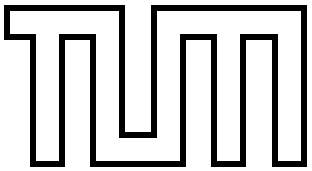


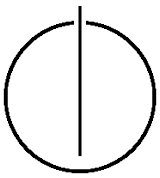
FAKULTÄT FÜR INFORMATIK

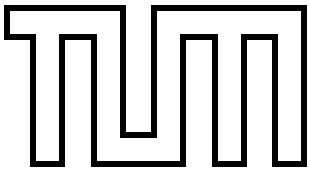
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diploma Thesis in Computer Science

An Ontology for Aircraft Design

Markus Ast





FAKULTÄT FÜR INFORMATIK

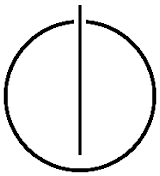
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diploma Thesis in Computer Science

An Ontology for Aircraft Design

Eine Ontologie für die Flugzeug-Entwicklung

Author:	Markus Ast
Supervisor:	Prof. Bernd Brügge, Ph.D.
Advisor (TUM):	Dipl.Inf. Tobias Röhm
Advisor (BHL):	Dipl.Ing. Martin Glas
Submission Date:	April 12, 2012



I assure the single handed composition of this diploma thesis only supported by declared resources.

Munich, April 12, 2012 Markus Ast

Acknowledgements

First of all, I would like to thank my two advisors, Tobias Röhm and Martin Glas, for their huge support, their constructive and helpful feedback, and their understanding regarding some unfortunate issues. A special thanks to Martin for keeping my spirit up in some difficult situations. Further, I would like to thank Arne Seitz and Patrick Vratny for their involvement in the evaluation. Also, I would like to thank Bauhaus Luftfahrt as a whole for providing me with a workplace, a wonderful Christmas event, and a nice working atmosphere.

On a special note I would like to thank my parents without whose loving support and patience, especially during difficult times, conducting and finishing this thesis would not have been possible. I dedicate this thesis to my late father.

This work was conducted using the Protégé resource, which is supported by grant LM007885 from the United States National Library of Medicine.

Abstract

Ontologies have become a viable alternative to classic data models as a means to capture and represent knowledge. They are even capable of capturing knowledge in a way that provides possibilities to infer more knowledge than is explicitly modeled and connect data without the need for explicit programming, as is the case in classic data representations. However, modelling knowledge in an ontology also requires more effort than creating a database model. Thus, the development of an ontology shares a resemblance with the development of a software project. In the field of software engineering, development methodologies are widespread and highly sophisticated. The field of ontology development, however, is relatively young by comparison, and sound methodologies have begun to emerge only in the last decade. In this diploma thesis, existing ontology development methodologies are summarized and presented. Furthermore, one methodology is selected and applied to an ontology development project in the domain of aircraft design, which is intended to be used as part of a solution for the problematic data exchange between heterogeneous aircraft models with different focuses. In this project an aircraft design ontology covering aircraft structure, aircraft aspects and aircraft parameters has been developed. The application of the methodology is presented in detail and discussed afterwards, also the resulting ontology is evaluated and described in detail.

Ontologien sind zu einer brauchbaren Alternative von klassischen Datenmodellen geworden um Wissen zu erfassen und zu repräsentieren. Sie können sogar Wissen derarat erfassen daß dies das Ableiten weiteren Wissens, als das was explizit modelliert ist, ermöglicht, und Daten ohne explizierte Programmierung miteinander verknüpfen, wie das sonst bei klassischen Datenrepräsentationen der Fall ist. Allerdings ist das Modellieren von Wissen in einer Ontologie aufwendiger als ein klassisches Datenbankmodell zu erstellen. Die Entwicklung einer Ontologie ähnelt mehr der Entwicklung eines Software-Projektes. Im Bereich der Software-Entwicklung sind Entwicklungsmethoden ausgereift und weit verbreitet. Das Themenfeld der Ontologienetwicklung ist jedoch relativ jung im Vergleich dazu, und ordentliche Entwicklungsmethoden sind erst im letzten Jahrzehnt aufgekommen. In dieser Diplomarbeit werden existierende Ontologie-Entwicklungsmethoden zusammengefaßt und präsentiert. Desweiteren wird eine Methode ausgewählt und in einem Ontologieentwicklungsprojekt in der Domäne des Flugzeug-Entwurfs angewandt, welches als Lösungsbaustein für das Problem des Datenaustausches zwischen heterogenen Flugzeug-Datenmodellen mit verschiedener Ausrichtung dienen soll. In diesem Projekt wird eine Ontologie entwickelt die Flugzeugstruktur, -aspekte und -parameter umfaßt. Die Anwendung der Methode wird detailliert vorgestellt und diskutiert, ebenso wird die entstandene Ontologie evaluiert und detailliert beschrieben.

Contents

List of Figures

List of Tables

1. Introduction

The design of aircrafts is a complex process that involves various specialists for the huge range of details and different aspects that have to be regarded. For each special discipline in this domain, there exist a variety of software tools, off the shelf or self-made by the engineers, to support the development in the current tasks at hand. Often, data has to be exchanged between those tools, and currently, this is often a mostly manual and needlessly time consuming task; data from one program has to be mapped accordingly for another program, which can become difficult in programs (especially self made ones using environments like mathematica) where often just some arbitrary floats and arrays get passed around. Also, this kind of mapping has to be done between all programs that exchange data between one another, in the worst case in both directions, which poses an n:m relationship regarding the data exchanges.

The basic idea behind this thesis was to use a central data model that all exchange data from tools is mapped to. This way, there would only have to be mappings between each tool and the central data model, changing the mapping relationship from n:m to n:1 and thus greatly reducing the effort. Furthermore, this central data model could also be used to perform plausibility checks on the given data to assert that the mappings are also correct, for example regarding dimensions, units, or certain integrity constraints. Instead of using a classic data model, the incentive was to see whether an ontology can fulfill this task. Ontologies usually at least form a thesaurus (or to be exact, a class hierarchy) for the given domain; furthermore, with the aid of a so-called reasoner, ontologies are also able to deduce knowledge that is only implicitly modeled. However, building an ontology is no trivial task. In order to get a sound result, following a sophisticated development methodology is necessary. But in order to select a methodology for this project, one has to know what kind of ontology development methodologies exist.

