or can be dynamic as explained in Subsection 8.1.2. In [Man07] when the metadata of documents explicitly refers to concepts of a specific ontology, it is considered a tightly coupled approach, while if documents are not committed to any predefined ontology, it is considered a loosely coupled approach. In this last case an appropriate ontology for a given domain may be chosen at run-time.

Annotated Documents with Non-ontological Metadata

This template represents one or more unstructured document collections annotated by metadata that conforms to a given non-ontological resource schema. Table 8.26 shows the representation of this template.

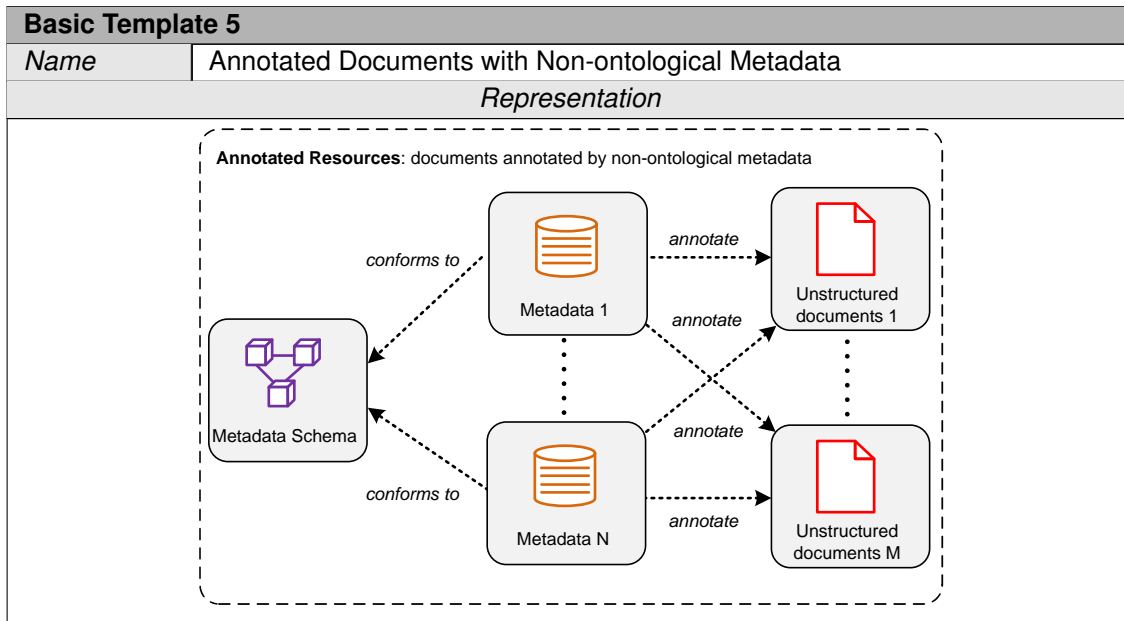


Table 8.26: Basic Template 5. Annotated Documents with Non-ontological Metadata

Annotated Ontologies with Ontological Metadata

This template represents one or more ontologies annotated with metadata instances that conform to a ontology. Table 8.27 shows the representation of this template.

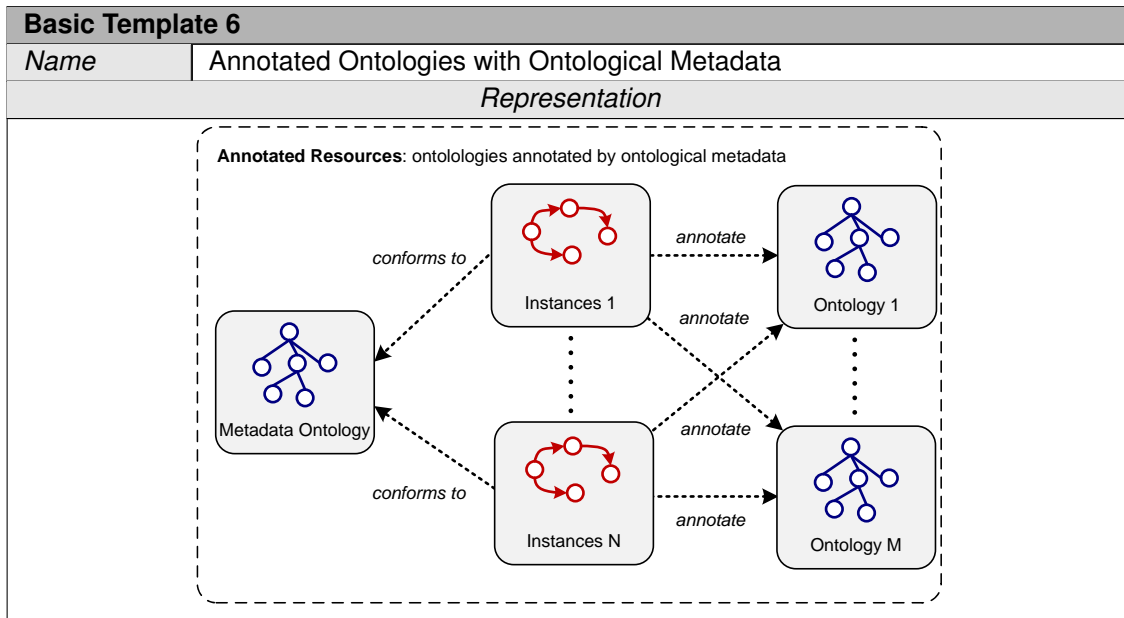


Table 8.27: Basic Template 6. Annotated Ontologies with Ontological Metadata

Annotated Ontologies with Non-ontological Metadata

This template represents one or more ontologies annotated by metadata that conforms to a given non-ontological resource schema. Table 8.28 shows the representation of this template.

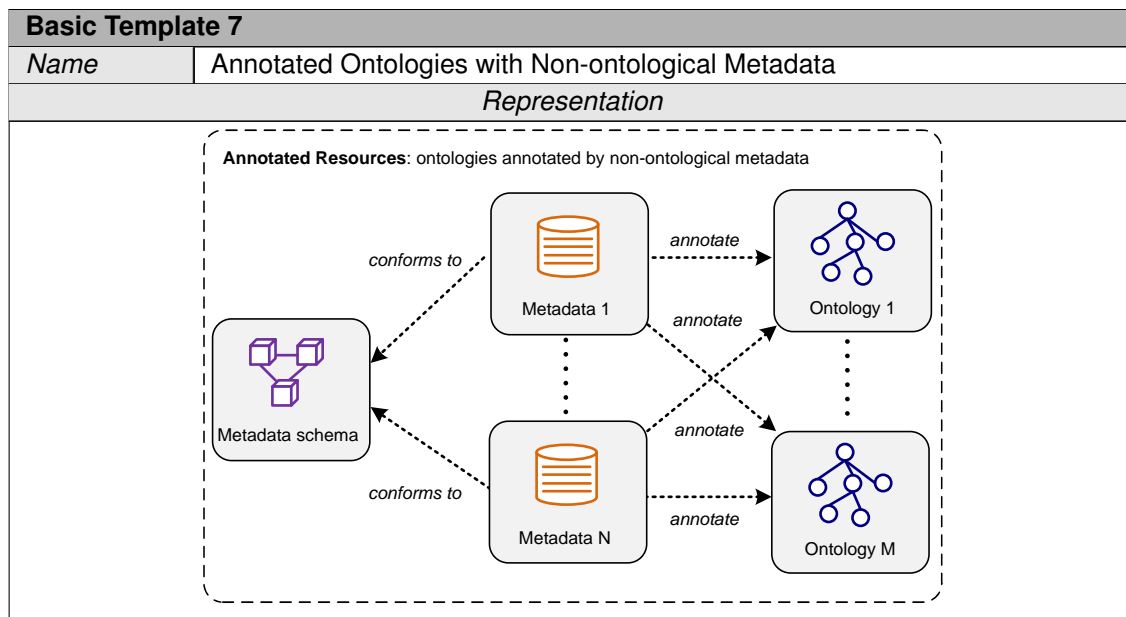


Table 8.28: Basic Template 7. Annotated Ontologies with Non-ontological Metadata

8.4 System Models

This section provides a catalogue of system models for the use cases provided in Chapter 7 and constrained by the semantic characteristics presented in Chapter 6.

Next, the system models presented in this section are enumerated and grouped according to the associated use case template.

- **UCT1: Query Information.**
 - System Model 1. Query Information with a Single Ontology/Schema Approach.
 - System Model 2. Query information with a Multiple Ontologies Approach.
 - System Model 3. Query information with a Hybrid Ontologies Approach.
- **UCT2: Search Resources.**
 - System Model 4. Search Resources.
- **UCT3: Browse Resources.**
 - System Model 5. Browse Annotated Resources.
 - System Model 6. Browse Ontological Resources.
- **UCT4: Extract Information.**
 - System Model 7. Extract Information.

- **UCT5: Manage Knowledge.**

- System Model 8. Edit Datasources.
- System Model 9. Edit Annotations.
- System Model 10. Populate.
- System Model 11. Learn.

8.4.1 Query Information System Models

The system models presented in this subsection correspond to the use case enumerated in Section 7.1. We have described the following models according to the state of the art of ontology based integration of information [WVV⁺01].

Query information with a Single Ontology/Schema Approach

This system model has to be used when the application queries data sources that conform to a given schema. Table 8.29 shows an application that queries data sources that conforms the given schema.

System Model 1	
<i>Name</i>	Query information with a Single Ontology/Schema Approach
<i>Representation</i>	
<pre> graph LR Application[Application] -.-> queries Datasources[Datasources with schema] </pre>	
<i>Use Case</i>	UCT1: Query Information
<i>Associated Characteristics</i>	
1	Single ontology typology.

Table 8.29: System Model 1. Query information with a Single Ontology/Schema Approach

Query Information with a Multiple Ontologies Approach

This system model has to be used when the application queries data sources that are expressed according to multiple aligned schemas. Table 8.30 shows an application that queries data sources that conform the given aligned schemas. Each of them can be any data source with a schema presented in Subsection 8.3.1.

System Model 2	
Name	Query information with a Multiple Ontology Approach
<i>Representation</i>	
Use Case	UCT1: Query Information
<i>Associated Characteristics</i>	
1	Network of ontologies typology.
13	Existence of ontologies' conceptual heterogeneity.

Table 8.30: System Model 2. Query information with a Multiple Ontology Approach

Query Information with a Hybrid Ontology Approach

This system model has to be used when the application queries data sources that are expressed according to multiple schemas aligned with a shared vocabulary. Table 8.31 shows and application that queries data sources aligned with the shared vocabulary. Each of them can be any data source with a schema presented in Subsection 8.3.1.

System Model 3	
Name	Query information with a Hybrid Ontology Approach
<i>Representation</i>	
Use Case	UCT1: Query Information

(continues on next page)

(comes from previous page)

Associated Characteristics	
1	Multiple Ontology Typology.
13	Ability to resolve conceptual heterogeneity in ontologies.

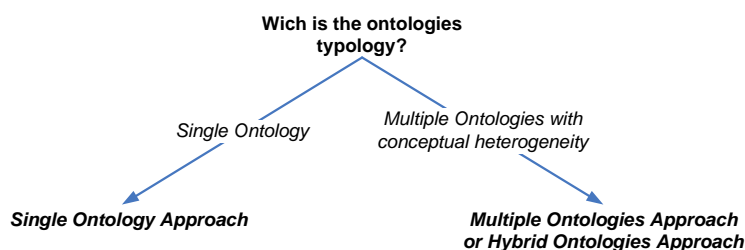
Table 8.31: System Model 3. Query information with a Hybrid Ontology Approach

Criteria for selecting the appropriate approach

The *single ontology* approach can be applied when the application is based on a single ontology typology (characteristic 1), so there is not conceptual heterogeneity (characteristic 13). According to [WVV⁺01], the implementation effort of this approach is straight-forward, but adding sources or removing them from the application requires some adaptation in the global ontology.

Both the *multiple ontology* and *hybrid ontology* approaches can be applied when the application is based on a network of ontologies typology (characteristic 1).

Figure 8.3 shows the criteria used for selecting the appropriate system model according to the elicited semantic application characteristics.

Figure 8.3: *Obtain Information* system model election criteria

If the ontologies typology is a network of ontologies, then for selecting the appropriate system model the selection between the *multiple ontology* and *hybrid ontology* approaches has to be made according to design decisions that regard factors such the implementation effort. According to [WVV⁺01], the implementation effort of the single ontology approach is costly because adding new resources requires to provide a new source ontology related to other ontologies, whereas the implementation effort of the *multiple ontology* approach is reasonable because adding new resources requires to provide a new source ontology aligned with the shared vocabulary.

8.4.2 Search Resources System Model

The system model presented in this subsection corresponds to the use case enumerated in Section 7.2.

Table 8.32 shows the system model associated to the use case *Search Resources*. The figure represents an application that searches resources annotated according to the *Annotated Documents* basic template explained in Subsection 8.3.2.

System Model 4	
Name	Search Resources
<i>Representation</i>	
Use Case	UCT2: Search Resources

Table 8.32: System Model 4. Search Resources

8.4.3 Browse Resources System Models

The system model templates presented in this subsection correspond to the use case described in Section 7.3.

Table 8.33 shows the system model template associated to the use case *Browse Resources* when the browsed resources consist of a set of annotated resources. The figure represents an application that browses resources annotated according to the *Annotated Resources* basic template explained in Subsection 8.3.2.

System Model 5	
Name	Browse Annotated Resources
<i>Representation</i>	
Use Case	UCT3: Browse Resources

Table 8.33: System Model 5. Browse Annotated Resources

Table 8.34 shows the system model template associated to the use case *Browse Resources* when the browsed resources consist of a set of ontological resources elements. The figure represents an application that navigates through ontological information expressed according to the *Data Sources with Schema* basic template explained in Subsection 8.3.1.

System Model 6	
Name	Browse Ontological Resources
<i>Representation</i>	
Use Case	UCT3: Browse Resources

Table 8.34: System Model 6. Browse Ontological Resources

8.4.4 Extract Information System Model

Table 8.35 shows the system model template associated to the use case *Extract Information* (described in Section 7.4). The figure represents an application that extracts meaningful information from resources annotated according to the *Annotated Documents* basic template explained in Subsection 8.3.2.

System Model 7	
Name	Extract Information
<i>Representation</i>	
Use Case	UCT4: Extract Information

Table 8.35: System Model 7. Extract Information

8.4.5 Manage Knowledge System Models

This subsection presents the system models associated to the use case described in Section 7.5.

Table 8.36 shows the system model template associated to the use case *Manage Knowledge* when editing ontology elements, instances, and alignments of resources that are organized according to one of the *Datasources with Schema* templates.

System Model 8	
Name	Edit Ontological Data Sources
<i>Representation</i>	
Use Case	UCT5: Manage Knowledge

Table 8.36: System Model 8. Edit Ontological Data Sources

Table 8.37 shows the system model template associated to the use case *Manage Knowledge* when editing annotation metadata and associated schemas that are organized according to one of the *Annotated Resources* templates.

System Model 9	
Name	Edit Annotations
<i>Representation</i>	
Use Case	UCT5: Manage Knowledge

Table 8.37: System Model 9. Edit Annotations

Table 8.38 shows the system model template associated to the use case *Manage Knowledge* when populating an ontology.

System Model 10	
Name	Populate
<i>Representation</i>	
Use Case	UCT5: Manage Knowledge

Table 8.38: System Model 10. Populate

Table 8.39 shows the system model template associated to the use case *Manage Knowledge* when learning an ontology from a document corpora.

System Model 11	
Name	Learn
<i>Representation</i>	
Use Case	UCT5: Manage Knowledge

Table 8.39: System Model 11. Learn

8.5 Examples

This section include three examples of system models showing the final result after applying some symbols, templates and models presented in the chapter.

8.5.1 Example 1

Figure 8.4 illustrates an example of the *single ontology* approach where the information integrated is either ontological or non-ontological. In the example all the resources belong to the same system where the application is executed, that is, the application consumes its own ontologies and data; hence, this example is characterized by the use of internal ontologies and data sources (characteristics 5 and 16).

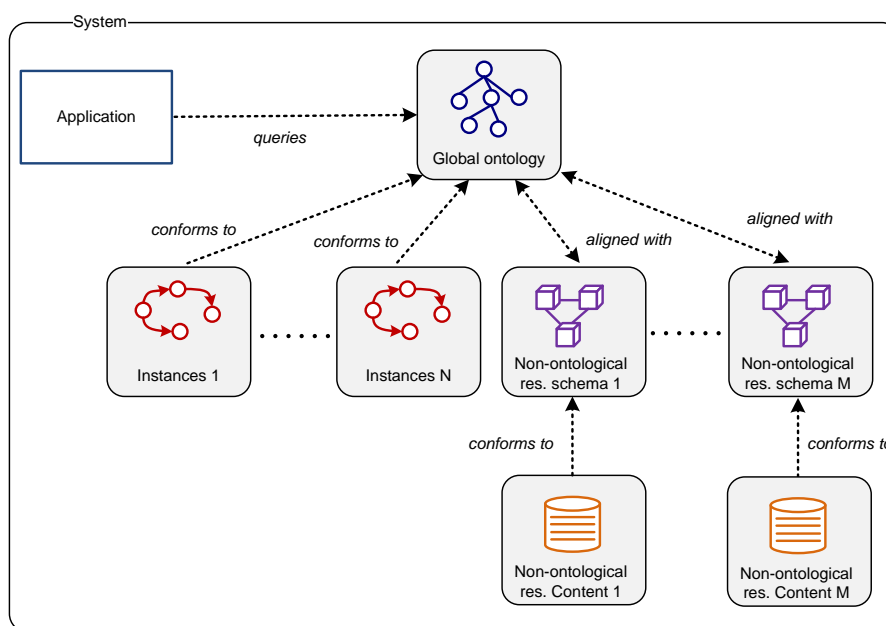


Figure 8.4: *Obtain Information* use case with an example of the single ontology approach

8.5.2 Example 2

Figure 8.5 illustrates an example of the *multiple ontology* approach where the information integrated is either ontological or non-ontological. In this example, the resources to be integrated belong to various systems different to the system where the application is executed, although different ontologies have to be created for integrating the non-ontological resources. Also, in the example, the application consumes own and foreign ontologies and data; hence this example is characterized by dealing with internal and external ontologies and external data sources (characteristics 5 and 16).

8.5.3 Example 3

Figure 8.6 illustrates an example of the *hybrid ontology* approach where the information integrated is either ontological or non-ontological. In this example, two resources belong to the system in which the application is executed. The resources have either ontological or non-ontological nature. Also, in the example, different ontological resources available in an external system are integrated. These ontologies and instances are not

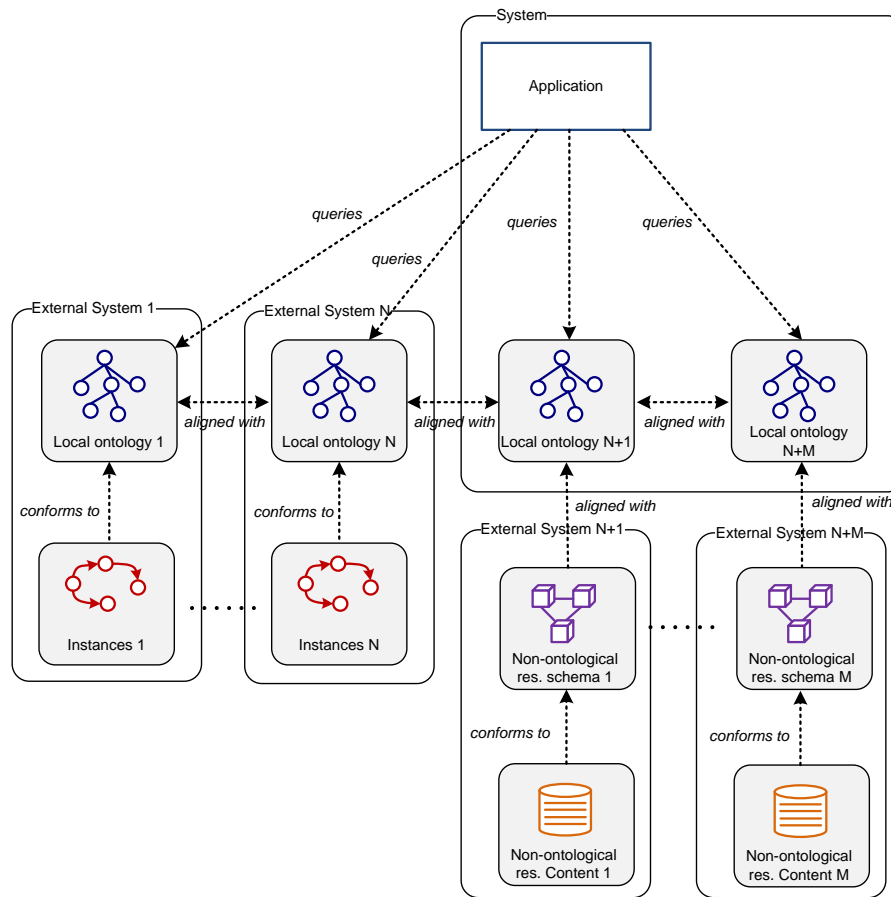


Figure 8.5: *Obtain Information* use case with an example of the multiple ontology approach

known in design-time, so symbols that represent dynamic resources (see Section 8.1.2) are used to model this situation. In the example, the application consumes own and foreign ontologies and data, thus this example is characterized by dealing with internal and external ontologies and data sources (characteristics 5 and 16). This example is also characterized by selecting ontologies and data either at design-time and at run-time (characteristics 2 and 20).

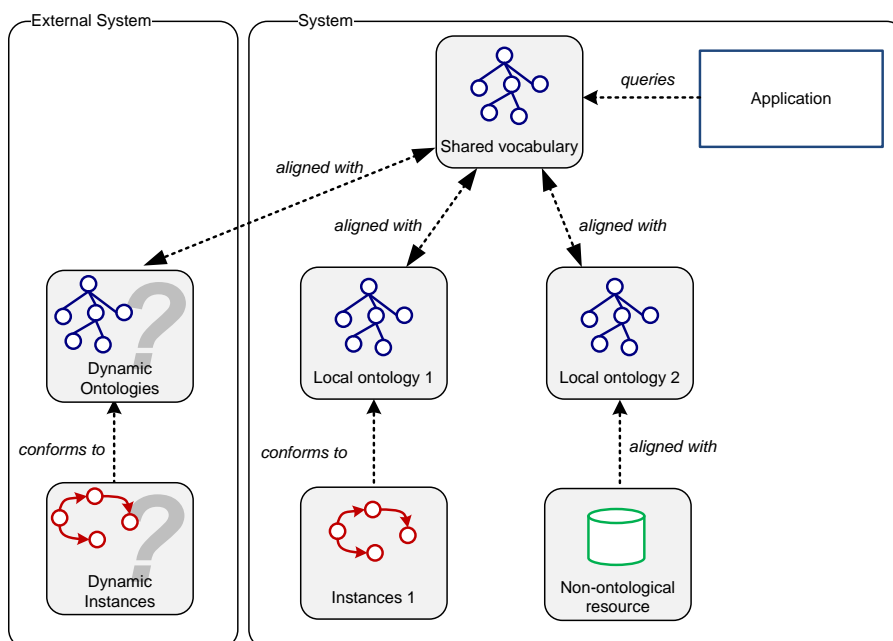


Figure 8.6: *Obtain Information* use case with an example of the hybrid ontology approach

Chapter 9

Architectural Patterns

This chapter describes the architectural patterns associated to the system models presented in the previous chapter. The components involved in the patterns presented in this chapter are those components defined in the Semantic Web Framework (see Section 3.3) plus other component, the *Non-ontological Resource Discovery and Ranking* component, which is not catalogued in the SWF because of its non-ontological nature.

Each of the patterns presented in this section can be selected taking into account the basic symbols, relationships and templates present in the system models.

The chapter is structured as follows:

- Section 9.1 describes the interfaces provided and required by the components of the SWF.
- Section 9.2 presents other components that will be used in the patterns and that are not described in the SWF.
- Section 9.3 describes the components involved in the architecture of an application and associated to the basic symbols in system models.
- Section 9.4 enumerates the components involved in the architecture of an application and associated to the relationships between symbols in system models.
- Section 9.5 describes the architectural patterns associated to the basic system model templates.
- Section 9.6 describes the architectural patterns associated to the system models.

9.1 Semantic Web Framework Component Interfaces Description

In order to facilitate the composition of the components their interfaces have to be explicitly named so a component that requires an interface must be attached to a component that provides the same interface. Currently the SWF enumerates the functionalities that each of the components provides as well as the dependencies between components. These functionalities describe the behavior and responsibilities of the components and the interface that the components provide to other components or consume from them. However the SWF does not explicitly name the interfaces that the components provide or require.

This section extends the definition of the components of the SWF that will be used in the presented architectural patterns by defining the interfaces that those components provide. The component descriptions are organized according to the SWF dimension to which they belong.

The components in the architectural patterns will be represented according to the UML 2.0 Component Diagram specification¹.

¹<http://www.uml.org>

Figure 9.1 shows the symbol used for representing components. As shown in the figure, these components will be annotated according to an UML stereotype that represents the SWF dimension to which the component belongs to. Table 9.1 shows the stereotypes used for identifying the SWF dimensions.



Figure 9.1: Semantic Web Framework component representation

Stereotype	Dimension
<i>SWF_DMM</i>	Data and metadata management
<i>SWF_QER</i>	Query and reasoning
<i>SWF_OEN</i>	Ontology Engineering
<i>SWF_OCU</i>	Ontology Customization
<i>SWF_OEV</i>	Ontology Evolution
<i>SWF_OIG</i>	Ontology Instance Generation
<i>SWF_SWS</i>	Semantic Web Services

Table 9.1: Stereotypes used for describing the SWF dimensions

9.1.1 Data and Metadata Management

Information Directory Manager

According to its definition in the SWF, the *Information Directory Manager* component provides functionalities for handling the distribution of a given query among the data providers, managing a content provider directory, identifying relevant information providers from a query and identifying provider self-descriptions, and handling the storage of and access to distributed ontologies and data, independent of the particular representation formalism [GMG⁺07].

Table 9.2 shows the representation of the *Information Directory Manager* component as well as its provided and required interfaces.

Information Directory Manager component	
<i>UML Diagram</i>	
<i>Interfaces Provided</i>	
<i>InfoAccess</i>	Acronym of <i>Information Access</i> . This interface supplies a unique mechanism for accessing different repositories that contain ontological and non ontological information. It supplies a layer for distributed query answering
<i>InformationMng</i>	Acronym of <i>Information Management</i> . This interface allows management (edition) of ontologies and ontology instances.
<i>Interfaces Required</i>	
<i>OntAccess</i>	Acronym of <i>Ontologies Access</i> . This interface is required to get query access to the local descriptions of ontologies and instances.
<i>DataAccess</i>	This interface is required to get access to the local data and annotated data sources.
<i>AlignmentAccess</i>	This interface is required to get access to the local alignments.
<i>MetadataAccess</i>	This interface is required to get access to the local metadata repositories.
<i>OntAndInstMng</i>	Acronym of <i>Ontologies and Instances Management</i> . This interface is required to provide access to local ontologies and ontology instances for normal management functions.
<i>DataManagement</i>	This interface is required to provide access to local data and annotated data for normal management functions.
<i>AlignmentMng</i>	Acronym of <i>Alignment Management</i> . This interface is required to provide access to local alignments for normal management functions.
<i>MetadataMng</i>	This interface is required to provide access to local metadata repositories for normal management functions.
<i>DataTranslator</i>	This interface is required to perform data translation.
<i>Matcher</i>	This interface is required to perform alignments between ontologies.

Table 9.2: Interfaces provided and required of the *Information Directory Manager* component

Ontology Repository

According to its definition in the SWF, the *Ontology Repository* component provides functionalities to store and access ontologies and ontology instances locally. Optionally, the ontology repository can be distributed and, therefore, it will provide transparent access to ontologies and ontology instances logically and physically distributed [GMG⁺07].

Table 9.3 shows the representation of the *Ontology Repository* component as well as its interfaces provided and required.

