NeOn-project.org

NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 – "Semantic-based knowledge and content systems"

D 6.1.1 Requirements on NeOn Architecture

Deliverable Co-ordinator: **Jose Manuel Gómez (iSOCO)**

Deliverable Co-ordinating Institution: **Intelligent Software Components (iSOCO)**

Other Authors: **Carlos Buil Aranda (iSOCO)**

Oscar Muñoz García (UPM)

Walter Waterfeld (Software AG)

Document Identifier:	NEON/2006/D6.1.1/v1.0	Date due:	August 31, 2006
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	**31 August 2006**
Project start date:	March 1, 2006	Version:	V1.0
Project duration:	4 years	State:	Final
		Distribution:	Report/Public

NeOn Consortium

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

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

Universidad Politécnica di Madrid (UPM)
Campus de Montegancedo
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

Intelligent Software Components S.A. (ISOCO)
Calle de Pedro de Valdivia 10
28006 Madrid
Spain
Contact person: Richard Benjamins
E-mail address: rbenjamins@isoco.com

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST – 655 avenue de l'Europe
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: jerome.euzenat@inrialpes.fr

Universität Koblenz-Landau (UKO-LD)
Universitätsstrasse 1
56070 Koblenz
Germany
Contact person: Steffen Staab
E-mail address: staab@uni-koblenz.de

Ontoprise GmbH. (ONTO)
Amalienbadstr. 36
(Raumfabrik 29)
76227 Karlsruhe
Germany
Contact person: Jürgen Angele
E-mail address: angele@ontoprise.de

Food and Agriculture Organization of the UN (FAO)
Viale delle Terme di Caracalla 1
00100 Rome
Italy
Contact person: Johannes Keizer
E-mail address: johannes.keizer@fao.org

Universität Karlsruhe – TH (UKARL)
Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB
Englerstrasse 28
D-76128 Karlsruhe, Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

Software AG (SAG)
Uhlandstrasse 12
64297 Darmstadt
Germany
Contact person: Walter Waterfeld
E-mail address: walter.waterfeld@softwareag.com

Institut 'Jožef Stefan' (JSI)
Jamova 39
SI-1000 Ljubljana
Slovenia
Contact person: Marko Grobelnik
E-mail address: marko.grobelnik@ijs.si

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

Consiglio Nazionale delle Ricerche (CNR)
Institute of cognitive sciences and technologies
Via S. Martino della Battaglia,
44 - 00185 Roma-Lazio, Italy
Contact person: Aldo Gangemi
E-mail address: aldo.gangemi@istc.cnr.it

Asociación Española de Comercio Electrónico (AECE)
C/Alcalde Barnils, Avenida Diagonal 437
08036 Barcelona
Spain
Contact person: Gloria Tort
E-mail address: gtort@fecemd.org

Atos Origin S.A. (ATOS)
Calle de Albarracín, 25
28037 Madrid
Spain
Contact person: Tomás Pariente
E-mail address: tomas.parientelobo@atosorigin.com

Table of contents

LIST OF TABLES

LIST OF FIGURES

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 writing parts of this document:

iSOCO

UPM

Ontoprise

Software AG

FAO

UKARL

Change Log

Version	Date	Amended by	Changes
0.1	31-07-2006	Carlos Buil-Aranda	Description of changes
0.2	18-08-2006	Jose Manuel Gómez-Pérez	Draft ready for Quality Assurance
0.3	25-09-2006	Carlos Buil-Aranda	Incorporated feedback
1.0	01-10-2006	Walter Waterfeld	Final version with enhanced risk analysis

Executive Summary

This document shows the requirements for the NeOn architecture and how these requirements have been extracted. The document is divided in two main blocks. The first block describes the methodology that has been used in order to gather the NeOn architecture requirements while the second one details the actual requirements that have been extracted from the different sources.

The methodology followed has lied on two main information sources: the technical annex and the NeOn Requirements and Vision [1] document. The third information source upon which the architectural requirements specification has been built is direct feedback from the consortium. Our aim has been to build the NeOn architecture requirements specification on top of both the expertise of system architecture core partners, technology and methodology providers, and the actual needs of the case studies.

All this information was gathered by means of a requirements sheet circulated among partners as a template in the successive requests for information. A large amount of information was collected whose effective classification demanded a well-defined software requirements specification methodology. With the aim of keeping as close to the standards as possible, we adopted the IEEE [3] approach.

The description of the requirements contained herein aim to analyze current practices in ontology engineering and gather information from the frontline that can be used to delimit strengths and weaknesses of current tools in order to develop a precise understanding of NeOn needs. These requirements aim to feed the design of NeOn architecture and toolkit.

Accordingly to the IEEE software requirement specification, an overall description of the requirements has been produced first. This description provides a perspective on the NeOn architecture and introduces the functions that shall be satisfied by it. Next, this information is analyzed in detail producing a complete collection of specific requirements.

1. Introduction

This document aims to analyze current practices in ontology engineering and gather information from the frontline that can be used to delimit strengths and weaknesses of current tools in order to develop a precise understanding of NeOn needs. These requirements aim to feed the design of NeOn architecture and toolkit. The NeOn architecture shall be modular, service-oriented, and open, enabling case studies as well as research and exploitation partners to accommodate NeOn technology, techniques and methods for ontology development, reasoning and collaboration in networked environments.

This effort shall eventually result in intensive tool support for the mechanisms developed in the technical work packages. Networked ontologies require these tools to be offered as an integrated NeOn infrastructure. Additionally, the collaborative framework inherent to NeOn requires further infrastructure: the distributed repository of networked ontologies and a set of distributing components working as a middleware between the NeOn toolkit itself and the distributed repository.

The distributed repository shall be able to manage networked ontologies and legacy information resources while distributed components shall be provided which offer the necessary functionalities to the application layer, i.e. the NeOn toolkit, such as reasoning or annotation. This NeOn middleware shall be transparent with respect to the particular formalism used for each particular networked, and possibly heterogeneous, ontology managed by NeOn.

NeOn shall provide a service-oriented interface that allows legacy components to be included with the smallest impact on the rest of the system, enforcing standard interface methods which provide polices for NeOn compatibility and extension. However, components built from scratch under the NeOn specification shall rather use more direct, higher performance methods for integration. In this direction, the NeOn architecture shall offer an API accessible by means of several client interfaces.

Functionalities provided by the NeOn infrastructure shall include, but are not limited to, simultaneously visualizing, browsing, and editing multiple ontologies, ontology alignment, and lifecycle support to collaborative ontology development, ontology change propagation, consistency maintenance, and versioning. On the other hand, non functional requirements on the NeOn infrastructure include robustness, scalability, concurrency, and reuse of technology and information resources.

2. NeOn architecture requirements: Methodology

The method followed in order to extract the architectural requirements of NeOn has heavily lied on two main sources: the technical annex and the NeOn Requirements and Vision document [1] . The technical annex contains the main guidelines for the development of NeOn technology and methodology. On the other hand, [1] is the first attempt of the NeOn consortium to define a series of functionalities that shall be satisfied. Additionally, this work produced a series of assumptions in terms of which the former functionalities were specified. Namely, these assumptions are ontology networking, dynamics, sharing, and context.

The third information source upon which the architectural requirements specification has been built is direct feedback from the consortium. First, architecture partners from WP6 were triggered, taking advantage of their specific knowledge on system architecture. Then, the request for information was extended to the rest of the partners, especially the case studies. Our aim was to complete the vision provided by the system architecture core partners with the specification of the requirements which eventually need to be satisfied in order to sustain the architectural needs of the case studies.

All this information was gathered by means of a requirements sheet circulated among partners as a template in the successive requests for information. A large amount of information was collected whose effective classification demanded a well-defined software requirements specification methodology. With the aim of keeping as close to the standards as possible, we adopted the IEEE approach [3] .

Next, we describe how we extracted information from our main textual sources i.e. [1] and the technical annex, for requirements gathering. Then, the tools used during requirements gathering are described. Finally, the IEEE software requirements specification methodology and its application to the actual NeOn architecture requirement classification are described.

2.1 Analysis of textual sources

NeOn requirements and vision

[1] Contains a large amount of valuable information for the NeOn architecture requirements. This information is mainly focused on the following three points:

- Data to be managed using the NeOn infrastructure, i.e. networked ontologies.
- Functionalities that need to be available throughout the entire lifecycle of networked ontologies.
- Users that need the NeOn infrastructure to develop networked ontologies.

The following key characteristics are assumed: ontologies will be networked, dynamic, shared and contextualized. These assumptions shall also be present in the architecture requirements extracted from this source. The NeOn architecture shall provide functionalities necessary to manage networked ontologies as well as their properties.

Another aspect to be reflected by the architecture requirements is the different types of ontology users. According to [1] users can be classified under two main categories, namely ontology experts and ontology editors. Ontology experts develop ontologies with the aim of modelling a given domain. However, not necessarily do they have deep knowledge about particular domains. On the other hand, ontology editors have a large expertise on the concrete domain to be modelled. Therefore, both have diverse needs which shall be reflected in the architecture requirements specification.

NeOn Technical Annex

The NeOn description of work introduces a preliminary vision of the NeOn architecture and the functionalities it aims to provide. We have used this specification as a wish list on top of which we have built an elaborated, domain grounded, and exhaustive architecture requirements collection that shall allow to accurately design the NeOn architecture.

2.2 Requirements sheet

The requirements sheet template is the main tool used to gather a preliminary collection of architecture requirements. It contains six main fields:

- Requirement id.
- Requirement name.
- Requirement description.
- Relevance of the requirement in the overall NeOn architecture. It can be either critical, average, or low.
- Main conceptual architecture layer where the requirement can be allocated. These layers are three: Distributed Repository layer, Distributed Components, and NeOn toolkit.
- Type of the requirement extracted, either functional or non functional.
- Traceability information. This can be either internal or external. Internal traceability relates a given requirement with other requirements it depends on, whereas external traceability collects information regarding the exact information source where the requirement is detected.

A preliminary version of the template was filled in with information from the textual sources, then it was circulated among partners and the collected data was merged.

This sheet allows classifying requirements hierarchically, from the more general to the more concrete, providing a stratified vision of the requirement information. However, the sheer amount of data collected is too large and the requirements sheet alone does not suffice in order to clearly display information contained within. In this direction, requirement categories have been created, following the IEEE methodology for software requirements specification, pursuing a more precise requirements description.

2.3 IEEE methodology for software requirements specification (SRS)

This methodology details how to write a good specification of software requirements, including the following information:

- Important characteristics of a requirements specification.
- Necessary sections to be included into the specification.
- Requirements hierarchy for each section. Each requirement fits into one section of the recommendation. Additionally, each requirement is bound to the previous hierarchy of requirements.

SRS is both a standard and a recommendation proposed by IEEE. The structure proposed by this methodology is comprehensible and classifies requirements in sections that relate requirements initially separated. As a standard, SRS is easily interpreted by the specialized reader.

Once the requirements sheet was populated, we classified this information in the form of an SRS compliant document. This document specifies the functionalities that a software product shall provide to the end user. The following software aspects are considered:

- **Functionality:** what the software is expected to do.
- **External interfaces:** how the described software shall interact with other software, hardware, and users.
- **Performance:** speed, availability, response time, or recovery time.
- **Attributes:** considerations on portability, correctness, maintainability, or security.
- **Design constraints** imposed on the future design and implementation due to the adoption of particular standards.

A good software requirements specification should also have the following properties:

- **Correctness**: every requirement stated therein must be met by the final product.
- **Unambiguity:** every requirement statement has only one possible interpretation
- **Completeness:** it includes all significant, functional and non functional requirements.
- **Internally consistent:** the document is aligned with other upper level documents, e.g. [1] and the neon technical annex.
- **Ranked for importance** and/or stability: the relevance of each requirement is detailed.
- **Verifiable:** a requirement is verifiable if some finite, cost-effective process exists against which it can be checked.
- **Modifiable:** its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style
- **Traceable**: the precedence of each requirement is clear and future referencing is possible.

This methodology proposes to divide the requirements specification document into three main sections which, on the other hand are not mandatory. In fact, during this work we have modified it as necessary. Nevertheless, the following outline has been included in the architecture requirements specification.

1. The overall description section describes the general factors that affect the software product and its requirements.
    - Product Perspective: puts the product into perspective with other related products. It details if the product is part of other larger product. Relations with other software, system, user, hardware, communications interfaces and memory constraints are specified.
    - Product Functions: a description about the main functions that the software shall provide.
    - User characteristics: software characteristics necessary to satisfy users.
    - Developer constraints such as regulatory policies, hardware limitations, audit functions, reliability requirements, etc.
    - Assumptions and dependencies of the software product, e.g. specific operating systems.
2. The specific requirements section details all the requirements that the software product shall eventually meet. Requirements contained therein include a short description. Every requirement shall be perceived by users, operators or other external systems:

- External Interfaces
- Functions
- Performance Requirements
- Logical Database Requirements
- Design Constraints
- Software System Attributes

Additionally, specific requirements shall be readable, uniquely identifiable, and cross-referenced to earlier documents.

2.4 Requirements lifecycle

Software requirements specification is not a sequential process with clear and well-defined beginning and end. Quite on the contrary, this is a spiral model where requirements themselves have their own lifecycle. Requirements are:

1. Detected,
2. Gathered and contextualized,
3. Decomposed into smaller requirements,
4. Evolved along with the software product,
5. Sometimes, no longer applicable and disappear from the system specification.

According to this life cycle, partners produced feedback on the different versions of the architecture requirements in order to obtain a more complete and comprehensive list. Specific contributions from the case studies are crucial. They help establishing actual relevance of architectural requirements in terms of their usefulness for application development. In general, user interaction is critical for requirements gathering and evolution as they provide first-hand insight.

This kind of iterations on the requirements list show how they are updated with new recommendations and how new requirements appear. The relevance of the requirements identified will be naturally fitted as NeOn evolves.

3. NeOn architecture requirements

Next, the NeOn architecture requirements are classified according to the IEEE software requirements specification methodology [3] . Please note that the document structure below may not exactly coincide with the one proposed by this methodology. Quite on the contrary, it has been adjusted to better fit our aim to describe the NeOn architecture requirements, object of this document.

3.1 Overall Description [1]

An overall description of the NeOn architecture requirements is first introduced. Then, specific, detailed requirements will be provided.

[1] This section corresponds to section 2 of the IEEE software requirements specification.

3.1.1 NeOn architecture perspective

This section puts the NeOn architecture into perspective with respect to the expected use of networked ontologies. System requirements are sketched within and all kind of interfaces describing the relation of the architecture with users, hardware, software, and communication are introduced.

3.1.1.1 System requirements

Table 1 shows the complete set of system requirements for the NeOn architecture. This table describes the main assumptions upon which the NeOn architecture shall be built. The requirements contained herein, each displayed with their own identifier, are divided in three main categories, as shown in Figure 1: NeOn architecture layers, NeOn services, and NeOn compatibility and extension.