Thus, the first major part of this thesis is to assemble an overview of available development methodologies, present and compare them. With these insights, it is then possible to select a methodology to be used for the ontology development. This is the second major part of this thesis, constructing the aircraft design ontology by following the selected ontology development methodology. The execution of the methodology will conclude with an evaluation of the built ontology, a discussion about this execution and potential deviations or special situations or findings, a detailed description of the built ontology, and lastly a final conclusion of what might be interesting follow ups regarding further developments.

As for the development of the ontology, example data from three software tools that are in use for aircraft design at Bauhaus Luftfahrt, a Munich aerospace research institute that is the cooperating partner in this thesis, is provided as a basis and reference. The ontology concentrates mainly on one aspect of an aircraft, which is the structure and static build-up.

This is only one very special aspect, but is regarded as sufficient for a first ontology, as this project is also intended to gain some experience in modelling and handling ontologies, and to gain deeper insight into the capabilities and also limits of ontologies. Regarding the content, the ontology should at least be capable of representing a standard fixed wing passenger transport aircraft, to be concrete an Airbus A320, on a rough abstraction level for the early stages of the aircraft design process.

The thesis itself is structured as follows:
Chapter 2 provides the basic foundation and background for the thesis and the ontology development project. First, subchapter 2.1 explains what an ontology is exactly, how they can be represented and how they work underneath. Afterwards, subchapter 2.2 gives an overview and a short presentation of available ontology development methodologies. Finally, subchapter 2.3 is a short introduction into the domain of aircraft design.
Chapter 3 follows and demonstrates the execution of the selected development methodology for building the aircraft design ontology.
Chapter 4 presents the details of the built ontology, how certain aspects were modeled, and how the single parts of the ontology fit together.
Chapter 5 finally provides a conclusion for this thesis, and also depicts what could be interesting aspects for potential further developments regarding ontologies in general, or regarding this aircraft design ontology in specific.

2. Theoretical Background

This chapter provides an introduction to some basic topics for this thesis. First of all, chapter 2.1 gives an overview of what ontologies are, how they work, and how they can be applied. After having established a basic notion for ontologies, chapter 2.2 presents existing methodologies for developing such an ontology. Finally, chapter 2.3 is a very short introduction into the domain of aircraft design, which is the domain in which the ontology development for this thesis takes place.

2.1. Ontology Basics

The following sub-chapters give an introduction to the concept and workings of ontologies. At first, the notion of ontologies is explained, followed by an introduction to description logics, which are a basis for various ontology languages. Some important languages are presented thereafter, with a special focus on the Ontology Web Language OWL, which will be used in this thesis. Finally, a short overview of selected ontology development tools is given. For more in-depth information on these topics and on ontologies in general, please refer to [2], [30], [28] and [11].

2.1.1. About Ontologies

The word "ontology" originates from classical Greek, where it depicted the philosophical study of being and existence, which deals with the question of what kinds of things exist and how they relate to one another based on similarities or differences.
In computer science the concept of an ontology has been adapted through the domain of artificial intelligence. Tom Gruber has defined it as a *"specification of a conceptualization"* [15], which has become widely accepted (and further adapted). It describes concepts and relationships that can be interpreted and used by an agent, that means it is a knowledge source that includes not only data itself but also semantics between that data, and is processable by a machine, i.e. a computer (program). An ontology is used to establish and especially share knowledge, and because of this it is necessary that this conceptualized knowledge is agreed upon by all participants in the development. Another definition for an ontology, that further underlines these aspects, is made in [29] where it is described as *"a formal explicit specification of a shared conceptualization of a domain of interest"*. This definition emphasizes the formality aspect (which is important for machine processability), the consensus, and also that an ontology always captures knowledge of a specific domain (with a certain interest in mind, that also has an influence on the view of the domain that the ontology captures).

There are some general aspects that are characterizing for all kinds of ontologies. The concepts in an ontology essentially form a thesaurus. These concepts are typically orga-

nized in a hierarchical structure, where some concepts are subsumed by others through a *"is a"* relation. The concepts themselves can be interrelated to one another by specifying relations, typically through some rules or logical statements; by combining those measures, it is also possible to formulate more complex axioms. Furthermore, concepts can also be attributed, that is, connected to data values. For the application of an ontology, analogous to object oriented programming, the concepts are instantiated, and these instances are then called "individuals".

Ontologies can be categorized in various ways. Figure 2.1 gives an example for such a categorization. Top level ontologies, also called upper ontologies, are ontologies that deal

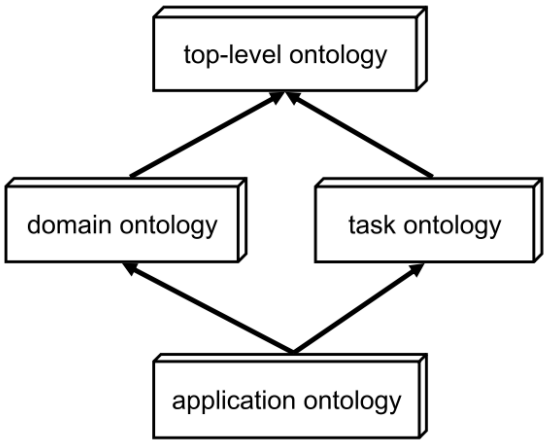


Figure 2.1.: Types of ontologies [30]

with very general and abstract terms, like notions of time, space, things, etc., at a high abstraction level. They are intended to formulate general concepts that are agreed on by a broad audience, and that can be reused and further refined and detailed in more specific ontologies. Examples for such upper ontologies are DOLCE[1] (Descriptive Ontology for Linguistic and Cognitive Engineering) and SUMO[2] (Suggested Upper Merged Ontology).
Domain- and task-ontologies are more specialized ontologies, that can (re)use upper ontologies for general concepts which they refine and specify in some way. A domain ontology usually captures knowledge of a certain domain (e.g. the domain of aircraft design in this thesis), whereas task ontologies capture knowledge for more or less general tasks (for example, cooking in general). The distinction on this level, however, is not always totally concise; typically, it is spoken of domain ontologies.
As a further specialization, there are application ontologies. These ontologies can (re)use knowledge from more general domain ontologies that capture knowledge from those domain(s) that a concrete application deals with. They are developed in conjunction with an application that is made for certain usage scenarios. Application ontologies are thus typically very specific for the tasks at hand, derived from established use cases, and thus more or less application dependent.