Ontology Repository component	
UML Diagram	
Interfaces Provided	
<i>OntAccess</i>	Acronym for <i>Ontologies Access</i> . This interface provides a defined protocol to provide query access to ontologies and instances and supports standard ontology query languages.
<i>OntAndInstMng</i>	Acronym for <i>Ontologies and Instances Management</i> . This interface allows access management to ontologies and ontology instances.
Interfaces Required	
<i>InfoAccess</i>	Acronym for <i>Information Access</i> . This interface is required to allow query access to ontologies and ontology instances located in other repositories or nodes through an <i>Information Directory Manager</i> . It can be required to provide fault tolerance mechanisms by ensuring query access to the system, regardless of whether any node of the distributed repository is temporarily unavailable.
<i>InformationMng</i>	Acronym for <i>Information Management</i> . This interface is required to provide access management to ontologies and ontology instances located in other repositories or nodes through an <i>Information Directory Manager</i> . It can be required to manage change propagation automatically when an ontology is updated, to provide high availability of ontologies and ontology instances by automatically distributing replicas, or to provide fault tolerance mechanisms by ensuring access management to the system regardless of whether any node of the distributed repository is temporarily unavailable.

Table 9.3: Interfaces provided and required of the *Ontology Repository* component

Data Repository

According to its definition in the SWF, the *Data Repository* component provides functionalities to locally store and access any type of data (text, images, etc.) and ontology annotated data locally. Optionally, the data repository can be distributed and, therefore, it will provide transparent access to data and annotated data logically and physically distributed [GMG⁺07].

Table 9.4 shows the representation of the *Data Repository* component as well as its interfaces provided and

required.

Data Repository component	
<i>UML Diagram</i>	
<i>Interfaces Provided</i>	
<i>DataAccess</i>	This interface provides a defined protocol to access data resources.
<i>DataManagement</i>	This interface allows the management of data resources.
<i>Interfaces Required</i>	
<i>InfoAccess</i>	Acronym for <i>Information Access</i> . This interface is required to allow access to data and data annotation resources located in other repositories or nodes through an <i>Information Directory Manager</i> . It can be required to provide fault tolerance mechanisms by ensuring access to the system regardless of whether any node of the distributed repository is temporarily unavailable.
<i>InformantionMng</i>	Acronym for <i>Information Management</i> . This interface is required to provide management of data and data annotations located in other repositories through an <i>Information Directory Manager</i> . It can be required to provide high availability of data and data annotations by automatically distributing replicas, or to provide fault tolerance mechanisms by ensuring access management to the system regardless of whether any node of the distributed repository is temporarily unavailable.

Table 9.4: Interfaces provided and required of the *Data Repository* component

Alignment Repository

According to its definition in the SWF, the *Alignment Repository* component provides functionalities for handling the storage of and access to distributed alignments [GMG⁺07].

Table 9.5 shows the representation of the *Alignment Repository* component as well as its interfaces provided and required.

Alignment Repository component	
<i>UML Diagram</i>	
<i>Interfaces Provided</i>	
<i>AlignmentAccess</i>	This interface provides access, via protocol, to different alignments.
<i>AlignmentMng</i>	Acronym for <i>Alignment Management</i> . This interface allows the different alignments to be published and unpublished into the repository.

Table 9.5: Interfaces provided and required of the *Alignment Repository* component

Metadata Registry

According to its definition in the SWF, the *Metadata Registry* component provides functionalities to store and access any metadata information (ontology metadata) locally. Optionally, the registry can be distributed and, therefore, it will provide transparent access to metadata information logically and physically distributed [GMG⁺07].

Table 9.6 shows the representation of the *Metadata Registry* component as well as its interfaces provided and required.

Metadata Registry component	
UML Diagram	
Interfaces Provided	
<i>MetadataAccess</i>	This interface provides a defined protocol to access metadata information.
<i>MetadataMng</i>	This interface allows the management of metadata information.
Interfaces Required	
<i>InfoAccess</i>	Acronym for <i>Information Access</i> . This interface is required to allow access to metadata information located in other registries or nodes through an <i>Information Directory Manager</i> . It can be required to provide fault tolerance mechanisms by ensuring access to the system regardless of whether any node of the distributed registry is temporarily unavailable.
<i>InformantionMng</i>	Acronym for <i>Information Management</i> . This interface is required to provide access management to metadata information located in other registries through an <i>Information Directory Manager</i> . It can be required to manage change propagation automatically when a metadata element is updated (i.e. to ensure consistency of dependent artifacts e.g. related ontologies.).

Table 9.6: Interfaces provided and required of the *Metadata Registry* component

9.1.2 Querying and Reasoning

Query Answering

According to its definition in the SWF, the *Query Answering* component takes care of all the issues related to the logical processing of a query by providing reasoning functionalities to search results in a knowledge base [GMG⁺07].

Table 9.7 shows the representation of the *Query Answering* component as well as its interfaces provided and required.

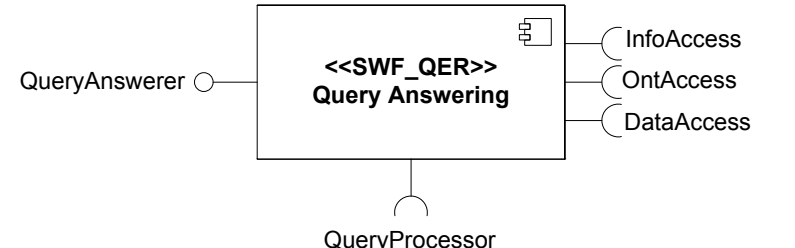
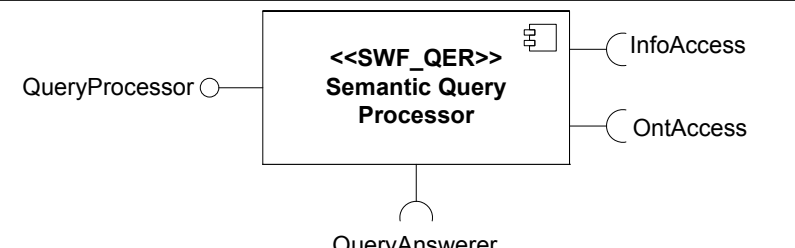
Query Answering component	
<i>UML Diagram</i>	
	
<i>Interfaces Provided</i>	
<i>QueryAnswerer</i>	This interface provides the logical processing of a given query.
<i>Interfaces Required</i>	
<i>QueryProcessor</i>	This interface is required to perform the physical processing of a query.
<i>InfoAccess</i>	This interface is required as a mediator for accessing various repositories.
<i>OntAccess</i>	This interface is required to get access to local ontologies and instances.
<i>DataAccess</i>	This interface is required to get access to the local data and annotated data sources.

Table 9.7: Interfaces provided and required of the *Query Answering* component

Semantic Query Processor

According to its definition in the SWF, the *Semantic Query Processor* component takes care of all the issues related to the physical processing of a query by providing reasoning functionalities to manage query answering over ontologies in distributed sources. This involves (among other functions) translating queries and their results from one ontology to another. It also involves merging results from different information sources into a consistent unified result which can be presented to the end user [GMG⁺07].

Table 9.7 shows the representation of the *Query Answering* component as well as its interfaces provided and required.

Semantic Query Processor component	
<i>UML Diagram</i>	
	
<i>Interfaces Provided</i>	
<i>QueryProcessor</i>	This interface provides the physical processing of a given query.

(continues on next page)

(comes from previous page)

<i>Interfaces Required</i>	
<i>QueryAnswerer</i>	This interface is required to translate queries to the ontologies used by distributed resources, to initiate query execution in remote sources and to obtain query results.
<i>InfoAccess</i>	This interface is required to answer queries over ontologies and instances in distributed sources.
<i>OntAccess</i>	This interface is required to get access to local ontologies and instances.

Table 9.8: Interfaces provided and required of the *Semantic Query Processor* component

Semantic Query Editor

According to its definition in the SWF, the *Semantic Query Editor* component takes care of all the issues related to interfacing with the user, and accomplish this by supporting user in formulating a query [GMG⁺07]. Table 9.9 shows the representation of the *Semantic Query Editor* component as well as its interfaces provided and required.

Semantic Query Editor component	
<i>UML Diagram</i>	
<pre> classDiagram class SemanticQueryEditor["<<SWF_QER>> Semantic Query Editor"] SemanticQueryEditor -- QueryEditor SemanticQueryEditor .. QueryProcessor </pre>	
<i>Interfaces Provided</i>	
<i>QueryEditor</i>	This interface provides access to all the issues related to the user interface.
<i>Interfaces Required</i>	
<i>QueryProcessor</i>	This interface is required to translate queries and their results from the user-friendly format to other formats and back again.

Table 9.9: Interfaces provided and required of the *Semantic Query Editor* component

9.1.3 Ontology Engineering

Ontology Editor

According to its definition in the SWF, the *Ontology Editor* component provides functionalities to create and modify ontologies, ontology elements, and ontology documentation [GMG⁺07].

Table 9.10 shows the representation of the *Ontology Editor* component as well as its interfaces provided and required.

Ontology Editor component	
<i>UML Diagram</i>	
<i>Interfaces Provided</i>	
<i>OntEditor</i>	This interface provides the functionalities to edit the ontology elements.
<i>Interfaces Required</i>	
<i>OntAccess</i>	This interface is required to get access to ontologies and ontology elements.
<i>OntAndInstMng</i>	This interface is required to store ontologies and ontology elements.
<i>QueryProcessor</i>	This interface is required to check if an ontology is satisfiable after performing changes.
<i>OntBrowser</i>	This interface is required to navigate through an ontology to insert, modify, or document its elements.

Table 9.10: Provided and required interfaces of the *Ontology Editor* component

Ontology Browser

According to its definition in the SWF, the *Ontology Browser* component provides functionalities to visually browse an ontology [GMG⁺07].

Table 9.11 shows the representation of the *Ontology Browser* component as well as its interfaces provided and required.

Ontology Browser component	
<i>UML Diagram</i>	
<i>Interfaces Provided</i>	
<i>OntBrowser</i>	This interface provides functionalities to visually browse an ontology.
<i>Interfaces Required</i>	
<i>OntAccess</i>	This interface is required to get access to ontologies.
<i>ViewCustomizer</i>	This interface is required to visualize the ontology to be browsed in a customized fashion.

Table 9.11: Interfaces provided and required of the *Ontology Browser* component

Ontology Matcher

According to its definition in the SWF, the *Ontology Matcher* component provides functionalities to match two ontologies and output some alignments. We can distinguish two main types of such components: those that provide only matching and those that directly use matching for processing another task (merging, mediating, etc.) [GMG⁺07].

Table 9.12 shows the representation of the *Ontology Matcher* component as well as its interfaces provided and required.

Ontology Matcher component	
<i>UML Diagram</i>	
<i>Interfaces Provided</i>	
<i>Matcher</i>	This interface provides an alignment (set of correspondences) from a list of two or more ontologies.
<i>AlignmentAccess</i>	This interface provides access, via protocol, to different alignments.
<i>AlignmentMng</i>	Acronym for <i>Alignment Management</i> . This interface allows the different alignments to be published and unpublished into the repository.
<i>AlignEditor</i>	Acronym for <i>Alignment Editor</i> . This interface provides the graphic representation and manipulation of the alignments.
<i>Transformer</i>	This interface is used for generating an ontology in a given language from one ontology written in another particular ontology language.
<i>Merger</i>	This interface is used for generating an ontology that contains the entities of two aligned ontologies.
<i>DataTranslator</i>	This interface is used for performing the translation of data according to an alignment between a source ontology or data source with regard to which the data is expressed and a target ontology to which it is translated.
<i>Mediator</i>	This interface is used to transform queries from one ontology or a data source into another ontology according to an alignment between them, and to transform the answers to the query with respect to the same alignment.

(continues on next page)

(comes from previous page)

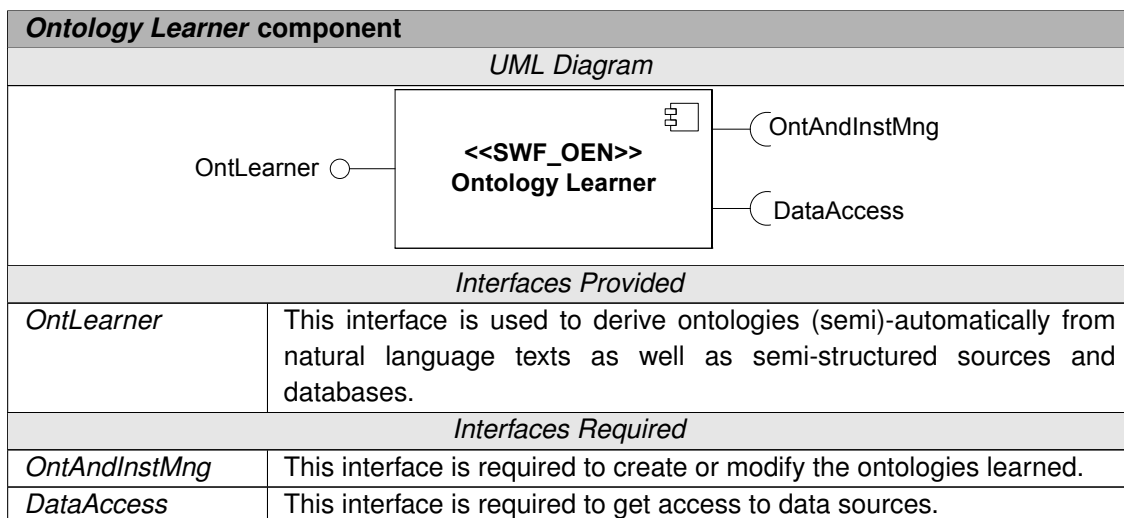
<i>Interfaces Required</i>	
<i>OntAccess</i>	This interface is required to get access to ontologies and instances.
<i>DataAccess</i>	This interface is required to get access to data and data annotations.
<i>AlignmentsAccess</i>	This interface is required to get access to alignments stored in other components.
<i>QueryProcessor</i>	This interface is required to perform various kinds of queries (containment, answering, consistency, etc.). This is particularly useful for providing semantic matching.
<i>ViewCustomizer</i>	This interface is required to perform alignments between views of ontologies.
<i>OntEditor</i>	This interface is required to perform the visual editing of alignments.

Table 9.12: Interfaces provided and required of the *Ontology Matcher* component

Ontology Learner

According to its definition in the SWF, the *Ontology Learner* component provides functionalities to acquire knowledge and generate ontologies of a given domain through some kind of (semi)-automatic process [GMG⁺07].

Table 9.13 shows the representation of the *Ontology Learner* component as well as its interfaces provided and required.

Table 9.13: Interfaces provided and required of the *Ontology Learner* component

Ontology Evaluator

According to its definition in the SWF, the *Ontology Evaluator* component provides functionalities to evaluate ontologies, either their formal model or their content, in the different phases of the ontology life cycle [GMG⁺07].

Table 9.14 shows the representation of the *Ontology Evaluator* component as well as its interfaces provided

and required.

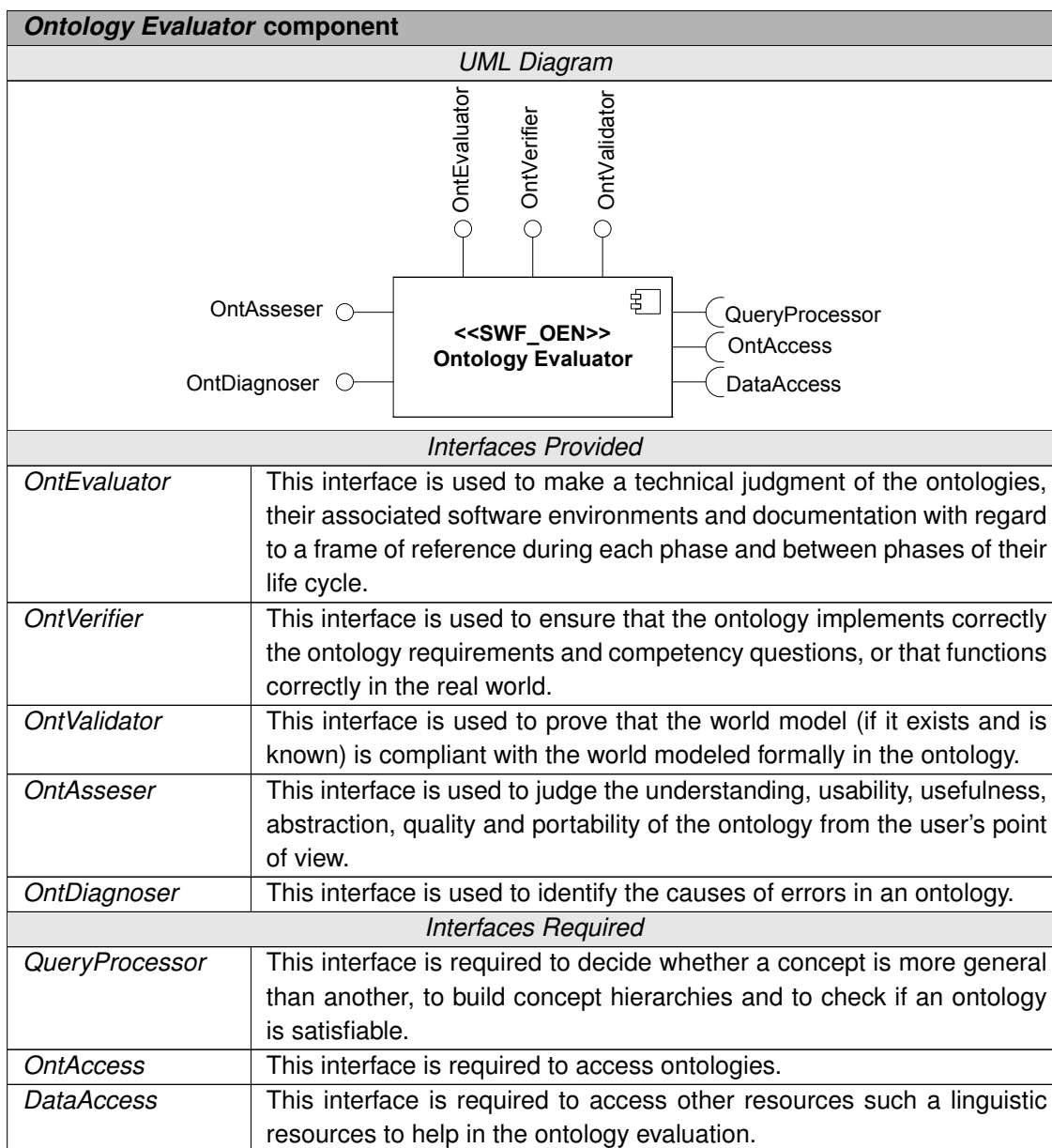


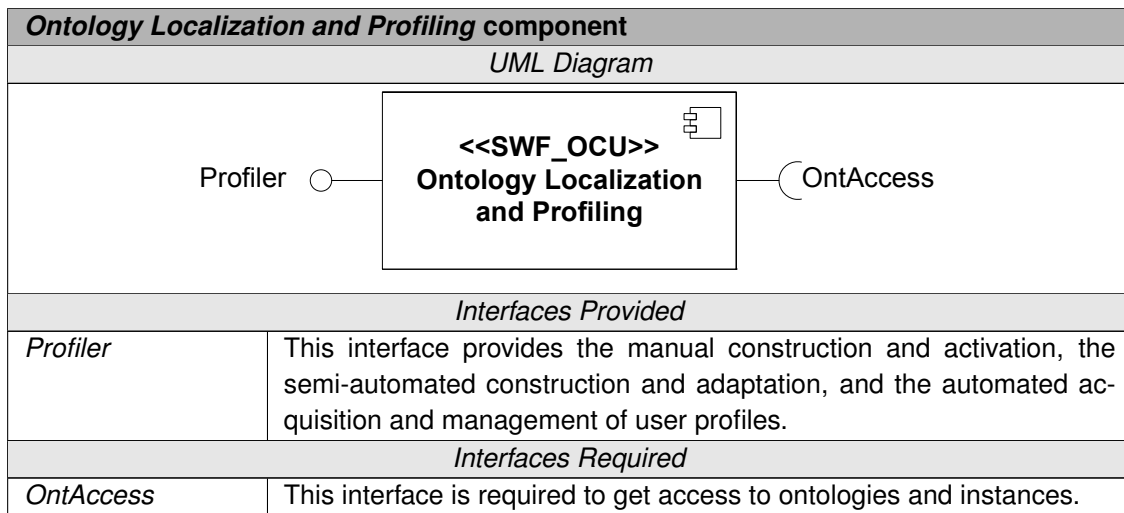
Table 9.14: Interfaces provided and required of the *Ontology Evaluator* component

9.1.4 Ontology Customization

Ontology Localization and Profiling

According to the definition in the SWF, the *Ontology Localization and Profiling* component is in charge of providing functionalities that adapt an ontology according to some context (e.g. communal or individual preferences, language, expertise, etc.) or some user profile [GMG⁺07].

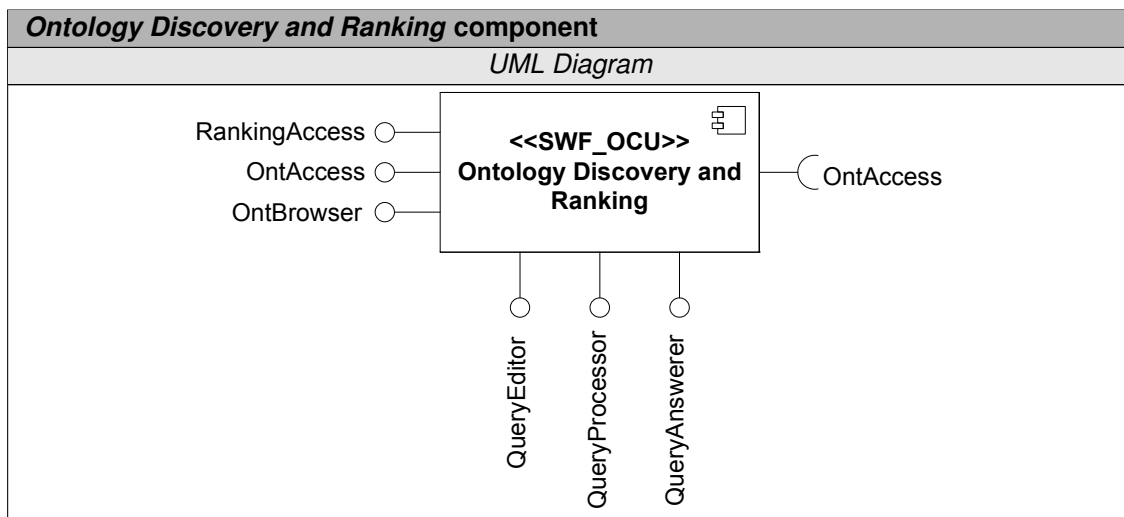
Table 9.15 shows the representation of the *Ontology Localization and Profiling* component as well as its interfaces provided and required.

Table 9.15: Interfaces provided and required of the *Ontology Localization and Profiling* component

Ontology Discovery and Ranking

According to its definition in the SWF, the *Ontology Discovery and Ranking* component is in charge of providing functionalities that find appropriate views, versions or sub-sets of ontologies, and then to rank them according to some criterion[GMG⁺07].

Table 9.16 shows the representation of the *Ontology Discovery and Ranking* component as well as its interfaces provided and required.



(continues on next page)

(comes from previous page)

<i>Interfaces Provided</i>	
<i>RankingAccess</i>	This interface provides a range of calculated quality measures about crawled ontologies.
<i>OntAccess</i>	This interface provides access to the ontologies discovered.
<i>OntBrowser</i>	This interface provides exploratory navigation through the ontologies discovered.
<i>QueryEditor</i>	This interface provides user-level (human-centric) queries (e.g. web forms).
<i>QueryProcessor</i>	This interface provides keyword/term level queries.
<i>QueryAnswerer</i>	This interface provides machine-level (content-centric) queries (e.g. SPARQL).
<i>Interfaces Required</i>	
<i>OntAccess</i>	This interface is required to get access to crawled ontologies and instances.

Table 9.16: Interfaces provided and required of the *Ontology Discovery and Ranking* component

Ontology Adaptation Operators

According to its definition in the SWF, the *Ontology Adaptation Operators* component is in charge of applying appropriate operators to the ontology in question, the result of which is an ontology customized according to some criterion (e.g. levels of trust or group preferences) [GMG⁺07].

Table 9.17 shows the representation of the *Ontology Adaptation Operators* component as well as the interfaces provided and required.

Ontology Adaptation Operators component	
<i>UML Diagram</i>	
<i>Interfaces Provided</i>	
<i>Adapter</i>	This interface is in charge of providing a customized ontology after applying the appropriate operators.
<i>Interfaces Required</i>	
<i>Profiler</i>	This interface is required to obtain the user profiles.
<i>OntAccess</i>	This interface is required to get access to ontologies and instances.

Table 9.17: Provided and required interfaces of the *Ontology Adaptation Operators* component

Ontology View Customization

According to its definition in the SWF, the *View Customization* component is responsible for enabling the user to change or amend a view on a particular ontology to fit a particular purpose (e.g. previewing, content-based view, topography, etc.) [GMG⁺07].

Table 9.18 shows the representation of the *Ontology View Customization* component as well as the interfaces provided and required.

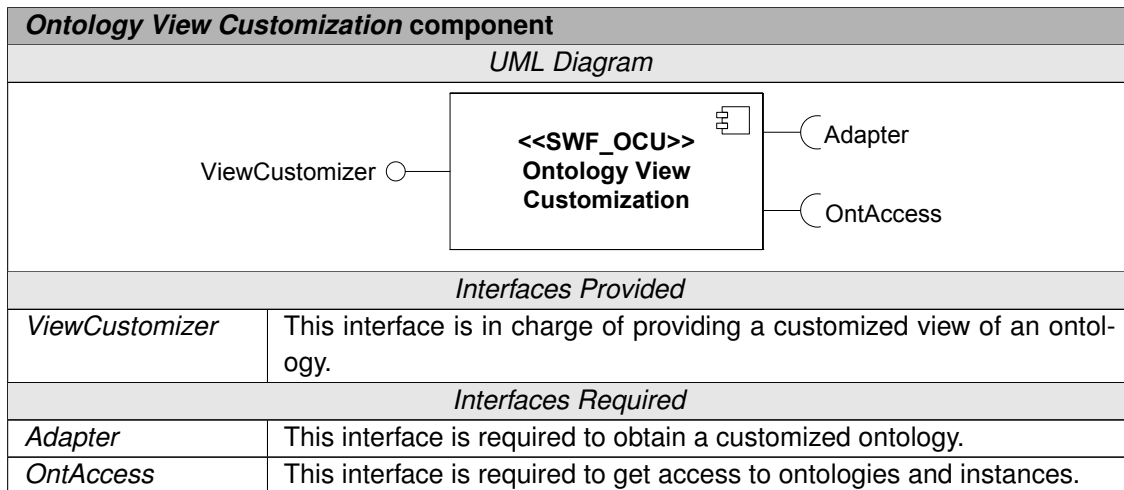


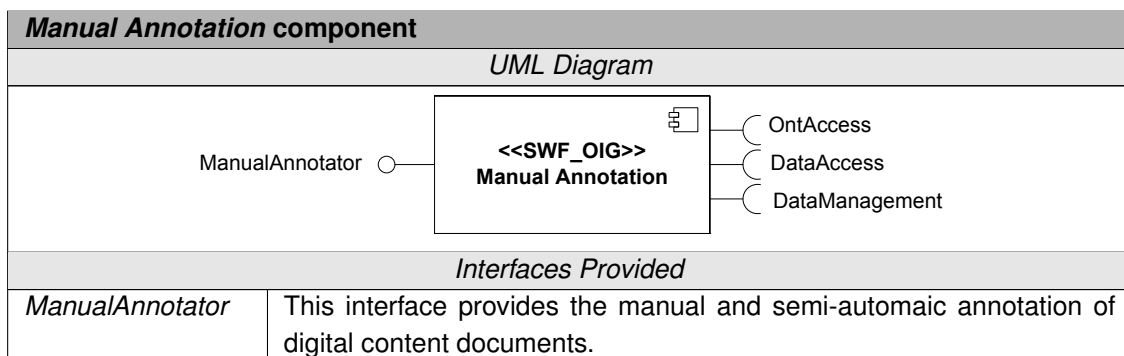
Table 9.18: Interfaces provided and required of the *Ontology View Customization* component

9.1.5 Ontology Instance Generation

Manual Annotation

According to the definition in the SWF, the *Manual Annotation* component is in charge of the manual and semi-automatic annotation of digital content documents (e.g. web pages) with concepts in the ontology. With respect to textual data, mentions of instances in the text which correspond to concepts in the ontology are annotated manually in the document. Similarly, for non-textual data (e.g. visual, audio and audiovisual sources), the concepts in the ontology, reflecting the meaning conveyed, are associated with the media content item [GMG⁺07].