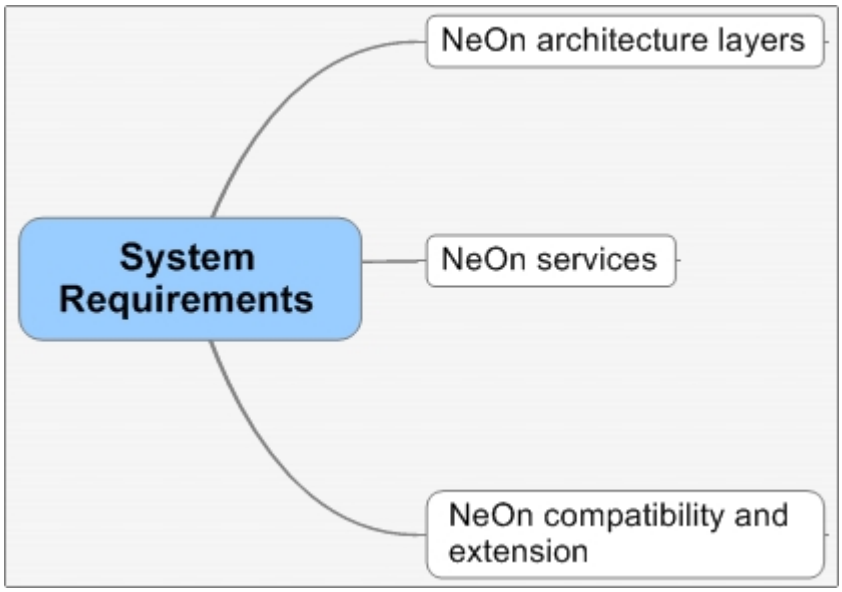

Figure 1: System requirements

NeOn architecture layers

The NeOn architecture will be structured in three main conceptual layers (see Figure 2: Reference architecture as a set of extension points) where information repositories, components and services, and user interfaces will be accommodated. These layers are the following, from lower to higher abstraction level:

- **Distributed repository layer:** Information resources will be potentially stored in a distributed repository managed by this architecture layer. The distributed repository layer allows NeOn to access resources both transparently both from their location and the nature of the knowledge being accessed.
- **Distributed components layer:** This middleware layer between the NeOn toolkit and the distributed repository hosts a number of services which allow users to exploit the information stored in the distributed repository.
- **NeOn toolkit:** User front-ends to the services provided by the second architecture layer as well as to the information repositories managed by the lower level layer.

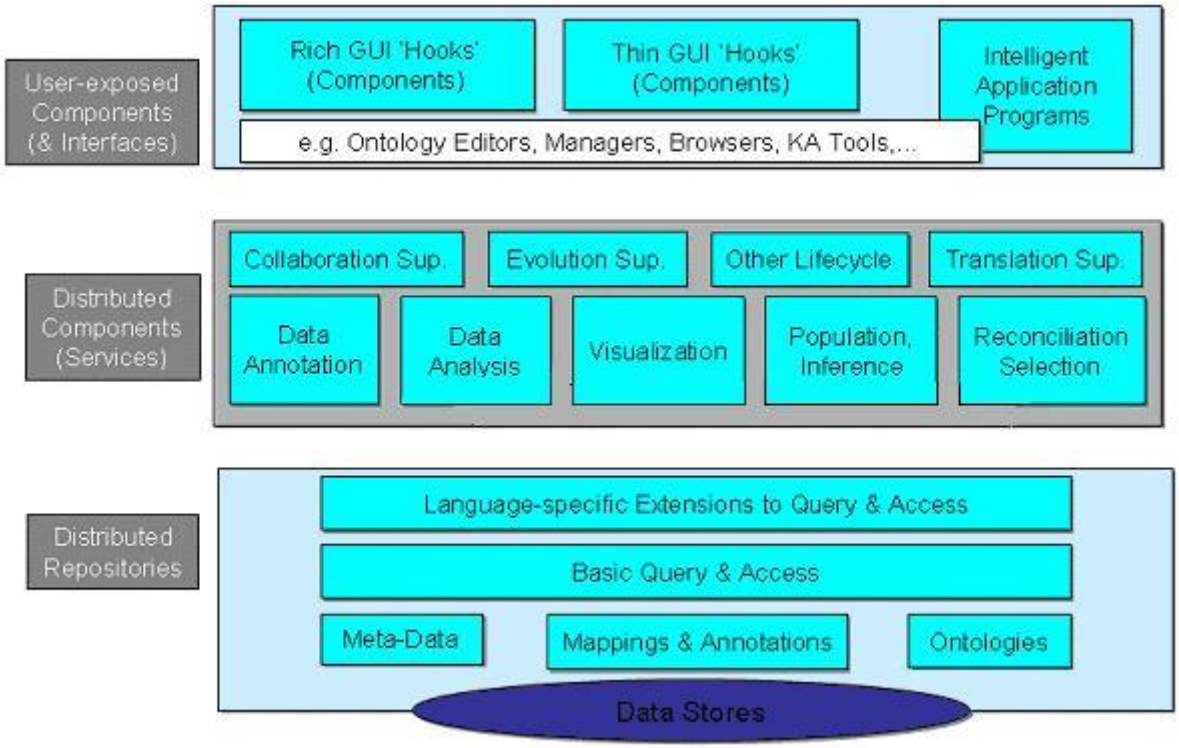

Figure 2: Reference architecture as a set of extension points

The different levels of the NeOn architecture need to work in a coordinated manner and a large amount of communication and service invocation is expected to take place between them. The services and information stores maintained by the NeOn architecture stack are at the same time service functionality clients and producers for each other. In this scenario, natural flow of service invocation will take place from higher level to lower level layers. On the other hand, push services can exist. For example, information refresh may be triggered upwards from the distributed repository layer in order to update user-level information displayed in the NeOn toolkit. Hence, it will be necessary to design and implement the necessary infrastructure which allows this kind of interaction. A possible solution is to include a semantic enterprise service bus, like the one proposed in EU-project DIP[2]. Due to performance constraints, special emphasis should be made on scalability in this regard.
The NeOn distributed repository layer is expected to deal with a large number of heterogeneous information resources. This includes not only formal knowledge in the form of ontological resources expressed in a variety of ontology languages, but also legacy resources containing information which is still valuable for the different organizations. In this direction, the following types of distributed information resources shall be integrated in the NeOn knowledge model:

- **Unstructured content**, like e.g. text.
- **Structured knowledge** with no clear semantics. This category would group e.g. legacy database systems and catalogues.
- **Formal knowledge:** Ontological knowledge and data.

NeOn services

NeOn aims to become the next generation ontology development framework and, as such, it must provide a large number of services which allow managing and supporting the entire ontology lifecycle, from design to exploitation. Therefore, in addition to ontology editing functionalities, other basic services are required to satisfy collateral needs of ontology developers. These services will be mainly localized in the second layer of the NeOn architecture but their functionalities will be supported by the distributed repository layer and user interfaces will be provided by the NeOn toolkit. NeOn services identified so far are the following (see also Figure 3):

- **Annotation service:** NeOn will provide the necessary infrastructure to semantically annotate information sources, using data annotation [9] , specialized text mining [10] techniques or tools like Text-Garden[3]. The annotation service will be eventually used to populate ontologies managed by the NeOn framework with

[2] See http://dip.semanticweb.org

[3] www.textmining.net

already existing data stored in legacy repositories. This feature will allow to automatically ontologize legacy information, i.e. to semantically describe it.

- **Context service:** This service will allow building user profiles and establishing behaviour patterns which allow building customized, user-focused services.
- **Reasoning service:** Reasoning capabilities will be injected in the NeOn framework. This service will provide the means to access these capabilities. Reasoning includes ontology consistency checking and inference.
- **Query service:** NeOn will allow users to formulate queries using common standard languages and execute them against the knowledge contained within. This knowledge includes both ontology data and legacy data.
- **Question formulation service:** A higher-level query interface will also be supported that allows to express queries more naturally, e.g. in free text. Questions formulated by means of this facility will be automatically translated into ontological query standard languages and submitted to the NeOn query service.
- **Summarization service:** It is fundamental for ontology users to count on information about the ontologies maintained by a system like NeOn. In this regard, the NeOn summarization service aims to build reports with the most relevant ontology properties. These parameters are still to be defined.
- **Provenance service:** Associated to the summarization service, this service will log and provide information about events occurred to particular ontologies throughout their lifecycle. This information will include changes, the users who introduced them, and the different existing versions of the ontology.
-

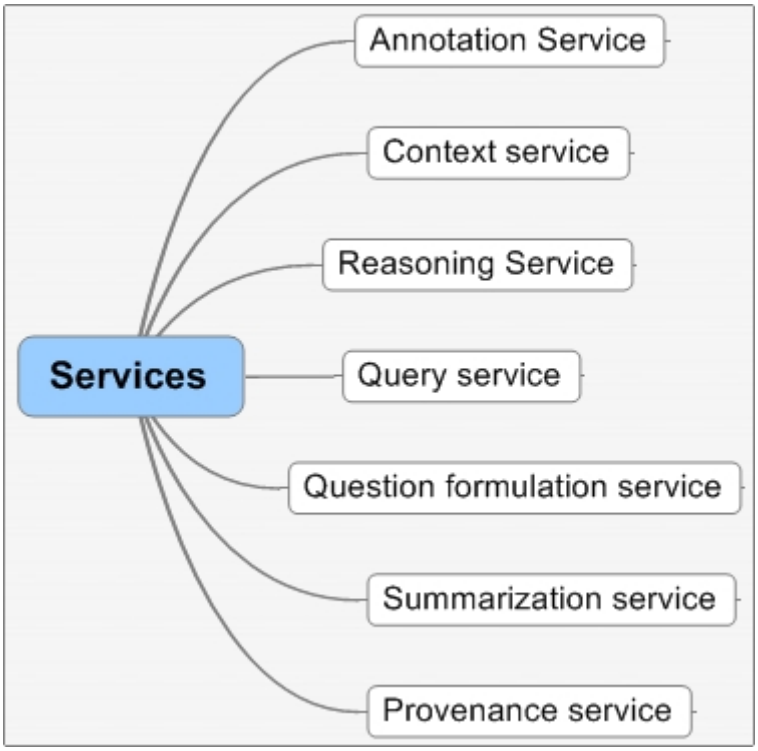

Figure 3: NeOn services

NeOn compatibility and extension

Both, the NeOn layered architecture and services contained within it must support and be built in order to be compliant with the following high-level policies. First of all, the NeOn architecture will be open in order to facilitate the inclusion and extension of state-of-the-art ontological technology like in [11] , either already present in the consortium, proceeding from alliances with other top projects like cBio[6] , or from contributions of other organizations. Thus, the necessary infrastructure must be included in the core of the NeOn architecture that allows loosely coupling of components and resources. This will require specifying a series of well-defined NeOn system extensibility interfaces that standardize the addition of external components into the framework.

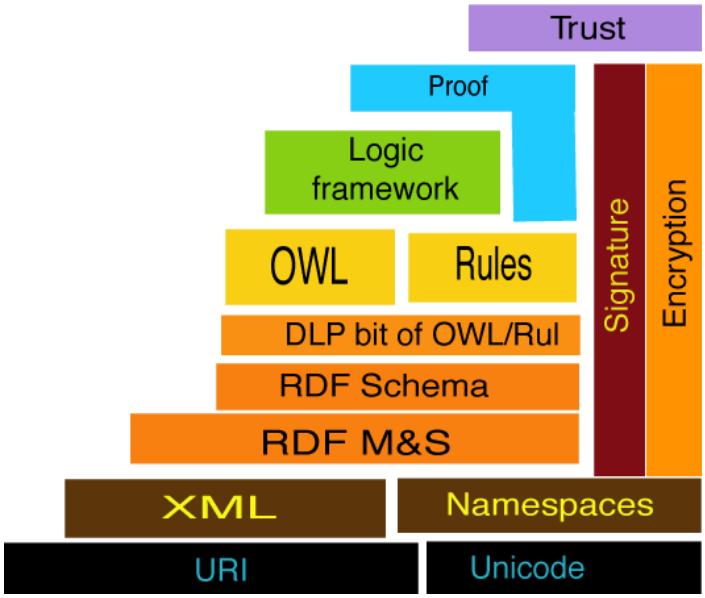

Figure 4: W3C Semantic Web stack[4]

Ontologies and data shall be loaded in a stand-alone server, based on OntoStudio and OntoBroker, which implements the backend for the reasoning service and the ontology editor. On the other hand, given that both OWL-like languages and Frame languages coexist in the Semantic Web stack (see Figure 4), the NeOn data model will observe the dual language approach, i.e. both language types will be supported.

Req #	Title	Description	Importance	External traceability
2.1.1.1	Distributed repository layer	Information resources will be potentially stored in a distributed repository managed by this architecture layer. NeOn shall access these resources transparently to their location	Critical	NeOn TA[5]
2.1.1.2	Middleware layer	This middleware layer between the NeOn toolkit and the distributed repository shall implement a number of services which allow users to exploit the information stored in the distributed repository	Critical	NeOn TA
2.1.1.3	NeOn toolkit layer	User frontends and high level services shall be supported by this layer	Critical	NeOn TA
2.1.1.4	Inter-layer communication mechanism	The NeOn architecture layers shall be at the same time service functionality clients and producers for each other. Natural flow of architecture service invocation goes from the higher level layers to the lower level layer. On the other hand, push services, e.g. information refresh may be triggered upwards by lower level layers, in this case, the distributed repository layer. The necessary infrastructure, e.g. a service bus, shall be implemented to allow this kind of service invocation	Critical	NeOn TA
2.1.1.5	Support for integration of heterogeneous information sources	The following types of distributed information resources shall be integrated in the NeOn knowledge model: - Unstructured content, e.g. text. - Structured knowledge with no clear semantics, e.g. DBs, catalogues. - Formal knowledge (ontologies and data) - Semantic annotations	Critical	NeOn TA
2.1.1.6	Annotation service	The second layer of the NeOn architecture shall include annotation mechanisms and associated services	Critical	NeOn TA

[4] Figure taken from [4]

[5] Technical Annex

2.1.1.7	Text mining service	The second layer of the NeOn architecture shall include text mining services for human language information retrieval	Average	NeOn TA
2.1.1.8	Summarization service	The second layer of the NeOn architecture shall include an informational service briefing ontology information	Critical	NeOn R&V
2.1.1.9	Context service	The second layer of the NeOn architecture shall include a service to gather and process context information for later exploitation	Critical	NeOn R&V
2.1.1.10	Reasoning service	The second layer of the NeOn architecture shall include reasoning mechanisms and associated services.	Critical	NeOn TA
2.1.1.11	Query service	The second layer of the NeOn architecture shall include query mechanisms and associated services.	Critical	NeOn R&V
2.1.1.12	Question formulation Service	The NeOn toolkit shall provide a question formulation service that eases query formulation by allowing more natural ways of expressing queries, e.g. natural language	Average	NeOn R&V
2.1.1.13	Provenance service	The NeOn middleware layer shall include a service which keeps track of the precedence of ontologies, i.e. how, when, and by whom ontologies evolve across their lifecycle.	Critical	NeOn R&V
2.1.1.14	Re-use and extension of pre-existing technology	The architecture shall be open enough and facilitate the inclusion and extension of state-of-the-art ontology platforms	Average	NeOn TA
2.1.1.15	Stand-alone server	Ontologies and data shall be loaded in a stand-alone server, based on OntoStudio and OntoBroker, which implements the backend for the reasoning service and the ontology editor.	Critical	NeOn TA
2.1.1.16	Dual language approach	Description logic-based languages, e.g. OWL, and Frame/Rule languages, e.g. FLogic, shall be supported	Critical	NeOn architecture partners
2.1.1.17	Integrated NeOn Infrastructure	The NeOn architecture shall integrate the technology developed in the technical work packages	Average	NeOn TA
2.1.1.18	Preconfigured and bundled infrastructure	NeOn infrastructure shall be preconfigured and bundled into sector-specific solutions, e.g. in the case studies	Average	NeOn TA
2.1.1.19	Loosely coupled architecture	NeOn architecture will be mainly based on a loosely couple concept	Critical	NeOn TA
2.1.1.19.1	Loosely coupled architecture	NeOn architecture shall allow loose coupling of components	Critical	NeOn TA
2.1.1.19.2	Loosely coupled architecture	NeOn architecture shall allow loosely coupling of resources	Critical	NeOn TA

Table 1: System requirements

3.1.1.2 User interface

Table 2 shows the logical characteristics of interfaces between NeOn and its potential users, including GUI configuration issues and customization. Figure 5 shows a classification of user interface requirements with three large categories: those associated to the NeOn editor, requirements for the NeOn browser, and requirements bound to the NeOn GUI. The requirements described herein represent the counterparts of the system functionalities described in section 0 in terms of GUI capabilities.