[1] http://www.loa.istc.cnr.it/DOLCE.html
[2] http://www.ontologyportal.org/

Some interesting characteristics of certain ontologies, depending on the underlying language and implementation, and true for today's modern ontologies that are based on the W3C standard OWL (Web Ontology Language; more on this topic later on in chapter 2.1.3), are the **open world assumption**, and the **non-unique names assumption**. The open world assumption is the opposite of the closed world assumption, which is common for classic data models. The meaning of the closed world assumption is that the 'world', that is modeled by some stated knowledge, is closed regarding this knowledge, which means that any questions regarding knowledge that is not stated can only result in a negative answer; everything beyond this delimited 'world' is simply not existent. The open world assumption, however, does not delimit the 'world' in such a way. Knowledge that is modeled is also a benefit for a closed world, but anything that is not modeled is not assumed to be non-existent, it is just regarded as unknown. As long as there are no statements in the knowledge that prohibit a certain fact, this fact may yet become true some time in the future, the according knowledge may simply not be available at the moment. Here follows a simple example. Imagine a database about "students" that has an entry "John", as well as an ontology about students, also with an entry about the same "John". Asking the database as well as the ontology whether John is a student produces a positive result. Asking the database whether "Joe" is a student (while there is no entry about him) results in a negative answer, whereas the same question directed at the ontology (which also has no entry about Joe) results in a "don't know". The only way to get a negative answer for this question is when the modeled knowledge in the ontology, directly or indirectly, states that Joe is not a student. The ramifications of this open world assumption may not yet be totally clear for people who are used to think in a closed world assumption (as is probably the case for the typical computer science student or user), but later on in this thesis (see, for example, chapter 4.4) they will become apparent.
The above mentioned non-unique naming assumption states that one concept or individual in an ontology may have more than just one name. The names "John" and "Joe" may refer to the same individual (imagine, for example, someone who uses web 2.0 services only with different pseudonyms for each web service), and as long as there is no statement that John and Joe are distinct individuals, they may still be the same.

Before presenting information about ontology languages, first an introduction to description logics is given, which is a foundation for many ontology languages, for example OWL (which is the language for the ontology development in this thesis).

2.1.2. Description Logics

Many ontology languages are based on a logical background. Besides classic first order predicate logic or a less expressive frame logic, description logics have been established as a common basis for most newer languages. They provide a rich expressiveness which may not go as far as first order predicate logic, but this tradeoff is compensated by being able to provide more efficient solutions for decision problems. Depending on the restrictions, it is also possible to guarantee decidability.

A profound and thorough book about description logics is provided by "The Description

Logic Handbook" [4]. The term "description logics" describes a family of languages for knowledge representation via logical axioms. The expressiveness of a description logic is determined by the kind of logical axioms (operators) it allows, providing for a range of languages.

Description logics have been driven by three basic ideas:

- atomic concepts, atomic roles and individuals are the basic building blocks
- expressive power is restricted by using a small set of constructors
- it is possible to infer implicitly given knowledge about concepts and individuals with the help of inference procedures ($\rightarrow$ reasoners)

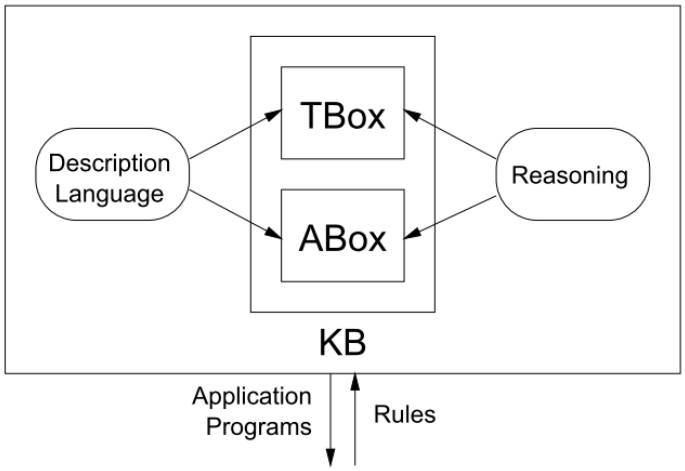


Figure 2.2.: Architecture of a description logics knowledge base [4]

The basic architecture that description logics knowledge bases follow is shown in Figure 2.2. The knowledge base entails two constructs, the ABox and the TBox. While the TBox contains the information about the terminology for the ontology (that is, the general concepts and relations), the ABox contains information about the assertions for the individuals that are made with the terms from the TBox. The contents of the ABox and TBox are described by the appropriate description language. This forms the explicitly stated knowledge in the knowledge base. Furthermore, a reasoner, which has access to both boxes, can possibly infer additional knowledge that was only implicitly given, and add this to the knowledge base for access by applications that interact with that knowledge base.

As said above, the expressiveness of description logics is defined via a set of constructors. For the following overview, the letters A and B stand for atomic concepts, R for atomic roles, and C and D for general concept descriptions. Description logic languages are built by adding additional constructors to a minimal language that is useful. This minimal description logic is called $\mathcal{AL}$, and possesses the following constructors [4] (semantics correspond to what one would intuitively expect):

C, D	$\rightarrow$	$A \mid$	(atomic concept)
		$\top \mid$	(universal concept)
		$\bot \mid$	(bottom concept)
		$\neg A \mid$	(atomic negation)
		$C \sqcap D \mid$	(intersection)
		$\forall R.C \mid$	(value restriction)
		$\exists R.\top$	(limited existential quantification)

This minimal description language can be extended to enhance its expressiveness by adding further constructors to it (which are represented by single letters), like the union of concepts, full existential quantification, the negation of concepts, etc.
By adding so-called "terminological axioms", statements about how roles and concepts relate to one another can be made. These have the form

$$C \sqsubseteq D \quad (R \sqsubseteq S) \quad \text{or} \quad C \equiv D \quad (R \equiv S)$$

with C and D being (general) concepts, R and S being (general) roles, $\sqsubseteq$ denoting inclusion, and $\equiv$ denoting equality. Further additions to make full use of ABox and TBox functionality are 'concept definition' (defining two concepts C and D to be equivalent), 'concept assertion' (asserting an individual 'a' to a concept C) and 'role assertion' (asserting two individuals to a role R). More extensions can be made by allowing (statements about) transitivity, symmetry, reflexivity and functionality for roles. Putting all those aspects together, diverse variations of description logics are possible. Table 2.1 gives an overview.

The describing letters from Table 2.1 are combined to express a certain description logic. For example, Protégé (see chapter 2.1.4) supports $\mathcal{SHOIN}^{\mathcal{D}}$, OWL 2 as a whole (see chapter 2.1.3) is based on $\mathcal{SROIQ}^{\mathcal{D}}$.