Table 9.19 shows the representation of the *Manual Annotation* component as well as its interfaces provided and required.



(continues on next page)

(comes from previous page)

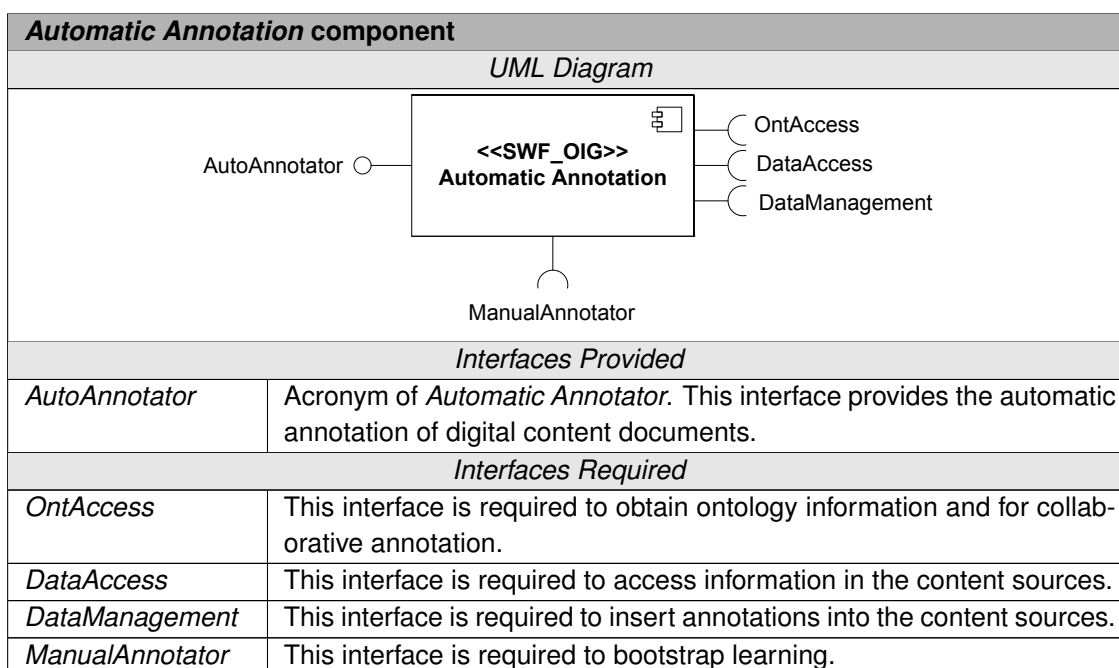
<i>Interfaces Required</i>	
<i>OntAccess</i>	This interface is required to obtain ontology information and for collaborative annotation.
<i>DataAccess</i>	This interface is required to access information in the content sources.
<i>DataManagement</i>	This interface is required to insert annotations into the content sources.

Table 9.19: Interfaces provided and required of the *Manual Annotation* component

Automatic Annotation

According to its definition in the SWF, the *Automatic Annotation* component is in charge of automatically annotating digital content documents (e.g. web pages) with concepts in the ontology. Occurrences in the content of instances of concepts in the ontology are automatically detected and subsequently annotated [GMG⁺07].

Table 9.20 shows the representation of the *Automatic Annotation* component as well as its interfaces provided and required.

Table 9.20: Interfaces provided and required of the *Automatic Annotation* component

Ontology Populator

According to its definition in the SWF, the *Ontology Populator* component is in charge of providing functionalities to automatically generate new instances in a given ontology from a data source. It links unique occurrences of instances in the content to instances of concepts in the ontology. Compared to the *Manual Annotation* component, this component will only create one instance in the ontology for a given occurrence, no matter how many times this is mentioned in the text. Compared to the *Automatic Annotation* component,

this component does not only disambiguate instances but also identifies co-referring instances [GMG⁺07]. Table 9.21 shows the representation of the *Ontology Populator* component as well as the interfaces provided and required.

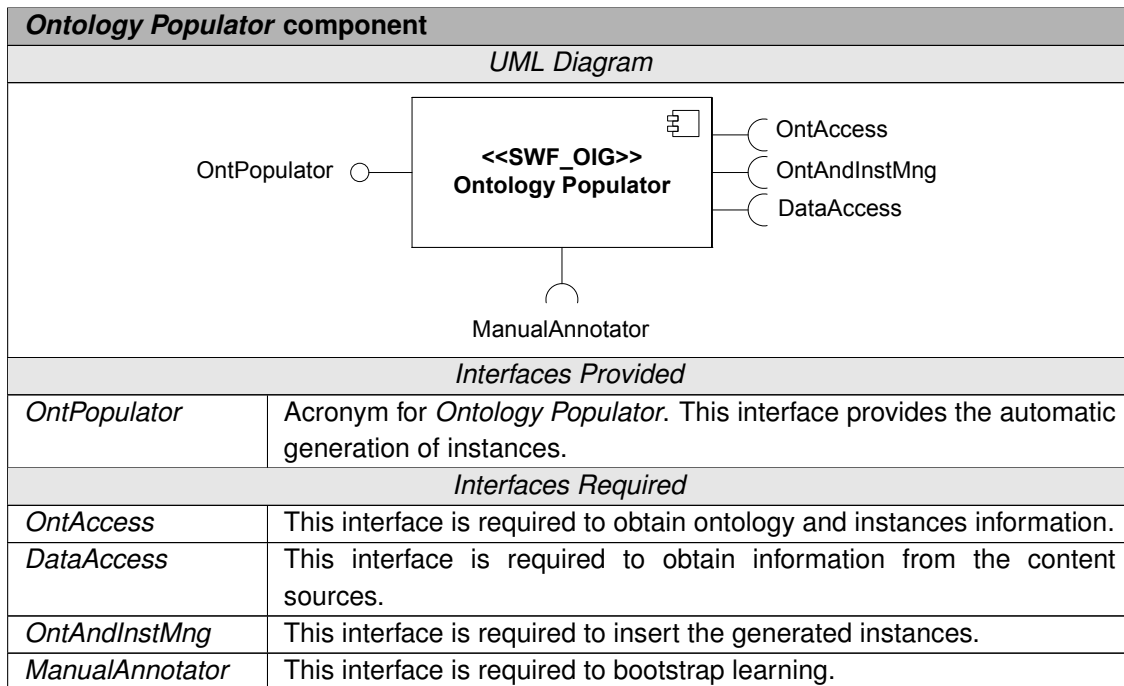


Table 9.21: Provided and required interfaces of the *Ontology Populator* component

9.2 Components Not Defined in the SWF

This subsection presents other components that will also be used in the patterns not described in the SWF.

9.2.1 Non-ontological Resource Discovery and Ranking

The *Non-ontological Resource Discovery and Ranking* component is in charge of providing functionalities to find appropriate non-ontological resources, and then to rank them according to some criterion. This component is not cataloged in the SWF because of its non-semantic nature.

Examples of the implementations of this component are Google² or Yahoo!³.

Table 9.22 shows the representation of the *Non-ontological Resource Discovery and Ranking* component as well as its interfaces provided and required.

²<http://www.google.com>

³<http://www.yahoo.com>

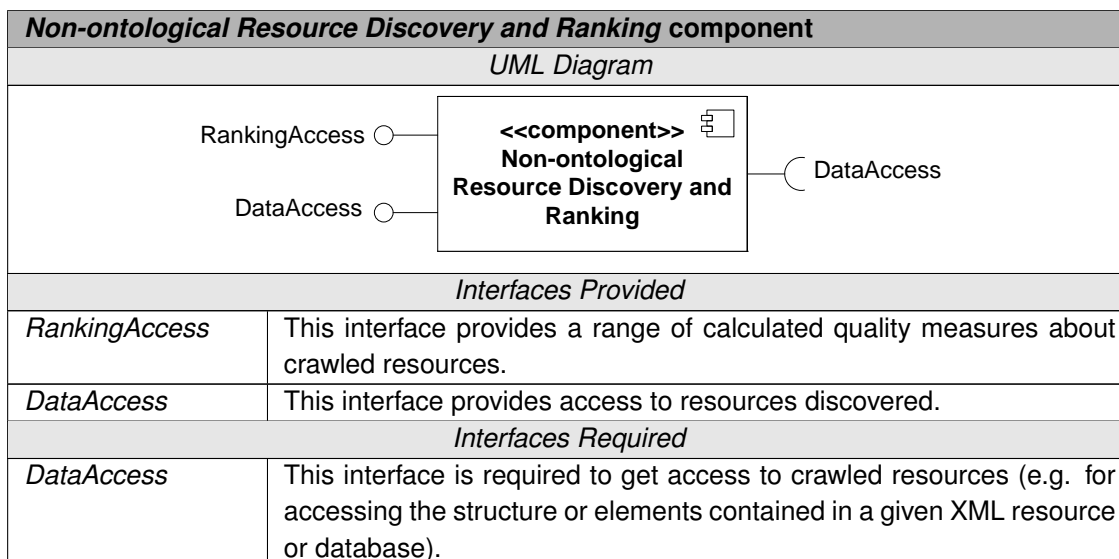


Table 9.22: Interfaces provided and required of the *Non-ontological Resource Discovery and Ranking* component

9.3 Components associated to Basic System Models Symbols

This section illustrates the components involved in the architecture of a large-scale semantic application and associated to the basic symbols in system models.

The differentiation between the types of resources has been explained in Subsection 8.1.1.

In the case of applications symbol, a number of components that make use of the interfaces provided by the patterns described in this chapter will be included in the architecture depending on their needs. At list of the System Dialogs and Facades described in Subsection 11.1.1 should also be included.

The presence of system limits can introduce different implementations of the same component in the architecture such, for example, data and ontology repositories, when the resources are stored in different repositories and each of them is comprised in a system limit.

9.3.1 Components Associated to Resources

This subsection describes the components involved in the architecture of a semantic application and associated to the basic symbols that represent resources in system models.

Components Associated to Ontological Resources

The *Ontology Repository* component will be used in the application whenever an ontological resource is present in a system model.

Table 9.23 shows the representation of the *Ontological Resource Access and Management* pattern.

Pattern 1	
Name	<i>Ontological Resource Access and Management</i>
<i>UML Diagram</i>	
Basic Symbols Associated	1 and 2.
Selection Criterion	Presence of a static ontological resources symbol.
<i>Roles of the Components</i>	
Ontology Repository	Provides access, management and storage of the ontology or the instances.
<i>Roles of the Interfaces</i>	
OntAccess	Provides access to the ontology or the instances.
OntAndInstMng	Provides management of the ontology or the instances.

Table 9.23: *Ontological Resource Access and Management*

Components Associated to Non-ontological Resources

When a non-ontological resource is present in a system model, the component involved in the application architecture is the *Data Repository* component, which provides functionalities to store and access any type of data (structured or non-structured).

Table 9.24 shows the representation of the *Non-ontological Resource Access and Management* pattern.

Pattern 2	
Name	<i>Non-ontological Resource Access and Management</i>
<i>UML Diagram</i>	
Basic Symbols Associated	3, 4 and 5.
Selection Criterion	Presence of a static non-ontological resource symbol.
<i>Roles of the Components</i>	
Data Repository	Provides access, management and storage of the non-ontological resource.
<i>Roles of the Interfaces</i>	
DataAccess	Provides access to the non-ontological resource.
DataManagement	Provides management of the non-ontological resource.

Table 9.24: *Non-ontological Resource Access and Management*

Components Associated to Dynamic Resources

In the case that dynamic resources is present within a given system model, then a component used for the discovery of such dynamic resources have to be used. These dynamic resources can be ontological or non-ontological.

1. When dynamic ontological resources are present in a system model, then the *Ontology Discovery and Ranking* component has to be included in the application architecture.

The architecture associated to dynamic ontological resources access can take two possible forms. In the pattern shown in Table 9.25, the *Ontology Discovery and Ranking* component is directly connected to the *Ontology Repository* component. As shown in the figure, the *Ontology Discovery and Ranking* component depends on the *Ontology Repository* component to access ontologies. In the second approach, shown in Table 9.26, a set of *Ontology Discovery and Ranking* components are federated in a graph to provide access to the ontologies and instances discovered. In this case, the outer nodes of the graph must be *Ontology Repository* components.

Pattern 3	
Name	<i>Dynamic Ontological Resource Access</i>
UML Diagram	
Basic Symbols Associated	6 and 7.
Selection Criterion	Presence of a dynamic ontological resource symbol.
Roles of the Components	
<i>Ontology Discovery and Ranking</i>	Permits discovering and ranking dynamic ontology or dynamic instances.
<i>Ontology Repository</i>	Provides the access to and storage of the ontology or the instances.
Roles of the Interfaces	
<i>OntAccess</i>	Provides access to ontology or the instances.
<i>RankingAccess</i>	Provides the ranking of ontological resources discovered.

Table 9.25: *Dynamic Ontological Resource Access*

Pattern 4	
Name	<i>Dynamic Ontological Resource Access via Federated Discoverers</i>
UML Diagram	
Basic Symbols Associated	6 and 7.

(continues on next page)

(comes from previous page)

<i>Selection Criterion</i>	Presence of a dynamic ontological resource symbol.
<i>Roles of the Components</i>	
<i>Ontology Discovery and Ranking</i>	Provides the discovery and ranking of the dynamic ontology or the instances.
<i>Ontology Repository</i>	Provides the access to and storage of the ontology or instances
<i>Roles of the Interfaces</i>	
<i>OntAccess</i>	Provides access to the ontology or instances.
<i>RankingAccess</i>	Provides rankings of the ontological resources discovered.

Table 9.26: *Dynamic Ontological Resource Access via Federated Discoverers*

2. Whenever dynamic non-ontological resources are present within a system model, then an *Non-ontological Resource Discovery and Ranking* component should be included in the application architecture.

Table 9.27 shows the representation of the pattern of *Dynamic Non-ontological Resource Access*.

Pattern 5	
<i>Name</i>	<i>Dynamic Non-ontological Resource Access</i>
<i>UML Diagram</i>	
<i>Basic Symbols Associated</i>	Dynamic non-ontological resources basic symbols.
<i>Selection Criterion</i>	8, 9 and 10.
<i>Roles of the Components</i>	
<i>Non-ontological Resource Discovery and Ranking</i>	Provides the discovery and ranking of the dynamic non-ontological resource.
<i>Data Repository</i>	Provides access to and storage of the non-ontological resource.
<i>Roles of the Interfaces</i>	
<i>DataAccess</i>	Provides access to the non-ontological resource.
<i>RankingAccess</i>	Provides rankings about discovered non-ontological resources.

Table 9.27: *Dynamic Non-ontological Resource Access*

9.4 Components associated to Relationships Between Symbols in System Models

This section enumerates the components involved in the architecture of an application and associated to the relationships between symbols in system models.

9.4.1 Conforms To

Since the *Ontology Repository* component permits access to and stores ontologies and instances, the *Conforms To* relationship does not introduce more components in the architecture apart from the *Ontology Repository* in the case of ontological resources.

An analogous situation occurs with non-ontological resources. The *Data Repository* will store both the schema and content of non-ontological resources.

In the case that the ontologies or schemas are separated from the data (e.g. stored within different system limits) then multiple repositories can be included within the application architecture.

9.4.2 Aligned With

When there is an alignment between two ontologies the *Ontology Repository* component stores and provides access to them and the *Alignment Repository* component handles the storage of the mappings that compose the alignment. The *Information Directory* component provides access to and management of the mappings that compose the alignment.

Table 9.28 shows the representation of the pattern of the *Alignment between Ontologies Access and Management*.

Pattern 6	
Name	<i>Alignment between Ontologies Access and Management</i>
UML Diagram	
<pre> classDiagram class InfoAccess class InformationMng class AlignmentAccess class AlignmentMng class OntAccess class InformationDirectoryManager["<<SWF_DMM>> Information Directory Manager"] class AlignmentRepository["<<SWF_DMM>> Alignment Repository"] class OntologyRepository["<<SWF_DMM>> Ontology Repository"] InformationDirectoryManager --> InfoAccess InformationDirectoryManager --> InformationMng InformationDirectoryManager --> AlignmentAccess InformationDirectoryManager --> AlignmentMng AlignmentRepository --> AlignmentAccess AlignmentRepository --> AlignmentMng OntologyRepository --> OntAccess InformationDirectoryManager --> OntologyRepository : OntAccess InformationDirectoryManager --> AlignmentRepository : AlignmentAccess InformationDirectoryManager --> AlignmentRepository : AlignmentMng </pre>	
Relationship Associated	3 (both static).
Selection Criterion	Presence of an alignment between two static ontologies.

(continues on next page)

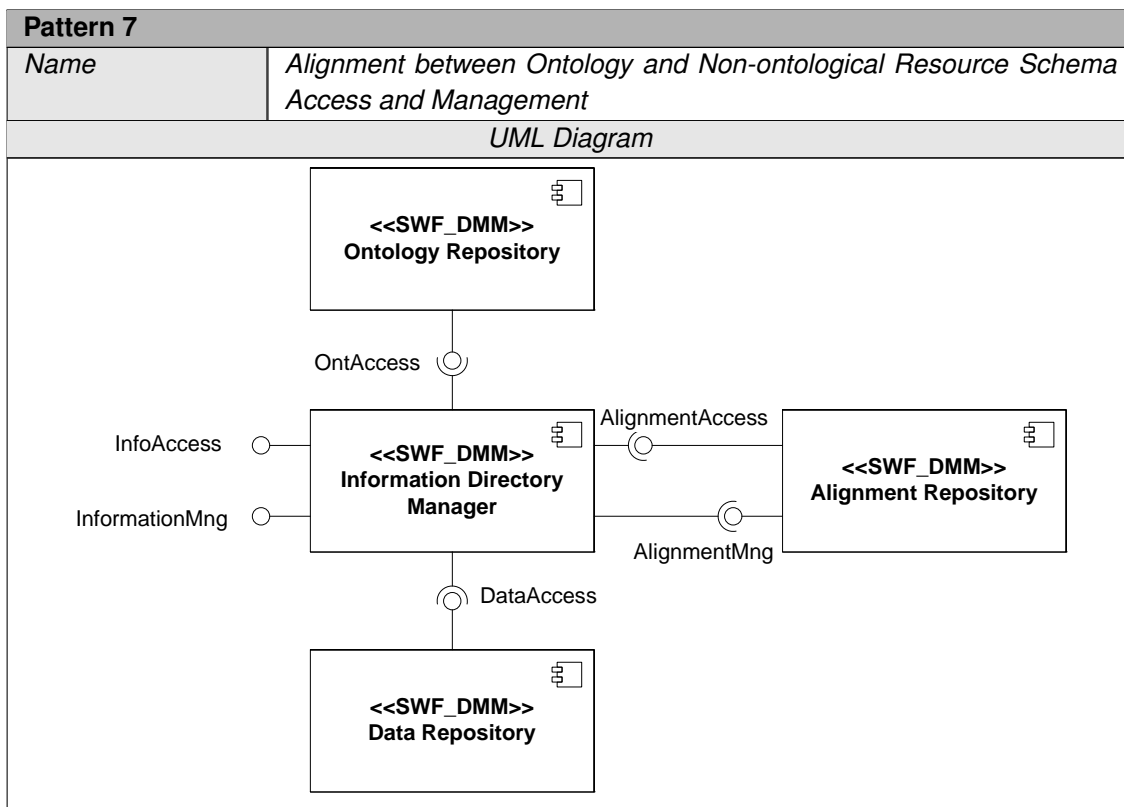
(comes from previous page)

<i>Roles of the Components</i>	
<i>Information Directory Manager</i>	Provides uniform access to ontologies and alignments and management of alignments.
<i>Ontology Repository</i>	Provides access to, management and storage of ontologies.
<i>Alignment Repository</i>	Provides access to, management and storage of the alignment.
<i>Roles of the Interfaces</i>	
<i>OntAccess</i>	Provides access to ontologies.
<i>AlignmentAccess</i>	Provides access to the alignments
<i>AlignmentMng</i>	Provides management of the alignment.
<i>InfoAccess</i>	Provides uniform access to the ontologies and the alignment.
<i>InformationMng</i>	Provides management of the alignment.

Table 9.28: *Alignment between Ontologies Access and Management*

When there is an alignment between an ontology and a non-ontological resource schema, the *Ontology Repository* component stores and permits access to the ontology, and the *Data Repository* component permits access to the data schema. The *Alignment Repository* component handles the storage of the mappings that compose the alignment. The *Information Directory* component provides access to and management of the mappings that compose the alignment.

Table 9.29 shows the representation of the pattern of *Alignment between an Ontology and a Non-ontological Resource Access and Management*.



(continues on next page)

(comes from previous page)

<i>Relationship Associated</i>	4 (both static).
<i>Selection Criterion</i>	Presence of an alignment between a static ontology and a static non-ontological resource.
<i>Roles of the Components</i>	
<i>Information Directory Manager</i>	Provides uniform access to the ontology, the non-ontological resource schema and the alignment, and to the management of the alignment.
<i>Ontology Repository</i>	Provides access to and management and storage of the ontology.
<i>Data Repository</i>	Provides access to and storage of the non-ontological resource schema.
<i>Alignment Repository</i>	Provides access to and management and storage of the alignment.
<i>Roles of the Interfaces</i>	
<i>OntAccess</i>	Provides access to the ontology.
<i>DataAccess</i>	Provides access to the non-ontological resources schema.
<i>AlignmentAccess</i>	Provides access to the alignment.
<i>AlignmentMng</i>	Provides management of alignment.
<i>InfoAccess</i>	Provides uniform access to the ontology, the non-ontological resource schema and the alignment.
<i>InformationMng</i>	Provides management of the alignment.

Table 9.29: Alignment between Ontology and Non-ontological Resource Schema Access and Management

If there is an alignment between two dynamic ontologies, the *Ontology Discovery and Ranking* component permits discovering the ontologies and the *Ontology Matcher* component handles the run-time generation of the alignment as well as the access to and management of the mappings that compose the alignment.

Table 9.30 shows the representation of the pattern of the *Alignment between Dynamic Ontology Access and Management*.

Pattern 8	
<i>Name</i>	<i>Alignment between Dynamic Ontology Access and Management</i>
<i>UML Diagram</i>	
<pre> classDiagram class OntologyMatcher["<<SWF_OEN>> Ontology Matcher"] { +Matcher -AlignmentAccess -AlignmentMng } class OntologyDiscovery["<<SWF_OCU>> Ontology Discovery and Ranking"] { +OntAccess +RankingAccess -OntAccess } OntologyMatcher -- > OntologyDiscovery : OntAccess OntologyMatcher -- > OntologyDiscovery : QueryProcessor OntologyMatcher .. > OntologyDiscovery : AlignmentAccess OntologyMatcher .. > OntologyDiscovery : AlignmentMng </pre>	
<i>Relationship Associated</i>	3 (both dynamic).

(continues on next page)

(comes from previous page)

<i>Selection Criterion</i>	Presence of an alignment between two dynamic ontologies.
<i>Roles of the Components</i>	
<i>Ontology Discovery and Ranking</i>	Provides the discovery and ranking of dynamic ontologies.
<i>Ontology Matcher</i>	Provides the automatic matching of ontologies discovered.
<i>Roles of the Interfaces</i>	
<i>Matcher</i>	Launches the matching of dynamic ontologies discovered.
<i>AlignmentAccess</i>	Provides access to the alignment.
<i>AlignmentMng</i>	Provides management of the alignment.
<i>OntAccess</i>	Provides access to ontologies.
<i>QueryProcessor</i>	Used to perform various kinds of queries (containment, answering, consistency, etc.) during semantic matching.
<i>RankingAccess</i>	Provides rankings of ontologies discovered.

Table 9.30: *Alignment between Dynamic Ontology Access and Management*

If there is an alignment between a dynamic ontology and a dynamic non-ontological resource schema, the *Ontology Discovery and Ranking* component lets to discover the ontology, the *Non-ontological Resource Discovery and Ranking* component is used to discover the schema; and the *Ontology Matcher* component handles the run-time generation of the alignment as well as the access to and management of the mappings that compose the alignment.

Table 9.31 shows the representation of the pattern of the *Alignment between a Dynamic Ontology and a Dynamic Non-ontological Resource Access and Management*.

Pattern 9	
<i>Name</i>	<i>Alignment between a Dynamic Ontology and a Dynamic Non-ontological Resource Access and Management</i>
<i>UML Diagram</i>	
<i>Relationship Associated</i>	4 (both dynamic).
<i>Selection Criterion</i>	Presence of an alignment between a dynamic ontology and a dynamic non-ontological resource schema.

(continues on next page)

(comes from previous page)

<i>Roles of the Components</i>	
<i>Ontology Discovery and Ranking</i>	Provides the discovery and ranking of the dynamic ontology.
<i>Non-ontological Resource Discovery and Ranking</i>	Provides the discovery and ranking of the dynamic non-ontological resource schema.
<i>Ontology Matcher</i>	Provides the automatic matching of the ontology discovered and the non-ontological resource schema discovered.
<i>Roles of the Interfaces</i>	
<i>DataAccess</i>	Provides access to the non-ontological resources schema.
<i>Matcher</i>	Launches the matching of the dynamic ontology discovered and the schema discovered.
<i>AlignmentAccess</i>	Provides access to the alignment.
<i>AlignmentMng</i>	Provides management of the alignment.
<i>OntAccess</i>	Provides access to the ontology.
<i>RankingAccess</i>	Provides rankings of discovered resources.

Table 9.31: *Alignment between a Dynamic Ontology and a Dynamic Non-ontological Resource Access and Management*

Heterogeneous dynamic and static ontologies and non-ontological resources schemas can be aligned through the use of the general pattern presented in tables 9.32 and 9.33.

Pattern 10	
<i>Name</i>	<i>Alignment between a Dynamic Ontology and a Static Non-ontological resource Access and Management</i>
<i>UML Diagram</i>	
<pre> classDiagram class DMM["<<SWF_DMM>> Data Repository"] class OEN["<<SWF_OEN>> Ontology Matcher"] class OCU["<<SWF_OCU>> Ontology Discovery and Ranking"] DMM --> OEN : DataAccess OEN --> OCU : OntAccess OEN --> OCU : RankingAccess OEN --> OCU : AlignmentAccess OEN --> OCU : AlignmentMng OCU --> OCU : OntAccess </pre>	
<i>Relationship Associated</i>	4 (dynamic ontology and static non-ontological resource).
<i>Selection Criterion</i>	Presence of an alignment between a dynamic ontology and a static non-ontological resource schema relationship.

(continues on next page)

(comes from previous page)

<i>Roles of the Components</i>	
<i>Ontology Discovery and Ranking</i>	Provides the discovery and ranking of the dynamic ontology.
<i>Data Repository</i>	Provides access to and storage of the static non-ontological resource schema.
<i>Ontology Matcher</i>	Provides the automatic matching of the discovered ontology and the non-ontological resource schema.
<i>Roles of the Interfaces</i>	
<i>DataAccess</i>	Provides access to the non-ontological resources schema.
<i>Matcher</i>	Launches the matching of the dynamic ontology discovered and the schema.
<i>AlignmentAccess</i>	Provides access to the alignment.
<i>AlignmentMng</i>	Provides management of the alignment.
<i>OntAccess</i>	Provides access to the ontology.
<i>RankingAccess</i>	Provides rankings about resources discovered.

Table 9.32: *Alignment between a Dynamic Ontology and a Static Non-ontological resource Access and Management*

Pattern 11	
<i>Name</i>	<i>Alignment between a Static Ontology and a Dynamic Non-ontological resource Access and Management</i>
<i>UML Diagram</i>	
<i>Relationship Associated</i>	4 (static ontology and dynamic non-ontological resource).
<i>Selection Criterion</i>	Presence of an alignment between a static ontology and a dynamic non-ontological resource schema relationship.