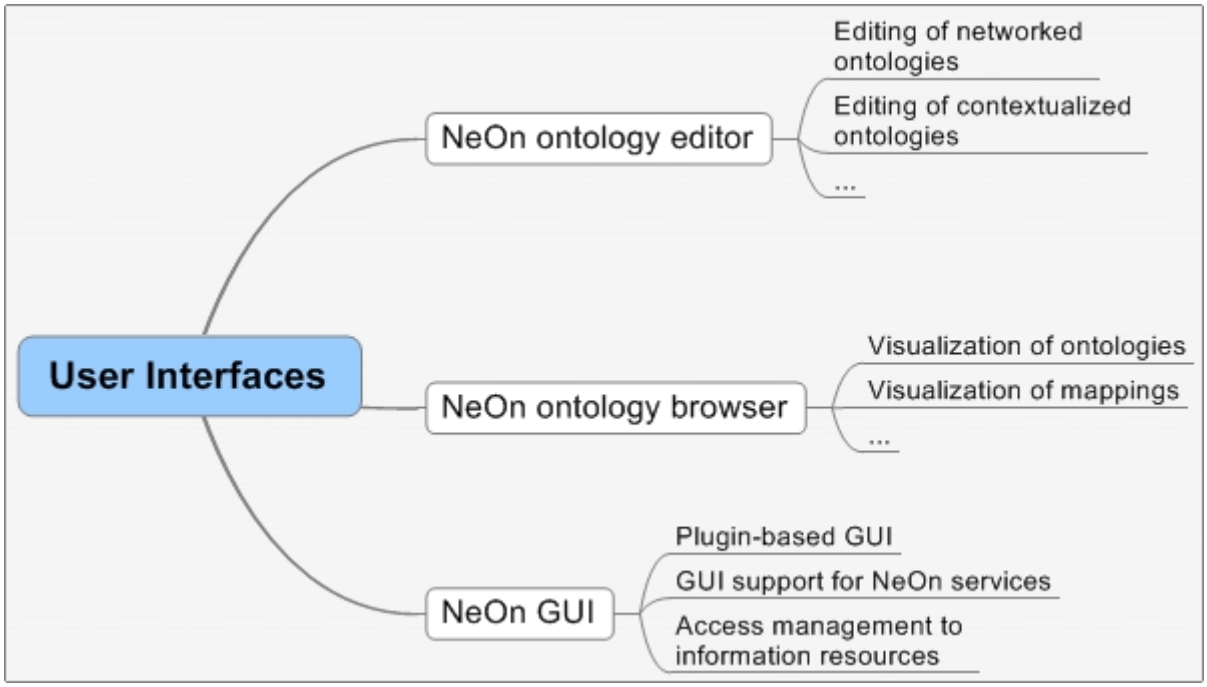

Figure 5: User interfaces requirements

NeOn Ontology Editor

These requirements appeal to the NeOn functionalities related to the editing of networked ontologies in the NeOn framework. The NeOn Editor, based on OntoStudio, which will be appropriately extended for the networked case, shall allow working on several networked ontologies simultaneously. The NeOn Editor shall allow selection and reuse of complete ontologies as well as of their parts. This can be achieved by making the NeOn Editor aware of the modules comprised by a particular ontology, including ontology module editing capabilities. The NeOn Ontology Editor shall allow editing all kind of ontology entities, including ontology mappings.

A fundamental feature, shed by the NeOn pharmaceutical [8] and fishery [7] case studies, is support for contextualized ontologies. The ontologies needed in these scenarios are contextualized, i.e. their properties change according to the particular context of application, and only a subset of these properties is valid in a certain context. For example, the invoice model, described by means of networked ontologies, of the pharmaceutical case study has different contexts of applications depending on the stakeholders involved in the invoice exchange. Although the invoice model is the same, the applicable information will change if the stakeholders involved are a laboratory and a pharmacy or a laboratory and a wholesaler. Thus, the NeOn Editor shall also allow users to define, select and edit ontology context.

NeOn Ontology Browser

NeOn shall allow visualizing the structure of networked ontologies, including ontology modularization, and the properties of the ontological entities contained within. Additionally, NeOn shall provide means to graphically represent changes in networked and shared ontologies, as well as ontology lifecycle provenance information, i.e. who and how interacted with a given ontology or with a set of them.
NeOn shall also provide mapping visualization. In this case, effective graphical representation of mapping information is expected to be especially helpful since NeOn will manage ontologies in a potentially complex networked environment. In this direction, NeOn shall allow visualizing all kind of dependencies between networked ontologies.
It is also certainly relevant that context information be accurately displayed. In this direction, NeOn shall provide contextualized views of ontologies. All the different types of visualization described in Table 2 shall be parameterized, and these parameters customizable.
With high probability, the NeOn architecture design will eventually integrate browsing capabilities with the NeOn Ontology Editor.

NeOn GUI

NeOn GUI requirements specify the rest of the properties that the NeOn user interface must comply with, in terms of facilities for user interface configuration and user access to services. These requirements can be classified under the following main categories:

- Plugin-based GUI
- Support for different types of clients
- GUI support for NeOn services
- Access management to information resources

As stated in the NeOn compatibility and extension requirements, in particular, requirement 2.1.1.19, NeOn extensibility and flexibility is a really important point to be considered under the architecture perspective. This issue is also reflected as part of the user interface requirements. Thus, NeOn shall allow easy configuration of the functionalities available in the NeOn toolkit by means of a plugin-based system. NeOn shall also provide users with reflective functionalities offering information about the plugins present in the interface at any time.
This plugin-based system will be part of the infrastructure that supports the implementation of rich clients to the NeOn functionalities. A rich client contains a large amount of logic within. On the other hand, thin clients, i.e. clients with hardly any logic within, will be based on web technology, merely a dumb interface that collects user requests, invokes the appropriate services, and shows results after execution. The main advantage of this kind of clients is that they can be decoupled from the component actually providing the functionality. Hence, they turn to be certainly useful in distributed systems like NeOn.
Part of the requirements on user interfaces correspond to front-ends of the NeOn services described in 0, i.e. NeOn shall provide GUI support to every service. Additionally, though this requirement is weaker than those described so far NeOn shall provide a user interface that enabled certain types of users with an administrator role to manage access rights to information resources, including ontological and non-ontological knowledge. On the other hand, information about resource accessibility shall be displayed to regular users.

Req #	Title	Description	Importance	External traceability	Internal traceability
2.1.2.1	NeOn Editor	The NeOn toolkit layer shall provide an ontology editor based on OntoStudio	Critical	NeOn TA NeOn case studies	2.1.1.3 3.2.11
2.1.2.1.1	Multi-ontology editing	The NeOn editor shall allow to work with several ontologies simultaneously	Critical	NeOn TA NeOn case studies	2.1.1.3 3.2.11
2.1.2.1.2	Support for contextualized ontologies	NeOn ontologies shall be contextualized, i.e. their properties depend on the current context	Critical	NeOn R&V NeOn case studies	2.1.1.3 2.1.1.9
2.1.2.1.2.1	Ontology context editing	The NeOn editor shall allow to select and edit ontology context	Critical	NeOn R&V NeOn case studies	3.2.16.2
2.1.2.1.3	Partial ontology selection	Partial selection and reuse of ontologies shall be available	Average	NeOn R&V	2.1.1.3 3.2.11.1.3
2.1.2.2	NeOn Browser	NeOn shall allow ontology browsing. Might be integrated with the NeOn editor	Critical	NeOn TA NeOn case studies	2.1.1.3 3.2.11.1.2
2.1.2.2.1	Networked ontologies Visualization	NeOn shall allow to browse and visualize networks of ontologies	Critical	NeOn TA NeOn case studies	2.1.1.3 3.2.11.1.2
2.1.2.2.2	Context-dependent Ontology visualization	NeOn shall provide contextualized views of ontologies	Critical	NeOn R&V NeOn case studies	3.2.16
2.1.2.2.3	Visualization of changes in networked ontologies	NeOn shall provide graphical ways of displaying changes in networked and shared ontologies	Critical	NeOn R&V	2.1.1.3 3.2.11.1.7.2

2.1.2.2.4	Cross-author change tracking	NeOn shall provide means to represent ontology lifecycle provenance information, who and how interacted with a given ontology	Average	NeOn R&V	2.1.1.3, 2.1.1.13
2.1.2.2.5	Mappings visualization	Mapping information shall be graphically displayed	Critical	NeOn R&V	2.1.1.3 3.2.9
2.1.2.2.6	Briefing on ontology entities	Information about ontology entities shall be displayed in a comprehensible way	Average	NeOn R&V NeOn case studies	2.1.1.3 3.2.11
2.1.2.2.7	Visualization of dependencies between networked ontologies	NeOn shall allow to visualize dependencies, i.e. connections and alignments between networked ontologies	Average	NeOn R&V	2.1.1.3 3.2.11.1.2
2.1.2.2.8	Visualization of Ontology modules	NeOn shall provide visualization of ontology modules	Average	NeOn R&V	2.1.1.3 3.2.3
2.1.2.2.9	Ontology summarization	NeOn shall display information about properties of ontologies in a summarized way.	Average	NeOn R&V	2.1.1.3 2.1.1.8
2.1.2.2.9.1	Ontology summarization	Summarization shall contain the main characteristics of the ontologies.	Average	NeOn R&V	3.2.7 2.1.1.8
2.1.2.2.9.2	Customizable summarization	NeOn shall allow to customize the parameters to be summarized	Average	NeOn R&V	3.2.7 2.1.1.8
2.1.2.2.9.3	Customizable summarization	Summarization parameters are to be defined, yet	Average	NeOn R&V	3.2.7 2.1.1.8
2.1.2.2.10	Briefing on non-ontological resources	NeOn shall provide information about non-ontological resources being used	Average	NeOn R&V	2.1.1.3 3.2.2.2
2.1.2.3	NeOn GUI	NeOn shall offer a user-friendly GUI to functionalities and services	Critical	NeOn TA	
2.1.2.3.1	Customisation facilities	NeOn shall allow to easily configure the functionalities available in the NeOn toolkit	Average	NeOn R&V	2.1.1.3, 2.1.1.9, 3.2.8
2.1.2.3.2	Accessibility to plugins	Plugin load features shall be accessible to the user	Average	NeOn R&V	3.1.1
2.1.2.3.2.1	Plugin acknowledgement	NeOn shall provide means to inform users of the different plugins installed.	Average	NeOn R&V	3.1.1
2.1.2.3.3	GUI support for NeOn services	NeOn shall provide user interfaces to every NeOn service	Critical	NeOn R&V	2.1.1
2.1.2.3.3.1	GUI support for the annotation service	NeOn shall provide a user interface to the annotation service	Average	NeOn R&V	2.1.1.6
2.1.2.3.3.2	GUI support for the text mining service	NeOn shall provide a user interface to the text mining service	Average	NeOn R&V	2.1.1.7
2.1.2.3.3.3	GUI support for reasoning and inference	NeOn shall provide a user interface to the reasoning and inference service	Average	NeOn R&V	2.1.1.10
2.1.2.3.3.3.1	GUI support for query formulation	NeOn shall provide a user interface to formulate queries in standard query languages	Average	NeOn R&V	2.1.1.11
2.1.2.3.3.3.2	GUI support for question formulation	NeOn shall provide a user interface to formulate questions	Average	NeOn R&V	2.1.1.12
2.1.2.3.3.3.3	GUI support for answer explanation	NeOn shall sensibly explain question answers in the terms of the formulated question	Average	NeOn R&V	2.1.1.12
2.1.2.3.3.4	GUI support for the summarization service	NeOn shall provide a user interface to the summarization service	Average	NeOn R&V	3.2.7 2.1.1.8

2.1.2.3.3.5	GUI support for the context service	NeOn shall provide a user interface to the context service	Average	NeOn R&V	2.1.1.9
2.1.2.3.3.6	GUI support for the provenance service	NeOn shall provide a user interface to the provenance service	Average	NeOn R&V	2.1.1.13 3.2.6
2.1.2.3.4	Access management to information resources	NeOn shall provide a GUI to show and manage access rights to resources	Average	NeOn R&V NeOn case studies	2.1.1.3 3.6.3.1
2.1.2.3.4.1	Access management to ontological resources	NeOn shall provide a GUI to show and manage access permission to ontological resources	Average	NeOn R&V	2.1.1.3 3.6.3.1.1
2.1.2.3.4.2	Access management to non-ontological resources	NeOn shall provide a GUI to show and manage access permission to non-ontological resources	Average	NeOn R&V	2.1.1.3 3.6.3.1.1
2.1.2.3.5	Rich client	NeOn shall provide support for rich clients, i.e. NeOn clients with a large amount of service logic within	Critical	NeOn TA	3.5.1.1
2.1.2.3.6	Thin client	NeOn shall provide support for thin clients, i.e. NeOn clients with hardly any logic within, like web clients.	Average	NeOn TA	3.5.1.2

Table 2: User interface requirements

3.1.1.3 Hardware requirements

This section specifies the logical characteristics of interfaces between the NeOn architecture and the hardware components of the system, including configuration characteristics. It also covers such matters as e.g. what devices are to be supported, how they are to be supported, and protocols. Hardware requirements gathered so far have been structured in

Table 3. This table will definitely change as the project evolves. Currently, the minimum processor and memory specifications are provided.

Req #	Title	Description	Importance	External traceability
2.1.3.1	Minimum processor	Pentium IV 2 GHz or better	Critical	NeOn TA
2.1.3.2	Minimum memory	1GB RAM	Critical	NeOn TA

Table 3: Hardware requirements

3.1.1.4 Software requirements

The use of additional software products, e.g. data management systems or operating systems, and interfaces with other applications, e.g. linkage with software libraries, are specified here. As Table 4 shows, NeOn shall be compatible with the main operating systems in the market, including the popular Microsoft family of windows systems and the different distributions of the open source alternative operating system, Linux. Additionally, Java will be the main technology underlying the implementation of NeOn.

Req #	Title	Description	Importance	External traceability
2.1.4.1	Compatibility with several platforms	The NeOn toolkit shall be compatible with several platforms	Critical	NeOn TA
2.1.4.1.1	Operating Systems	NeOn infrastructure shall be compatible with several Operating Systems	Critical	NeOn TA
2.1.4.1.1.1	Windows XP/2000 family	NeOn infrastructure shall be compatible with the Windows XP/2000 family	Average	NeOn TA
2.1.4.1.1.2	Linux family	NeOn infrastructure shall be compatible with the Linux family	Average	NeOn TA
2.1.4.1.1.3	Mac OS X	NeOn infrastructure shall be compatible with Mac OS X	Average	NeOn TA
2.1.4.1.2	Java Platform	NeOn infrastructure shall run on all Java 1.4.2-compatible JVMs	Average	NeOn TA

Table 4: Software requirements

3.1.1.5 Communication interfaces

Table 5 shows two meta-requirements for communication in the NeOn infrastructure. Communication interfaces are seen under the perspective of the different components which form part of the NeOn platform. Hence, it is certainly important that NeOn shall offer a service-oriented interface that allows legacy components to be included with the smallest impact on the rest of the system. This way, the standard interface methods described as part of the NeOn compatibility and extension policies in section 0 shall be enforced. However, components built from scratch under the NeOn specification shall rather use more direct, higher performance methods for integration. In this direction, the NeOn architecture shall offer an API accessible by means of several client interfaces.