2.1.3. Ontology Languages

Regarding possibilities to describe ontologies, there is a variety of languages available. There are a number of traditional ontology languages that were used especially before a clear notion of the semantic web was established and the semantic web stack of W3C standards began to develop. Examples for these languages are CycL, which was (and is still) used for representing knowledge in the Cyc Knowledge Base, F-Logic (a frame logic language), KIF or LOOM. However, after the propagation of XML[3] with XML Schema[4] (which provide a syntactic basis for the semantic web stack) and the introduction of RDF[5] and RDFS[6], the development of ontology languages has taken a more focused direction. RDF and RDFS have been established as a standard for describing knowledge resources, and build the foundation even for the latest standards. One of the first modern ontology languages that used RDF and RDFS as a foundation was the Ontology Inference Layer (OIL). Another

[3] http://www.w3.org/XML/
[4] http://www.w3.org/XML/Schema
[5] http://www.w3.org/TR/2002/WD-rdf-concepts-20021108/
[6] http://www.w3.org/TR/2002/WD-rdf-schema-20021112/

$\mathcal{AL}$	attributive language (allowing atomic negation, concept intersection, (universal) value restrictions and limited existential quantification)
$\mathcal{FL}$	frame based description language (allowing concept intersection, (universal) value restrictions, limited existential quantification and role restriction)
$\mathcal{EL}$	allowing concept intersection and existential restrictions (of full existential quantification)
$\mathcal{C}$	complex concept negation
$\mathcal{S}$	abbreviation for $\mathcal{ALC}$ with transitive roles
$\mathcal{H}$	role hierarchy
$\mathcal{R}$	limited complex role inclusion axioms; reflexivity and irreflexivity; role disjointness
$\mathcal{O}$	nominals (enumerated classes or object value restrictions)
$\mathcal{I}$	inverse roles
$\mathcal{N}$	cardinality restrictions
$\mathcal{Q}$	qualified cardinality restrictions
$\mathcal{F}$	functional roles
$\mathcal{E}$	full existential quantification
$\mathcal{U}$	concept union
$^{(\mathcal{D})}$	use of data type attribution in roles, data values or data types

Table 2.1.: Description logics overview

one of those early modern languages that referred to RDF was the DARPA Agent Markup Language (DAML). The efforts and results of the projects that produced these two ontology languages were later on combined, which led to an extended language DAML+OIL. The draft for this language was submitted to the W3C in 2002. It provided the starting point for the development of the Web Ontology Language (OWL[7]), which was released as a W3C recommendation in a first version in February 2004. More than five years later, in November 2009, a revision of OWL was released as OWL 2[8]. Figure 2.3 shows the dependencies of the mentioned parts of the semantic web stack.

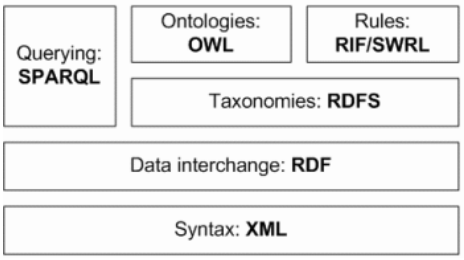


Figure 2.3.: Excerpt from the W3C semantic web stack

As mentioned in the previous sub-chapter, the foundation for the semantics and capabilities of OWL are description logics. In OWL, concepts are called classes, roles are called

[7] http://www.w3.org/TR/owl-features/
[8] http://www.w3.org/TR/owl2-overview/

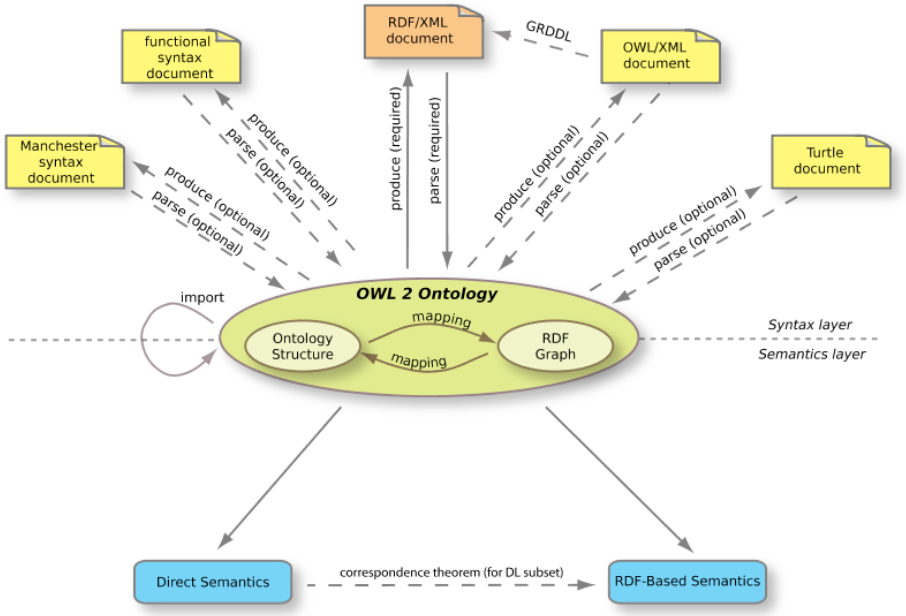


Figure 2.4.: Structure of OWL 2[9]

properties and are available as object properties (properties that connect two classes or their appropriate individuals in a relation) or data type properties (that connect classes with data types or individuals with data values), and the members (or instances) of classes are called individuals.
The first version of OWL is available in three dialects with different expressiveness. It is based on the $\mathcal{SH}$-family of description logics. The least expressive dialect is OWL-Lite (based on $\mathcal{SHIF}^{(\mathcal{D})}$). It provides basic features for creating a classification hierarchy and simple constraints. OWL-DL is an extension of OWL-Lite (based on $\mathcal{SHOIN}^{(\mathcal{D})}$). It lifts constraint restrictions, and allows more complex constructs like union, intersections or disjoints. OWL-DL is the most expressive dialect that still retains computational completeness, and provides a sufficient applicability for most scenarios. OWL-Full is a further extension of OWL-DL. It lifts the last restriction of type separation (a class can not also be an individual or property, a property can not also be an individual or class). This enables greatest expressiveness and maximum flexibility, but at the cost of sacrificing computational guarantees; it becomes possible that inference becomes undecidable (i.e. is stuck in endless computation).