(continues on next page)

(comes from previous page)

Roles of the Components	
<i>Non-ontological Resource Discovery and Ranking</i>	Provides the discovery and ranking of the dynamic non-ontological resource schema.
<i>Ontology Repository</i>	Provides access to and management and storage of the ontology.
<i>Ontology Matcher</i>	Provides the automatic matching of the ontology and the non-ontological resource schema discovered.
Roles of the Interfaces	
<i>DataAccess</i>	Provides access to the non-ontological resource schema.
<i>Matcher</i>	Launches the matching of the dynamic ontology discovered and the schema.
<i>AlignmentAccess</i>	Provides access to the alignment.
<i>AlignmentMng</i>	Provides management of the alignment.
<i>OntAccess</i>	Provides access to the ontology.
<i>RankingAccess</i>	Provides rankings about resources discovered.

Table 9.33: Alignment between a Static Ontology and a Dynamic Non-ontological resource Access and Management

9.4.3 Annotate

As seen in Subsection 8.2.3 the relationship *Annotate* can represent either unstructured documents or ontologies annotated.

Next, patterns for each of these approaches are presented.

1. In the case of documents annotated by a set of instances, annotations can be stored either within the own annotated document or as physically separated instances from the document.

If annotations are stored within the source document, then the pattern shown in Table 9.34 is used. In this pattern, the *Data Repository* component is used for proving access and management of unstructured documents and annotations. This pattern is also valid for documents annotated by non-ontological metadata stored outside or inside the documents.

Pattern 12	
<i>Name</i>	<i>Annotations within documents Access and Management</i>
UML Diagram	
<i>Relationship Associated</i>	5 (both static) and 6.
<i>Selection Criterion</i>	Annotations stored within the annotated documents.

(continues on next page)

(comes from previous page)

<i>Roles of the Components</i>	
<i>Data Repository</i>	Provides access to, management and storage of the document and the annotations.
<i>Roles of the Interfaces</i>	
<i>DataAccess</i>	Provides access to the document and its annotations.
<i>DataManagement</i>	Provides management of the document and its annotations.

Table 9.34: Annotations within documents Access and Management

If annotations are stored in a separate space within instances, then the pattern shown in Table 9.35 is used. In this pattern, the *Data Repository* component is used for providing access to unstructured documents; the *Ontology Repository* component it used for providing access and management of annotations; and the *Information Directory Manager* component is used for providing access to un-structured documents and annotations and management of annotations.

Pattern 13	
<i>Name</i>	<i>Annotations outside documents Access and Management</i>
<i>UML Diagram</i>	
<pre> classDiagram class OntologyRepository["<<SWF_DMM>> Ontology Repository"] class InformationDirectoryManager["<<SWF_DMM>> Information Directory Manager"] class DataRepository["<<SWF_DMM>> Data Repository"] OntologyRepository -- InformationDirectoryManager : OntAccess OntologyRepository -- InformationDirectoryManager : OntAndInstMng InformationDirectoryManager -- DataRepository : DataAccess </pre>	
<i>Relationship Associated</i>	5 (both static).
<i>Selection Criterion</i>	Annotations stored as instances outside the annotated documents.
<i>Roles of the Components</i>	
<i>Data Repository</i>	Provides access to and management and storage of the document.
<i>Ontology Repository</i>	Provides access to and management and storage of the annotations.
<i>Information Directory Manager</i>	Provides uniform access to the document and its annotations.

(continues on next page)

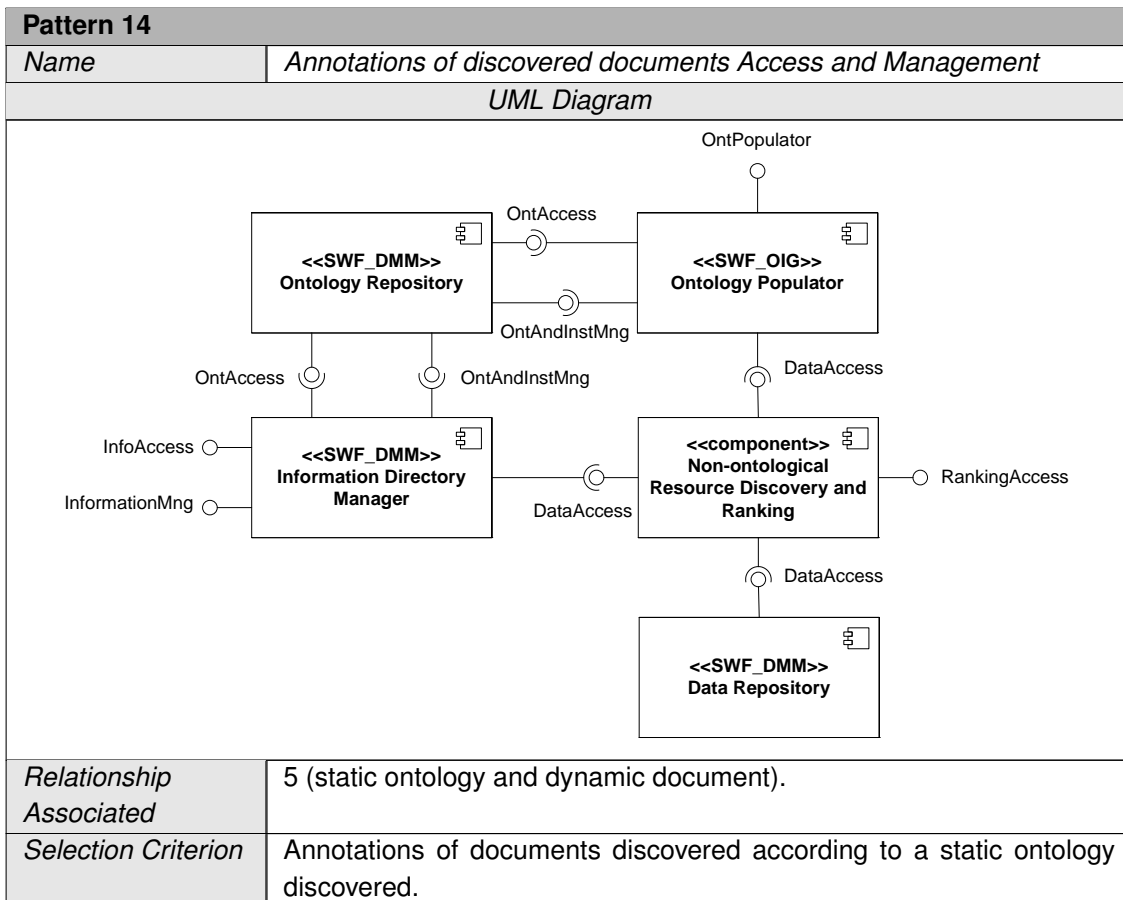
(comes from previous page)

Roles of the Interfaces	
<i>DataAccess</i>	Provides access to the document.
<i>OntAccess</i>	Provides access to the annotations.
<i>OntAndInstMng</i>	Provides management of the annotations.
<i>InfoAccess</i>	Provides uniform access to the document and its annotations.
<i>InformationMng</i>	Provides management of the annotations.

Table 9.35: Annotations outside documents Access and Management

As seen in Subsection 8.2.3, annotations can be generated for being accessed in a dynamic fashion. In this case there exists three possibilities:

- (a) To discover documents at run-time and annotate them with dynamically generated instances that conform to a given ontology selected or engineered at design-time. The pattern shown in Table 9.36 provides an architectural design for this scenario. In this pattern, the *Non-ontological Resource Discovery and Ranking* component is in charge of discovering and providing access to the annotated documents; the *Ontology Repository* component stores the local ontology as well as the generated annotations. The *Ontology Populator* component annotates at run-time the documents discovered and accesses the *Ontology Repository* component to store the annotations and to take access to the ontology. Finally the *Information Directory Manager* component provides access to and management of the annotations.



(continues on next page)

(comes from previous page)

Roles of the Components	
<i>Non-ontological Resource Discovery and Ranking</i>	Provides the discovery and ranking of the document.
<i>Data Repository</i>	Provides access to and management and storage of the document.
<i>Ontology Repository</i>	Provides access to and management and storage of the annotations.
<i>Information Directory Manager</i>	Provides uniform access to the document and its annotations.
<i>Ontology Populator</i>	Annotates at run-time the documents discovered.
Roles of the Interfaces	
<i>DataAccess</i>	Provides access to the document.
<i>RankingAccess</i>	Provides rankings of the documents discovered.
<i>OntAccess</i>	Provides access to the annotations.
<i>OntAndInstMng</i>	Provides management of the annotations.
<i>OntPopulator</i>	Launches the automatic generation of annotations.
<i>InfoAccess</i>	Provides uniform access to the document and its annotations.
<i>InformationMng</i>	Provides management of the annotations.

Table 9.36: Annotations of discovered documents Access and Management

- (b) To discover a suitable ontology at run-time for annotating a set of documents. The pattern shown in Table 9.37 provides an architectural design for this scenario. In this pattern, the *Ontology Discovery and Ranking* component is in charge of discovering and providing access to the ontology; the *Ontology Repository* component stores the generated annotations. The *Data Repository Component* stores the documents to be annotated; the *Ontology Populator* component annotates at run-time the documents according to the discovered ontology and accesses the *Ontology Repository* component to store the annotations; finally the *Information Directory Manager* component provides access to and management of the annotations.

Pattern 15	
Name	<i>Annotations of Documents according to a Discovered Ontology Access and Management</i>
UML Diagram	
Relationship Associated	5 (dynamic ontology and static document).

(continues on next page)

(comes from previous page)

<i>Selection Criterion</i>	Annotations of documents according to a run-time ontology discovered.
<i>Roles of the Components</i>	
<i>Ontology Discovery and Ranking</i>	Provides the discovery and ranking of the dynamic ontology.
<i>Data Repository</i>	Provides access to and management and storage of the document.
<i>Ontology Repository</i>	Provides access to and management and storage of the annotations.
<i>Information Directory Manager</i>	Provides uniform access to the document and its annotations.
<i>Ontology Populator</i>	Annotates at run-time the documents discovered.
<i>Roles of the Interfaces</i>	
<i>DataAccess</i>	Provides access to the document.
<i>RankingAccess</i>	Provides rankings about documents discovered.
<i>OntAccess</i>	Provides access to the annotations.
<i>OntAndInstMng</i>	Provides management of the annotations.
<i>OntPopulator</i>	Launches the automatic generation of annotations.
<i>InfoAccess</i>	Provides uniform access to the document and its annotations.
<i>InformationMng</i>	Provides management of the annotations.

Table 9.37: Annotations of Documents according to a Discovered Ontology Access and Management

- (c) To discover at run-time both the ontology and the documents. The pattern shown in Table 9.38 provides an architectural design for this scenario. In this pattern, the *Non-ontological Resource Discovery and Ranking* component is in charge of discovering and providing access to the annotated documents; the *Ontology Discovery and Ranking* component is in charge of discovering and providing access to the ontology; the *Ontology Repository* component store the annotations generated; the *Ontology Populator* component annotates at run-time the documents discovered according to the run-time ontology discovered and accesses the *Ontology Repository* component to store the annotations; finally, the *Information Directory Manager* component provide access to and management of the annotations.

Pattern 16	
Name	<i>Annotations of Discovered Documents according to a discovered Ontology Access and Management</i>
UML Diagram	
Relationship Associated	5 (dynamic ontology and dynamic document).
Selection Criterion	Annotations of documents discovered according to a run-time ontology discovered.
Roles of the Components	
<i>Ontology Discovery and Ranking</i>	Provides the discovery and ranking of the dynamic ontology.
<i>Non-ontological Resource Discovery and Ranking</i>	Provides the discovery and ranking of the document.
<i>Data Repository</i>	Provides access to and management and storage of the document.
<i>Ontology Repository</i>	Provides access to and management and storage of the annotations.
<i>Information Directory Manager</i>	Provides uniform access to the document and its annotations.
<i>Ontology Populator</i>	Annotates at run-time the documents discovered.
Roles of the Interfaces	
<i>DataAccess</i>	Provides access to the document.
<i>RankingAccess</i>	Provides rankings about discovered documents.
<i>OntAccess</i>	Provides access to the annotations.
<i>OntAndInstMng</i>	Provides management of the annotations.
<i>OntPopulator</i>	Launches the automatic generation of annotations
<i>InfoAccess</i>	Provides uniform access to the document and its annotations.
<i>InformationMng</i>	Provides management of the annotations.

Table 9.38: *Annotations of Discovered Documents according to a discovered Ontology Access and Management*

- In the case of ontologies annotated, the annotations can be stored by a set of instances or by the content of a non-ontological resource.

(a) In the case of ontologies annotated by a set of instances the pattern shown in Table 9.39 is used.

Pattern 17	
Name	<i>Ontologies Annotated with Ontological Metadata Access and Management</i>
<i>UML Diagram</i>	
<pre> classDiagram class MetadataRegistry["<<SWF_DMM>> Metadata Registry"] class InformationDirectoryManager["<<SWF_DMM>> Information Directory Manager"] class OntologyRepository["<<SWF_DMM>> Ontology Repository"] MetadataRegistry --> InformationDirectoryManager : MetadataAccess MetadataRegistry --> InformationDirectoryManager : MetadataMng InformationDirectoryManager --> OntologyRepository : OntAccess </pre>	
Relationship Associated	7.
Selection Criterion	Annotations of ontologies with ontological metadata (instances).
<i>Roles of the Components</i>	
<i>Ontology Repository</i>	Provides access to the ontology.
<i>Metadata Registry</i>	Provides access to and management and storage of the annotations.
<i>Information Directory Manager</i>	Provides uniform access to the ontology and its annotations.
<i>Roles of the Interfaces</i>	
<i>OntAccess</i>	Provides access to the ontology.
<i>MetadataAccess</i>	Provides access to the annotations.
<i>MetadataMng</i>	Provides management of the annotations.
<i>InfoAccess</i>	Provides uniform access to the ontology and its annotations.
<i>InformationMng</i>	Provides management of the annotations.

Table 9.39: *Ontologies Annotated with Ontological Metadata Access and Management*

(b) In the case of ontologies annotated by a the content of a non-ontological resource the patterns shown in Table 9.40 is used.

Pattern 18	
<i>Name</i>	<i>Ontologies Annotated with Non-ontological Resource Content Access and Management</i>
<i>UML Diagram</i>	
<pre> classDiagram class OntologyRepository["<<SWF_DMM>> Ontology Repository"] class InformationDirectoryManager["<<SWF_DMM>> Information Directory Manager"] class DataRepository["<<SWF_DMM>> Data Repository"] OntologyRepository --> InformationDirectoryManager : OntAccess InformationDirectoryManager --> DataRepository : DataAccess, DataManagement InformationDirectoryManager --> InformationDirectoryManager : InfoAccess, InformationMng </pre>	
<i>Relationship Associated</i>	8.
<i>Selection Criterion</i>	Annotations of ontologies with non-ontological resource content meta-data.
<i>Roles of the Components</i>	
<i>Ontology Repository</i>	Provides access to the ontology.
<i>Data Repository</i>	Provides access to and management and storage of the annotations.
<i>Information Directory Manager</i>	Provides uniform access to the ontology and its annotations.
<i>Roles of the Interfaces</i>	
<i>OntAccess</i>	Provides access to the ontology.
<i>DataAccess</i>	Provides access to the annotations.
<i>DataMng</i>	Provides management of the annotations.
<i>InfoAccess</i>	Provides uniform access to the ontology and its annotations.
<i>InformationMng</i>	Provides management of the annotations.

Table 9.40: *Ontologies Annotated with Non-ontological Resource Content Access and Management*

9.5 Components Associated to the Basic Templates

This section describes the architectural patterns associated to the basic system model templates presented in Section 8.3.

9.5.1 Components Associated to Data Sources with Schema

In Subsection 8.3.1 three different kinds of data sources with an associated schema have been shown: ontological data sources with an associated ontological schema, non-ontological data sources with an associated non-ontological schema, and a mixed template that incorporates heterogeneous data sources with ontological and non-ontological schemas. This subsection provides the patterns used for dealing with the three different kinds of data sources.

Components Associated to Ontological Data Sources with Ontological Schema

When multiple instances data sources that conform to a given schema, the *Information Directory Manager* component is used for providing access to and management of the ontology and the instances. In this pattern, *Information Directory Manager* requires the presence of any component that provides the *OntAccess* and *OntAndInstMng* interfaces. This component can be either the *Ontology Repository* component for the design-time ontologies selected or in case of dynamic resources, the *Ontology Discovery and Ranking* component. Table 9.41 shows the representation of the pattern of the *Ontological Data Sources Access and Management*.

Pattern 19	
Name	<i>Ontological Data Sources Access and Management</i>
UML Diagram	
Basic Template Associated	1.
Selection Criterion	Presence of ontological data sources expressed according to an ontological schema.
Roles of the Components	
<i>Information Directory Manager</i>	Provides uniform access to ontology and instances.
Roles of the Interfaces	
<i>InfoAccess</i>	Provides uniform access to the ontology and instances.
<i>InformationMng</i>	Provides uniform management of the ontology and instances.
<i>OntAccess</i>	Provides access to the ontology and instances.
<i>OntAndInstMng</i>	Provides management of the ontology and instances.

Table 9.41: *Ontological Data Sources Access and Management*

Components Associated to Non-ontological Data Sources with Non-ontological Schema

When there is multiple non-ontological content that conforms to a given non-ontological resource schema, the *Information Directory Manager* component is used for providing access to and management of the schema as well as to the non-ontological resource content. In this pattern, the *Information Directory Manager* requires the presence of any component that provides the *DataAccess* and *DataManagement* interfaces. This component can be the *Data Repository* component for design-time non-ontological resources selected, or in the case of dynamic resources, the *Non-ontological resource Discovery and Ranking* component.

Table 9.42 shows the representation of the pattern of the *Non-ontological Data Sources Access and Management*.

Pattern 20	
Name	<i>Non-ontological Data Sources Access and Management</i>
<i>UML Diagram</i>	
<i>Basic Template Associated</i>	2.
<i>Selection Criterion</i>	Presence of non-ontological data sources expressed according to a non-ontological schema.
<i>Roles of the Components</i>	
<i>Information Directory Manager</i>	Provides uniform access to non-ontological resource contents and their schema.
<i>Roles of the Interfaces</i>	
<i>InfoAccess</i>	Provides uniform access to non-ontological resource contents and their schema.
<i>InformationMng</i>	Provides uniform management of the non-ontological resource contents and their schema.
<i>DataAccess</i>	Provides access to non-ontological resource contents and their schema.
<i>DataManagement</i>	Provides management of non-ontological resource contents and their schema.

Table 9.42: *Non-ontological Data Sources Access and Management*

Components Associated to Ontological and Non-ontological Data Sources with Ontological Schema

When there is multiple ontological content that conforms to a given ontological schema mixed with non-ontological content that conforms to a given non-ontological resource schemas aligned with the ontology, then the *Information Directory Manager* component is used for providing access to and management of the ontology, the non-ontological schemas, the instances and the non-ontological resource content. In this pattern, the *Information Directory Manager* requires the presence of any component that provides the following interfaces: *OntAccess*, *OntAndInstMng*, *DataAccess* and *DataManagement* interfaces. The *AlignmentAccess* and *AlignmentMng* interfaces are used for accessing and managing the alignments between the ontology and the non-ontological schemas. If there is any kind of dynamic resource, then the *Matcher* interface is required to run-time align the different schemas with the ontology.

Table 9.43 shows the representation of the pattern of the *Ontological and Non-ontological Data Sources Access and Management*.

Pattern 21	
Name	<i>Ontological and Non-ontological Data Sources Access and Management</i>
<i>UML Diagram</i>	
Basic Template Associated	3.
Selection Criterion	Presence of multiple ontological content that conforms to a given ontological schema mixed with non-ontological content that conforms to a given non-ontological resource schemas aligned with the ontology.
<i>Roles of the Components</i>	
Information Directory Manager	Provides uniform access to ontologies, instances, and non-ontological resource contents and schemas.
<i>Roles of the Interfaces</i>	
InfoAccess	Provides uniform access to the ontology, instances, and non-ontological resource contents and their schema.
InformationMng	Provides uniform management of the ontology, instances, and non-ontological resource contents and their schema.
OntAccess	Provides access to the ontology and instances.
OntAndInstMng	Provides management of the ontology and instances.
DataAccess	Provides access to non-ontological resource contents and schemas.
DataManagement	Provides management of non-ontological resource contents and schemas.
AlignmentAccess	Provides access to the alignments between the ontology and the non-ontological resources schemas.
AlignmentMng	Provides management of the alignments.
DataTranslator	Performs the data translation of the non-ontological resource contents in terms of the ontology.
Matcher	When dynamic resources are involved it provides the run-time matching of the ontology and the schemas.

Table 9.43: *Ontological and Non-ontological Data Sources Access and Management*

9.5.2 Components Associated to Annotated Resources

Any of the patterns explained in Subsection 9.4.3 can be used to store multiple resources annotated and their annotations.

9.6 Components Associated to System Models

9.6.1 Query Information

This pattern is used to draw the architecture of the *Query Information* use case. The interfaces required in this pattern must be attached to the interfaces provided in any of the approaches in subsection 9.5.1.

Table 9.44 shows the pattern associated to *Query Information* system models.

Pattern 22	
Name	Query Information
UML Diagram	
<pre> classDiagram class QueryDialog["<<component>> Query Dialog"] class QueryFacade["<<component>> Query Facade"] class SemanticQueryEditor["<<SWF_QER>> Semantic Query Editor"] class SemanticQueryProcessor["<<SWF_QER>> Semantic Query Processor"] class QueryAnswering["<<SWF_QER>> Query Answering"] QueryDialog -- QueryFacade : QueryFacade QueryFacade -- SemanticQueryEditor : QueryEditor SemanticQueryEditor -- SemanticQueryProcessor : QueryProcessor SemanticQueryProcessor -- QueryAnswering : QueryAnswerer SemanticQueryProcessor -- SemanticQueryEditor : QueryProcessor SemanticQueryProcessor -- InfoAccess SemanticQueryProcessor -- OntAccess QueryAnswering -- InfoAccess QueryAnswering -- OntAccess QueryAnswering -- DataAccess </pre>	
Basic Template Associated	1, 2 and 3.
Selection Criterion	Presence of an application that queries a set of integrated information.

(continues on next page)

(comes from previous page)

<i>Roles of the Components</i>	
<i>Semantic Query Editor</i>	Takes care of all issues related to the user interface.
<i>Semantic Query Processor</i>	Takes care of all issues related to the physical processing of a query.
<i>Query Answering</i>	Takes care of all the issues related to the logical processing of a query.
<i>Query Dialog</i>	Implements the <i>Query Information</i> use case logic, that is, the software that handles the dialog between the <i>Primary Actor</i> and the System according to the use case flows.
<i>Query Facade</i>	Provides the operations to meet the use case responsibilities.
<i>Roles of the Interfaces</i>	
<i>QueryEditor</i>	Provides access to all the issues related to the user interface.
<i>QueryProcessor</i>	Provides the physical processing of a given query.
<i>QueryAnswerer</i>	Provides the logical processing of a given query.
<i>InfoAccess</i>	Provides uniform access to the ontologies, instances, and non-ontological resource contents and their schemas.
<i>OntAccess</i>	Provides access to the ontology and instances.
<i>DataAccess</i>	Provides access to non-ontological resource contents and schemas.
<i>QueryFacade</i>	Provides the operations to meet the use case responsibilities.

Table 9.44: *Query Information*

9.6.2 Search Resources

This pattern is used to draw the architecture of the *Search Resources* use case. The interfaces required in this pattern must be attached to the interfaces provided in any of the approaches in Subsection 9.4.3.

Table 9.45 shows the pattern associated to the *Search Resources* system models.

Pattern 23	
Name	Search Resources
UML Diagram	
<pre> classDiagram class SearchDialog["<<component>> Search Dialog"] class SearchFacade["<<component>> Search Facade"] class SemanticQueryEditor["<<SWF_QER>> Semantic Query Editor"] class SemanticQueryProcessor["<<SWF_QER>> Semantic Query Processor"] class QueryAnswering["<<SWF_QER>> Query Answering"] class QueryEditor class QueryProcessor class QueryAnswerer SearchDialog -- SearchFacade : SearchFacade SearchFacade -- QueryEditor : QueryEditor SemanticQueryEditor -- QueryProcessor : QueryProcessor SemanticQueryProcessor -- InfoAccess : InfoAccess SemanticQueryProcessor -- OntAccess : OntAccess QueryAnswering -- QueryAnswerer : QueryAnswerer QueryAnswering -- QueryProcessor : QueryProcessor QueryAnswering -- InfoAccess : InfoAccess QueryAnswering -- OntAccess : OntAccess QueryAnswering -- DataAccess : DataAccess </pre>	
Basic Template Associated	4.
Selection Criterion	Presence of an application that search for annotated resources.
Roles of the Components	
Semantic Query Editor	Takes care of all issues related to the user interface.
Semantic Query Processor	Takes care of all issues related to the physical processing of a search query.
Query Answering	Takes care of all the issues related to the logical processing of a search query.
Search Dialog	Implements the <i>Search Resources</i> use case logic, that is, the software that handles the dialog between the <i>Primary Actor</i> and the System according to the use case flows.
Search Facade	Provides the operations to meet the use case responsibilities.

(continues on next page)

(comes from previous page)

Roles of the Interfaces	
<i>QueryEditor</i>	Provides access to all the issues related to the user interface.
<i>QueryProcessor</i>	Provides the physical processing of a given search query.
<i>QueryAnswerer</i>	Provides the logical processing of a given search query.
<i>InfoAccess</i>	Provides uniform access to the ontologies, annotations, and non-ontological resource contents and their schemas.
<i>OntAccess</i>	Provides access to the ontology and annotations in the form of instances.
<i>DataAccess</i>	Provides access to the annotated data sources.
<i>ExtractFacade</i>	Provides the operations to meet the use case responsibilities.

Table 9.45: Search Resources

9.6.3 Browse Resources

This pattern is used to draw the architecture of the *Browse Resources* use case.

Table 9.46 shows the pattern associated to the *Browse Resources* system models.

Pattern 24	
Name	<i>Browse Resources</i>
UML Diagram	
<i>Basic Template Associated</i>	5 and 6.
<i>Selection Criterion</i>	Presence of an application that browse resources.

(continues on next page)

(comes from previous page)

<i>Roles of the Components</i>	
<i>Ontology Browser</i>	Takes care of all issues related to the visual browsing.
<i>Ontology View Customization</i>	It is responsible for enabling the user to change or amend a view on a particular ontology to fit a particular purpose.
<i>Ontology Adaptation Operators</i>	It is in charge of applying appropriate operators to the ontology in question, the result of which is an ontology customized according to some criterion.
<i>Ontology Localization and Profiling</i>	It is in charge of providing functionalities that adapt an ontology according to some context or some user profile.
<i>Browse Dialog</i>	Implements the <i>Browse Resources</i> use case logic, that is, the software that handles the dialog between the <i>Primary Actor</i> and the System according to the use case flows.
<i>Browse Facade</i>	Provides the operations to meet the use case responsibilities.
<i>Roles of the Interfaces</i>	
<i>InfoAccess</i>	Provides access to each of the result resources obtained.
<i>OntBrowser</i>	Provides functionalities related to the visual browsing.
<i>ViewCustomizer</i>	Provides the customized views of the ontologies.
<i>Adapter</i>	Provides customized ontologies after applying the appropriate operators.
<i>Profiler</i>	Provides the manual construction and activation, the semi-automated construction and adaptation, and the automated acquisition and management of user profiles.
<i>OntAccess</i>	Provides access to the ontologies and instances.
<i>ExtractFacade</i>	Provides the operations to meet the use case responsibilities.

Table 9.46: *Browse Resources*

9.6.4 Extract Information

This pattern is used to draw the architecture of the *Extract Information* use case. The interfaces required in this pattern must be attached to the interfaces provided in any of the approaches in Subsection 9.4.3.

Table 9.47 shows the pattern associated to the *Extract Information* system model.