Req #	Title	Description	Importance	External traceability	Internal traceability
2.1.5.1	Service oriented architecture	NeOn shall offer a service oriented interface in order to ease the inclusion of new components into any of the three architecture layers	Critical	NeOn TA	2.1.1.19
2.1.5.2	API –based interface	The NeOn backend will be accessible by means of several client interfaces, e.g. Java, .NET.	Critical	NeOn TA	2.1.1.5

Table 5: Communication interfaces

3.1.2 Functions of the NeOn architecture

This section contains a list of the major functions that the NeOn architecture shall perform, to be detailed in section 0. Next, we summarize these functions, organized following a classification that groups them in self-contained categories.

Geographically-transparent access to distributed repository: NeOn shall allow access to ontological and non ontological information pieces transparently from their location.

Support of heterogeneous knowledge: Different types of information sources will be managed by NeOn, which shall provide the necessary means to include them all into its knowledge model.

Dual language approach: NeOn shall be compliant with the current and future Semantic Web standards. Additionally, other languages shall be supported as required.

Ontology modularization: Large ontologies, where information tends to be dispersed, shall be properly managed by means of a modular structure.

Multilinguality support: As pointed out by the NeOn pharmaceutical and fishery case studies, support for multilingual ontologies shall be provided.

Mapping support: NeOn shall provide means to simplify alignment between entities of two or more globally inconsistent ontologies.

Support for contextualized ontologies: NeOn shall provide the means to parameterize ontologies with respect to their different contexts.

Lifecycle support for ontology development: NeOn shall support knowledge acquisition and ontology development and maintenance throughout their lifecycle, extending standard ontology development facilities to the networked arena.

Reasoning and inference: The NeOn backend shall provide reasoning capabilities on top of the aggregated knowledge contained in a network of ontologies.

Query support: Standard ontology query languages like e.g. SPARQL shall be supported by NeOn, allowing the retrieval of information contained within.

Question formulation: Question formulation facilities shall be built on top of ontology querying that will allow users to build more complex and expressive questions for information retrieval.

Semantic annotation: NeOn shall incorporate semantic annotation functionalities which will provide information characterization in terms of the model described in a set of networked ontologies.

Access management to information resources: Access rights to information resources managed by NeOn need to be administrated as shown by the case studies

User profiling: NeOn shall accumulate information about user profiles during system interaction and provide a customized set of functionalities according to the inferred profile.

Ontology summarization: NeOn shall provide users with the means to summarize ontology state and properties.

Provenance support: Ontology traceability shall be kept by automatically storing relevant information like authors or contributors.

System documentation and help: The different user types of NeOn shall be provided with extensive help and training materials, including tutorials.

Support for different client types: Rich and thin clients shall be supported by the NeOn architecture, as described in section 0.

Service access to the NeOn backend: Access to the NeOn backend shall be provided to NeOn clients.

3.1.3 Users characteristics

We have focused the contents of this section under the perspective of the properties that NeOn shall satisfy in order to allow ontology developers to easily and quickly get acquainted with the system. First of all, each functionality provided by the system shall be accessible, i.e. the effort required by the user in order to reach them shall be kept certainly low. This requirement is mainly focused on the NeOn toolkit layer of the architecture but shall also be supported by the other two layers.

We plan to achieve this goal by following the conclusions of the user study carried out in [5] where a complete and deep analysis on current ontology engineering tools was made. This user study provided as a result a set of guidelines for the development of successful, human-ontology interaction compliant tools.

On the other hand, NeOn shall be delivered to the end user as a self-contained software package ready for installation. We foresee to provide two different types of installation packages, one containing the open source version of NeOn and a different one for the full fledged commercial version of the software. Both shall be automatically deployable, and an installation wizard will be provided along.

3.1.4 Constraints

Herein we describe the factors that shall constrain the development process of the NeOn infrastructure, including quality assurance criteria. Table 6 contains a detailed collection of these requirements.

Development constraints can be classified under four main categories, also graphically shown in Figure 6:

- **Cost estimation:** Different methods shall be used that allow evaluating the cost of each component of the NeOn infrastructure in a preliminary stage, after implementation design.
- **Testing:** Unit and functional tests shall be used to check the NeOn software.
- **Bug tracking and fixing:** Continuous bug tracking and fixing shall be carried out during the lifecycle of NeOn. The necessary infrastructure shall be provided in order to support this effort.
- **Software licensing:** NeOn or at least part of it shall be licensed as open source software, with the corresponding license, like e.g. (L)GPL.

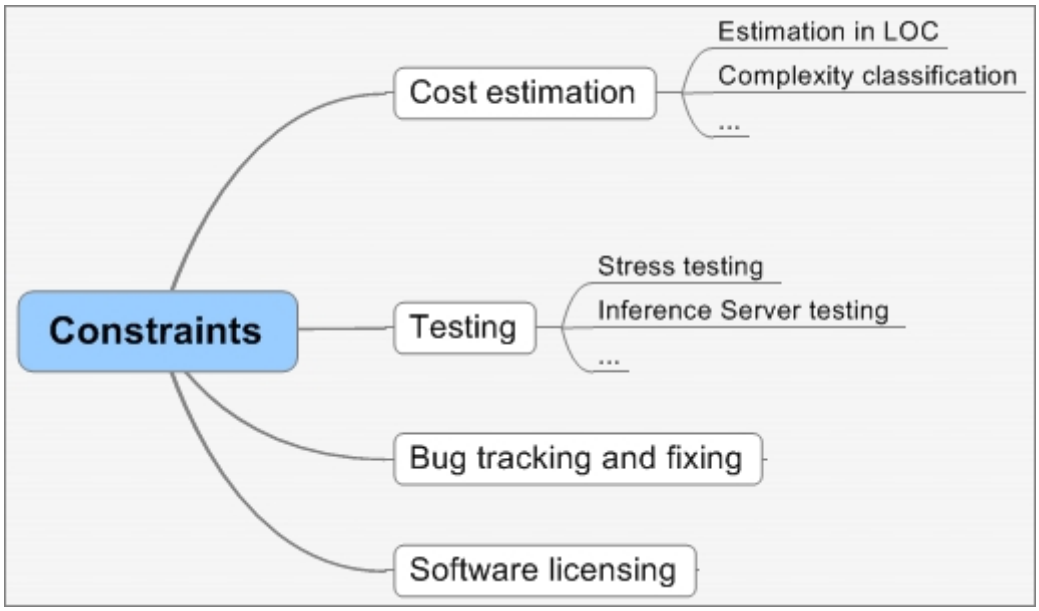

Figure 6: Constraints classification

The methods considered for cost estimation include classification of each component implementation complexity. Values ranging from 1 to 4 will be assigned that quantify this complexity. Additionally, as soon as preliminary prototypes are ready, estimations of the software complexity in terms of lines of codes (LOC) shall be made. Finally, when software maturity allows, a port-mortem analysis will be performed that evaluate the accuracy of predictions by comparing estimated cost with actual cost after implementation.

Tests will be carried out aiming to evaluate two main issues: i) system performance and ii) usability. Performance tests will be focused in measuring the throughput of the reasoning and inference engine of the NeOn backend in different situations. In this direction, a benchmark suite shall be built and executed against large, real-world ontological and non-ontological knowledge repositories. Connectivity shall also be evaluated in order to determine quality of service in terms of throughput and latency with respect to the number of components as well as external clients supported by the architecture.

Usability testing shall eventually reproduce the user study conducted in [5] to evaluate NeOn in terms of human-ontology interaction. The questionnaire created therein will be the perfect tool in order to compare NeOn with the ontology engineering tools evaluated already. Nevertheless, this kind of user study needs a certain degree of stability. Until then, other tests can be performed in order to ensure usability satisfaction. These measures include, but are not limited to, cognitive walkthroughs and interviews with representative user groups throughout the different stages of the NeOn lifecycle, periodic user observation in order to assess GUI intuitiveness and acceptance, and questionnaires that shall help to collect early feedback from test users.

Bug tracking infrastructure shall be available both for the members of the NeOn consortium and end users though different interfaces, focused on the particular needs of each groups of users, will be provided. Apart from the

infrastructure, it is also necessary to establish policies for bug fixing. Thus, resources shall be allocated for swift bug fixing.

Finally, open source versions of NeOn shall be licensed in order to achieve maximal dissemination of the framework among the community that shall coexist with commercial versions, as introduced in section 3.2.3. Sites like SourceForge shall be used to distribute open source versions of NeOn.

Req #	Title	Description	Importance	External traceability
2.4.1	Development process	NeOn components shall be developed following a predefined design and implementation method	Critical	NeOn TA
2.4.1.1	Implementation design validation	The implementation design shall be checked against the functional specification of the component	Critical	NeOn TA
2.4.1.2	Implementation changes	Implementation design of the components shall be modified according to eventual changes	Critical	NeOn TA
2.4.1.3	Cost estimation	The cost of each component shall be estimated after implementation design	Critical	NeOn TA
2.4.1.3.1	Estimation in LOC	Implementation cost shall be measured in terms of lines of code	Critical	NeOn TA
2.4.1.3.2	Predicted cost versus real cost analysis	The estimated cost shall be compared against the actual cost after implementation	Critical	NeOn TA
2.4.1.3.2.1	Deviation analysis	Deviation analyses shall be used	Critical	NeOn TA
2.4.1.4	Testing	Unit and functional tests shall be used to check the software	Critical	NeOn TA
2.4.1.4.1	Stress testing	Stress testing using large, real-world ontologies	Critical	NeOn TA
2.4.1.4.2	Inference server testing	A benchmark suite shall be used to guarantee performance of the inference server	Critical	NeOn TA
2.4.1.4.2.1	Server scalability tests	How many concurrent clients the server can handle	Critical	NeOn TA
2.4.1.4.2.2	Client/server I/O performance test	Throughput and latency	Critical	NeOn TA
2.4.1.4.2.3	Reasoning and inference kernel performance testing	The inference kernel shall be intensively tested with large, real-world ontologies	Critical	NeOn TA
2.4.1.4.3	Large-scale, real-world integration scenarios	Industrial partners shall use NeOn technology in their commercial products, e.g. Software AG shall apply it to the EII product using real-world integration scenarios from several customers	Critical	NeOn TA
2.4.1.4.3.1	Transparent use of non-ontological resources	Transparent use of non-ontological resources shall be evaluated	Critical	NeOn TA
2.4.1.4.3.2	Runtime access to external data sources	Runtime access to external data sources via inferencing on integration ontologies	Critical	NeOn TA
2.4.1.4.4	Usability and user-centered tests	Usability and user-centered tests shall be executed	Critical	NeOn TA

2.4.1.4.4.1	Cognitive walkthroughs and structured interviews	Cognitive walkthroughs and structured interviews with small, focused, representative user groups throughout the different stages of NeOn toolkit/methodology design and development	Critical	NeOn TA
2.4.1.4.4.2	User observation	User observation shall asses intuitiveness and acceptance of user interface features	Critical	NeOn TA
2.4.1.4.4.3	Questionnaires	Users shall fill in questionnaires in order to evaluate their perception on NeOn	Critical	NeOn TA
2.4.1.4.4.4	Involvement of NeOn toolkit	Involvement of NeOn toolkit, methodology and techniques used by ontology developers in acquiring and analyzing user-centered test; also performing usability studies on the developers themselves	Critical	NeOn TA
2.4.1.4.4.5	Involvement of cross-disciplinary teams	Involvement of cross-disciplinary teams (incl. Experts on HCI, psychology of programming, practicing developers, accessibility for less abled users, etc.)	Critical	NeOn TA
2.4.1.5	Infrastructure for continuous bug tracking and fixing	Infrastructure for continuous bug tracking and bug fixing management	Critical	NeOn TA
2.4.1.5.1	Infrastructure for continuous bug tracking and fixing for NeOn partners	Infrastructure for continuous bug tracking and bug fixing management available for the consortium. A bug fixing methodology shall accompany this infrastructure.	Critical	NeOn TA
2.4.1.5.2	Infrastructure for continuous bug tracking and fixing for the public	Infrastructure for continuous bug tracking and bug fixing management available for the public and participating developers	Critical	NeOn TA
2.4.1.5.2.2	Bugzilla	Bugzilla shall be used as bug tracking main infrastructure	Critical	NeOn TA
2.4.2	Software licensing	NeOn or at least part of it shall be licensed as open source software, with the corresponding license, e.g. (L)GPL.	Critical	NeOn TA
2.4.2.1	SourceForge	SourceForge shall be used to distribute the open source version of NeOn	Average	NeOn TA

Table 6: Development constraints

3.2 Specific requirements

This section details all the requirements of the NeOn architecture that shall be implemented. These requirements are aimed towards designers and testers.

3.2.1 External interfaces

This section of the NeOn architecture requirements specification has been focused on how NeOn shall be related with existing software in order to ease integration of this technology into the NeOn framework. With this aim, we approach the integration issue from the plugin paradigm. Figure 1 shows how we have classified this kind of requirements. A complete set of requirements on external interfaces is shown in Table 7. Integration of components has also been discussed in SEKT project (http://www.sekt-project.org/) and some of the components developed in the scope of this project will be integrated in the architecture of NeOn. The techniques used in SEKT will be analysed in order to integrate the different components in the NeOn architecture.

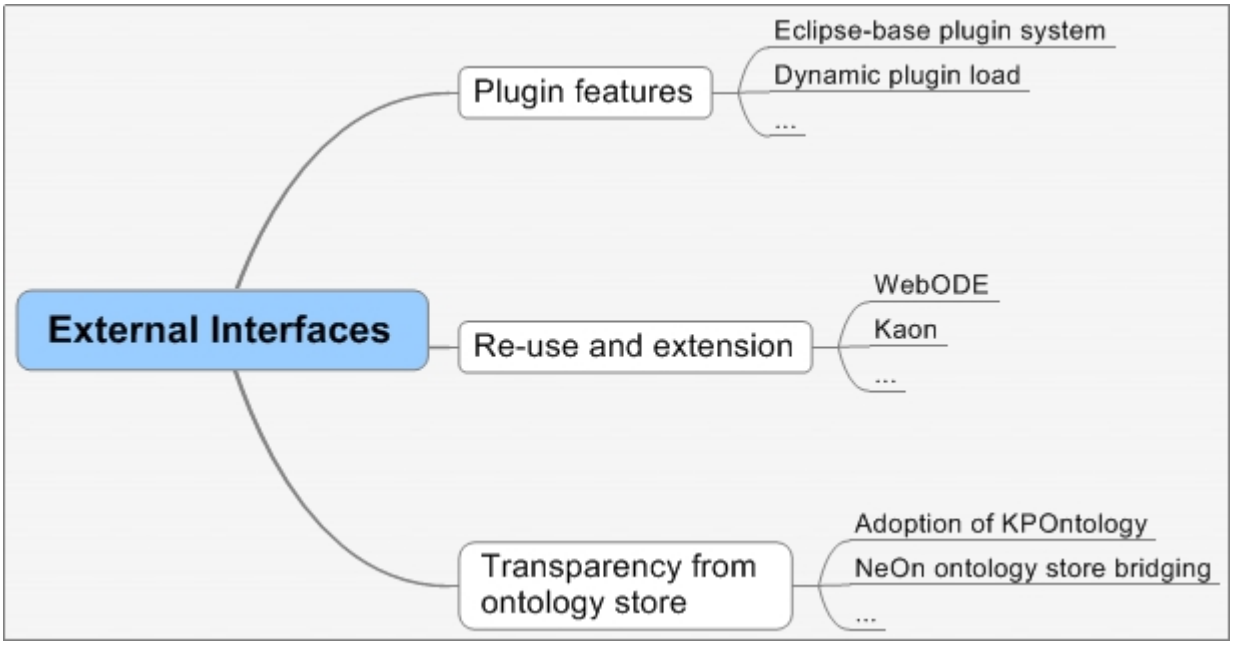

Figure 7: Classification of external interfaces requirements

NeOn shall inherit from Eclipse, on top of which the NeOn infrastructure will be implemented, a simple plugin mechanism that will allow easy integration of new components. This plugin mechanism will be endowed with summarization capabilities that will provide information about the plugin properties as required.