This first version of OWL, however, had some shortcomings, which were rectified in its second version. Hence OWL 2 is based on another description logic: $\mathcal{SROIQ}^{(\mathcal{D})}$. Figure 2.4 gives an overview of the structure of OWL 2. OWL 2 enables the use of several different syntaxes. The one mandatory syntax that all OWL 2 compatible tools must support is the RDF/XML syntax, which is the basic interchange syntax. The support of other syntaxes is optional, but they may provide some advantages in certain cases: OWL/XML for easier processing with XML tools, the Functional Syntax for better readability (to see the formal structure), Manchester Syntax for easier reading/writing of DL ontologies, and the Turtle

Syntax for easier reading/writing of RDF triples.
The semantics of OWL 2 ontologies are provided by either the 'OWL 2 RDF-Based Semantics' or the 'OWL 2 Direct Semantics'. Ontologies that satisfy the conditions of the first one are called OWL 2 Full, whereas ontologies that satisfy the conditions of the latter are called OWL 2 DL ontologies.
For further flexibility, OWL 2 offers three different profiles that provide sub-languages which are restricted in certain ways, to provide special usefulness. These three dialects do not form any inclusion relation between one another. Those three dialects are:

- OWL 2 EL, intended for large, class-expression oriented ontologies
- OWL 2 QL, intended for use in conjunction with standard relational database technology
- OWL 2 RL, intended for applications that require scalable reasoning

For more details on OWL 2 and some example ontologies that explain the used constructs in more detail, please refer to the OWL 2 Web Ontology Language Primer[10].

2.1.4. Ontology Development Tools

For the development (or support during the development) of ontologies, diverse software is available, be it open source, free of charge, or commercial. In this subchapter, a short overview of a few selected (current and still under continuing development) programs shall be given.
Probably one of the most well known and oldest programs for developing ontologies is **Protégé**[11], an ontology editor developed by the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine, that has a history of far over ten years. It deviates a little from the standard look and feel of many other editors or programs, but compensates this with a rich functionality and a lot of comfort features (like easily creating class hierarchies, declaring defined classes or refactoring support). It is extendable via plug-ins for which an established community provides support, supports the current OWL 2 standard, and is freely available. Furthermore, there are several reasoners available as plug-ins that directly integrate into Protégé, for example Pellet[12], FaCT++[13], HermiT[14] or Racer[15]. Another editor was made available in the course of the NeOn project[16], the **NeOn Toolkit**[17]. It is based on the Eclipse framework[18], supports the current OWL version, and is also extendable via plugins. Since it was developed during the course of the NeOn project, it also provides some special support for the NeOn methodology for ontology development (see chapter 2.2.7 and 3). It is also freely available.
A third popular ontology editor is the **TopBraid Composer**[19], developed by TopQuad-

[10] http://www.w3.org/TR/2009/REC-owl2-primer-20091027/
[11] http://protege.stanford.edu/
[12] http://clarkparsia.com/pellet/
[13] http://owl.man.ac.uk/factplusplus/
[14] http://hermit-reasoner.com/
[15] http://www.racer-systems.com/
[16] http://www.neon-project.org/
[17] http://neon-toolkit.org/
[18] http://www.eclipse.org/
[19] http://www.topquadrant.com/products/TB_Composer.html

rant, which is available as a commercial program, but also in a basic version free of charge. Besides supporting OWL and RDF(S), one of its special features is its extended functionality to make use of SPARQL[20], a query language for RDF resources.
Finally, a fully commercial program shall be mentioned. **OntoStudio**[21] is a full grown development environment that allows for graphical development, extendability via plugins, provides full support for current standards and enables collaborative development.

Besides these few mentioned programs, there exists a wide variety of other editors; however, most do not provide support for the current OWL 2 standard. The same often also holds true for specialized tools that provide functionality selected tasks like e.g. machine learning or ontology mapping. For a summary of a lot of diverse tools (also older and discontinued ones), please refer to The Sweet Compendium of Ontology Building Tools[22]

2.2. Ontology Development Methodologies

This section introduces ontology development methodologies, beginning with the first approaches of a systematic and methodological course of action in the mid 1990s, over to the well known methodologies in scientific literature that were published until now, in a roughly chronological order. Finally, the presented methodologies are compared to one another in a short overview.

2.2.1. Early Approaches

When ontologies became recognized as an instrument for knowledge modelling and sharing in the early 1990s, there were not yet any methodologies and development guidelines available. Ontologies were built from scratch, and the experiences made during these early developments led to the first identifications and propositions of development concepts. Uschold and King [43] provide an early, noteworthy, proposition for an ontology development methodology. They describe four consecutive stages that the development should follow:

1. Identification of the purpose
2. Building the ontology (which comprises capturing the underlying concepts of the ontology, coding these agreed upon concepts, and possibly integrating already existing ontologies)
3. Evaluation
4. Documentation

For each stage they offer some rough descriptions, rather of what to do and consider, as to how to do it. Nevertheless, it is a first basic approach for a development methodology. Another noteworthy publication was made by Grünninger and Fox [16]. They introduced

[20] http://www.w3.org/TR/rdf-sparql-query/
[21] http://www.ontoprise.de/de/produkte/ontostudio/
[22] http://www.mkbergman.com/862/the-sweet-compendium-of-ontology-building-tools/

the notion of so called "competency questions", which have become widely accepted and integrated into many of the later ontology development methodologies that are described later on. These competency questions are a special way to describe the requirements for an ontology. In a basic sense, they are informal natural language question directed at the target ontology, which this ontology shall be able to answer. They don't propose anything about the internal structure of the ontology, but of its expressiveness, and as such are not only a useful tool for formulating content-related requirements, but also for evaluating the built ontology; one could also say that they can serve as simple test cases or unit tests for the development.
Other projects which produced some guidelines and suggestions for ontology development, which shall be mentioned here, are the development of the Cyc knowledge base [22], the KACTUS project [3], IDEF5 [5] and Sensus [39]. An interesting summary of development guidelines an of what to pay attention to during ontology development is provided by the Ontology Development 101 [24]. It is not a (formal) ontology development methodology per se, but rather an informal guide on how to develop an ontology, directed at inexperienced ontology developers.

In the following sections, the more sophisticated development methodologies that were published from the late 1990s onward will be presented.