Pattern 25	
Name	<i>Extract Information</i>
<i>UML Diagram</i>	
<pre> classDiagram class ExtractInformationDialog["<<component>> Extract Information Dialog"] class ExtractInformationFacade["<<component>> Extract Information Facade"] class SemanticQueryEditor["<<SWF_QER>> Semantic Query Editor"] class SemanticQueryProcessor["<<SWF_QER>> Semantic Query Processor"] class QueryAnswering["<<SWF_QER>> Query Answering"] ExtractInformationDialog --> ExtractInformationFacade : ExtractFacade ExtractInformationFacade --> SemanticQueryEditor : QueryEditor ExtractInformationFacade ..> InfoAccess : InfoAccess SemanticQueryEditor --> SemanticQueryProcessor : QueryProcessor SemanticQueryProcessor ..> InfoAccess : InfoAccess SemanticQueryProcessor ..> OntAccess : OntAccess SemanticQueryProcessor ..> QueryAnswerer : QueryAnswerer SemanticQueryProcessor ..> QueryProcessor : QueryProcessor QueryAnswering ..> InfoAccess : InfoAccess QueryAnswering ..> OntAccess : OntAccess QueryAnswering ..> DataAccess : DataAccess QueryAnswering ..> QueryAnswerer : QueryAnswerer QueryAnswering ..> QueryProcessor : QueryProcessor </pre>	
<i>Basic Template Associated</i>	7.
<i>Selection Criterion</i>	Presence of an application that extract meaningful information by analyzing search results.
<i>Roles of the Components</i>	
<i>Semantic Query Editor</i>	Takes care of all issues related to the user interface.
<i>Semantic Query Processor</i>	Takes care of all issues related to the physical processing of a search query.
<i>Query Answering</i>	Takes care of all the issues related to the logical processing of a search query.
<i>Extract Information Dialog</i>	Implements the <i>Extract Information</i> use case logic, that is, the software that handles the dialog between the <i>Primary Actor</i> and the System according to the use case flows.
<i>Extract Information Facade</i>	Provides the operations to meet the use case responsibilities, performing the required analysis of the search results.

(continues on next page)

(comes from previous page)

Roles of the Interfaces	
<i>QueryEditor</i>	Provides access to all the issues related to the user interface.
<i>QueryProcessor</i>	Provides the physical processing of a given search query.
<i>QueryAnswerer</i>	Provides the logical processing of a given search query.
<i>InfoAccess</i>	Provides uniform access to the ontologies, annotations, and non-ontological resource contents and their schemas.
<i>OntAccess</i>	Provides access to the ontology and annotations in the form of instances.
<i>DataAccess</i>	Provides access to the annotated data sources.
<i>ExtractFacade</i>	Provides the operations to meet the use case responsibilities.

Table 9.47: Extract Information

9.6.5 Edit

This pattern is used to draw the architecture of the *Manage Knowledge* use case when editing ontology elements, instances, alignments and annotations.

Table 9.48 shows the pattern associated to the *Edit* system models.

Pattern 26	
Name	<i>Edit</i>
UML Diagram	
Basic Template Associated	8 and 9.
Selection Criterion	Presence of an application that edits ontology elements, instances, alignments and annotations.

(continues on next page)

(comes from previous page)

<i>Roles of the Components</i>	
<i>Ontology Editor</i>	Takes care of creating and modifying the ontologies and instances.
<i>Ontology Matcher</i>	It is responsible of editing the alignments.
<i>Automatic Annotation</i>	Is in charge of automatically annotating resources with concepts in the ontologies.
<i>Manual Annotation</i>	Is in charge of the manual and semi-automatic annotation of the the resources with concepts in the ontology.
<i>Edit Dialog</i>	Implements the <i>Edit</i> logic, that is, the software that handles the dialog between the <i>Primary Actor</i> and the System according to the use case flows.
<i>Edit Facade</i>	Provides the operations to meet the use case responsibilities.
<i>Roles of the Interfaces</i>	
<i>OntEditor</i>	This interface provides the functionalities to edit the ontology elements.
<i>AlignEditor</i>	This interface provides the graphic representation and manipulation of the alignments.
<i>ManualAnnotator</i>	This interface is required for the manual and semi-automatic annotation of the resources and for bootstrapping learning when automatic annotations are performed.
<i>AutoAnnotator</i>	Provides the automatic annotation of the resources.
<i>OntAccess</i>	Provides access to the ontologies and instances and for collaborative annotation.
<i>OntAndInstMng</i>	Provides management of the ontology and instances.
<i>QueryProcessor</i>	This interface is required to check if an ontology is satisfiable after performing changes and for providing semantic matching.
<i>OntBrowser</i>	This interface is required to navigate through an ontology to insert, modify, or document its elements.
<i>ViewCustomizer</i>	This interface is required to perform alignments between customized views of ontologies.
<i>AlingmentAccess</i>	This interface is required to access to alignments stored in other components.
<i>DataAccess</i>	This interface is required to access the annotated resources.
<i>DataManagement</i>	This interface is required to insert annotations into the annotated resources.
<i>EditFacade</i>	Provides the operations to meet the use case responsibilities.

Table 9.48: *Edit*

9.6.6 Populate

This pattern is used to draw the architecture of the *Manage Knowledge* use case when populating an ontology.

Table 9.49 shows the pattern associated to the *Populate* system model.

Pattern 27	
Name	Populate
UML Diagram	
<pre> classDiagram class PopulateDialog["<<component>> Populate Dialog"] class PopulateFacade["<<component>> Populate Facade"] class OntologyPopulator["<<SWF_OIG>> Ontology Populator"] class ManualAnnotation["<<SWF_OIG>> Manual Annotation"] class PopulateFacadeInterface["PopulateFacade"] class OntPopulatorInterface["OntPopulator"] class ManualAnnotatorInterface["ManualAnnotator"] class OntAccessInterface["OntAccess"] class OntAndInstMngInterface["OntAndInstMng"] class DataAccessInterface["DataAccess"] PopulateDialog -- PopulateFacadeInterface : PopulateFacade PopulateFacadeInterface < -- PopulateFacade PopulateFacade -- OntPopulatorInterface : OntPopulator OntPopulatorInterface < -- OntologyPopulator OntologyPopulator -- ManualAnnotatorInterface : ManualAnnotator ManualAnnotatorInterface < -- ManualAnnotation OntologyPopulator -- OntAccessInterface : OntAccess OntologyPopulator -- OntAndInstMngInterface : OntAndInstMng OntologyPopulator -- DataAccessInterface : DataAccess ManualAnnotation -- OntAccessInterface : OntAccess ManualAnnotation -- DataAccessInterface : DataAccess </pre>	
Basic Template Associated	10.
Selection Criterion	Presence of an application that populates an ontology.
Roles of the Components	
Ontology Populator	Automatically generates new instances in a given ontology from a data source.
Manual Annotation	Bootstraps learning.
Populate Dialog	Implements the <i>Populate</i> use case logic, that is, the software that handles the dialog between the <i>Primary Actor</i> and the System according to the use case flows.
Populate Facade	Provides the operations to meet the use case responsibilities, performing the required analysis of the search results.
Roles of the Interfaces	
OntPopulator	Provides the automatic generation of instances.
ManualAnnotator	This interface is required to bootstrap learning.
QueryAnswerer	Provides the logical processing of a given search query.
OntAccess	Provides access to the ontologies and instances.
DataAccess	This interface is required to access the annotated resources.
OntAndInstMng	This interface is required to insert the generated instances.
PopulateFacade	Provides the operations to meet the use case responsibilities.

Table 9.49: *Populate*

9.6.7 Learn

This pattern is used to draw the architecture of the *Manage Knowledge* use case when learning an ontology from a document corpora.

Table 9.50 shows the pattern associated to the *Learn* system model.

Pattern 28	
Name	<i>Learn</i>
UML Diagram	
<pre> classDiagram class LearnDialog["<<component>> Learn Dialog"] class LearnFacade["<<component>> Learn Facade"] class OntologyLearner["<<SWF_OEN>> Ontology Learner"] class OntologyRepository["<<SWF_DMM>> Ontology Repository"] LearnDialog --> LearnFacade : LearnFacade LearnFacade --> OntologyLearner : OntLearner OntologyLearner --> OntologyRepository : OntAndInstMng OntologyLearner .. DataAccess </pre>	
<i>Basic Template Associated</i>	11.
<i>Selection Criterion</i>	Presence of an application that learns an ontology from a document corpora.
Roles of the Components	
<i>Ontology Learner</i>	Acquires knowledge and generates ontologies of a given domain.
<i>Ontology Repository</i>	Stores the ontologies learned.
<i>Learn Dialog</i>	Implements the <i>Learn</i> use case logic, that is, the software that handles the dialog between the <i>Primary Actor</i> and the System according to the use case flows.
<i>Learn Facade</i>	Provides the operations to meet the use case responsibilities, performing the required analysis of the search results.

(continues on next page)

(comes from previous page)

<i>Roles of the Interfaces</i>	
<i>OntLearner</i>	This interface is used to derive ontologies (semi)-automatically from natural language texts.
<i>OntAndInstMng</i>	This interface is required to create or modify the ontologies learned.
<i>DataAccess</i>	This interface is required to access to the documents.
<i>LearnFacade</i>	Provides the operations to meet the use case responsibilities.

Table 9.50: *Learn*

Chapter 10

Requirements Engineering Process

This chapter provides the NeOn methodological guidelines for carrying out the *Requirements Elicitation and Analysis* activity. We do not propose methodological guidelines for the rest of the activities involved in the *Requirements Engineering* process because we have not specialized them with respect to existing software engineering methods.

The NeOn methodology proposes to write the requirements document incrementally at the start of the project and as the project proceeds at the beginning of every application release development. The process will make use of the use cases technique and system models will be drawn in order to provide a graphical representation of the requirements.

10.1 Proposed Guidelines for Requirements Elicitation and Analysis

The results of this activity are the *Semantic Application Requirements document*, the *Release Planning* and the *Set of Ontological Needs*.

For carrying out this activity both the use cases technique and system models will be used. Within this activity semantic characteristics of the application will be discovered.

The general description of the *Requirements Elicitation and Analysis activity* is presented in Table 10.1, including the definition, goal, inputs and outputs, who carries out the activity, and when the activity should be carried out.

The tasks for carrying out the requirements elicitation and analysis activity can be seen in Figure 10.1.

Next, each task is introduced:

- In Task 1 the scenarios that the application will deal with are obtained by applying the use cases technique.
- In Task 2 the characteristics of the semantic application are identified as well as the set of ontological needs.
- In Task 3 the system models of the application taking into account the results of Task 1 and Task 2 are elaborated.
- In Task 4 the *use cases*, the *semantic characteristics*, and the *system models* are integrated within a single document in a given standardized format.
- Tasks 5 and 6 has been obtained by adapting the way in which in Extreme Programming the development team estimates the effort required for implementing a certain task and then the customer prioritizes the stories for implementation.

Table 10.1: General description of the *Requirements Elicitation and Analysis* activity

Requirements Elicitation and Analysis	
<i>Definition</i>	
Requirements Elicitation and Analysis refers to the activity of eliciting and understanding the requirements of the persons or groups who will be affected by the system, directly or indirectly.	
<i>Goal</i>	
The requirements elicitation and analysis activity states the functionalities that the system should provide as well as the constraints on the system.	
<i>Input</i>	<i>Output</i>
A set of <i>Business Requirements</i> , the <i>Use Cases Catalogue</i> , the <i>Semantic Application Questionnaires</i> and the <i>System Models Catalogue</i> .	The <i>Semantic Application Requirements document</i> , the <i>Release planning</i> and the <i>Set of Ontological Needs</i> .
<i>Who</i>	
Software developers, who form the application development team in close collaboration with the customer or final application user. It is required to some member of the development team to have expertise in semantic technology as well as in the requirements engineering process.	
<i>When</i>	
This activity must be carried out at the start of the project inside the requirements engineering process and at the beginning of every application release development. <ul style="list-style-type: none"> • At the start of the project, the developers and customers try to identify all the really significant requirements they can. • As the project proceeds, the customers should continue to state new business requirements. Thus the activity of requirements elicitation and analysis will not shut off until the project is over. 	
<i>Why</i>	
To specify the requirements of the large-scale semantic application being developed.	
<i>Where</i>	
In a place (e.g. meeting room) appropriate to maintain the interviews with the customer.	

10.1.1 Task 1. To Identify the Use Cases

The objective of this task is to gather information about the application and to distill scenarios from this information. The technique that we will use for discovering the scenarios is based on use cases. This task must be carried out by the development team.

The inputs to this task are the *Business Requirements* facilitated by the customer and the *Use Cases Catalogue*. The output of this task are the use cases themselves.

The NeOn methodology provides a catalogue of use cases (see Chapter 7) with the aim of identifying use cases related to semantic applications. When performing this task these common use cases can be selected, adapted and appended to the identified set of requirements. The methodology does not provide specific recommendations for other use cases non related to semantics.

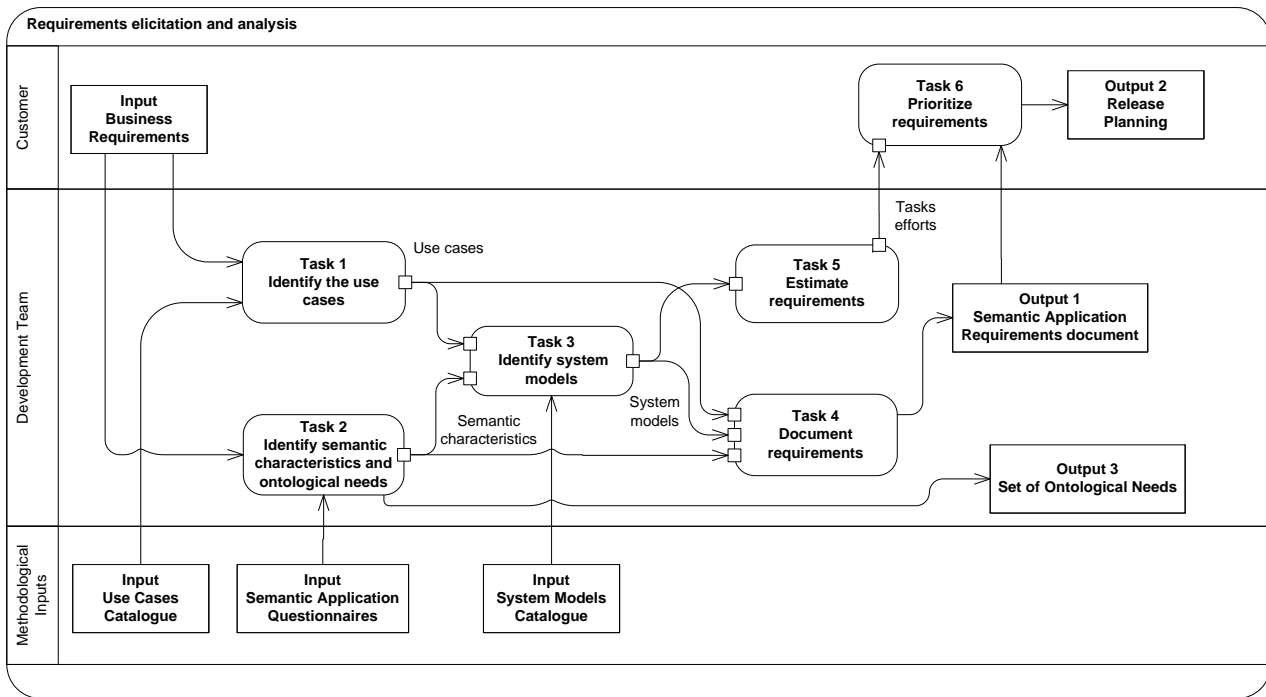


Figure 10.1: Tasks in the *Requirements Elicitation and Analysis* activity

10.1.2 Task 2. To Identify the Semantic Characteristics and Ontological Needs

The objective of this task is to collect the characteristics of the semantic application being developed, the typology of the resources that the application will deal with, and the *Set of ontological needs*. This task must be carried out by the development team.

The input to this task are the business requirements, and the output of this task are a the *Semantic characteristics* of the application and the *Set of Ontological Needs*.

The methodology presented in this deliverable provides a set of questionnaires used for identifying the characteristics (see Section 14) and *Set of ontological needs*.

10.1.3 Task 3. To Identify System Models

The objective of this task is to preliminary specify the system in the form of system models. This task must be carried out by the development team.

The input to this task are the *Use cases*, the *Semantic characteristics*, and the *System models catalogue*. The system models will reflect the scenarios identified during the use cases identification task constrained by the semantic characteristics of the application.

The output of this task are the *System Models*, which permit representing the system from the following different perspectives [Som07]:

- An external perspective, where the context or environment of the application is modelled.
- A structural perspective, where a preliminary architecture of the system or the structure of the data processed by the system is modelled.

The methodology presented in this deliverable provides a catalogue of system models that describe common behaviours of semantic applications (see Section 8). When performing this task these common system models can be selected according to the identified use cases and semantic characteristics. In consequence,

the system model catalogue is indexed by the related use case and characteristics associated to a given system models.

The system model catalogue provides a set of symbols (e.g. ontological and non ontological resources, applications and relationships between these symbols) that reflects the aforementioned structural perspective of the system. For reflecting the external perspective the system model will show the limits of the external systems that will be related with the system under development (see Chapter 8).

The following guidelines can be applied in order to elaborate a system model:

Step 1. To associate a basic symbol to each of the resources that the application will deal with.

Step 2. To identify the existing basic relationships between the basic resources.

Step 3. To identify the system model template associated to the identified use cases.

Step 4. To combine the symbols, relationships and system model templates in order to conform a unique system model.

10.1.4 Task 4. To Document Requirements

The objective of this task is to consolidate the requirements discovered in the previous tasks in a single description as the official statement of what the application developers should implement. This task must be carried out by the development team.

The inputs to this task are the *Use cases*, the *Semantic characteristics*, and the *System Models*.

The output of this task is the *Semantic Application Requirements Document*. The NeOn methodology suggest a template for writing the requirements document based on the proposed in [Som07], that at the same time, is based on the standard IEEE/ANSI 830-1998 [IEE98]. The requirements document should contain the slots shown in Table 10.2.

10.1.5 Task 5. To Estimate Requirements

The objective of this task is to estimate the effort needed for implementing a set of requirements. This task must be carried out by the development team.

The input to this task are the system models. As happens in Extreme Programming, the developers work together to break the system models (those not implemented in previous application releases) down into development tasks. A task is something that one developer can implement in a short period of time. Tasks are enumerated as completely as possible.

The estimation made by the development team is reflected in the output of this task. This output reflects the estimation cost for each identified task. Tasks are grouped according to the system model wich they belong.

10.1.6 Task 6. To Prioritize Requirements

The objective of this task is to prioritize the development tasks estimated in Task 5 by the development team. This task must be carried out by the customer.

The input to this task is the tasks budget. According to this input, the customers choose the system models that they want to be implemented in the following application release.

The order of implementation of the selected system models within the current iteration is a technical decision. The developers implement the system models selected by the customer in the order that makes the most technical sense. The customers cannot change the system models to be implemented in the next application release once the requirements process has finished and the design process is started. They are free to change or reorder any other system model in the project, but not the ones that the developers are currently working on.

Chapter	Description
<i>Preface</i>	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version [Som07].
<i>Introduction</i>	This should describe the need for the system. It should briefly describe its functions and explain how it will work with other systems. It should describe how the system fits into the overall business or strategic objectives of the organisation commissioning the software [Som07].
<i>Glossary</i>	This should define the technical terms used in the document. Assumptions about the experience or expertise of the reader cannot be made [Som07].
<i>User requirements</i>	The functionalities provided for the user and the non-functional system requirements should be described in this section. This description may use natural language, diagrams or other notations that are understandable to customers. Product and process standards which must be followed should be specified [Som07].
<i>Use cases</i>	This should describe the functional requirements in more detail using the use cases-based representation.
<i>Application semantic characteristics</i>	This should describe the non-functional requirements in terms of application characteristics and constraints.
<i>System models</i>	This should set out one or more system models showing preliminary relationships between key system components and between the system and its environment.
<i>System evolution</i>	This should describe the fundamental assumptions on which the system is based and should anticipate changes because of changing user's needs, etc. [Som07].
<i>Appendices</i>	These should provide detailed, specific information that is related to the application being developed [Som07].
<i>Index</i>	Several indexes to the document may be included. In addition to a normal alphabetic index, there may be an index of diagrams, etc. [Som07].

Table 10.2: The structure of the requirements document

The output of this task is the release planning that reflects the system models to be implemented for the next application release.

Chapter 11

Design Process

This chapter provides the NeOn methodological guidelines for carrying out the *Component Identification* activity. The rest of the activities are out of the scope of this deliverable and their guidelines will be provided in the next version.

Distributed systems architecture and user interface designs techniques are relevant for semantic application designing and will be considered in future versions of the methodology.

11.1 Proposed Guidelines for Component Identification

The goal of the *Component Identification* activity is to create an initial set of interfaces and component specifications, hooked together into a first-cut component architecture [CD01]. This activity is launched after the *Requirements Engineering* process is finished and takes the *Semantic Application Requirements Document*, the *Release Planning* and the *Architectural Patterns* as inputs.

The NeOn methodology proposes the filling card, presented in Table 11.1, for the component identification activity, including the definition, goal, inputs and outputs, who carries out the activity and when the activity should be carried out. This filling card has been adapted from the definition of the *Component Identification* activity described in the state of the art of this activity.

The tasks for carrying out the requirements elicitation and analysis activity can be seen in Figure 11.1. The result of this activity is the *Initial Architecture*.

The tasks proposed in these guidelines have been adapted from the activity description presented in [CD01]. Table 11.2 shows a mapping between the tasks that conform the *Component Identification* activity as it has been explained on the state of the art in Section 2.5.1 and the tasks that propose the NeOn methodology for the development of large-scale semantic applications.

As can be seen in Table 11.2 the first task in the state of the art (*Develop Business Type Model*) has been not adapted to the methodology proposed in this deliverable. The reason is that the data structures that semantic application uses are mainly ontologies or existing data schemas to be integrated within the application. In the case it is needed to create any ontology (or network of ontologies) to be used by the application, then the processes and activities defined in the NeOn methodology for building ontology networks can be applied.

11.1.1 Task 1. To Identify Dialogs and System Facades

The objective of this task is to identify the components that will be in charge of system interactions. The system interfaces are focused on, and derived from, the system's interaction. This task must be carried out by the development team.

The input to this task are the *Use Cases* gathered in the *Semantic Application Requirements Document* and the *Release Planning*, so the *System Dialogs and Facades* emerge from the use cases contemplated in the current release.

Table 11.1: Component Identification Filling Card

Component Identification	
<i>Definition</i>	
Component Identification refers to the activity of discovering the repositories that contains the information to be accessed and managed, which interfaces are needed to access and manage it, what components are needed to provide that functionality, and how they will fit together.	
<i>Goal</i>	
The component identification activity creates an initial set of interfaces and component specifications, hooked together into a preliminary component architecture.	
<i>Input</i>	<i>Output</i>
The <i>Semantic Application Requirements Document</i> , the <i>Release Planning</i> and the <i>architectural patterns</i> .	The <i>Initial Architecture</i>
<i>Who</i>	
Software developers, who form the application development team. It is required to some member/s of the development team to have expertise in semantic technology and they master the Semantic Web Framework components and the architectural patterns described by this methodology.	
<i>When</i>	
This activity must be carried out within the design process after the requirements engineering process for every application release development.	
<i>Why</i>	
To identify the components making up the semantic application and their relationships.	
<i>Where</i>	
At the software developers's workspace.	

For each use case two components will be defined:

- A *Dialog* component that implements the use case logic, that is, the software that handles the dialog between the primary actor and the system according to the specified use case.
- A *System Facade* component. For every step within a use case, the *System Facade* component may need to provide zero, one or multiple operations to meet its responsibilities.

The *Dialog* component will make use of the *System Facade* component. Thus the *Dialog* will have a dependency on the *System Facade* component.

The outputs of this task are a enumeration of the *System Dialogs* and *Facades* components. For every facade component a list of its operations must be specified.

11.1.2 Task 2. To Identify Knowledge Sources

The objective of this task is to catalogue the repository components that contain the ontological and non-ontological data that the application will make use of. This task must be carried out by the development team.

The input to this task are the *System Models* gathered in the *Requirements Document*, the *Release Planning*, and the *Architectural Patterns* so the knowledge sources emerge from the system models contemplated in the current release planning.

In this task, for each ontological and non-ontological resource reflected in the system models we identify its containing repository. Please note that several resources of a given kind (ontological or non-ontological) can

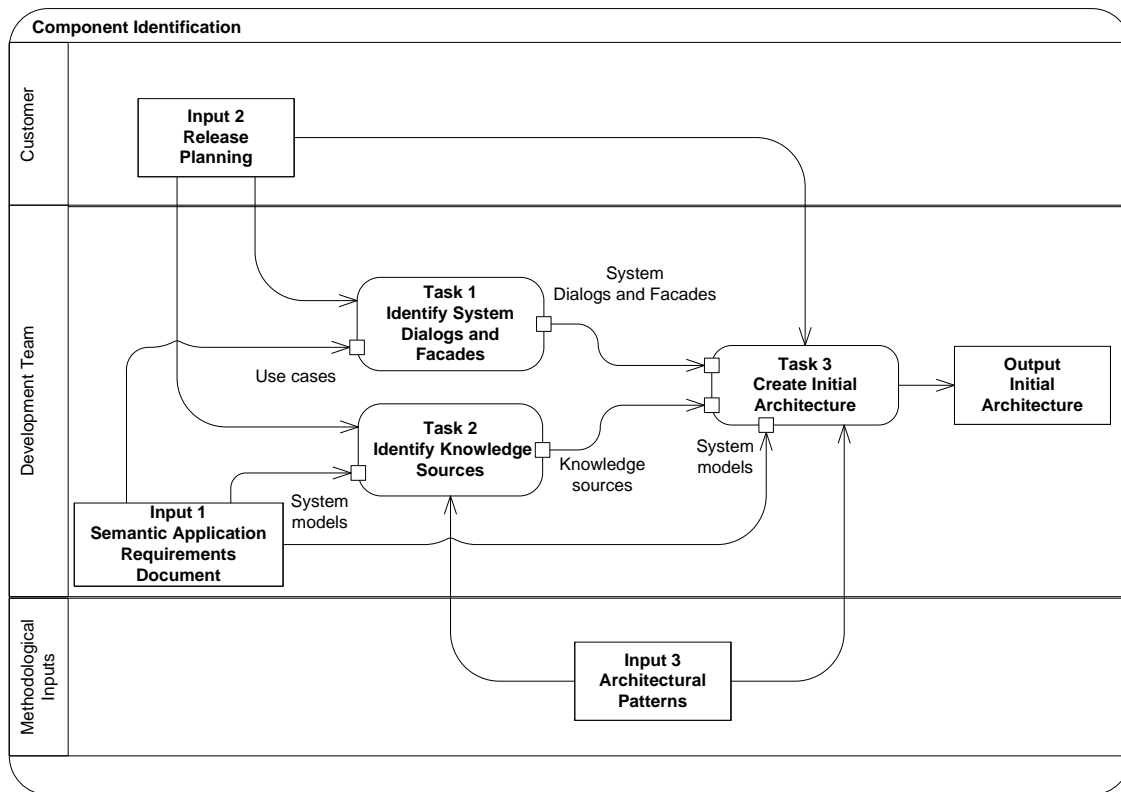


Figure 11.1: The component identification activity

be stored in the same repository.

As can be seen in Chapter 8 dynamically run-time discovered resources can be used within semantic applications. The methodology facilitates architectural patterns that will solve this issue through the use of gateway (*Ontology Discovery and Ranking* and *Non-ontological Resource Discovery and Ranking*) components presented in Chapter 9.

The output of this task is the set of identified *Knowledge Sources*.

The methodology presented in this deliverable provides a set of repository components and interfaces that offer the functionalities for accessing, managing and storing semantic and non-semantic resources, among other functionalities.

For each basic symbol identified in a system model there exists an architectural pattern that can be selected with the purpose of interfacing with the correspondent resource.

Software Engineering task	NeOn methodology task
Task 1. Develop Business Type Model	<i>none</i>
Task 2. Identify Business Interfaces	Task 1. To Identify Dialogs and System Facades
Task 3. Identify System Interfaces and Operations	Task 2. To Identify Interfaces with Knowledge Sources
Task 4. Create Initial Component Specifications and Architecture	Task 3. To Create the Initial Architecture

Table 11.2: Mapping between the tasks described in the state of the art and the tasks proposed by the NeOn methodology for the *Component Identification* activity

11.1.3 Task 3. To Create the Initial Architecture

The objective of this task is to draw an initial application architecture so developers can get an idea of how the components might fit together. This task must be carried out by the development team.