On the other hand, consortium members count with a certainly representative sample of Semantic Web and information integration technology that shall be reused and extended in the context of NeOn. Examples of this technology are e.g. KAON[6], WebODE, Magpie, OntoStudio, OntoBroker, GATE[7], or Tamino. Nevertheless, the use of such technology shall not be enforced but take place as a natural consequence of specific user needs. In this direction, the NeOn case studies will have a prominent role.

Directly related to section 0, NeOn shall be transparent from the different types of existing ontology stores. Thus, the main ontology stores shall be supported like e.g. JENA or Sesame. This can be achieved by means of KPOntology, a software library which abstracts away the underlying ontology store, providing access to the ontologies contained therein. The distributed repository layer shall use an extension of KPOntology that provides this functionality in a distributed environment.

[6] http://kaon2.semanticweb.org/ (Ontology management tool)

[7] http://gate.ac.uk/ (GATE Information extraction library)

Req. #	Functionality	Requirement Extracted	Description	Importance	Type	Layer of architecture	External Traceability	Internal Traceability
3.1.1	Plugin Features	Plugin System	NeOn shall be composed by means of plugins in order to allow easy integration of new components	Critical	FR	Neon toolkit	NeOn TA	3.6.4.2
3.1.1.1	Plugin Features	Eclipse-based plugin system	NeOn shall adopt Eclipse's plugin mechanism	Critical	FR	Neon toolkit	NeOn TA	2.1.1.3
3.1.1.2	Plugin Features	Dynamic plugin load	Plugin installation shall be a simple process	Critical	NFR	Neon toolkit	NeOn R&V	3.6.4.2
3.1.1.3	Plugin Features	Plugin summarization	NeOn shall provide means to summarize the functionalities and characteristics of plugins	Average	FR	Neon toolkit	NeOn R&V	3.6.4.2
3.1.2	Extension of state-of-the-art semantic technology	Re-use and extension	Re-use and extension of existing technology	Critical	FR	System-wide	NeOn TA	
3.1.2.1	Extension of state-of-the-art semantic technology	WebODE	NeOn shall re-use and extend WebODE	Average	FR	Distributed components	NeOn TA	
3.1.2.2	Extension of state-of-the-art semantic technology	KAON	NeOn shall re-use and extend KAON	Average	FR	Distributed components	NeOn TA	
3.1.2.3	Extension of state-of-the-art semantic technology	Magpie	NeOn shall re-use and extend Magpie	Average	FR	Distributed components	NeOn TA	
3.1.2.4	Extension of state-of-the-art semantic technology	OntoBroker and OntoStudio	NeOn shall re-use and extend Ontobroker and Ontostudio	Average	FR	Distributed components	NeOn TA	
3.1.2.5	Extension of state-of-the-art semantic technology	GATE	NeOn shall re-use and extend GATE	Average	FR	Distributed components	NeOn TA	
3.1.2.6	Extension of state-of-the-art semantic technology	XML-based repository	NeOn shall re-use and extend XML-based repository from SAG	Average	FR	Distributed repository	NeOn TA	
3.1.3	Transparency from ontology store	Transparency from ontology store	NeOn ontology repository shall be transparent from actual implementation	Critical	FR	Distributed repository	NeOn TA	3.2.1.1 3.2.1.2
3.1.3.1	Transparency from ontology store	Adoption of KPOntology	The distributed repository layer shall be partially built as an extension of the KPOntology library	Average	FR	Distributed repository	NeOn TA	3.2.1.1 3.2.1.2

3.1.3.2	Transparency from ontology store	NeOn ontology store bridging	The NeOn ontology repository shall be bridged to a number of existing technologies	Average	FR	Distributed repository	NeOn TA	3.2.1.1 3.2.1.2
3.1.3.2.1	Transparency from ontology store	Bridge to JENA	JENA shall be interfaced from the neon repository	Average	FR	Distributed repository	NeOn TA	3.2.1.1 3.2.1.2
3.1.3.2.2	Transparency from ontology store	Bridge to Sesame	Sesame shall be interfaced from the neon repository	Average	FR	Distributed repository	NeOn TA	3.2.1.1 3.2.1.2

Table 7: External interfaces

3.2.2 Detailed functions of the NeOn architecture

In this section we detail the main functions, introduced in section 0, that the NeOn infrastructure shall perform. Table 8 shows the complete collection of requirements on the functions of the NeOn architecture.

3.2.2.1 Geographically-transparent access to distributed repository

NeOn shall allow access to ontological and non ontological information pieces transparently from their location. Load and storage of information resources will be virtualized by the NeOn infrastructure aiming to abstract away the actual location of these resources. In this direction, the NeOn distributed repository will internally manage the physical resource where information is actually stored by means of a virtual file system which provides the user with a unique storage space. Figure 8 provides a high-level description of these statements.

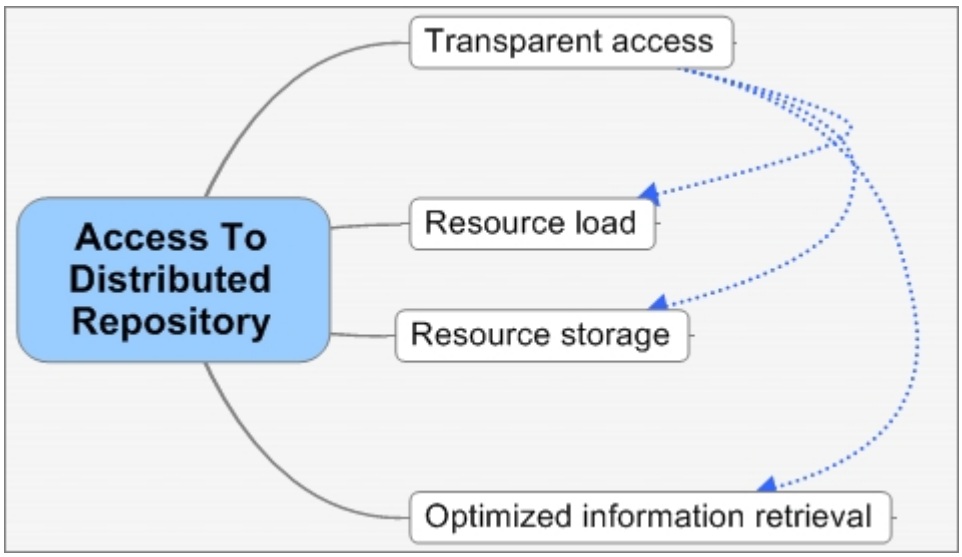

Figure 8: Transparent access to Distributed Repository

Distributed systems like NeOn need to implement optimizations that allow fast recovery of information since it will usually be geographically dispersed. These optimizations will extend the basic mechanism of information retrieval in terms of enhanced performance and flexibility. Foreseen optimizations are:

- **High availability of information resources:** The NeOn distributed repository will automatically distribute and maintain replicas of information resources among the nodes of the system. This will allow increasing performance of information retrieval by means of enabling local access to information.

- **Information caching:** Information resources will be cached once they are accessed by the first time. This will facilitate access to frequently used information. At the same time, it will require establishing a cache expiration time to dispose of unused cached information.

In both cases, NeOn shall implement a mechanism that ensures replica and cache consistency. Additionally, the distributed NeOn repository will provide an API to programmatically allow management of the knowledge resources contained within.

3.2.2.2 Support of heterogeneous knowledge

Four different types of information sources will be managed by NeOn, i.e. unstructured content like e.g. free text, structured knowledge without clear semantics, formal knowledge i.e. ontologies and data, and data annotations. In summary, we can classify knowledge as ontological and non-ontological knowledge. As shown in Figure 9, NeOn shall provide the necessary means to load and store non-ontological resources into the NeOn knowledge model, including methods that allow conversion from the original format into the NeOn data model. Additionally NeOn shall provide operational transparency, i.e. operations like e.g. query execution shall be executed transparently from the particular type of knowledge and information repository.

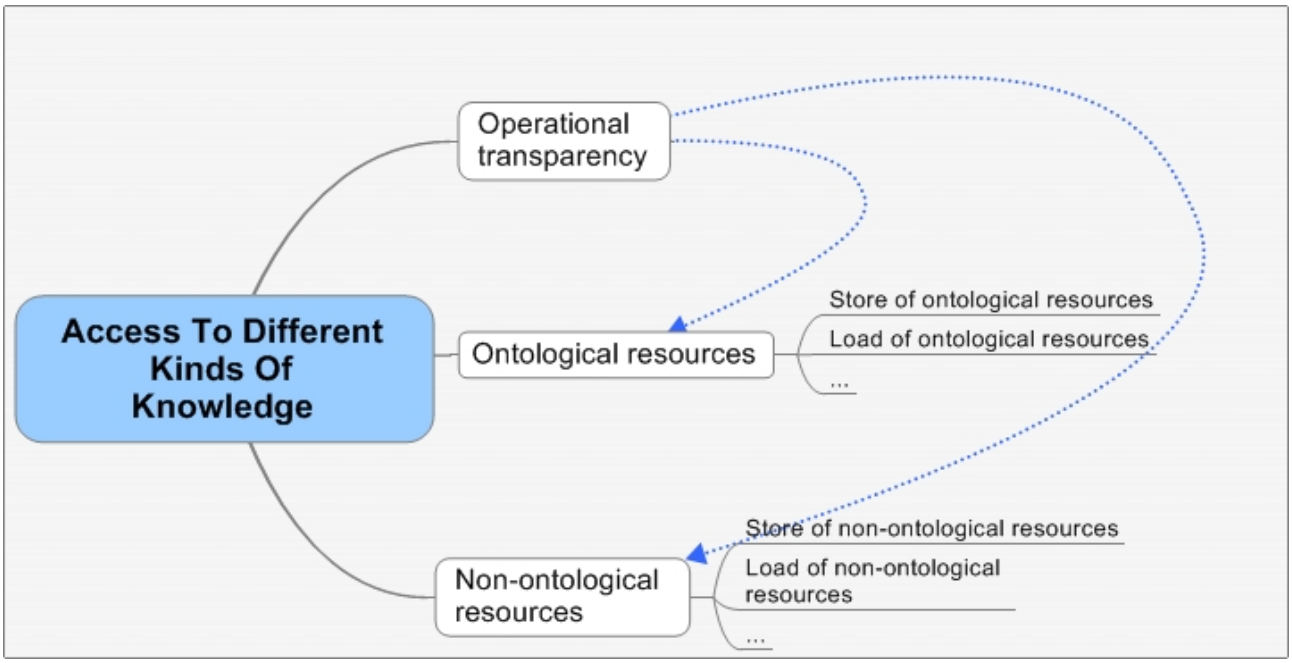

Figure 9: Support of heterogeneous knowledge

3.2.2.3 Dual language approach

NeOn shall support OWL-based and Rule languages.

3.2.2.4 Ontology modularization

Nowadays, one of the main problems in ontology engineering is to manage large ontologies where information tends to be dispersed. In such scenarios, consistent updates are certainly complicated as it is not straightforward to detect which parts of the ontology need to be modified, for example. As a consequence, ontologies tend to grow chaotically and the model contained within drifts further from the actual domain.

A way to avoid this kind of problems is to use ontology modules, by means of which ontology developers can decide how to structure their ontologies in self-contained units. NeOn shall support ontology modularization and provide the means to treat modules as first class citizens in the ontological realm. By extension, developers will have the means to manage ontologies to the desired module granularity level, i.e. facilities like e.g. ontology load and store shall also be applicable to ontology modules.

3.2.2.5 Multilinguality support

A fundamental feature, shared by the NeOn pharmaceutical and fishery case studies, is support for multilingual ontologies. It is the key for their respective domains that the models described by these ontologies are available in a series of different languages. Possible ways to implement multilinguality is by means of contextualized ontologies.

3.2.2.6 Mapping support

NeOn shall provide means to simplify alignment between entities of two or more globally inconsistent ontologies. In this direction, as in the case of ontology modules, mappings will be treated as first class ontology citizens (see Figure 10), with their own lifecycle. This will allow proper manipulation, creation, editing, and deletion of mappings.

On the other hand, ontologies evolve and change, and existing mappings between entities belonging to different networked ontologies need to be updated. NeOn shall automatically check mapping consistency throughout their entire lifecycle.

Additionally, mappings can be implicitly present either by means of ontology properties and relations or other mappings. Treatment of mappings as first class ontology entities, together with reflexive properties, are expected to allow reasoning on ontology mappings which will allow to bring out this hidden alignment knowledge.

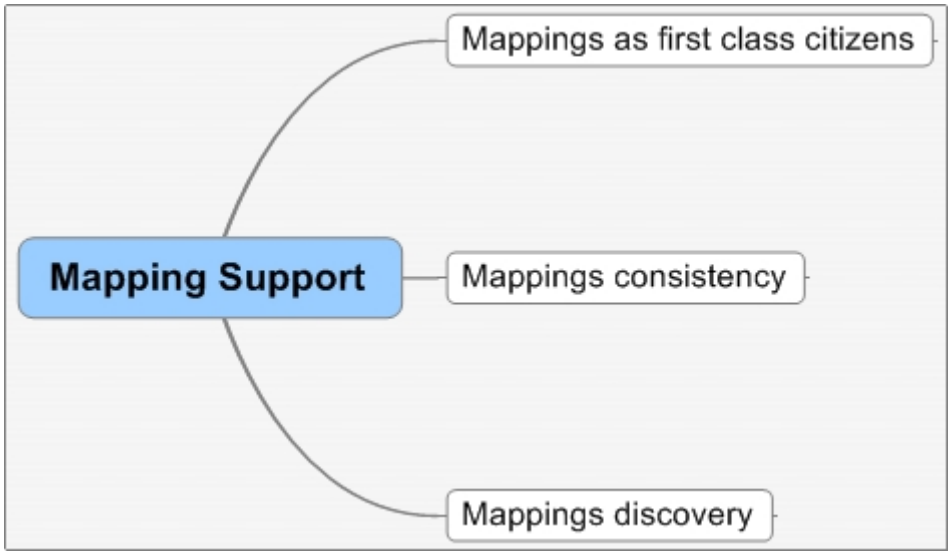

Figure 10: Requirements on mapping support

3.2.2.7 Support for contextualized ontologies

As introduced in previous sections, context support in networked ontologies is especially critical for NeOn. We define context as the set of all circumstances, properties, and facts within which the ontology has the desired semantics. Thus, (networked) ontologies are dependent on the context in which they are built or in which they are used. Additionally, different contexts can be valid for a particular ontology and all of them must be available when necessary. As a consequence, NeOn will provide the means to parameterize ontologies with respect to its different contexts.