2.2.2. METHONTOLOGY

METHONTOLOGY [14, 23] is a methodology that is based on the experiences of building an ontology in the domain of chemistry. It is an approach to further guide the development of ontologies from arts to a comprehensible and reproducible engineering discipline and uses aspects of the first methodological approaches mentioned in chapter 2.2.1.
METHONTOLOGY identifies several activities that have to be carried out during the execution of the methodology: planify [sic], specify, acquire knowledge, conceptualize, formalize, integrate, implement, evaluate, document and maintain. These activities pertain to certain states in the ontology life cycle, which is proposed to be an evolving prototype (which means from any state within the life cycle, the developer can move back to an earlier state to make modifications, if the need arises). Figure 2.5 gives a quick overview.

At the beginning of a project, all the activities have to be planned regarding estimated time and needed resources. The basic order of the states that the development process follows after that is "specification, conceptualization, implementation", and maintenance when the development is completed. Each state requires a certain activity, that is usually accompanied by one or more support activities that are relevant at all states of the process. During the specification, one has to describe the details about the purpose of the ontology, i.e. its uses, users, application scenarios, etc., as well as its scope and the level of formality. Simultaneously starts the knowledge acquisition, which suggests to use a range of techniques to acquire the required knowledge, i.e. brainstorming, expert interviews, formal or informal text analysis, etc., as applicable. Also simultaneously executed is the evaluation, which ensures for each phase that the right things (validation) were done right (verification), as well as the documentation, which requires the production of a document that describes the results of that phase (that can later be used to retrace the development). The next step is the conceptualization, which suggests building a complete glossary of terms from the

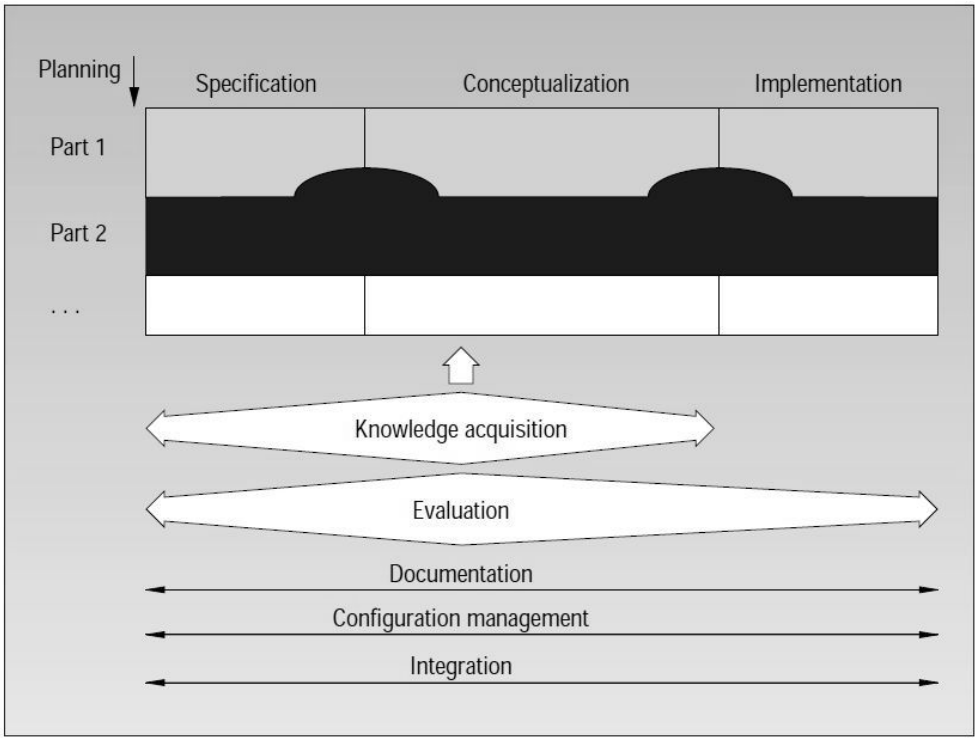


Figure 2.5.: Overview of METHONTOLOGY life cycle [23]

acquired knowledge, and to group these terms into concepts and verbs and further refine these groups. The integration activity is used to find appropriate (upper-)ontologies that already model the identified artifacts and can possibly be used for (partial) reuse. These inputs now allow for a further formalization and thus the implementation of the ontology in the target language, which concludes the ontology development.

The contribution METHONTOLOGY has made to further the engineering aspect of ontology development, compared to the early approaches, lies in the identification of the set of development activities, the proposition of the ontology life cycle, and the requirement to evaluate and especially document the outcomes of each development phase.

2.2.3. On-To-Knowledge (OTK)

On-To-Knowledge ([31], [28]) is a methodology that was developed within the On-To-Knowledge[23] project to establish a knowledge meta process, i.e. identify and structure knowledge (or in other words: develop and deploy an ontology) within the context of knowledge management (that means the handling and application of knowledge) within a corporation or between cooperating corporations.

Figure 2.6 provides a graphical overview over the OTK process, which is in fact called the knowledge meta process, i.e. the process that leads to the installation of a knowledge base, in contrast to the knowledge process which describes how the knowledge base is filled

[23] http://www.aifb.kit.edu/web/On-To-Knowledge

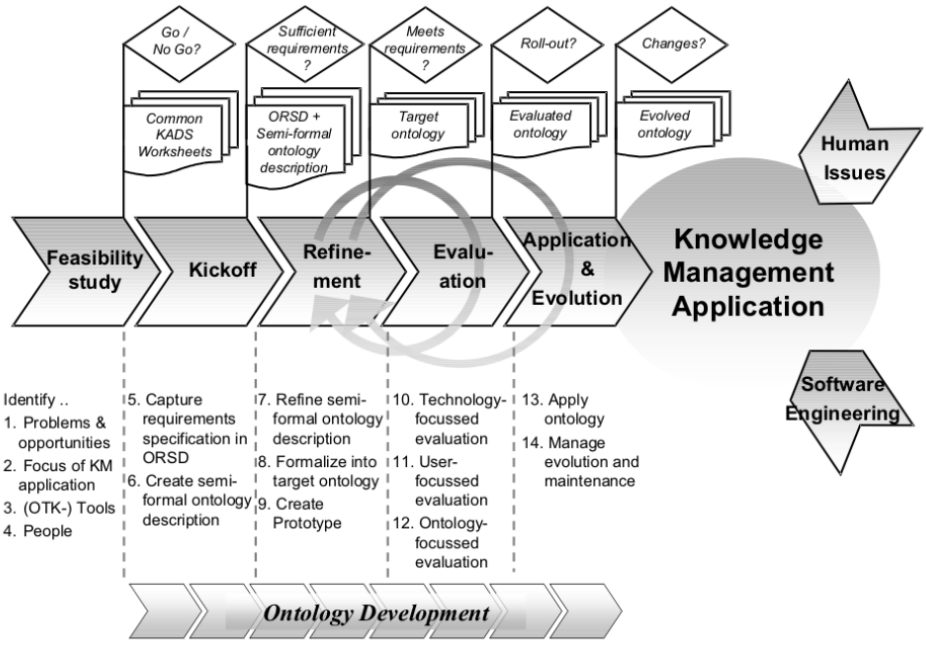


Figure 2.6.: OTK Knowledge Meta Process [31]

with data and used in practice.
OTK differentiates between five major steps (or phases) in the development process, shown as the big arrows in a line in the center of the figure. Each step itself is again divided into sub-steps, which are depicted under the appropriate arrow. At the end of each phase, a major decision has to be taken in order to advance to the next phase, which is based on the products of that phase; the figure shows these questions and phase products as "flags" at the end of an arrow. One can also see in the Figure that the later phases may require or lead to iterative cycles within the ontology development.