The inputs to this task are the *System Models* gathered in the *Requirements Document*, the *Release Planning*, the *System Dialogs and Facades*, the *Knowledge Sources* and the *Architectural Patterns*.

The architecture is obtained by composing the components identified in tasks 1 and 2 together with other components and interfaces, obtained from the *Architectural Patterns* by locating the patterns that correspond to each relationship and system model template (within the application system limit), until the architecture is completed. The composition of all the patterns can be done by attaching the required interfaces of a given pattern to the interfaces provided by another pattern. Attaching a required interface to a provided interface requires that both interfaces be of the same type.

The output of this task is the *Initial Architecture* of the application.

Chapter 12

Fictitious Example

This chapter presents an example showing how to carry out the *Requirements Elicitation and Analysis* and *Component Identification* activities, given a fictitious case study.

12.1 Requirements Elicitation and Analysis Activity

This section presents how to carry out the three first tasks of an hypothetical *Requirements Elicitation and Analysis* episode starting from a given set of *Business Requirements*.

12.1.1 Business Requirements

A logistics company has proved that setting dynamic shipment routes will decrease their shipment risks and delivery time and will increase its income because of factors such as weather, transport companies availability and fares, etc.

The company wants to upgrade its system to enable intelligent search of optimal routes taking into account weather information coming from different internet providers and information owned by transport companies, such as delivery times, transportation costs, availability of service for a certain route stretch, etc. The candidate routes are obtained from cartographies available in the Web.

Besides searching for the most adequate routes and transport companies, the logistics company wants to make use of the aforementioned integrated information to provide its clients with real time tracking of their shipments.

The information that the new application will use is encoded according to different formats: the weather information providers expose their information as instances expressed according to a given ontology; the transport companies provide a set of XML resources to facilitate the interoperability with the logistics companies; and the cartographies are published in the Semantic Web as instances that conform to a given ontology. Additionally, the logistics company will also use information stored in a relational database comprised in its own information systems.

The logistics company works with several transport companies and with only one weather information provider. Weather-information providers are not defined at the beginning of the project of the application development. That is, the application will discover information about weather at run-time and then integrate this information with the rest of the information.

None of the different XML schemas or ontologies that providers use to specify its information formats are unique, that is, each provider models its information according to different criteria.

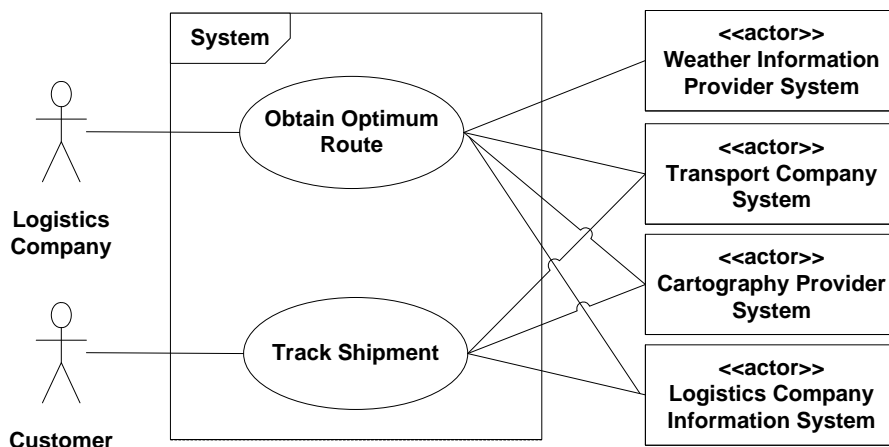


Figure 12.1: Identified use cases of the fictitious case study

12.1.2 Task 1. To Identify the Use Cases

Taking into account the *Business Requirements* facilitated by the logistics company, the development team starts identifying the use cases, finding the two use cases shown in Figure 12.1.

The first use case, *Obtain Optimum Route*, identifies the individual interactions between the logistics company system and the different external systems when an optimum route is obtained. The purpose of the second use case, *Track Shipment*, is to show the interactions between the customer of the logistics company with the system and the interactions of the system with the external information provider systems.

Both use cases can be seen as realisations of the use case template *UCT1 (Query Information)* presented in Section 7.1. The template in the catalogue has to be instantiated by the development team by identifying the concrete primary actor and the set of stakeholders including the external systems and modifying the flow of the main success scenario and extensions.

The development team locates the template in the catalogue of common use cases and then instantiates the use case specification as shown in tables 12.1 and 12.2.

Obtain Optimum Route	
<i>Scope</i>	Shipment Management System of the Logistics Company
<i>Level</i>	User-goal
<i>Primary Actor</i>	
Logistics Company	
<i>Stakeholders and Interests</i>	
<ul style="list-style-type: none"> • The <i>Logistics Company</i> requires to calculate the optimal route for a given shipment. • The <i>Weather Information Provider Systems</i> provide the System with information about weather conditions for a given location. • The <i>Transport Companies Systems</i> provide the System with information of the routes they follow to deliver a shipment and of their postal rates. • The <i>Cartography Provider System</i> provides the System with cartographical information of the geographical points contained in a given route. • The <i>Logistics Company Information System</i> provides the System management with information of shipment orders. 	
<i>Preconditions</i>	
<ul style="list-style-type: none"> • The <i>Logistics Company</i> can access the System. 	
<i>Success Guarantee (or Postconditions)</i>	
The System returns the optimum route for the given shipment to the <i>Logistics Company</i> .	

(continues on next page)

(comes from previous page)

Main Success Scenario (or Basic Flow)

1. The *Logistics Company* sends the shipment order identifier to the System.
2. The System sends the shipment order identifier to the *Logistics Company Information System*.
3. The *Logistics Company Information System* responds to the System with the departure and arrival places of the shipment.
4. The System sends the shipment departure and arrival places to the *Cartography Provider System*.
5. The *Cartography Provider System* responds to the System with the locations situated between the departure place of the shipment and the arrival places.
6. The System sends one location contained between the departure place of the shipment and the arrival place to a *Weather Information Provider System*.
7. The *Weather Information Provider System* responds to the System with the weather information of the given location.

Steps 6-7 are repeated until there are no more locations between the shipment departure and arrival places for querying about their weather conditions.

9. The System sends to a *Transport Company System* the departure and arrival places.
10. The *Transport Company System* responds to the System with its availability of deliver from the departure to the arrival places as well as with its postal fares.

Steps 9-10 are repeated until there are no more transport companies to be queried.

11. The System evaluates the gathered information on transportation costs and routes and weather conditions and responds to the *Logistics Company* with the optimal route and transportation company for delivering the required shipment.

(continues on next page)

*(comes from previous page)**Extensions (or Alternative Flows)*

*a. At any time, System fails:

1. The System informs the *Logistics Company* about the failure.

2-3a. At any time the communication with the *Logistics Company Information System* fails.

1. The System informs the *Logistics Company* about the failure.

3a. The *Logistics Company Information System* does not find the requested order.

1. The System informs the *Logistics Company* about the failure.

4-5a. At any time the communication with the *Cartography Provider System* fails.

1. The System informs the *Logistics Company* about the failure.

2. The System continues without taking into account route weather conditions in Step 9.

5a. The *Cartography Provider System* is not able to provide a route between the departure and arrival places.

1. The System informs the *Logistics Company* about the failure.

2. The System continues without taking into account route weather conditions in Step 9.

6-7a. At any time the communication with the *Weather Information Provider System* fails.

1. The System informs the *Logistics Company* about the failure.

2. The System continues without taking into account route weather conditions in Step 9.

7a. The *Weather Information Provider System* is not able to provide the weather conditions for a given location.

1. The System informs the *Logistics Company* about the failure.

2. The System continues without taking into account the route weather conditions for the given location in Step 6.

9-10a. At any time the communication with the *Transport Company System* fails.

1. The System informs the *Logistics Company* about the failure.

2. The System continues with the next transport company to be queried in Step 9.

10a. The *Transport Company* does not deliver from the departure to the arrival places.

1. The System continues with the next transport company to be queried in Step 9.

(continues on next page)

(comes from previous page)

<i>Special Requirements</i>
<ul style="list-style-type: none"> • The application will discover in the Web the weather information providers at run-time.
<i>Technology and Data Variations List</i>
<ul style="list-style-type: none"> • The <i>Logistics Company Information System</i> consist of an Oracle relational database accessible by standard APIs (e.g. ODBC, JDBC). • The <i>Weather Information Provider Systems</i> will publish their information in the Semantic Web as instances expressed according to a given ontology. • The <i>Transport Companies Systems</i> provide their information as Web accessible XML resources. • The <i>The Cartography Provider System</i> provide its information as instances expressed according to a given ontology.
<i>Frequency of Occurrence</i>
High
<i>Miscellaneous</i>
<ul style="list-style-type: none"> • Open issue: some security aspects of several system are not clear for the moment.

Table 12.1: *Obtain Optimum Route* use case

Track Shipment	
<i>Scope</i>	Shipment Management System of the Logistics Company
<i>Level</i>	User-goal
<i>Primary Actor</i>	
Customer	
<i>Stakeholders and Interests</i>	
<ul style="list-style-type: none"> • The <i>Customer</i> of the logistics company requires to track the state and position of a given shipment. • The <i>Transport Companies Systems</i> provide the System with information about the state and position of the shipment. • The <i>Cartography Provider System</i> provides the System with cartographical information about the route that the shipment is following. • The <i>Logistics Company Information System</i> provides to the System management information about shipment orders. 	

(continues on next page)

*(comes from previous page)***Preconditions**

- The *User* can access the System.
- The shipment order has been sent to the transport company.
- The transport company has assigned to the shipment order its own identifier and this has been sent to the logistics company.

Success Guarantee (or Postconditions)

The System returns the the state and position of the shipment to the *Customer*.

Main Success Scenario (or Basic Flow)

1. The *User* sends to the System the shipment order identifier.
2. The System sends the shipment order identifier to the *Logistics Company Information System* .
3. The *Logistics Company Information System* responds to the System with the identifier assigned to the shipment by the transport company, the departure place of the shipment and the arrival place.
4. The System sends the shipment departure and arrival places to the *Cartography Provider System* .
5. The *Cartography Provider System* responds to the System with the route from the departure place to the arrival place.
6. The System sends the identifier assigned to the shipment order by the transport company to a *Transport Company System*.
7. The *Transport Company System* responds to the System with the current status and position of the shipment order.
8. The System responds to the *Customer* with the route of the shipment, marking on it the current position as well as the current shipment status.

(continues on next page)

(comes from previous page)

Extensions (or Alternative Flows)

*a. At any time, System fails:

1. The System informs the *Customer* about the failure.

2-3a. At any time the communication with the *Logistics Company Information System* fails.

1. The System informs the *Customer* about the failure.

3a. The *Logistics Company Information System* does not find the requested order.

1. The System informs the *Customer* about the failure.

4-5a. At any time the communication with the *Cartography Provider System* fails.

1. The System sends the identifier assigned to the shipment order by the transport company to a *Transport Company System*.
2. The *Transport Company System* responds to the System with the current status and position of the shipment order.
3. The System responds to the *Customer* with the current status of the shipment and its position, without returning the complete route.

5a. The *Cartography Provider System* is not able to provide a route between the departure and arrival places.

1. The System sends to a *Transport Company System* the identifier assigned to the shipment order by the transport company.
2. The *Transport Company System* responses to the System with the current status and position of the shipment order.
3. The System responses to the *Customer* with the current status of the shipment and its position without returning the complete route.

6-7a, 5a.1-2a. At any time the communication with the *Transport Company System* fails.

1. The System informs the *Customer* about the failure.

Technology and Data Variations List

- The *Logistics Company Information System* consist of an Oracle relational database accessible by standard APIs (e.g. ODBC, JDBC).
- The *Transport Companies Systems* provide their information as Web accessible XML resources.
- The *The Cartography Provider System* provides its information as instances expressed according to a given ontology.

Frequency of Occurrence

High

(continues on next page)

(comes from previous page)

Miscellaneous

- Open issue: some security aspects of several systems are not clear for the moment.

Table 12.2: *Track Shipment* use case

12.1.3 Task 2. To Identify Application Characteristics and Ontological Needs

In this task, the development team proceeds to identify those non-functional requirements that will affect the behaviour of the whole application.

As previously seen, the set of characteristics that commonly appear on semantic applications is intended to help developers identify the semantic requirements of the application under development. Next, the application is characterized according to the dimensions explained in Chapter 6.

Ontologies Dimension

The application is characterized by the consumption of own and foreign ontologies (those owned by the cartography providers and the weather information providers). Thus, the application is not bound to the particular domain of the logistics company. The ontologies also are not centralized in a single resource but distributed in multiple sources, so there is *Distribution of ontologies*.

With respect to the *Design-time or run-time ontology selection* characteristic, the ontologies owned by the cartography providers are identified and selected at design time. Nevertheless, the weather information provider' ontologies must be discovered and queried at run-time so the application is characterized by partial *Design-time ontology selection*.

With respect to the *Scalability regarding the number of ontologies* characteristic, the application will not operate at scale with a huge number of ontologies.

Because the application will deal with different ontologies produced by different organizations, these ontologies may have different encodings (e.g., RDF(S) or OWL) and can exhibit differences in quality, computational heterogeneity (lightweight and heavyweight ontologies), can be modeled according to different criteria and will exhibit conceptual heterogeneity that should be solved with the use of mappings; thus, the *Ability to resolve conceptual heterogeneity in ontologies* characteristic is guaranteed.

The use of different ontologies that somehow have to be interrelated will lead to a ontology network typology.

Data Dimension

The application is characterized by the consumption of own and foreign data. In consequence, there is *Use of internal and external data sources*.

As happened with the ontologies, the data is not centralized in a single resource but distributed among different companies, that is, there is *Data Distribution*.

Also, the application is characterized by the use of dynamic data that is constantly changing so there is *Data Dynamicity*. The data is gathered, selected, combined and processed at run-time.

With respect to the *Data scalability* characteristic, the application has to operate at scale with a huge number of instances.

There is *Data Encoding Heterogeneity* because data is encoded in two different forms, that is, semantic and non-semantic (ontology instances, XML, records in a relational database.).

Reasoning Dimension

With respect to the *Reasoning Dimension*, the development team can advance that heavyweight logical reasoning is not a key aspect of the application. Because the application has to deal with the heterogeneity of ontologies and instances explained before, the reasoning has to be hybrid and enabled by the combination of subsumption reasoning and other kinds of reasoning. Besides, the application should deal with contradictory information coming from contradictory weather previsions. And, furthermore, it is not guaranteed that the information used will be always complete since the system will process information from different providers that the logistics company does not own.

Non-functional Dimension

With respect to the *Interoperability with other applications*, the system will interoperate with legacy databases and will use the XML information provided by the transport companies.

12.1.4 Task 3. To Identify System Models

The development team starts by identifying the typology of the resources that the application will deal with. Table 12.3 shows the resources identified with its associated basic symbols, which are taken from Section 8.1 after analyzing the characteristics of each resource.

Resource Identifier	Resource Description	Basic Symbol
Cartography Ontology	Ontology of the cartography provider	1. Static Ontology
Cartography Instances	Instances of the cartography provider	2. Static Instances
Transport Schema	XML schema of the transport company provider	3. Static Non-ontological Resource Schema
Transport Data	XML data of the transport company provider	4. Static Non-Ontological Resource Content
Weather Ontologies	Ontologies of the weather information providers	6. Dynamic Ontology
Weather Instances	Instances of the weather information providers	7. Dynamic Instances
Logistics DB	Corporate database of the logistics company	13. Non-ontological Resource Content that Conforms To a given Schema Abbreviation

Table 12.3: Symbols associated to the resources used by the example application

Then the development team identify the existing basic relationships between the basic resources. Table 12.4 shows the relationships identified, obtained from Section 8.2.

As it has been seen in subsection 12.1.2 both use cases are associated to the use case template *UCT1 (Query Information)* presented in Section 7.1. By taking a look to the system model catalogue, the development team realizes that they count with three possible system models.

The *System Model 1 (Query Information with a Single Ontology Approach)* is discarded because of the *Multiple Ontology Typology* characteristic identified in the previous task. From the other system models, the development chooses the third system model (*Query Information with a Hybrid Ontologies Approach*) because of a design decision.

Resource 1	Resource 2	Relationship
Cartography Instances	Cartography Ontology	1. Instances that Conform To a Given Ontology
Transport Data	Transport Schema	2. Non-ontological Resource Content that Conforms To a Given Schema
Weather Instances	Weather Ontologies	1. Instances that Conform To a Given Ontology

Table 12.4: Relationships between the resources used in the example application

As the system model chosen indicates, it is needed to create and incorporate another ontology consisting of a shared vocabulary that will be aligned with the rest of the ontological and non-ontological schemas to facilitate information integration. In consequence, several *Aligned With* relationships must be included in the final system model.

By integrating the basic symbols and their relationships with the *Query Information with a Hybrid Ontology Approach*, the system model resulting is the one shown in Figure 12.2.

12.2 Component Identification Activity

This section presents how to carry out the three tasks of the *Component Identification* activity, taking into account the use cases and system model obtained in the previous section.

12.2.1 Task 1. To Identify Dialogs and System Facades

Within this task the development team introduces in the architecture a system dialog and a facade for each use case identified. The components obtained are the following:

- **Obtain Optimum Route Dialog** component. It implements the *Obtain Optimum Route* use case logic, that is, the software that handles the dialog between the *Logistics Company* and the system according to the specified use case. This component will make use of the *Obtain Optimum Route Facade* component.
- **Obtain Optimum Route Facade** component. For every step within the *Obtain Optimum Route* use case, it will provide operations to meet its responsibilities.
- **Track Shipment Dialog** component. It implements the *Track Shipment* use case logic, that is, the software that handles the dialog between the *Customer* and the system, according to the specified use case. This component will make use of the *Track Shipment Facade* component.
- **Track Shipment Facade** component. For every step within the *Track Shipment* use case, it will provide operations to meet its responsibilities

12.2.2 Task 2. To Identify Interfaces to Knowledge Sources

Within this task the development team catalogues the sources or repositories containing the ontological and non-ontological data that the application will make use of. For each ontological and non-ontological resource reflected in the previously system model obtained, its containing repository is identified. Table 12.5 shows the system and patterns associated to each resource.

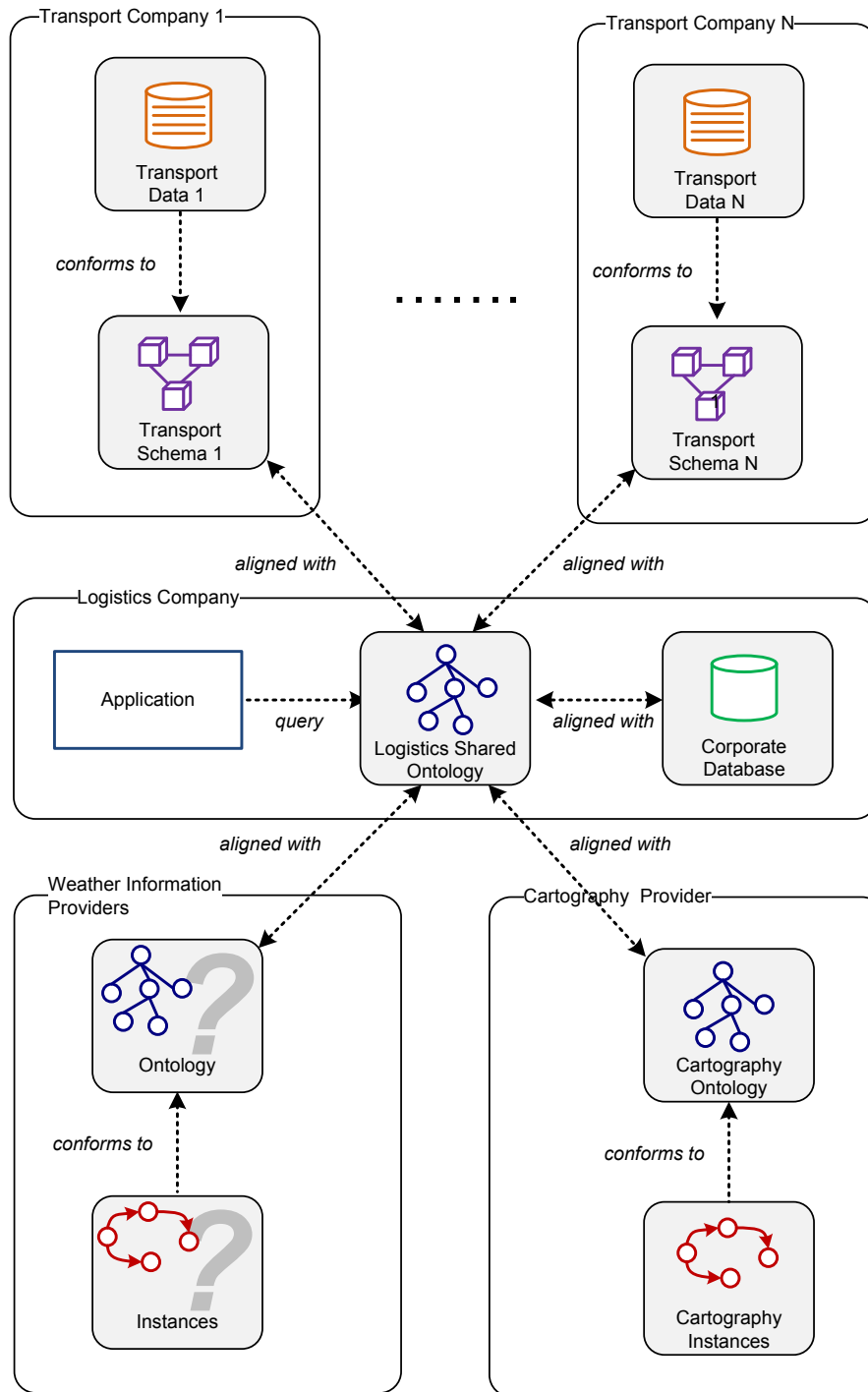


Figure 12.2: System model identified for the fictitious example

12.2.3 Task 3. To Create the Initial Architecture

By integrating the all the components and patterns the architecture shown in Figure 12.3 is obtained.

Resource	System	Pattern
Transport Data	Transport Companies	2. Data Repository
Transport Schema	Transport Companies	2. Data Repository
Logistics Shared Ontology	Logistics Company	1. Ontology Repository
Corporate Database	Logistics Company	2. Data Repository
Weather Ontology	Weather Information Providers	3. Dynamic Ontological Resource Access
Weather Instances	Weather Information Providers	3. Dynamic Ontological Resource Access
Cartography Ontology	Cartography Providers	1. Ontology Repository
Cartography Instances	Cartography Providers	1. Ontology Repository

Table 12.5: Patterns associated to the repositories used by the example application

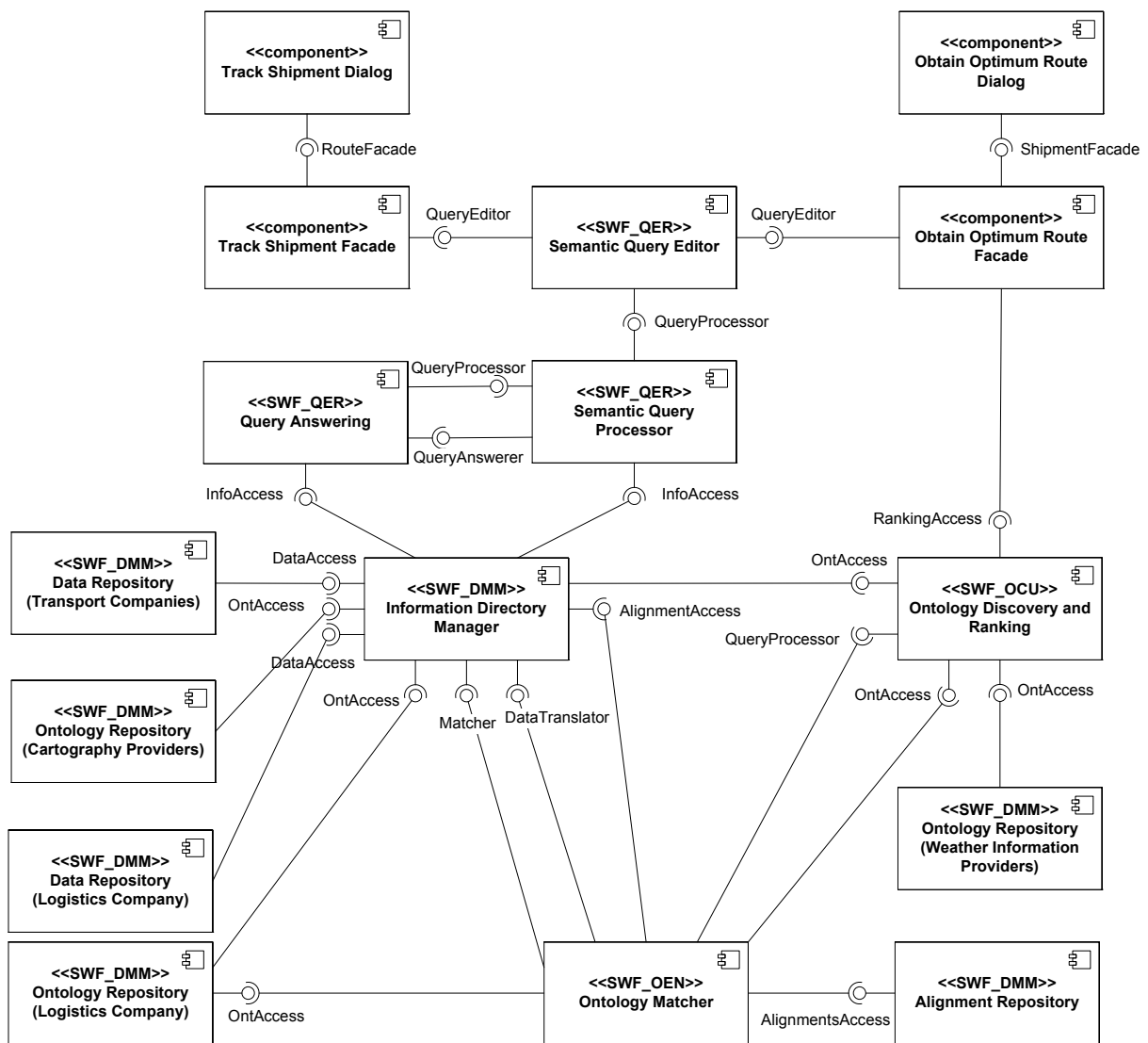


Figure 12.3: Example Application Architecture

Chapter 13

Real Example. FAO case study

This chapter presents an real example showing how have been carried out the *Requirements Elicitation and Analysis* and *Component Identification* activities for the AGROVOC Concept Server Workbench 3.0 have been carried out. The AGROVOC Concept Server Workbench 3.0 web-based distributed collaborative tool for managing multilingual ontologies about agriculture.

13.1 Requirements Elicitation and Analysis Activity

This section presents how the three first tasks of the *Requirements Elicitation and Analysis* of the system have been carried out, starting from a given set of *Business Requirements*.

Since the full requirements of the application are quite long and extensive, we are reporting here the most important sections.

13.1.1 Business Requirements

The AGROVOC Concept Server Workbench can be a good reference system for all users in the agricultural domains who would need to search or manage terminologies or ontological concepts for agriculture, in multiple languages. It uses as a resource the FAO AGROVOC multilingual thesaurus which has been re-engineered and extended to have some of the characteristics of a full ontology.

Currently, the AGROVOC maintenance tool is a stand-alone system and it is not available online, which implies an increment of work for the FAO staff that manages the different AGROVOC languages and other updates (incorporate changes, new terms, new language versions).

The need to develop of a new tool that could maintain the AGROVOC Concept Server arises from the following objectives:

- to promote and facilitate the enrichment of the agricultural terminology and knowledge outside the usual AGROVOC managing organisations;
- to have an enriched platform for managing terminological and ontological data, and make these available via web services, or via download facilities (currently provided with a separate ftp area¹);
- to provide a knowledge base from where users can download modules to further build domain-specific ontologies;
- to have a unique collaborative tool available online for AGROVOC;
- to provide a framework in which terminologists and domain experts can benefit from semantic-based techniques, such as corpus analysis, term extractions, etc;

¹ftp://ftp.fao.org/gi/gil/gilws/aims/kos/agrovoc_formats

- to allow local terminology to be represented and available for regional or sub-regional users and/or applications without having to centralize all languages for all domains in a unique repository .