3.2.2.8 Lifecycle support for ontology development

The main goal of NeOn is to support knowledge acquisition and ontology development and maintenance throughout their lifecycle, extending standard ontology development facilities to the arena of networked knowledge, where networking appeals both to the physical distribution of the available resources and to the connections between the entities of different ontologies. Figure 11 summarizes these functions.

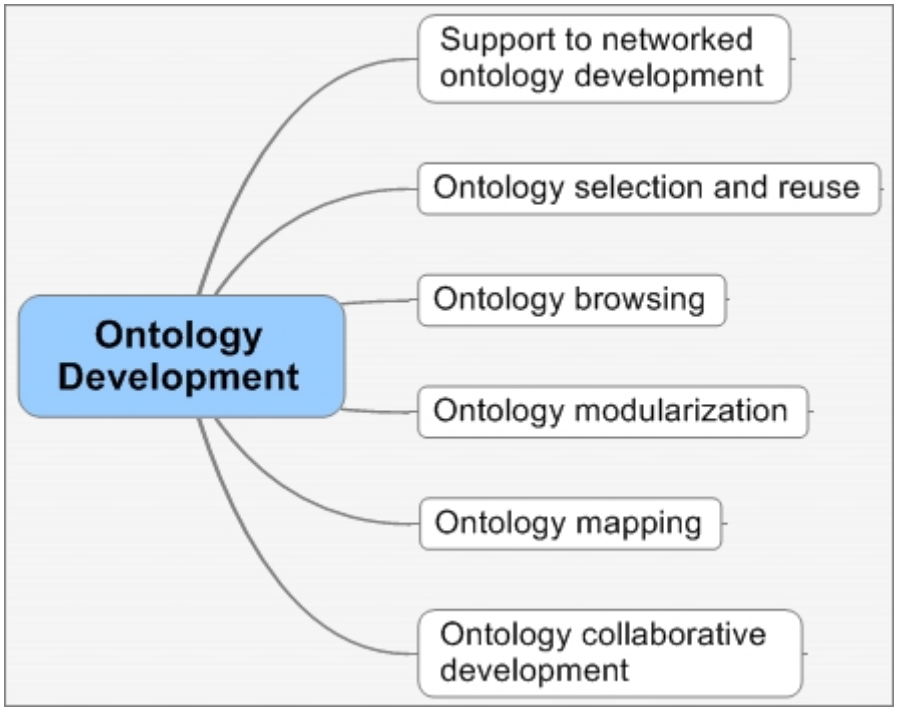

Figure 11: Functions for ontology development

The main functions that need to be supported are the following:

- **Networked ontology editing and building:** NeOn will extend the creation and editing of ontologies to the networked case. This task shall be assisted by means of consistency checkers and reasoners scaled to a network of ontologies.
- **Browsing and visualization of networked ontologies:** NeOn will extend ontology browsing and visualization to the case where several ontologies are semantically connected, creating a knowledge network.
- **Ontology selection and reuse:** NeOn will allow selection and reuse of ontologies and ontology entities from a network of ontologies.
- **Modularization in networked ontologies:** NeOn will support the creation and maintenance of trans-ontological modules, i.e. ontology modules built with parts of different ontologies simultaneously.

Additionally, NeOn shall support collaborative ontology development by means of adopting the CVS metaphor for collaborative work on shared ontological resources. This includes the following functions:

- **Synchronization of ontology concurrent updates:** In a networked, collaborative environment, concurrent updates of shared information are a very real possibility. In such cases, the necessary means to synchronize changes must be provided. CVS-like techniques will be implemented that allow committing changes safely.
- **Consistency checking for ontology concurrent updates:** Safe commitment of ontology updates needs prior detection of conflicts between concurrent changes. Inconsistencies can also appear between the updated portion of the ontology and the rest of the knowledge contained within. Both need to be detected and, when possible, remedial actions shall be automatically applied.
- **Ontology versioning:** The CVS metaphor shall be supported for collaborative ontology development.

Finally, ontology changes shall be propagated across the network of ontologies with which a particular ontology is connected. For consistency reasons, it might be impractical to automatically update dependent ontologies but the change can be evaluated and reported to the ontology developer by means of the corresponding GUI (see section 0).

3.2.2.9 Reasoning and inference

The NeOn backend shall provide reasoning capabilities on top of the aggregated knowledge contained in a network of ontologies. Regardless of ontology distribution, inference will take place on a common knowledge meta-model. Thus, it is necessary to ensure consistency of the ontologies and data to be reasoned upon. A nice optimization would be to allow the encapsulation of the knowledge participating in the reasoning process, avoiding executing it against all the knowledge held by the system.

3.2.2.10 Query support

Standard ontology query languages like e.g. SPARQL shall be supported by NeOn, allowing the retrieval of information contained within.

3.2.2.11 Question formulation

Question formulation functionality shall be built on top of ontology querying that will allow users to build more complex and expressive questions for information retrieval. Natural language questions shall be supported that after automatic translation into the corresponding ontology query standard will be submitted to the query service and run against the information contained in the system.

Usually, this kind of interfaces show a gap between what the user tried to express in the form of a free text question and the information retrieved by the system as the answer to this question. Thus, it is necessary to explain the results obtained in terms of the question helping the user understand the result set.

3.2.2.12 Semantic annotation

NeOn shall incorporate semantic annotation functionalities (see Figure 12), which will provide information characterization in terms of the model described in a set of networked ontologies. The different types of knowledge sources available in NeOn apart from ontological knowledge, i.e. unstructured knowledge like e.g. free text, and structured, non-semantic knowledge like e.g. an XML store, will be subject of semantic annotation. Different annotation techniques shall be provided, including data and text mining.

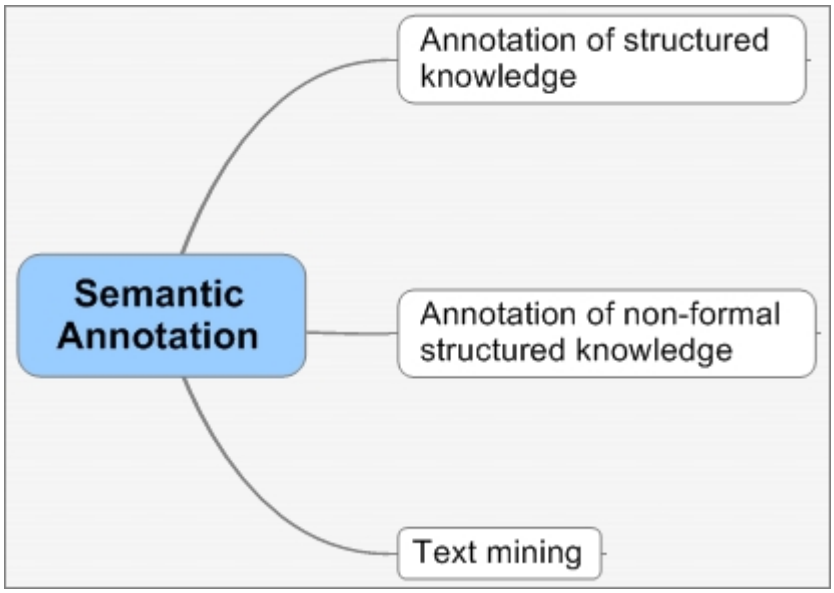

Figure 12: Semantic annotation functionalities

3.2.2.13 Access management to information resources

Access rights to information resources managed by NeOn need to be administrated. Case studies have shown that not all kinds of users can have access to all types of information. For example, a particular ontology describing

an invoice model must remain stable unless a change in the law or the business dynamics requires it to be updated. Hence, the capability to see, modify, or delete complete or parts of ontologies shall be constrained by the author.

3.2.2.14 User profiling

NeOn shall accumulate information about user profiles during system interaction and provide a customized set of functionalities according to the inferred profile.

3.2.2.15 Ontology summarization

NeOn shall provide users with the means to summarize ontology state and properties.

3.2.2.16 Provenance support

Ontology characteristics and state provided by the functionality described in section 0 are closely related to ontology provenance information. Ontology traceability shall be kept by automatically storing relevant information like authors or contributors, as well as versioning information. This information will be reported to ontology developers by means of the corresponding GUI.

3.2.2.17 System documentation and help

The different user types of NeOn shall be provided with extensive help and training materials, including tutorials, which will allow improving understanding of the system, either by means of providing information that solves specific problems or by helping the novice ontology developer get acquainted with the system. A help subsystem shall be built that provides contextualized help, in terms of the materials available in each situation and the particular user profile.

3.2.2.18 Support for different client types

Rich and thin clients will be supported by the NeOn architecture, as described in section 0.

3.2.2.19 Service access to the NeOn backend

Access to the NeOn backend shall be provided both to rich and thin clients. A programmatic Java API will be provided to rich clients while a socket-based interface will be available for thin clients.

Req. #	Functionality	Requirement Extracted	Description	Imp.	Type	Layer of architecture	External Traceability	Internal Traceability
3.2.1	Access to distributed repository	Transparent access to information resources	NeOn shall allow access to ontological and non ontological information pieces transparently from their location	Critical	FR	Distributed repository	NeOn TA	2.1.1.1
3.2.1.1	Access to distributed repository	Distributed Repository access API	NeOn distributed repository API shall provide access to knowledge contained in the repository	Critical	FR	Distributed repository	NeOn TA	2.1.1.1
3.2.1.2	Transparent access to distributed repository	Resource transparent storage	NeOn shall provide transparent access to resource storage. The NeOn Distributed Repository shall internally manage the physical location where the resource is actually stored.	Critical	NFR	Distributed repository	NeOn R&V	2.1.1.1

3.2.1.3	Transparent access to distributed repository	Resource transparent load	NeOn shall provide a virtual file system which abstracts away the actual resource location, offering a virtual unique storage space	Critical	NFR	Distributed repository	NeOn R&V	2.1.1.1
3.2.1.4	Access to distributed repository	Optimized information retrieval	NeOn shall implement optimizations to the basic mechanism of information retrieval that allow to increase performance and flexibility	Average	FR	Distributed repository	NeOn TA	2.1.1.1
3.2.1.4.1	Access to distributed repository	High availability of information resources	The NeOn architecture shall automatically distribute replicas of information resources to facilitate local access to information.	Average	FR	Distributed repository	NeOn TA	2.1.1.1
3.2.1.4.1.1	Access to distributed repository	Consistency maintenance of replicas	NeOn shall implement a device to ensure consistency of distributed replicas	Average	FR	Distributed repository	NeOn TA	2.1.1.1
3.2.1.4.1	Access to distributed repository	Caching	The NeOn architecture shall implement a caching mechanism to improve performance	Average	FR	Distributed repository	NeOn TA	2.1.1.1
3.2.2	Support of heterogeneous knowledge	Access to the different kinds of knowledge	NeOn shall provide access to the main four different kinds of information i.e. unstructured content, structure knowledge without a clear semantics, formal knowledge (ontologies and data), and semantic annotations	Critical	FR	Distributed repository	NeOn TA	2.1.1.5
3.2.2.1	Support of heterogeneous knowledge	Operational transparency	Operations, like the query execution mechanism of NeOn, shall be transparent from the kind of information and repository	Critical	NFR	Distributed components	NeOn TA	2.1.1.5
3.2.2.2	Reuse of legacy non-ontological resources	Load non-ontological resources	NeOn shall provide facilities to load non-ontological resources in the NeOn knowledge model	Critical	FR	Distributed components Distributed repository	NeOn R&V NeOn case studies	2.1.1.5
3.2.2.2.1	Reuse of non-ontological resources	Converting non-ontological resources	NeOn shall provide the appropriate conversion	Critical	FR	Distributed components	NeOn R&V	2.1.1.5

			methods between the non-ontological resources and the specifics of the NeOn data model					
3.2.2.3	Reuse of non-ontological resources	Store non-ontological resources	NeOn shall provide facilities to store non-ontological resources in the NeOn knowledge model	Critical	FR	Distributed components	NeOn R&V	2.1.1.5
3.2.2.4	Reuse of existing ontological resources	Load ontological resources	NeOn shall provide facilities to load ontological resources in the NeOn knowledge model	Critical	FR	Distributed components	NeOn R&V	2.1.1.5
3.2.2.5	Reuse of ontological resources	Store ontological resources	NeOn shall provide facilities to store ontological resources in the NeOn knowledge model	Critical	FR	Distributed components	NeOn R&V	2.1.1.5
3.2.3	Ontology modularization	Management of ontology modularization	NeOn shall allow to structure ontologies using modules	Critical	FR	Distributed components	NeOn architecture partners NeOn case studies	
3.2.3.1	Ontology modularization	Treatment of ontology modules as first class citizens	NeOn shall allow users to manage ontologies to the module granularity level, i.e. ontology facilities like ontology load and store, shall be applicable to ontology modules, too.	Critical	FR	NeOn toolkit	NeOn architecture partners	
3.2.4	Semantic annotation	Semantic annotation	NeOn shall allow to annotate information sources	Critical	FR	Distributed components	NeOn TA NeOn case studies	2.1.1.6, 2.1.1.5
3.2.4.1	Data annotation	Semantic Annotation of unstructured knowledge	NeOn shall allow to annotate unstructured content, e.g. text	Critical	FR	Distributed components	NeOn TA	2.1.1.6, 2.1.1.5
3.2.4.2	Data annotation	Semantic Annotation of non-formal structured knowledge	NeOn shall allow to annotate structured knowledge without clear semantics, e.g. database and XML repositories	Critical	FR	Distributed components	NeOn TA	2.1.1.6, 2.1.1.5
3.2.4.3	Text mining	Text mining	NeOn shall support text mining	Average	FR	Distributed components	NeOn TA	2.1.1.7
3.2.5	Access to Distributed Components	Middleware layer accessible by an API	The middleware layer API shall provide programmatic access to the NeOn distributed components	Critical	FR	Distributed components	NeOn TA	2.1.1.4
3.2.6	Provenance Support	Ontology provenance service	NeOn shall keep ontology traceability by automatically storing relevant	Average	FR	Distributed components	NeOn R&V	2.1.1.13

			information like authors or contributors, as well as timeline and versions					
3.2.7	Summarization	Ontology briefing	NeOn shall provide users with the means to summarize the state and characteristics of the ontology	Average	FR	Distributed components	NeOn R&V	2.1.1.8
3.2.8	User profiling	User profiling	NeOn shall accumulate information about user profiles during system interaction and provide a customized set of functionalities according to that profile	Average	FR	Distributed components	NeOn R&V NeOn case studies	2.1.1.3, 2.1.1.13
3.2.9	Mapping support	Mapping capabilities	NeOn shall provide techniques for simplifying the mapping process between entities of two or more ontologies that are globally inconsistent	Critical	FR	Distributed components	NeOn TA NeOn case studies	2.1.1.2
3.2.9.1	Mapping support	Treatment of mappings as first class citizens	NeOn shall provide means to treat mappings as ontological entities themselves, with their own lifecycle, in order to allow easy manipulation, creation, editing, and deletion.	Critical	FR	Distributed components	NeOn R&V	
3.2.9.2	Mapping support	Mapping consistency checking	NeOn shall automatically check the consistency of mappings throughout their lifecycle	Critical	FR	Distributed components	NeOn R&V	
3.2.9.3	Mapping support	Discovery of implicit mappings	NeOn shall have reflexive capabilities which allow reasoning on ontology mappings with the aim of checking them as well as inferring and automatically applying new mappings	Critical	FR	Distributed components	NeOn R&V	2.1.1.10
3.2.10	Extended support for ontology relations	Extended support for ontology relations	NeOn shall provide the means to improve acquisition of relations	Critical	FR	Distributed components	NeOn TA	
3.2.10.1	Extended support for ontology relations	Multiple inheritance relations	NeOn shall support multiple inheritance relations	Average	FR	Distributed components	NeOn TA	
3.2.10.2	Extended support for ontology	Complex slot compositions	NeOn shall support multi slot composition	Average	FR	Distributed components	NeOn TA	