The **Feasibility Study** is used to identify potential problems with possible solutions, as well as opportunities for the project, on the economic and technical level, and select the most appropriate focus for the project, as well as a target solution. This target solution is again used to gain insight into the inner- and interdepencencies, regarding business tasks, involved actors, performance, organizational aspects and acceptance and the possible needs for changes in these fields. It culminates in a "go / no go" decision for the project.
A positive "go" decision initiates the project **Kickoff**. This phase leads to the creation of a requirements specification document, which contains information about the goal, domain and scope of the (ontology) project, design guidelines for the ontology, the knowledge sources that will be used for the development, information about the users and usage scenarios, a draft of the knowledge management application that will make use of the ontology, and finally competency questions (known from [16], see chapter 2.2.1) to describe the usage scenarios in more detail. Based on this specification, a semi formal ontology description is created.
A sufficient capturing of the requirements leads to the **Refinement** phase. Here, the on-

tology description is refined (using a top-down, bottom-up, or middle-out approach) and formalized into the target ontology. Also important is the creation of a prototype of the knowledge application, especially to further the **Evaluation**, which is the subsequent phase. The evaluation is performed regarding three viewpoints. The technology-focussed evaluation comprises technical aspects like language conformity, scalability, performance, etc. The user-focussed evaluation checks the ontology (application) against the requirements specification document and especially the competency questions, and also takes user feedback from the prototype application into account. Finally, the ontology-focussed evaluation deals with verification (is the system correctly built) and validation (is it the correct system), possibly using formal methods. Depending on the outcome of the evaluation, a re-iteration of the refinement may be required (resulting in possibly several iterative cycles), or the ontology and system may be deployed.
This leads to the **Application & Evolution** phase, which deals with applying the ontology and knowledge application in the organization, maintenance, and the evolution of the system, which will eventually produce another version of the ontology (and knowledge application) by re-performing the refinement and following steps. This is, all in all, an iterative process.

2.2.4. Methodology for DIstributed, Loosely-controlled and evlolvInG Engineering of oNTologies(DILIGENT)

The methodologies so far have only dealt with a centralized development, which is purely done by experts which have full autonomy over the built ontology. DILIGENT ([40], [25], [28]) tries to address these issues and describes a development methodology for a decentralized environment, where the participants have only partial autonomy over the ontology (development) and non-expert ontology builders are involved. DILIGENT differentiates four roles of participants:

1. domain experts (expertise in the domain)
2. ontology engineers (expertise in building ontologies)
3. knowledge engineers (expertise in using ontologies for building information systems)
4. users (use the ontology for their own purposes)

Participants may have more than one role, and a subset of the participants forms the so-called "board", which is responsible for the shared ontology. Figure 2.7 and 2.8 give a graphical overview of the development process (the representation of Figure 2.8 is analogous to Figure 2.6 in chapter 2.2.3), which is explained consecutively. The development is comprised of five phases. The first phase is the **Build (1)**, where the board builds an initial core ontology, using established development methodologies like e.g. OTK. This initial ontology is then released and will be used and adapted by the users for their own purposes. The basic idea in this phase is to produce a small and useful ontology to get the development started, which can be released as early as possible.
Following the release is the **Local Adaption (2)** phase, where users use and adapt the ontology for their own purposes. For this they first have to understand the shared ontology, identify similarities between their own and the shared conceptualization and map these,

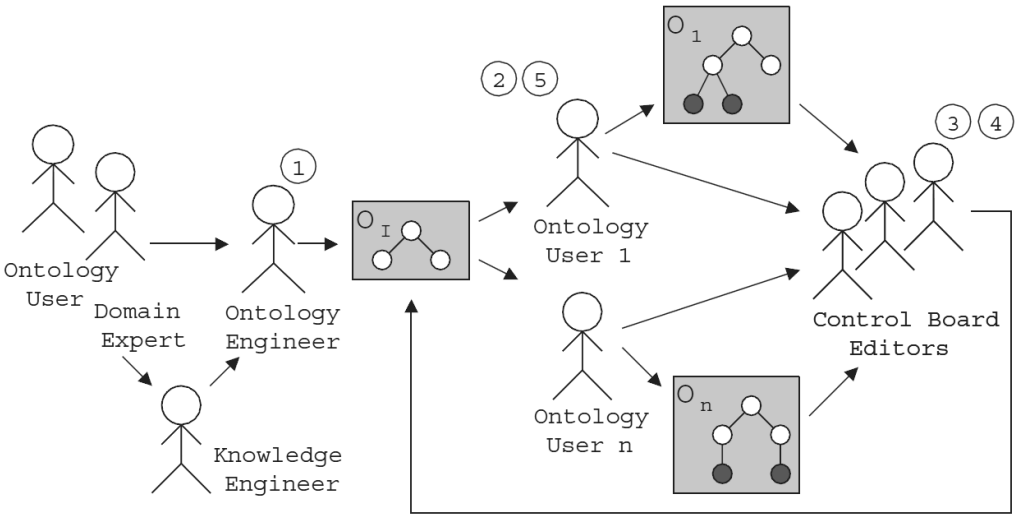


Figure 2.7.: DILIGENT distributed development overview [40]