While mentioning those, it is important also to indicate the reasons which drove the reengineering of the AGROVOC full system (content and infrastructure):

- the current thesaurus is keyword-oriented and it is not easy to distinguish a conceptual entity (concept) from the terms that represent it;
- the current AGROVOC may be used for more services (we wish to incorporate more semantics in the original thesaurus structure);
- the AOS initiative promotes the ability to represent, share and distribute local knowledge, terminology in multiple languages including language variances, term and string level relationships.

The main business reasons for the development of this new tool for managing the concept server data are:

- to reduce AGROVOC maintenance work;
- easier construction of sub-domain ontologies;
- more complete and coherent agricultural terminology;
- better connection (mapping) with FAOTERM and other systems;
- enriched collaboration within the agricultural community and other agricultural projects oriented versus the semantic Web (e.g. Agropedia Indica).

In more detail, and to clarify the business requirements used in this deliverable, we have to highlight that

- agricultural users would need to be able to find a concept related to agriculture by using any of the possible solutions that they could apply, e.g., search using a term in a specific language that represents a concept, search by URI if already known, search by creator or owner, by creation or modification date, etc.
- agricultural users would need to operate on the terminological base that underpins the ontological structure, this means that they would need to add a term or word that identifies a concept, and they would need to edit already inserted terms, or eliminate them if needed.
- agricultural users would need to be facilitated the management of the terminological and ontological base with some automatic actions today available in the international community through several tools (e.g. corpus analysis, relationships refinement, etc).

13.1.2 Task 1. To Identify the Use Cases

Taking into account the *Business Requirements* facilitated by FAO, the use cases identified are those shown in Figure 13.1.

Next all the use cases are explained

- The first use case, *Search Concept*, identifies the interactions between any kind of system user and the application when finding and visualizing an AGROVOC concept.

As an example scenario, let us think of a fishery expert writing a document about the tuna fish who would like to identify which type of species are available. Then, the user navigates or ask to the system to find the concept "tuna" and to visualize it in the main concept window. The system shows the concept window with the "tuna" concept selected and visualized in full hierarchy.

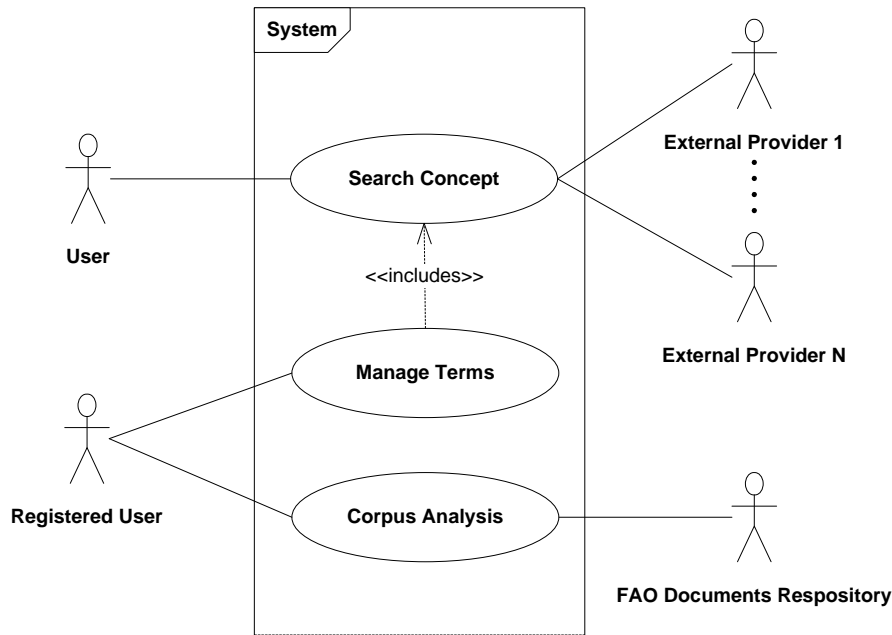


Figure 13.1: Use cases identified in the FAO case study

One of the advantages of using this use case for the FAO organization is that an expert would not ask the resource owner but would carry out the task himself, consequently saving time to the expert and to the resource maintainer.

This use case can be seen as a realisation combining the use cases templates *UCT1 (Query Information)*, *UCT2 (Search Resources)* and *UCT3 (Browse Resources)* presented in sections 7.1, 7.2 and 7.3 respectively. After locating the correspondent templates in the catalogue of use cases and instantiating them, the use case specification shown in 13.1 is obtained.

Search Concept	
<i>Scope</i>	AGROVOC Concept Server Workbench 3.0
<i>Level</i>	User-goal
<i>Primary Actor</i>	
User	
<i>Stakeholders and Interests</i>	
<ul style="list-style-type: none"> - The <i>User</i> makes a request to the system to get a concept. This <i>User</i> role represents any kind of user (registered and non-registered users). - The <i>External Providers</i> may provide answer to the system, which make them available to the final customer, who is the <i>User</i>. 	
<i>Preconditions</i>	
<ul style="list-style-type: none"> - None. 	

(continues on next page)

*(comes from previous page)***Success Guarantee (or Postconditions)**

The System returns the requested concept information to the *User*. The concept may be found within the own AGROVOC knowledge base or not. In the latter case, results from external resources may be shown.

Main Success Scenario (or Basic Flow)

1. The *User* submits a search query following several patterns such as URI, a term string in a specific language, date in which a concept may have been created, owner, status, and relationships with other concepts or term codes.
2. The System looks within the AGROVOC knowledge base, finding a set of results.
3. The System returns the results ranked by an specific criteria to the *User*.
4. The *User* asks the system to return an specific concept information.
5. The System responds to the *User* with the information associated to the concept chosen (including its terms).

Extensions (or Alternative Flows)

*a. At any time, System fails:

1. The System informs the *User* about the failure.
- 1a. The *User* navigates to a concept.
1. The use case continues in step 4.
- 2a. No results are found within the AGROVOC knowledge base.
1. The System sends the search query to an *External Provider*.
 2. The *External Provider* responds to the System with the requested information.
- Steps 2a.1-2a.2 are repeated until there are no more external providers to be queried.*
3. The use case continues on step 3.
- 2a.1-2a.2a. The communication with the *External Provider* fails.
1. The system keeps requesting other Information Providers continuing in step 2a.1.
- 4a. The *User* asks the system to return an specific concept information in a particular language.
1. The System responds to the *User* with the information associated to the concept chosen according to the language chosen.

Special Requirements

- Concepts are stored as ontology classes.
- Terms are linguistic information associated to a concept stored within a non-ontological resource.

Miscellaneous

3. The criteria for ranking the results is not currently specified.

(continues on next page)

(comes from previous page)

<i>Frequency of Occurrence</i>
High

Table 13.1: *Search Concept* use case

- The second use case, *Manage Terms*, identifies the interactions between a registered user and the application when assigning terms to a given concept.

As an example scenario, let us think of a student located in south Italy who after browsing the AGROVOC concept server data, finds a concept represented in English but not in Italian. Then the user proceeds to assign a new translation for an English term by creating a new term, for example “acqua”, associated to the concept represented by the English term “water”. Finally the term “acqua” is added to the knowledge base and associated with the appropriate concept.

An advantage of this use case for the FAO organisation is that AGROVOC data are enriched and available to more users worldwide since it increases sharing, collaboration and decreases individual efforts.

This use case can be seen as a realisation of the use case *UCT5 (Manage Knowledge)* presented in section 7.5. After locating the correspondent template in the catalogue of use cases and instantiating it, the use case specification shown in 13.2 is obtained.

Manage Terms	
<i>Scope</i>	AGROVOC Concept Server Workbench 3.0
<i>Level</i>	User-goal
<i>Primary Actor</i>	
Registered User	
<i>Stakeholders and Interests</i>	
<ul style="list-style-type: none"> – The <i>Registered User</i> who have privileges to modify the knowledge base. 	
<i>Preconditions</i>	
<ul style="list-style-type: none"> – The <i>Registered User</i> must be registered and logged-in. 	
<i>Success Guarantee (or Postconditions)</i>	
Modifications requested by the <i>Registered User</i> are correctly saved in the knowledge base by adding a new translation and linking it to the appropriate concept.	
<i>Main Success Scenario (or Basic Flow)</i>	
<ol style="list-style-type: none"> 1. Find a concept and obtain its associated terms: Includes <i>Search Concept</i>. 2. The <i>Registered User</i> request the System to add an specific term associated to the concept. 3. The System commits the changes sent by the <i>Registered User</i>. 	

(continues on next page)

(comes from previous page)

<i>Extensions (or Alternative Flows)</i>
<p>*a. At any time, System fails:</p> <ol style="list-style-type: none"> 1. The System informs the <i>Registered User</i> about the failure. <p>2a. The <i>Registered User</i> wants to modify an specific term.</p> <ol style="list-style-type: none"> 1. The <i>Registered User</i> sends the modifications to the System . 2. The System continues in Step 3. <p>2b. The <i>Registered User</i> wants to remove a given term.</p> <ol style="list-style-type: none"> 1. The <i>Registered User</i> sends to the System the term to be deleted. 2. The System continues in Step 3.
<i>Special Requirements</i>
<ul style="list-style-type: none"> – Concepts are stored as ontology classes. – Terms are linguistic information associated to a concept stored within a non-ontological resource.
<i>Frequency of Occurrence</i>
High

Table 13.2: *Manage Terms* use case

- The third use case, *Corpus Analysis*, identifies the interactions between a registered user and the application when identifying new terms, concepts or relationships to enrich the knowledge base from a existing document.

Within this use case multiple terms that are not in the knowledge base can be added after extracting them from several documents of a specific domain and in a specific language. The documents are stored within the FAO document repository.

As an example scenario, let us think of an AGROVOC manager in Spain who would like to enrich the ontology section for nutrition. The user selects a language, and a domain. Then the system performs an analysis and proposes a list of concepts, terms or relationship between terms (synonymy, etc) for being added to the ontology system.

Maintaining the current AGROVOC and the future concept server is a very expensive action in terms of the time spent by the experts involved. It requires knowledge on the domain, experience in the IM area, and dedication to the task. The corpus analysis functionality will speed up this process allowing the user to minimise the efforts.

This use case can be seen as a realization of the use case *UCT2 (Search Resources)* presented in section 7.2 when filtering documents by language and domain. This use case also realizes the use case *UCT5 (Manage Knowledge)* presented in section 7.5 when performing ontology learning. After locating the correspondent template in the catalogue of use cases and instantiating it, the use case specification shown in 13.3 is obtained.

Corpus Analysis	
<i>Scope</i>	AGROVOC Concept Server Workbench 3.0
<i>Level</i>	User-goal

(continues on next page)

(comes from previous page)

<i>Primary Actor</i>
Registered User
<i>Stakeholders and Interests</i>
<ul style="list-style-type: none"> – The <i>Registered User</i> who requests the System to enrich the knowledge base by analysing documents in a specific domain and language. – The <i>FAO Documents Repository</i> is an external repository that provides the system with documents (and associated metadata) from which to extract the information.
<i>Preconditions</i>
<ul style="list-style-type: none"> – Documents and associated metadata in the <i>FAO Documents Repository</i> exist. – The <i>Registered User</i> must be logged-in and have specific privileges to perform the use case.
<i>Success Guarantee (or Postconditions)</i>
The system enriches the knowledge base with the information extracted from the <i>FAO Documents Repository</i> after validating the System proposal by the <i>Registered User</i> .
<i>Main Success Scenario (or Basic Flow)</i>
<ol style="list-style-type: none"> 1. The <i>Registered User</i> selects the domain and the language he wishes to work on. 2. The System request the <i>FAO Document Repository</i> a set of documents filtered by the selected domain and language. 3. The <i>FAO Document Repository</i> provides the System with the set of documents. 4. The System analyses the corpora and provides the <i>Registered User</i> with the list of learned concepts, terms and relationships between terms. 5. The <i>Registered User</i> selects from the list the items to be added and sends them back to the System. 6. The System modifies the knowledge base by incorporating the items selected by the <i>Registered User</i>.
<i>Extensions (or Alternative Flows)</i>
<p>*a. At any time, System fails:</p> <ol style="list-style-type: none"> 1. The System informs the <i>Registered User</i> about the failure.
<i>Special Requirements</i>
<ul style="list-style-type: none"> – The <i>FAO Document Repository</i> contains the metadata needed to filter the documents by domain and language. – Concepts are stored as ontology classes. – Terms are linguistic information associated to a concept stored within a non-ontological resource.

(continues on next page)

(comes from previous page)

<i>Frequency of Occurrence</i>
High

Table 13.3: *Corpus Analysis* use case

13.1.3 Task 2. To Identify Application Characteristics and Ontological Needs

In this task, the FAO development team has identified the semantic application characteristics and constraints. Tables 13.4, 13.5, 13.6 and 13.7 show the applicability of the characteristics and the motivation behind their selection.

Characteristic	Applicability	Motivations
Use of internal or external ontologies	Internal	The application does not consume ontologies produced by other applications.
Generation of new ontologies	Yes	From this system users maybe export ontologies that can be consumed by other applications or have different uses.
Centralization or distribution of ontologies	Distribution	The AGROVOC concept server application may be replicated and may access distributed ontologies. First releases of the system may use a centralised repository, but further extensions will make use of distributed repositories.
Ontologies Dynamicity	Yes	Ontologies will change at run-time with the addition of new concepts.
Scalability regarding the number of ontologies	Yes	The application is planned to use multiple ontologies or modules. These may be very large.
Use of a single ontology or a network of ontologies	Network of ontologies	The application will make use of remote distributed ontologies.

Table 13.4: Characteristics of the FAO case study with respect to the dimension of the ontologies.

Characteristic	Applicability	Motivations
Use of internal or external data sources	Both	The application may make use of external documents and associated annotations.
Data Distribution	Yes	Instances may be distributed.
Data Dynamicity	Yes	Data will change over time.
Data scalability	Yes	Data may not only a lot, but in multiple languages.
Data Encoding Heterogeneity	Yes	Mostly the system will make use of similar textual data expressed according to different formats.

Table 13.5: Characteristics of the FAO case study with respect to the data dimension.

Characteristic	Applicability	Motivations
Kind of semantic reasoning	Subsumption	The application will reason with taxonomies.
Hybrid reasoning	Yes	The application will make use of linguistic techniques for ontology learning
Dealing with contradictory data	No	The data that the application will make use of will not be contradictory.
Dealing with incomplete data	No	The application will assume that the information processed will be always complete.

Table 13.6: Characteristics of the FAO case study with respect to the reasoning dimension.

Characteristic	Applicability	Motivations
Interoperability with other applications	Yes	The application will interoperate with existing FAO Document repositories and with other External Providers.

Table 13.7: Characteristics of the FAO case study with respect to the non-functional dimension.

13.1.4 Task 3. To Identify System Models

The typology of the resources that the application will deal with are shown in Table 13.8 together with its associated basic symbols taken from Section 8.1.

Table 13.9 shows the identified relationships between the basic resources obtained from Section 8.2.

The first use case, *Search Concept* is a realisation of the use cases templates *UCT1 (Query Information)*, *UCT2 (Search Resources)* and *UCT3 (Browse Resources)*. Thus the following system model templates will be adapted in the application system model:

- *Search Resources (System Model 4)* for searching the concepts in the AGROVOC ontologies using the annotations (or terms) of those concepts.
- *Browse Annotated Resources (System Model 5)* for browsing the terms associated to the concepts in the AGROVOC ontologies.
- *Query Information with a Single Ontology Approach (System Model 1)* when querying each of the external providers when there are no results found within the AGROVOC knowledge base.

The second use case, *Manage Terms* is a realisation of the use cases templates *UCT5 (Manage Knowledge)* with the goal of editing the terms within the AGROVOC knowledge base. Thus the system model template *Edit Annotations (System Model 9)* is included within the application system model.

The third use case, *Corpus Analysis* is a realisation of the use cases templates *UCT2 (Search Resources)* and *UCT5 (Manage Knowledge)*. Thus the following system model templates will be adapted in the application system model:

- *Search Resources (System Model 4)* for searching the documents in the FAO documents repository using the metadata of these documents (language, domain, etc.).
- *Learn (System Model 11)* for learning the AGROVOC ontologies from the searched documents within the FAO documents repository.

By integrating the basic symbols with their relationships the final system model is the one shown in Figure 13.2.

Resource Identifier	Resource Description	Basic Symbol
AGROVOC Ontologies	Ontologies containing the AGROVOC concepts	1. Static Ontology
AGROVOC Terms	Multilingual terms associated to the AGROVOC concepts	4. Static Non-Ontological Resource Content
Linguistic Model	Schema of the multilingual terms associated to the AGROVOC concepts	3. Static Non-ontological Resource Schema
FAO Documents	Text documents stored in the FAO documents repository	5. Static Unstructured Document
FAO Documents Metadata	Metadata about text documents stored in the FAO documents repository	5. Static Non-ontological Resource Content
FAO Documents Metadata Schema	Schema of the Metadata about text documents stored in the FAO documents repository	5. Static Non-ontological Resource Schema
External Contents	External non-semantic data resources	4. Static Non-Ontological Resource Content
External Resources Schema	Schema of the external contents	3. Static Non-ontological Resource Schema
External Instances	Instances stored in external semantic resources	2. Static Instances
External Ontologies	Ontologies stored in external semantic resources	1. Static Ontology

Table 13.8: Symbols associated to the resources used by the AGROVOC Concept Server application

Resource 1	Resource 2	Relationship
AGROVOC Terms	Linguistic Model	2. Non-ontological Resource Content that Conforms To a Given Schema
AGROVOC Terms	AGROVOC Ontologies	8. Ontology Annotated by a Non-ontological Metadata
Documents Metadata	FAO Documents	6. Unstructured Document Annotated by a Non-ontological Metadata
Documents Metadata	Documents Metadata Schema	2. Non-ontological Resource Content that Conforms To a Given Schema
External Contents	External Resources Schemas	2. Non-ontological Resource Content that Conforms To a Given Schema
External Instances	External Ontologies	1. Instances that Conforms To a Given Ontology

Table 13.9: Relationships between the resources used by the example application

13.2 Component Identification Activity

This section presents how to carry out the three tasks of the *Component Identification* activity, taking into account the use cases and system model obtained in the previous section.

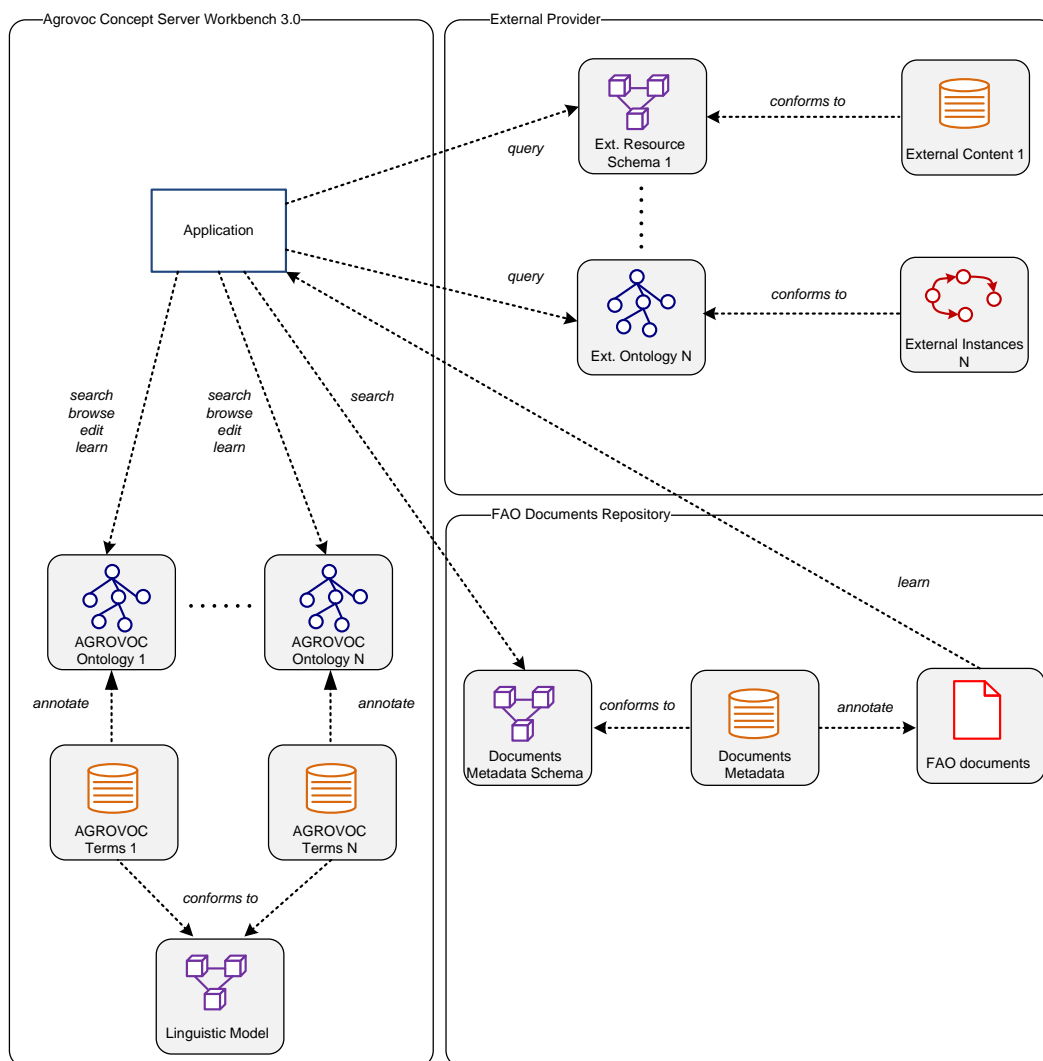


Figure 13.2: Identified system model for the FAO case study

13.2.1 Task 1. To Identify Dialogs and System Facades

Within this task the development team introduces in the architecture a system dialog and a facade for each use case identified. The components obtained are the following:

- **Search Concept Dialog** component. It implements the *Search Concept* use case logic, that is, the software that handles the dialog between the *User* and the system according to the specified use case. This component will make use of the *Search Concept Facade* component.
- **Search Concept Facade** component. For every step within the *Search Concept* use case, it will provide operations to meet its responsibilities.
- **Manage Terms Dialog** component. It implements the *Manage Terms* use case logic, that is, the software that handles the dialog between the *Registered User* and the system, according to the specified use case. This component will make use of the *Manage Terms Facade* component.
- **Manage Terms Facade** component. For every step within the *Manage Terms* use case, it will provide operations to meet its responsibilities
- **Corpus Analysis Dialog** component. It implements the *Corpus Analysis* use case logic, that is,

the software that handles the dialog between the *Registered User* and the system, according to the specified use case. This component will make use of the *Corpus Analysis Facade* component.

- **Corpus Analysis Facade** component. For every step within the *Corpus Analysis* use case, it will provide operations to meet its responsibilities

13.2.2 Task 2. To Identify Interfaces to Knowledge Sources

Within this task the development team catalogues the sources or repositories containing the ontological and non-ontological data that the application will make use of. For each ontological and non-ontological resource reflected in the previously system model obtained, its containing repository is identified. Table 13.10 shows the system and patterns associated to each resource.

Resource	System	Pattern
AGROVOC Ontologies	AGROVOC Concept Server Workbench 3.0	1. Ontology Repository
AGROVOC Terms	AGROVOC Concept Server Workbench 3.0	2. Data Repository
Linguistic Model	AGROVOC Concept Server Workbench 3.0	2. Data Repository
FAO Documents	FAO Documents Repository	2. Data Repository
FAO Documents Meta-data	FAO Documents Repository	2. Data Repository
FAO Documents Meta-data Schema	FAO Documents Repository	2. Data Repository
External Contents	External Provider	2. Data Repository
External Resources Schema	External Provider	2. Data Repository
External Instances	External Provider	1. Ontology Repository
External Ontologies	External Provider	1. Ontology Repository

Table 13.10: Patterns associated to the repositories used by the FAO application

13.2.3 Task 3. To Create the Initial Architecture

The architecture obtained by integrating all the components and patterns has been divided into three different figures (13.3, 13.4 and 13.5) in order to be clearly visualized. Each of these figures show the architecture corresponding to each of the three use cases.