	relations							
3.2.11	Lifecycle support for ontology development	Knowledge Acquisition	NeOn shall support knowledge acquisition and ontology development	Critical	FR	NeOn toolkit Distributed components	NeOn case studies	2.1.1.1, 2.1.1.2, 2.1.1.3, 2.1.1.4
3.2.11.1	Lifecycle support for ontology development	Extended ontology support to the networked paradigm	NeOn shall extend standard ontology development facilities to the distributed, networked case	Critical	FR	NeOn toolkit Distributed components	NeOn R&V	
3.2.11.1.1	Lifecycle support for ontology development	Networked Ontology editing and building	NeOn shall extend ontology creation and editing to the networked case	Critical	FR	NeOn toolkit Distributed components	NeOn R&V	
3.2.11.1.1.1	Lifecycle support for ontology development	Ontology checking and consistency	NeOn shall assist the ontology editing and building process by providing users with consistency checkers and reasoners	Average	FR	Distributed components	NeOn R&V	
3.2.11.1.2	Lifecycle support for ontology development	Browsing and visualization of networked ontologies	NeOn shall extend ontology browsing and visualization to the networked case	Critical	FR	NeOn toolkit Distributed components	NeOn R&V	
3.2.11.1.3	Lifecycle support for ontology development	Ontology selection and reuse	NeOn shall provide the means to access one or several ontology repositories for ontology selection. Then, the selected ontology/ies will be imported by means of the ontology load functionality	Critical	FR	NeOn toolkit Distributed repository	NeOn R&V	
3.2.11.1.4	Lifecycle support for ontology development	Selection and reuse of networked ontologies	NeOn shall allow to select and reuse ontologies and ontology entities from a network of ontologies	Critical	FR	NeOn toolkit Distributed components	NeOn R&V	
3.2.11.1.5	Lifecycle support for ontology development	Modularization in networked ontologies	NeOn shall allow to create and maintain trans-ontological modules	Critical	FR	NeOn toolkit Distributed components	NeOn R&V	
3.2.11.1.6	Lifecycle support for ontology development	Mappings across networked ontologies	Mappings are inherent to the concept of ontology networking. NeOn shall provide the means to support them.	Critical	FR	Distributed components	NeOn R&V	
3.2.11.1.7	Ontology collaborative development	Ontology collaborative development	NeOn shall allow collaborative ontology development	Critical	FR	NeOn toolkit Distributed components	NeOn TA	
3.2.11.1.7.1	Ontology collaborative development	Adoption of CVS development model	NeOn shall adopt the CVS metaphor for collaborative work on shared ontological resources	Critical	FR	NeOn toolkit Distributed repository	NeOn TA	

3.2.11.1.7.1.1	Ontology collaborative development	Synchronization of ontology concurrent updates	NeOn shall allow synchronization of ontologies edited by multiple users	Critical	FR	Distributed repository	NeOn TA	
3.2.11.1.7.1.2	Ontology collaborative development	Consistency checking for ontology concurrent updates	NeOn shall provide the means to detect the introduction of inconsistencies when a shared ontology is updated. In such case remedial actions should be automatically taken (when possible) and affected users notified	Critical	FR	Distributed components Distributed repository	NeOn R&V	
3.2.11.1.7.1.3	Ontology collaborative development	Ontology versioning	NeOn shall provide support for ontology versioning based on the CVS metaphor	Critical	FR	NeOn toolkit Distributed repository	NeOn R&V	
3.2.11.1.7.1.3.1	Ontology collaborative development	Global awareness of local ontology versioning	NeOn shall ease the adoption of new versions of an ontology in an already existing network of ontologies	Average	FR	NeOn toolkit Distributed repository	NeOn R&V	
3.2.11.1.7.2	Ontology collaborative development	Change propagation in a ontology network	NeOn shall automatically propagate updates in a particular ontology throughout the network of interrelated ontologies	Average	FR	Distributed components Distributed repository	NeOn R&V	
3.2.11.1.7.2.1	Ontology collaborative development	Ontology network consistency maintenance	NeOn shall maintain partial and local consistency among a network of ontologies	Critical	FR	Distributed components Distributed repository	NeOn TA	
3.2.12	Reasoning & Inferencing	Reasoning with networked ontologies	The NeOn backend shall support reasoning capabilities using the aggregated knowledge contained in a set of networked ontologies	Critical	FR	Distributed components	NeOn TA	2.1.1.10
3.2.12.1	Reasoning & Inferencing	Unique data model for modelling and reasoning infrastructure	NeOn reasoning and inference operates on the same data model as the NeOn knowledge formation	Critical	NFR	Distributed components	NeOn TA	
3.2.12.2	Reasoning & Inferencing	Reasoning with up-to-date data	NeOn shall ensure ontology and data are consistent for reasoning and inference	Critical	NFR	Distributed components	NeOn TA	
3.2.13	Query support	Query support	NeOn shall provide the means, e.g. by supporting standard query	Critical	FR	Distributed components	NeOn R&V	2.1.1.11

			languages, to query the information contained within					
3.2.14	Question formulation	Question formulation	NeOn shall provide the means that allow users to express queries naturally, e.g. by free test questions	Average	FR	NeOn toolkit	NeOn R&V	2.1.1.12
3.2.14.1	Question formulation	Answer Explanation	NeOn shall explain the results upon question answering in the terms of the formulated question	Critical	FR	Distributed components	NeOn R&V	2.1.1.12
3.2.15	Multilinguality Support	Multilinguality	NeOn shall support multilingual ontologies	Critical	FR	Distributed components	NeOn R&V	
3.2.16	Support for contextualized ontologies	Context support in networked ontologies	Ontologies are dependent on the context in which they are built or in which they are used. NeOn shall support context in networked ontologies	Critical	FR	Distributed components	NeOn R&V NeOn case studies	
3.2.16.1	Support for contextualized ontologies	Multi-context ontologies	Different contexts shall be applied to networked ontologies	Critical	FR	Distributed components	NeOn R&V NeOn case studies	
3.2.16.2	Support for contextualized ontologies	Support for ontology context parameters	NeOn shall provide means to parameterize ontologies with respect to a given context	Critical	FR	Distributed components	NeOn R&V NeOn case studies	

Table 8: Functions provided by the NeOn infrastructure

3.2.3 Performance requirements

This section contains a number of factors that need to be taken into account in order to maximize performance during the execution of the functionalities provided by NeOn. Additionally, the key aspects where performance is required are highlighted. Figure 13 shows these factors and aspects.

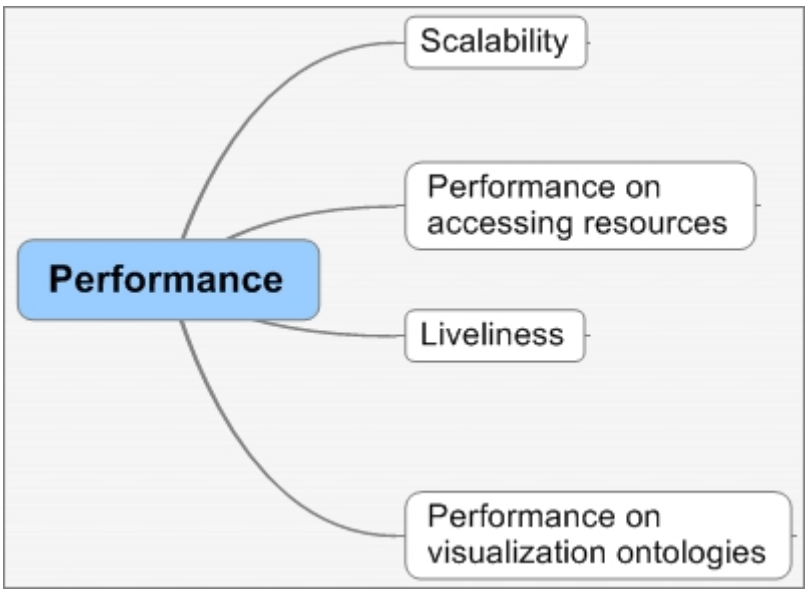

Figure 13: Performance factors and application contexts

Performance is closely related to scalability. If a distributed system like NeOn is not scalable, then, there is a high probability, it will not be meeting performance targets either. There are three different ways in which NeOn shall be scalable. Nevertheless, there is a limit up to which a distributed system can be scaled. Above that limit, performance decreases.

- **Load scalability:** NeOn shall allow expansion and contraction of its resource pool without performance decrease. This applies both to ontological and non-ontological resources.
- **Geographic scalability:** NeOn shall maintain usefulness and performance regardless of how apart its resources and components are located.
- **Administrative scalability:** NeOn shall allow increasing concurrent access without performance loss.

Additionally, performance shall not suffer from the heterogeneity of the knowledge resources maintained by NeOn. It shall remain homogenous with respect to the nature of this knowledge, either unstructured, structured, or formal. Operational transparency regarding the sources implied in the execution of a NeOn service shall be observed in such way that performance be independent from them.

We have identified a number of circumstances where performance is certainly a critical requirement:

- **Load and storage of ontological and non-ontological resources** shall be timely and efficient.
- **Ontology browsing and visualization:** NeOn visualization techniques shall guarantee quick access to the required ontology elements and easily browse taxonomy and data. By extension, this also applies to networked ontologies.

The execution of NeOn services shall also comply with the following properties:

- **Liveliness:** NeOn services and functions shall guarantee that results of execution are returned in a limited amount of time.
- **Progress awareness:** NeOn GUIs shall keep users informed of the current state of service execution.

Req. #	Functionality	Requirement Extracted	Description	Importance	Type	Layer of architecture	Internal Traceability
3.3.1	Access to distributed Information repositories	Scalability	NeOn shall be scalable	Critical	NFR	Distributed repository Distributed components	3.2.1 3.4.1.1

3.3.1.1	Access to distributed Information repositories	Load scalability	NeOn shall allow to expand and contract its resource pool with no performance decrease	Critical	NFR	Distributed repository	3.2.1 3.4.1.1
3.3.1.1.1	Access to distributed Information repositories	Load scalability for non-ontological resources	NeOn shall allow to expand and contract its pool of non formal resources with no performance decrease	Critical	NFR	Distributed repository	3.2.1 3.4.1.1
3.3.1.1.2	Access to distributed Information repositories	Load scalability for ontological resources	NeOn shall allow to expand and contract its ontology pool with no performance decrease	Critical	NFR	Distributed repository	3.2.1 3.4.1.1
3.3.1.2	Access to distributed Information repositories	Geographic scalability	NeOn shall maintain usefulness and performance, regardless how apart its resources and components are	Critical	NFR	Distributed repository Distributed components	3.2.1 3.4.1.1
3.3.1.3	Access to distributed Information repositories	Administrative scalability	NeOn shall allow increasing concurrent access without performance loss	Critical	NFR	Distributed repository	3.2.1 3.4.1.1
3.3.1.4	Access to distributed Information repositories	Scalability limitations	There is a limit up to which we can scale a distributed system, and above that the performance of the system degrades	Critical	NFR	Distributed repository Distributed components	3.2.1 3.4.1.1
3.3.2	Support of heterogeneous knowledge	Performant access to heterogeneous knowledge	Performance of access to knowledge repository shall be homogenous regardless the nature of this knowledge, i.e. unstructured, structured, or formal	Average	NFR	Distributed repository Distributed components	3.2.2 3.4.1.2
3.3.2.1	Support of heterogeneous knowledge	Performant operational transparency	Operational transparency shall not imply performance loss	Average	NFR	Distributed repository Distributed components	3.2.2.1 3.4.1.2
3.2.2.2	Reuse of ontological Resources	Efficient reuse of existing ontological resources	NeOn shall provide efficient support for reuse of existing ontological resources	Critical	NFR	Distributed repository Distributed component	3.2.2.4 3.4.1.2.1

3.3.2.2.1	Reuse of ontological Resources	Performant load of ontological resources	Loading ontological resources into the NeOn knowledge model shall be timely and efficient	Critical	NFR	Distributed repository Distributed component	3.2.2.4 3.4.1.2.1
3.3.2.2.1	Reuse of ontological Resources	Performant storage of ontological resources	Storing updated ontological resources in the repository shall be timely and efficient	Critical	NFR	Distributed repository	3.2.2.5 3.4.1.2.1
3.2.2.3	Reuse of non-ontological resources	Efficient reuse of existing non-ontological resources	NeOn shall provide efficient support for reuse of existing non-ontological resources	Critical	NFR	Distributed repository Distributed component	3.2.2.2 3.4.1.2.2
3.3.2.3.1	Reuse of non-ontological resources	Performant load of non-ontological resources	Loading non-ontological resources into the NeOn knowledge model shall be timely and efficient	Critical	NFR	Distributed repository Distributed component	3.2.2.2 3.4.1.2.2
3.3.2.3.1	Reuse of non-ontological resources	Performant storage of ontological resources	Storing updated non-ontological resources in their native repository shall be timely and efficient	Critical	NFR	Distributed repository	3.2.2.3 3.4.1.2.2
3.3.3	Execution of NeOn services and functionalities	Liveliness	All NeOn services and functions shall guarantee that execution results are returned in a limited amount of time	Critical	NFR	Distributed components	
3.3.3.1	Execution of NeOn services and functionalities	Progress user-awareness	NeOn shall keep users informed of service execution progress	Average	NFR	NeOn toolkit	
3.3.4	Visualization	Fluent ontology browsing and visualization	NeOn visualization techniques shall guarantee quick access to the required ontology elements and easily browse ontology taxonomy and data	Critical	NFR	NeOn toolkit	
3.3.4.1	Visualization	Fluent networked ontology browsing and visualization	NeOn visualization techniques shall guarantee quick access to the required ontology elements and easily browse networked ontologies taxonomy and data	Critical	NFR	NeOn toolkit	

Table 9: Performance requirements

3.2.4 Logical database requirements

Next, requirements on the information repositories maintained by NeOn are described. The whole list of requirements is contained in Table 1. NeOn information repositories will be distributed and heterogeneous. Accordingly with section 0, ontological repository stores like e.g. JENA, Sesame, OWLIM, and Kowari shall be supported by NeOn. Additionally NeOn shall manage information contained in other information repositories. The most representative of these non-ontological information repositories are relational databases. NeOn shall support widespread database systems like Oracle, MySQL, and Postgres. Special interesting would be to manage XML information repositories built on top of SAG's Tamino.