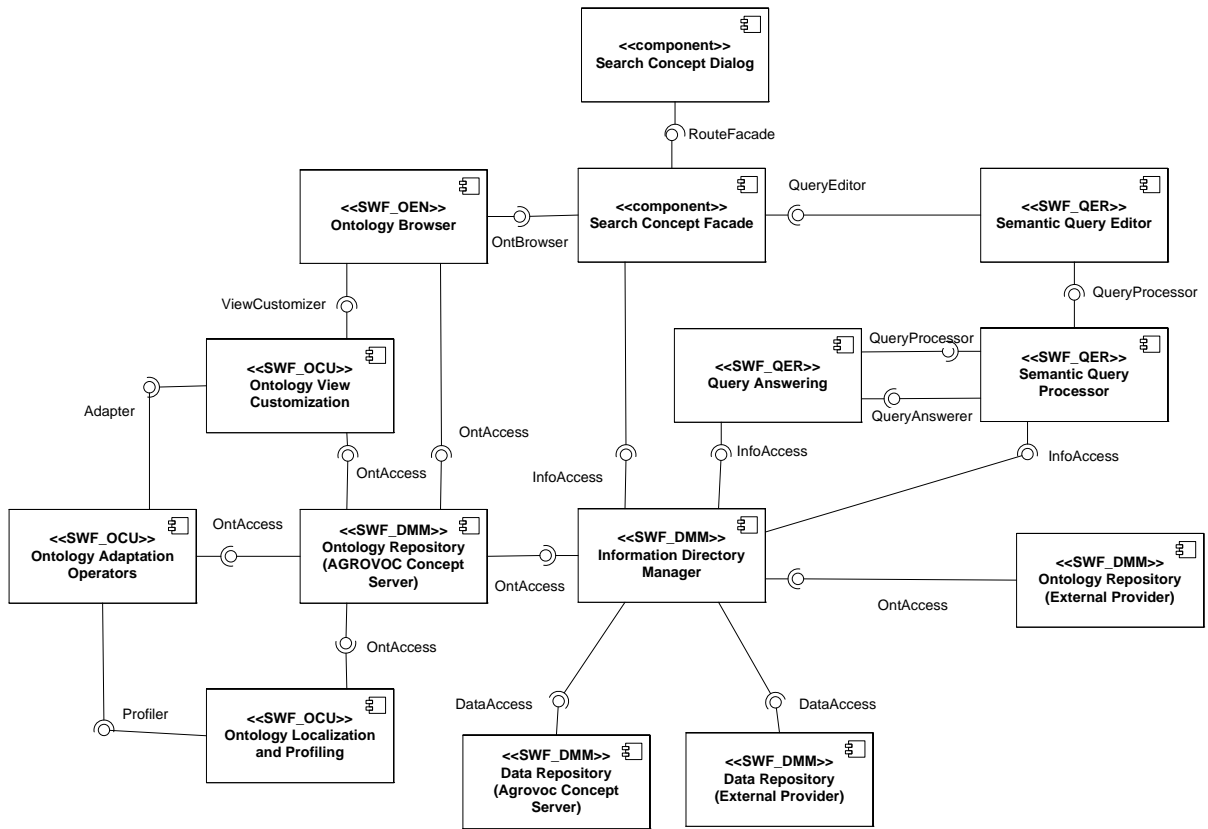


Figure 13.3: FAO Application Architecture (Search Concept)

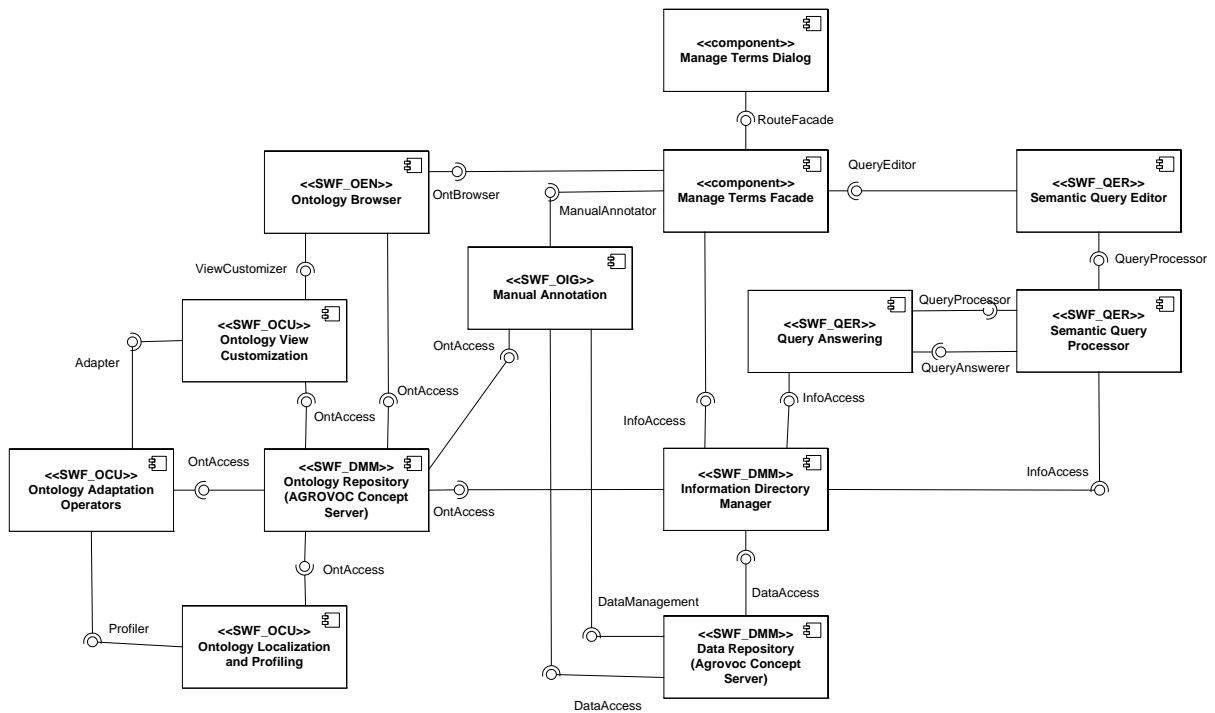


Figure 13.4: FAO Application Architecture (Manage Terms)

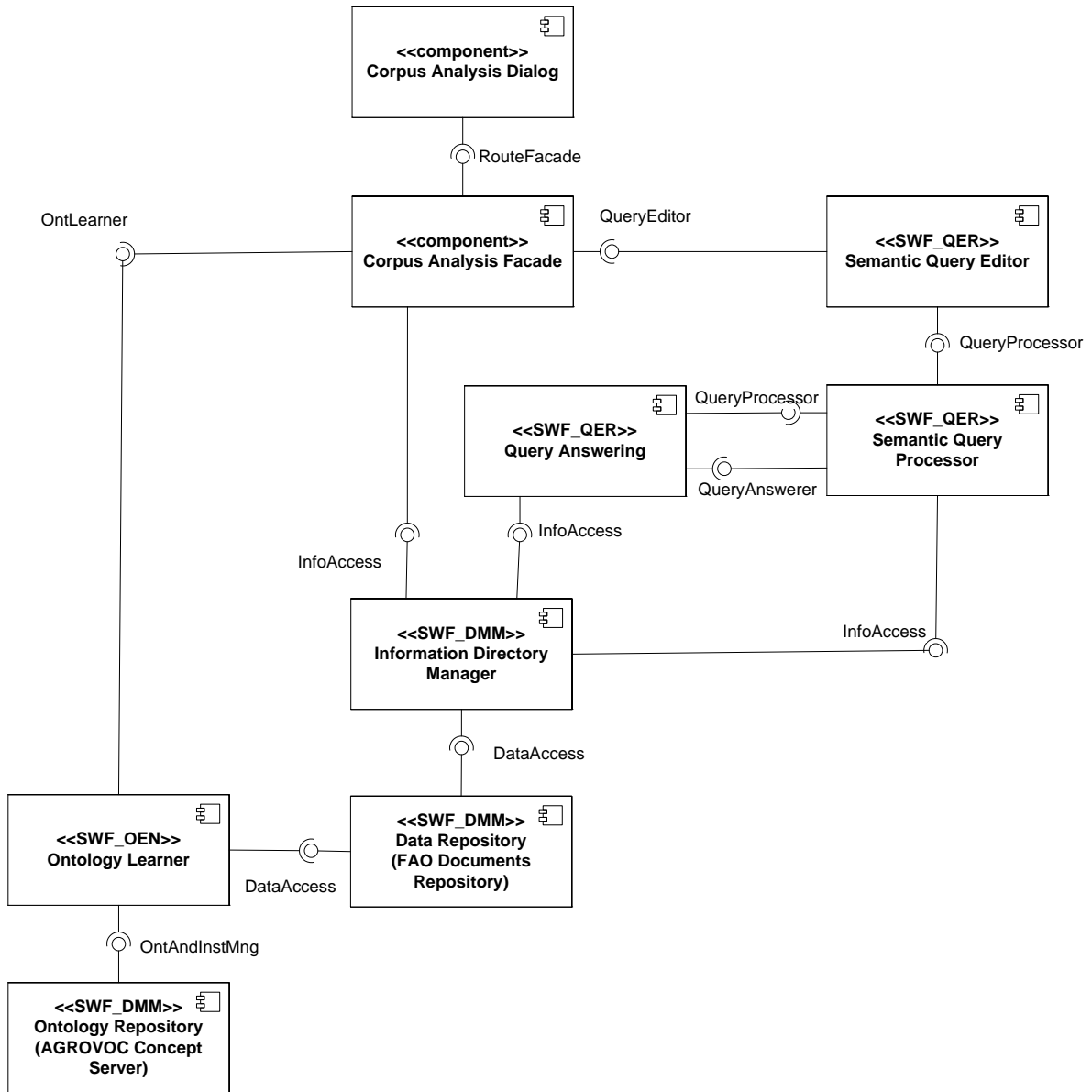


Figure 13.5: FAO Application Architecture (Corpus Analysis)

Chapter 14

Conclusions and Future Work

The development of applications that contain semantic functionalities is difficult for Software Engineers outside the semantic community.

With this first version of the NeOn methodology for the development of large-scale semantic applications, we aim to facilitate the adoption of semantic technologies to Software Engineers.

For this purpose, we have adapted the *Requirements Engineering* and *Design* processes and different techniques from software engineering and we have provide a set of catalogues.

We have analyzed the state of the art of classifications and categorisations of semantic applications in order to provide 32 common characteristics and constraints of semantic applications that can be selected and included in a requirements document as non-functional requirements. The characteristics and constraints associated to a given large-scale semantic application will impact during the whole development process.

The methodology adopts the use cases technique and provides 5 use case templates by adapting state of the art scenarios for building semantic applications from the viewpoint of the user's goals. This is useful for start analysing the application functional requirements from the user's concrete functional requirements.

In addition, this deliverable provides a new way to make a graphical representation of the macro-structure and environment of a semantic application by describing 11 system models, basic templates, relationships and basic symbols. The system models can be selected once the previously mentioned application characteristics and constraints and use-cases are identified.

We also provide 28 architectural patterns with the aim of facilitating the architectural design of a large-scale semantic application. The components involved in these architectural patterns have been obtained from the Semantic Web Framework [GGMN08] defined in the context of the Knowledge Web Project (FP6-507482). The architectural patterns can be selected once the system model of the application have been identified.

The methodology presented in this deliverable has not been evaluated yet in any large-scale development project finished. A first evaluation of the methodology will be provided by the NeOn case studies and included in the following version of the methodology.

The catalogues and patterns presented in the deliverable can be extended, and for this purpose a collaborative space (e.g. a wiki) will be enabled to facilitate the feedback, extension and enrichment to the community. Immediate lines of work include continuing defining the rest of the processes: *Provisioning and Development*, *Integration* and *Testing*.

Another future line of work is to extend the methodology for integrate large-scale semantic web applications with other paradigms such the Web 2.0 and to specialize the methodology in order to specifically deal with particular settings and tendencies such as the Open Linked Data initiative¹ or the Semantic Grid.

Also, an interesting result should be to give software support to the methodology by building or adapting an existing CASE tool and by formalising the processes, activities, methods, catalogues and patterns of the methodology with ontologies with the aim of automatically documenting the large-scale semantic application development process, or of supporting the application code generation. For this last point, it is necessary to

¹<http://linkeddata.org/>

define the rest of the processes and to provide interoperable implementations of the components that can be involved in the semantic application.

The questionnaires will be also used to characterize and categorize existing semantic applications to realize an analysis of the current panorama of semantic applications.

Appendix A

Questionnaires for Identifying the Characteristics of Semantic Applications

This section provides a set of questionnaires that can be used by the application developers to identify the characteristics of a semantic application given during their interviews with the customer or to identify the characteristics of an existing semantic application. The questions are also valid for identifying the ontologies and data sets that the application will deal with and to detect the ontological needs.

The questions are grouped into the following questionnaires:

- *Questionnaire about Ontologies.* To help the application developers to determine the characteristics of the ontologies that the application will make use of.
- *Questionnaire about Data.* To help the application developers to determine the characteristics of the data that the application will consume or manipulate and its relation with the ontologies or data schemas which data may conform to.
- *Questionnaire about Reasoning.* To help the application developers to determine the characteristics of the reasoning that the application will apply to the ontologies and data.
- *Questionnaire about Non-functional Characteristics.* To help the application developers to determine the other non-functional characteristics of the application with other applications.

Bellow, each of the questionnaires is provided. For each question their purpose and possible answers are provided. A question may have secondary questions, if needed, to specify the answer more.

Questionnaire 1 Characteristics of the ontologies

The following questions must be answered by considering the whole set of ontologies that the application will use.

1. Will the application combine ontologies for being exploited jointly or each ontology will be processed by separate?
 - A Jointly
 - B By separate
 - C Mixed
 - D Unknown

Characteristic: *Use of a single ontology or a network of ontologies.*

2. Will be the ontologies be identified by developers at design-time or located by the application at run-time?
 - A Design-time
 - B Run-time
 - C Mixed
 - D Unknown

Characteristic: *Design-time or run-time ontology selection.*

- a) If the answer is "run-time" or "mixed", specify the selection criteria.
 - A Quality
 - B Complexity
 - C Richness
 - D User's evaluation
 - E Unknown
 - F Other(s): _____

Purpose: to identify the run-time selection criteria.

3. Are all of the ontologies bound to the same particular domain?
 - A Yes
 - B No
 - C Unknown

Characteristic: *Use of generic or domain-specific ontologies.*

4. Will the application produce new ontologies?
 - A Yes
 - B No
 - C Unknown

Characteristic: *Generation of new ontologies.*

5. Will the ontologies be centralized in a single resource or distributed?
 - A Centralized
 - B Distributed
 - C Unknown

Characteristic: *Centralization or distribution of ontologies.*

6. How many ontologies will the application deal with?

- A 1
- B 2-5
- C 5-15
- D 15-50
- E 50-
- F Unknown

Characteristic: *Scalability with respect to the number of ontologies.*

The following questions must be answered by considering each of the existing ontologies.

7. Which is the domain described by the ontology?

- A _____
- B Unknown

Purpose: to identify the domain described by the ontology.

8. Is the ontology obtained from an external resource?

- A Yes. Specify from where: _____.
- B No
- C Unknown

Characteristic: *Use of internal or external ontologies* (when at least one ontology is obtained from an external resource).

Purpose: to identify the external resource from where the ontology have been obtained.

9. Does the ontology need to be reengineered?

- A Yes
- B No
- C Unknown

Purpose: To detect the ontological need of reengineering the considered ontology.

Characteristic: *Ontologies reuse and reengineering.*

10. Which will be the purpose of the ontology within the application?

- A Its own consumption
- B External consumption
- C Both
- D Unknown

Purpose: to identify if the ontology will be consumed by the application, by other application, or by both.

11. Will the ontology change during application execution time?

- A Yes
- B No
- C Unknown

Characteristic: *Ontology Dynamicity* (for the considered ontology).

12. How many ontology elements does the ontology contains?

- A -50
- B 50-100
- C 100-200
- D 200-1000
- E 1000-
- F Unknown

Characteristic: *Scalability with respect to the number of ontology elements* (for the considered ontology).

13. In which ontology language is the ontology formalized?

- A RDFS
- B OWL
- C Other: _____
- D Unknown

Purpose: to identify the ontology language in which the ontology is formalized

Characteristic: *Ontologies encoding heterogeneity* (if different ontology languages are used for different ontologies).

14. Is there any need of aligning the ontology with other ontologies due to conceptual heterogeneity?

- A Yes (specify the ontologies that need to be aligned with the ontology considered):

- B No
- C Unknown

Purpose: to identify the ontologies that need to be aligned with the ontology considered.

Characteristic: *Ability to resolve conceptual heterogeneity in ontologies* (when at least two ontologies need to be aligned).

a) If the answer is "Yes", is there any existing alignment?

- A Yes (specify the existing alignments): _____
- B No
- C Unknown

Purpose: to discover the existing alignments.

b) Specify the alignments that need to be created.

- A _____
- B Unknown

Purpose: to identify the ontological need of creating new alignments.

i – Specify for each of the previous alignments when they will be created.

- A Design-time: _____
- B Run-time: _____
- C Unknown: _____

Purpose: to identify if each of the alignments will be created at design-time or automatically at run-time.

Questionnaire 2 Characteristics of the data

The following questions must be answered by taking into account the whole set of data sets (or collections of data) that the application will access or manage.

1. Are all of the data sets bound to the same particular domain?

A Yes
B No
C Unknown

Characteristic: *Data domain dependence.*

2. Will the application produce new data sets?

A Yes
B No
C Unknown

Characteristic: *Data generation.*

3. Will the data be linked?

A Yes
B No
C Unknown

Characteristic: *Use of linked data.*

4. Will the existing data sets be centralized in a single resource or distributed in multiple resources?

A Centralized
B Distributed
C Unknown

Characteristic: *Data distribution.*

5. Will the data sets be identified by developers at design-time or located by the application at run-time?

A Design-time
B Run-time
C Mixed
D Unknown

Characteristic: *Design-time or run-time data selection.*

6. How many data sets will the application deal with?

A 1
B 2-5
C 5-15
D 15-50
E 50-
F Unknown

Characteristic: *Data scalability.*

7. Will the application aggregate non-semantic data?

- A Yes
- B No
- C Unknown

Characteristic: *Use of non-semantic data.*

The following questions must be answered by considering each of the data sets.

8. Which will be the purpose of the data set within the application?
- A Own consumption
 - B External consumption
 - C Both
 - D Unknown

Purpose: to identify if the data set will be consumed by the application, by other application or by both.

9. Will the data set be taken from an external resource?
- A Yes. Specify from where: _____
 - B No
 - C Unknown

Characteristic: *Use of internal or external data sources* (when at least one data source is obtained from an external resource). Purpose: to identify the external resource from where the ontology have been obtained.

10. Will the data set be changing during the application execution time?
- A Yes
 - B No
 - C Unknown

Characteristic: *Data dynamicity* (for the data set considered).

11. Which will be the size of the data set?
- A Various KB
 - B Various MB
 - C Various GB
 - D Various TB
 - E More than various TB
 - F Unknown

Characteristic: *Data scalability* (for the data set considered).

12. Which will be the format in which triples will be expressed?
- A RDF/XML
 - B N3
 - C Other: _____
 - D Unknown

Purpose: to identify the format in which the data set is represented.

Characteristic: *Data encoding heterogeneity* (if different formats are used for multiple data sets).

13. Will the data conform to a given ontology or to another kind of non-ontological data schema?

A Yes, it conforms to an ontology. Specify the ontology: _____

B Yes, it conforms to a non-ontological data schema. Specify the non-ontological data schema:

C No, it is unstructured data.

D Unknown

Purpose: to identify the ontology or schema which the data set conforms to.

a) If the answer is “conforms to a non-ontological data schema”, will the data set be reengineered for obtaining an ontological resource?

A Yes

B No

C Unknown

Purpose: to detect the ontological need of performing a non-ontological resource reengineering.

i – If the answer is “no”, will the application integrate the data set schema in order to exploit jointly the ontologies and non-semantic data?

A Yes

B No

C Unknown

Purpose: to determine whether data should be treated as ontology instances.

Questionnaire 3 Characteristics of reasoning

1. Do you know at this stage what are the kinds of semantic reasoning that the application will apply?

- A Consistency checking
- B Inference of new data
- C Automatic classification
- D Instances classification
- E Subsumption reasoning
- F Unknown

Characteristic: *Kind of semantic reasoning.*

2. Do you know at this stage if the application will apply sound reasoning?

- A Yes, the application will apply sound reasoning
- B Yes, the application won't apply sound reasoning.
- C No

Characteristic: *Sound reasoning.*

3. Do you know at this stage if the application will apply complete reasoning?

- A Yes, the application will apply complete reasoning
- B Yes, the application won't apply complete reasoning
- C No

Characteristic: *Complete reasoning.*

4. Do you know at this stage if the application will have to reason simultaneously with knowledge sources with non-ontological nature and with ontological sources?

- A Yes, by applying machine learning techniques
- B Yes, by applying linguistic techniques
- C Yes, by applying statistical techniques
- D Yes, by applying graph matching techniques
- E Yes, by applying _____techniques
- F No

Characteristic: *Hybrid reasoning.*

5. Will the application deal with contradictory data?

- A Yes
- B No
- C Unknown

Characteristic: *Reasoning with contradictory data.*

6. Will the application deal with incomplete data?

- A Yes
- B No
- C Unknown

Characteristic: *Reasoning with incomplete data.*

7. Do you know at this stage if the application will reason with uncertainty?

- A Yes, the application will reason with uncertainty
- B Yes, the application won't reason with uncertainty
- C No

Characteristic: *Reasoning with uncertainty.*

8. Do you know at this stage if the application will distribute the reasoning?

- A Yes, the application will distribute the reasoning
- B Yes, the application won't distribute the reasoning
- C No

Characteristic: *Distributed reasoning.*

Questionnaire 4 Non-functional characteristics

1. Will the application access programmatically other applications?

- A Yes
- B No
- C Unknown

Characteristic: *Interoperability with other applications.*

a) If the answer is “yes”, specify the kind of interfaces that the external systems provide to facilitate the interoperability.

- A API
- B Web Service
- C Semantic Web Service
- D Other: _____

Purpose: to identify the mechanism that facilitates the interoperability when the application access programmatically other application.

2. Will other applications access programmatically to the application?

- A Yes
- B No
- C Unknown

Characteristic: *Interoperability with other applications.*

a) If the answer is “yes”, specify the kind of interfaces that the external systems require to facilitate the interoperability.

- A API
- B Web Service
- C Semantic Web Service
- D Other: _____

Purpose: to identify the mechanism that facilitates the interoperability when other application access programmatically the application.

Bibliography

- [AMS⁺08] Mathieu d' Aquin, Enrico Motta, Marta Sabou, Sofia Angeletou, Laurian Gridinoc, Vanessa Lopez, and Davide Guidi. Towards a New Generation of Semantic Web Applications. *IEEE Intelligent Systems*, 23(3), May/June 2008.
- [BBB⁺01] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development. <http://www.agilemanifesto.org/>, 2001.
- [BCKB03] Len Bass, Paul Clements, Rick Kazman, and Ken Bass. *Software Architecture in Practice*, chapter 11. Addison-Wesley, Boston, second edition, 2003.
- [Bec99a] Kent Beck. Embrace Change with Extreme Programming. *IEEE Computer*, pages 70–77, October 1999.
- [Bec99b] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, October 1999.
- [Boh79] Barry W. Bohem. Software Engineering; R & D trends and defense needs. In P. Wegner, editor, *Research Directions in Software Technology (Ch. 22)*, pages 1–9, Cambridge, MA, 1979. MIT Press.
- [CD01] John Cheesman and John Daniels. *UML Components. A Simple Process for Specifying Component-Based Software*. Component Software Series. Addison-Wesley, 2001.
- [CLC03] David Cohen, Mikael Lindvall, and Patricia Costa. Agile Software Development. State-of-the-art report, Data and Analysis Center for Software, 775 Daedalian Dr. Rome, New York 13441-4909, January 2003.
- [Coc00] Alistair Cockburn. Selecting a project's methodology. *IEEE Software*, 17(4):64–71, 2000.
- [DF08] John Domingue and Dieter Fensel. Towards a Service Web: Integrating the Semantic Web and Service Orientation. *IEEE Intelligent Systems*, January 2008.
- [FKN⁺92] A. Finkelsetin, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2, 1992.
- [FM01] Dieter Fensel and Enrico Motta. Structured Development of Problem Solving Methods. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):913–932, 2001.
- [GGMN08] Raúl García-Castro, Asunción Gómez-Pérez, Oscar Muñoz-García, and Lyndon J.B. Nixon. Towards a Component-Based Framework for Developing Semantic Web Applications. In J. Domingue and C. Anutariya, editors, *3rd Asian Semantic Web Conference (ASWC 2008)*, Lecture Notes in Computer Science, pages 197–211, Bangkok, Thailand, December 2008. Springer-Verlag.

- [GMG⁺07] Raúl García-Castro, Oscar Muñoz-García, Asunción Gómez-Pérez, Stefania Costache, Diana Maynard, Stamatia Dasiopoulou, Raúl Palma, Vit Novacek, Freddy Lécué, Ying Ding, Monika Kaczmarek, Ruzica Piskac, Dominik Zyskowski, Jérôme Euzenat, and Martin Džbor. D1.2.5. Architecture of the Semantic Web Framework v2. Technical report, Knowledge Web, December 2007.
- [Gru93] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–200, 1993.
- [GS08] Asunción Gómez-Pérez and Mari Carmen Suárez-Figueroa. NeOn Methodology: Scenarios for Building Networks of Ontologies. /16th International Conference on Knowledge Engineering and Knowledge Management Knowledge Patterns (EKAW 2008). Conference Poster, Italy, 2008.
- [Hig02] Jim Highsmith. *Agile Software Development Ecosystems*. The Agile Software Development Series. Addison-Wesley Professional, April 2002.
- [HOC00] Jim Highsmith, Ken Orr, and Alistair Cockburn. Extreme Programming. E-business application delivery, Cutter Consortium, February 2000.
- [IEE90] IEEE. IEEE Standard Glossary of Data Management Terminology. IEEE Standard 610.5-1990, Standards Coordinating Committee of the IEEE Computer Society, August 1990.
- [IEE95a] IEEE. IEEE Guide for Software Quality Assurance Planning. IEEE Standard 730.1-1995, Software Engineering Standards Committee of the IEEE Computer Society, December 1995.
- [IEE95b] IEEE. IEEE Standard for Developing Software Life Cycle Processes. IEEE Standard 1074-1995, IEEE Computer Society, September 1995.
- [IEE97] IEEE. IEEE Standard for Developing Software Life Cycle Processes. IEEE Standard 1074-1997, IEEE Computer Society, December 1997.
- [IEE98] IEEE. *IEEE Recommended practice for software requirements specifications*, chapter 6. Number IEEE/ANSI 830-1998 in IEEE Software Engineering Standards Collection. IEEE Computer Society Press, Los Alamitos, CA, 1998.
- [Jac92] Ivar Jacobsen. *Object Oriented Software Engineering: A Use Case Driven Approach*. ACM Press. Addison-Wesley Professional, July 1992.
- [Joh97] Ralph E. Johnson. Frameworks = (Components + Patterns). *Communications of the ACM*, 40(10):39–42, October 1997.
- [JU99] Robert Jasper and Mike Uschold. A Framework for Understanding and Classifying Ontology Applications. In *Twelfth Workshop on Knowledge Acquisition Modeling and Management KAW 99*, 1999.
- [KHS⁺08] Kouji Kozaki, Yusuke Hayashi, Munehiko Sasajima, Shinya Tarumi, and Riichiro Mizoguchi. Understanding Semantic Web Applications. In *3rd Asian Semantic Web Conference (ASWC 2008)*, 2008.
- [Kru92] Charles W. Krueger. Software Reuse. *ACM Comput. Surveys*, 24(2):131–183, June 1992.
- [Kru00] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Object Technology. Addison-Wesley, second edition, March 2000.
- [KSF08] Reto Krümmenacher, Elena Simperl, and Dieter Fensel. Scalability in Semantic Computing: Semantic Middleware. In *Proceedings of the IEEE Conference on Semantic Computing*, pages 538–544, 2008.

- [Lar05] Craig Larman. *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Software Engineering/Object-Oriented Analysis and Design. Prentice Hall, Upper Saddle River, NJ 07458, third edition, August 2005.
- [Man07] Christoph Mangold. A survey and classification of semantic search approaches. *International Journal of Metadata, Semantics and Ontologies*, 2(1):23–34, January 2007.
- [Mot99] Enrico Motta. *Reusable Components for Knowledge Modelling*. IOS Press, Amsterdam, The Netherlands, 1999.
- [MS06] Enrico Motta and Marta Sabou. Next Generation Semantic Web Applications. In *1st Asian Semantic Web Conference*, Beijing, September 2006.
- [Obe06] Daniel Oberle. *Semantic Management of Middleware*. Semantic Web and Beyond. Springer, 2006.
- [Par01] Luis F. Paradela-González. *Una Metodología para la Gestión del Conocimiento*. PhD thesis, Universidad Politécnica de Madrid, Madrid, Spain, 2001.
- [PF02] Stephen R. Palmer and John M. Felsing. *A Practical Guide to Feature-Driven Development*. The Coad Series. Prentice Hall PTR, February 2002.
- [SAA⁺00] Guus Schreiber, Hans Akkermans, Anjo Anjewierden, Robert de Hoog, Nigel Shadbolt, Walter Van de Velde, and Bob Wielinga. *Engineering and Managing Knowledge: The CommonKADS Methodology*. M.I.T. Press, 2000.
- [SAA⁺07] Marta Sabou, Sofia Angeletou, Mathieu d' Aquin, Jesús Barrasa, Klaas Dellschaft, Aldo Gangemi, Jos Lehmann, Holger Lewen, Diana Maynard, Dunja Mladenic, Malvina Nissim, Wim Peters, Valentina Presutti, and Boris Villazón. D2.2.1 Methods for Selection and Integration of Reusable Components from Formal or Informal User Specifications. Technical report, NeOn Project, April 2007.
- [SAB⁺07] Mari Carmen Suárez-Figueroa, Guadalupe Aguado de Cea, Carlos Buil, Caterina Caracciolo, Martin Dzbor, Asunción Gómez-Pérez, German Herrero, Holger Lewen, Elena Montiel-Ponsoda, and Valentina Presutti. D5.3.1. NeOn Development Process and Ontology Life Cycle. Technical report, NeOn Project, August 2007.
- [SAB⁺08] Mari Carmen Suárez-Figueroa, Guadalupe Aguado de Cea, Carlos Buil, Klaas Dellschaft, Mariano Fernández-López, Andrés García, Asunción Gómez-Pérez, German Herrero, Elena Montiel-Ponsoda, Marta Sabou, Boris Villazón-Terrazas, and Zheng Yufei. D5.4.1. NeOn Methodology for Building Contextualized Ontology Networks. Technical report, NeOn Project, February 2008.
- [Sch01] Ken Schwaber. *Agile Software Development with SCRUM*. Agile Software Development. Prentice Hall, October 2001.
- [SFG⁺08] Mari Carmen Suárez-Figueroa, Mariano Fernández-López, Asunción Gómez-Pérez, Klaas Dellschaft, Holger Lewen, and Martin Dzbor. D5.3.2 Revision and Extension of the NeOn Development Process and Ontology Life Cycle. Technical report, NeOn Project, November 2008.
- [Som07] Ian Sommerville. *Software Engineering*. International Computer Science Series. Addison-Wesley, eighth edition, 2007.
- [SW99] Desmond Francis D' Souza and Alan Cameron Wills. *Objects, Components, and Frameworks with UML. The Catalysis Approach*. Object Technology. Addison-Wesley, 1999.
- [Szy98] Clemens Szyperski. *Component Software, Beyond Object Oriented Programming*. Addison-Wesley, 1998.

- [WVV⁺01] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information - a survey of existing approaches. In *IJCAI workshop on Ontologies and Information Sharing*, pages 108–117, 2001.