Req. #	Functionality	Requirement Extracted	Description	Importance	Type	Layer of architecture	Internal Traceability
3.4.1	Information repositories	Information repositories	NeOn shall manage and run on information repositories	Critical	FR	Distributed repository	2.1.1.1 2.1.1.5 2.1.2.3.4
3.4.1.1	Distributed repositories	Distributed Information repositories	Information repositories shall be geographically distributed	Critical	FR	Distributed repository	2.1.1.1 3.2.1
3.4.1.2	Heterogeneous knowledge sources	Heterogeneous knowledge sources	NeOn shall manage heterogeneous knowledge sources	Critical	FR	Distributed repository	2.1.1.5 3.2.2
3.4.1.2.1	Ontological repositories	Ontological repositories	NeOn shall manage formal knowledge in the form of ontologies	Critical	FR	Distributed repository	2.1.1.5 2.1.2.3.4.1 3.5.1.5.1 3.5.1.5.2
3.4.1.2.1.1	Ontological repositories	Ontological repository stores	NeOn shall support widespread ontology store, e.g. JENA, Sesame, OWLIM, Kowari	Critical	FR	Distributed repository	2.1.1.5 3.1.3.2
3.4.1.2.2	Non-ontological repositories	Non-ontological repositories	NeOn shall manage non-formal, legacy information sources	Critical	FR	Distributed repository	2.1.1.5 2.1.2.3.4.2
3.4.1.2.2.1	Non-ontological repositories	Free text	NeOn shall support free text information sources	Critical	FR	Distributed repository	2.1.1.5

3.4.1.2.2.2	Non-ontological repositories	Structured knowledge	NeOn shall support structured knowledge with no clear semantics	Critical	FR	Distributed repository	2.1.1.5
3.4.1.2.2.2.1	Non-ontological repositories	XML knowledge	NeOn shall support Software AG XML information repository	Average	FR	Distributed repository	2.1.1.5 3.1.2.6
3.4.1.2.2.2.1	Non-ontological repositories	Relational databases	NeOn shall support widespread database systems, e.g. Oracle, MySQL, Postgres	Average	FR	Distributed repository	

Table 10: Logical database requirements

3.2.5 Standards compliance

NeOn shall be compliant with the current and future Semantic Web standards. Table 11 details the complete set of architecture requirements focused on compliance with standards.

Several types of clients based on a number of technologies shall be integrated with NeOn. As advanced in section 0, the NeOn backend will provide an API-based interface to rich clients and a socket-based interface to thin clients. The API-based interface shall support Java clients like e.g. JSP and also COM clients like e.g. ASP, .NET, and VB.

As shown in Figure 4, OWL and Rule languages coexist in the Semantic Web arena. Thus, NeOn, as the next generation ontology framework, will take a dual language approach and support both trends, allowing users to exploit the properties of their respective paradigms conveniently. Additionally, other languages shall be supported as required.

Finally, SPARQL shall be adopted as standard query language.

Req. #	Functionality	Requirement Extracted	Description	Importance	Type	Layer of architecture	External Traceability	Internal Traceability
3.5.1.1	Service access to the NeOn backend	Java-based interface to NeOn backend	The NeOn backend shall provide a Java-based interface to rich clients	Critical	FR	Distributed components	NeOn TA	2.1.1.15
3.5.1.2	Service access to the NeOn backend	Socket-based interface to NeOn backend	The NeOn backend shall provide a socket-based interface to thin clients	Average	FR	Distributed components	NeOn TA	2.1.1.15
3.5.1.3	Client support	Client support	Several types of clients shall connect to the NeOn backend by means of the above-mentioned interfaces	Critical	FR	Neon toolkit	NeOn TA	3.5.1.1 3.5.1.2
3.5.1.3.1	Client support	Support of Java Clients	Java clients	Average	FR	Distributed components	NeOn TA	3.5.1.1 3.5.1.2

3.5.1.3.1.1	Client support	JSP client	JSP clients	Average	FR	Distributed components	NeOn TA	3.5.1.1 3.5.1.2
3.5.1.3.2	Client support	Support of COM Clients	COM clients	Average	FR	Distributed components	NeOn TA	3.5.1.1 3.5.1.2
3.5.1.3.2.1	Client support	ASP client	ASP clients	Average	FR	Distributed components	NeOn TA	3.5.1.1 3.5.1.2
3.5.1.3.2.2	Client support	.NET client	.NET clients	Average	FR	Distributed components	NeOn TA	3.5.1.1 3.5.1.2
3.5.1.3.2.3	Client support	Visual Basic client	Visual Basic clients	Average	FR	Distributed components	NeOn TA	3.5.1.1 3.5.1.2
3.5.1.3.3	Client support	Support other kind of clients	Support other kind of clients	Average	FR	Distributed components	NeOn TA	3.5.1.1 3.5.1.2
3.5.1.4	Support for common ontology query languages	Support for common ontology query languages	Query language standards shall be supported by the NeOn query service	Critical	FR	Distributed components	NeOn R&V	3.2.18
3.5.1.4.1	Support for common ontology query languages	Support for common ontology query languages	SPARQL shall be supported by the NeOn query service	Critical	FR	Distributed repository	NeOn R&V	3.2.18
3.5.1.5	Dual language approach	Semantic Web Standards	A family of Semantic Web Standards shall be used as formal representation languages	Critical	FR	Distributed repository	NeOn TA	2.1.1.16
3.5.1.5.1	Dual language approach	OWL support	Several OWL flavours	Average	FR	Distributed repository	NeOn architecture partners	2.1.1.16
3.5.1.5.2	Dual language approach	Support for rule-based languages, e.g. FLogic, DL	Other languages shall be used	Average	FR	Distributed repository	NeOn TA	2.1.1.16

Table 11: Standards compliance

3.2.6 Software system attributes

Some characteristics that the NeOn architecture shall meet can be included as non-functional requriements in the specification. These requirements include properties like reliability and security.

3.2.6.1 Reliability

Reliability requirements have focused on system availability, including resource and service availability. Since NeOn is a distributed system, special concern deserves fault tolerance issues. Both in the case of resource and service availability, NeOn shall guarantee normal access to the overall system in case some problem occurs in a determined node. Table 12 shows complete information on reliability requirements.

Req #	Functionality	Requirement Extracted	Description	Importance	Type	Layer of architecture	External Traceability	Internal Traceability
3.6.1.1	System availability	System availability	NeOn shall guarantee system availability	Critical	NFR	NeOn toolkit Distributed components Distributed repository	[1]	
3.6.1.1.1	Resource availability	Resource fault tolerance	Regardless of whether any node of the distributed repository is temporarily unavailable, NeOn shall guarantee normal access to the rest of the system	Critical	NFR	Distributed repository	[1]	3.4.1.1
3.6.1.2.1	Service availability	Service fault tolerance	NeOn services shall be decoupled from each other to guarantee overall system availability in case a particular service is temporarily disabled	Critical	NFR	NeOn toolkit Distributed components	[1]	

Table 12: Reliability requirements

3.2.6.2 Security

This section specifies the requirements necessary to prevent malicious access to NeOn data or functionality from occurring in the NeOn framework with the aim to avoid system malfunction and unexpected behaviour. Ontology access rights shall allow to specify what users can access the ontologies and data maintained by NeOn and under what conditions. The analogy with a Unix filesystem applies here, i.e. resource owners shall specify what rights the rest of the users will have on them. Additionally, certain information, both contained in ontological and non ontological requirements, shall be constrained to a reduced group of users.Table 13 shows specific requirements on security for NeOn.

Req. #	Functionality	Requirement Extracted	Description	Importance	Type	Layer of architecture	External Traceability	Internal Traceability
3.6.3.1	Access management to Information resources	Access management to Information resources	Access to information resources shall be administrated	Average	FR	Distributed repository	NeOn R&V Case studies	3.2.1
3.6.3.1.1	Access management to Information	Access management to ontological	NeOn shall allow to specify access permissions to	Average	FR	Distributed repository	NeOn R&V Case studies	3.2.1 3.4.1.2.1

	resources	resources	ontological resources					
3.6.3.1.2	Access management to Information resources	Access management to non-ontological resources	NeOn shall allow to specify access permissions to non-ontological resources	Average	FR	Distributed repository	NeOn R&V Case studies	3.2.1 3.4.1.2.2

Table 13: Security requirements

3.2.6.3 Maintainability

Modular design is a key factor in order to build a complex system like NeOn, which at the same time allows simple and cheap maintenance in terms of effort. As shown in Table 14, loose coupling is NeOn's bet in this regard. Loose coupling of components and resources shall allow increasing software modularity. Thus, the cost of software update, substitution of old components with new versions, and uptake of external resources will be reduced. It will also be limited to the directly attained components, since interfaces with the rest of the system will be kept in most occasions, allowing connected components to be unaware of such change.

Req. #	Functionality	Description	Importance	Type	Layer of architecture	External Traceability	Internal Traceability
3.6.4.1	Loosely Coupling	The loosely coupled design of the architecture shall help keeping maintenance effort low and focused	Critical	FR	Distributed components	NeOn architecture partners	2.1.1.19
3.6.4.1.1	Loosely Coupling	The loosely coupled design of the architecture shall help keeping maintenance effort low and constrained to the local component	Critical	FR	Distributed components	NeOn architecture partners	2.1.1.19.2
3.6.4.1.2	Loosely Coupling	The loosely coupled design of the architecture shall help keeping maintenance effort low and constrained to the local resource	Critical	FR	Distributed components	NeOn architecture partners	2.1.1.19.1

Table 14: Maintainability requirements

3.2.6.4 User documentation

Complex systems like NeOn need to make a special effort and incorporate high quality documentation and help that allow users to fully grasp all the possibilities offered by the software. Besides, as shown in the user study conducted in [5] it is certainly frustrating for users to find particular problems or bogus system behaviours which the system itself does not explain. Thus, it is fundamental for NeOn to provide users with a complete and sound documentation, including reliable tutorials and examples.

It is also interesting for NeOn developers to rely on standard methods that help simplifying the process of building the software. These methods shall include functional specification and implementation design of each system component using standard design tools like e.g. UML.

Table 15 includes a full description of documentation requirements.

Req. #	Functionality	Requirement Extracted	Description	Importance	Type	Layer of architecture	External Traceability
3.7.1	Documentation	Documentation	Industry-strength documentation shall be written for NeOn components, toolkit, etc.	Critical	NFR	Neon toolkit	NeOn TA
3.7.1.1	Documentation	Functional specification	A functional specification of each component shall be written	Average	NFR	Neon toolkit	NeOn TA
3.7.1.2	Documentation	Implementation design	An implementation design of each component shall be written	Average	NFR	Neon toolkit	NeOn TA
3.7.1.3	Documentation	UML compliance	NeOn design shall be UML compliant	Average	NFR	Neon toolkit	NeOn TA
3.7.2	Support to the modelling process	Support to the process of modelling ontologies	Samples of ontologies and tutorials for the novice users shall be provided	Average	FR	Neon toolkit	NeOn TA
3.7.2.1	Support to the modelling process	Accessibility	The help features shall be accessible to the user.	Critical	NFR	Neon toolkit	NeOn R&V
3.7.2.2	Support to the modelling process	Reliability	The tutorials and the sample ontologies shall be reliable so that the user can trust them	Average	NFR	Distributed components	Case studies

Table 15: Requirements on user documentation

4. Preliminary risk analysis

The requirements specified herein cover an ambitious set of functional and non-functional needs. The resulting requirements are not purely requirements for the NeOn architecture but also process requirements and requirements for concrete components for populating the NeOn architecture. As described in the document, this specification is not final and, across the lifecycle of the NeOn project, requirements will evolve along.

Many of these requirements represent approaches which complement each other in order to provide complete solutions to existing problems in networked ontologies management and use. Thoroughly implementing all these approaches might result in overkill for the project. This is especially true as the architecture cannot contain all components but must be extensible to add such components by many NeOn members. Additionally at some point it shall be necessary to focus on concrete choices for particular requirements. For example, this might be the case of the rich vs. thin client dicotomy which repeatedly appears in this deliverable.

On the other hand, all this technology aims at both i) reducing the complexity of ontology engineering in a world where the amount of distributed, networked ontologies is larger and larger and ii) allowing its use in real life applications like the ones described in the case studies of NeOn. Hence, as the project evolves, the participation of both types of users, i.e. ontology engineers and domain experts, will be more and more important in order to keep NeOn on the right track. For this aim, methodological requirements, like e.g. the ones described in section 0, shall be enforced if we intend to produce industry-strength software which serves to this purpose.

Finally, it seems quite sensible that a number of core functionalities be specified in the upcoming architecture design that provides at the same time the fundaments of the NeOn infrastructure and establishes the guidelines that flexibly allow future extensions.

5. Conclusion

We have provided a description of the NeOn architecture requirements and the methodology that has been used, based on the IEEE proposal for software requirements specification, in order to gather them. Our aim has been to build the NeOn architecture requirements specification on top of both the expertise of system architecture core partners, technology and methodology providers, as well as the actual needs of the case studies. The description of the requirements contained herein analyzes current practices in ontology engineering and gathers information that can be used to identify strengths and weaknesses of current tools in order to develop a precise understanding of NeOn needs. These requirements shall feed the design of the NeOn architecture and overall infrastructure.

6. Annex I: Requirements sheet templates

Next, the templates used for NeOn architecture requirements gathering are described.

The following sheet shows the fields that describe the requirements listed in Section 0. These fields are *Req.#* specifying the number of the requirement, *Title* which specifies the name of the requirement, a short *description*, the *importance* of the requirement in the NeOn architecture (critical, average or low) and the *traceability* to the document where the requirement was extracted.

Req #	Title	Description	Importance	External traceability

The following sheet shows the main fields for specifying the specific requirements of Section 0. It contains *Req.#*, that specifies the requirement number, *Functionality* specifying the function that is associated to this requirement, a short *description* of the requirement, the *importance* of the requirement for the NeOn architecture (critical, average or low), the *layer* of the architecture that the requirement belongs to and the *traceability* of the requirement, internal (to the requirements related) and external (to the document that this requirement has been extracted).

Req. #	Functionality	Description	Importance	Type	Layer of architecture	External Traceability	Internal Traceability

References

[1] Sabou M.R, Gómez-Pérez J.M. et al. NeOn Requirements and Vision, 2006

[2] Lamport L, Shostak R, Pease M. The Byzantine generals' problem. ACM Transactions on Programming Languages and Systems, 4(3):382—401. ACM, July 1982

[3] IEEE-SA Standards Board, IEEE Recommended Practice for Software Requirements Specifications, 1998.

[4] Horrocks I, Parsia B, Schneider P, Hendler J. Semantic Web Architecture: Stack or Two Towers? Principles and Practice of Semantic Web Reasoning (PPSWR), 2005.

[5] Dzbor M, Gómez-Pérez J.M, Buil C. Analysis of user needs, requirements, and behaviours with respect to user interfaces for ontology engineering. NeOn Deliverable D4.1.1, 2006.

[6] Project cBio: Advancing biology and medicine with tools and methodologies for the structured organization of knowledge.

[7] Iglesias M, Caracciolo C. Specification of users and user requirements and detailed use cases. NeOn Deliverable D7.1.1, 2006.

[8] Gómez-Pérez J.M, Pariente T, Daviaud C, Herrero G. Analysis of the pharma domain and requirements. NeOn Deliverable D8.1.1, 2006.

[9] Kiryakov A., Popov B., Terziev I., Manov D. and Ognyanoff D. Semantic annotation, indexing, and retrieval. Journal of Web Semantics, 2, Issue 1, 2005.

[10] Hearst M. A., Untangling Text Data Mining. Proceedings of ACL'99: the 37th Annual Meeting of the Association for Computational Linguistics, 1999

[11] Davies J., Studer R., Warren P. (Eds), Semantic Web Technologies: Trends and Research in Ontology-based Systems. John Wiley & Sons. June 2006.