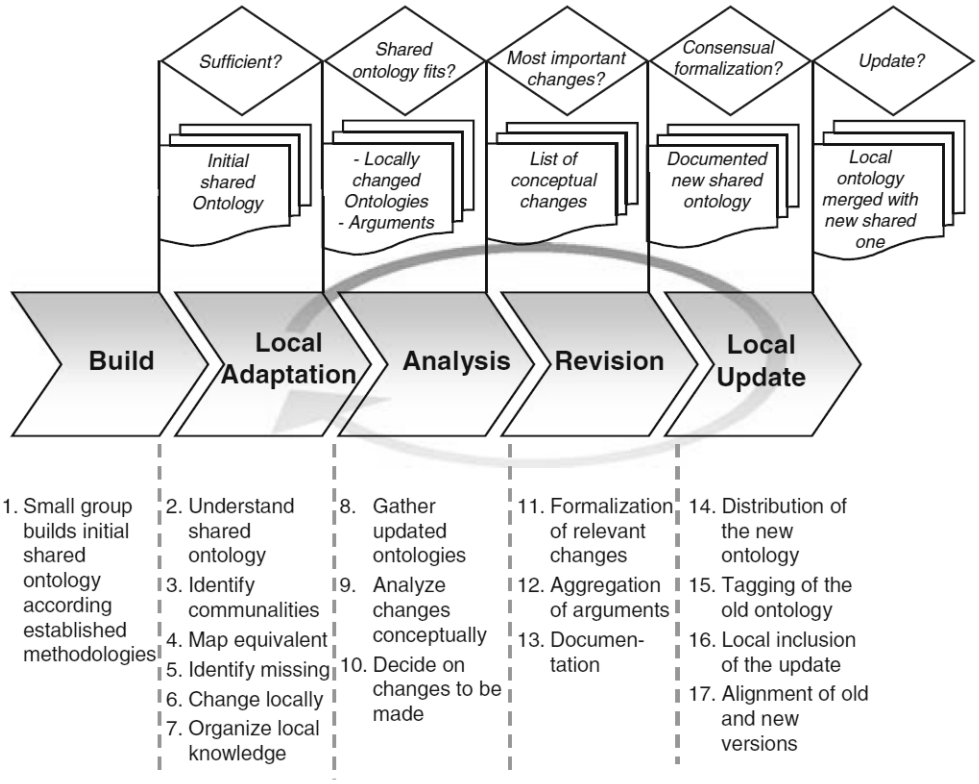


Figure 2.8.: DILIGENT process overview [40]

and identify missing concepts in the shared ontology. Then they can change the conceptualization of the ontology to introduce these missing concepts in their local copy, or make change requests to the board. These changes and requests must be reasonably explained and documented. Finally, the user can organize his local knowledge to utilize and reflect this new conceptualization.
In the **Analysis (3)** phase, the board collects change requests and locally updated ontologies and analyzes them based on the documented arguments. Afterwards, it decides which requests or changes will be adapted into the shared ontology, or whether there are other changes or refinements to be made. The frequency of this analysis phase is based on the frequency of local changes and incoming change requests.
Based on the decided changes from the analysis phase, the board formalizes and implements these changes in the **Revision (4)** phase. The arguments for these changes have to be aggregated and well formulated, for they also have to be retrievable by the users so that these can understand the changes. These arguments also form the basis for the documentation. Finally, this updated shared ontology will then be distributed to all users.
The users now have to perform the **Local Update (5)** phase. They themselves decide which of the changes made to the shared ontology they want to adapt. For that, the users have to analyze the new shared ontology and integrate it into their local version. They tag outdated versions, include the up-to-date version, and incorporate any local adaptions that they had made but were not reflected in the updated shared ontology (if they still deem them necessary).

This leads to an iterative development, where users take, adapt and change according to their need, and the board decides which of these changes and adaptions are viably applicable to the shared ontology, forming an evolution of the ontology which is closest to its meaning in the field of biology (distributed changes are made, but only the "best" changes prevail).

2.2.5. Human-Centered Ontology Engineering Methodology (HCOME)

HCOME ([21]) is a methodology similar to DILIGENT, in that it employs a form of distributed development. However, it follows a more radical approach. HCOME only distinguishes between two roles: domain experts and knowledge workers. Domain experts are mainly used as a knowledge source for domain topics, but they don't have an active role in the development. The participants who shall develop the ontology are also the ones that shall use them: the knowledge workers. They are no experts in ontology engineering, and neither are they guided by a supervising entity. The whole development process is based on a shared agreement between all participants; to support the knowledge workers in the development and provide them with tools adequate to their ontology engineering capabilities, HCOME suggests to use HCONE (Human Centered ONtology Engineering Environment); details about that can be found in [21], the following part only focuses on the HCOME process alone.

Figure 2.9 provides an overview of the life cycle phases and associated tasks. HCOME differentiates between a shared space (S) where all shared documents and discussions etc. are placed, and a personal space (P) that exists for each knowledge worker; Figure 2.9 marks where tasks are performed accordingly. As can be seen, HCOME has three major phases that form one life cycle iteration:

Ontology life cycle phases	Goals	Tasks
Specification	Define aim/scope/ requirements/teams	Discuss requirements (S)
		Produce documents (S)
		Identify collaborators (S)
Conceptualisation	Acquire knowledge	Import from ontology libraries **(P)**
		Consult generic top ontology **(P)**
		Consult domain experts by discussion (S)
	Develop and maintain ontology	Improvise **(P)**
		Manage conceptualisations **(P)**
		Merge versions **(P)**
		Compare own versions **(P)**
		Generalize/specialize versions **(P)**
		Add documentation **(P)**
Exploitation	Use ontology	Browse ontology **(P)**
		Exploit in applications
	Evaluate ontology	Initiate arguments and criticism (S)
		Compare others' versions (S)
		Browse/exploit agreed ontologies (S)
		Manage the recorded discussions upon an ontology (S)
		Propose new ontology versions by incorporating suggested changes (S)

Figure 2.9.: HCOME process overview [21]

1. Specification
2. Conceptualization
3. Exploitation

The **Specification** phase is about creating specification documents for the ontology. All discussions about the requirements and all resulting documents are performed and created in the shared space that any knowledge worker can access. This specification is created via a dialogue between all participants, and has to be consensually agreed on.
The second phase that follows is the **Conceptualisation** phase. Here, knowledge workers (alone, or as a collaboration) develop the ontology respective to the specification in their own private space. They can follow the development the way they see fit, reusing existing ontologies and adapting them, using existing knowledge resources as input (via machine learning techniques), improvising a from-scratch-development from their perspective, etc. Examples, comments and further documentation are recommended. Any expert questions and discussions that may arise during this phase are, however, performed within the shared space, so that all knowledge workers participating in the development can access them.
Following as the third and last phase of an iteration cycle is the **Exploitation** phase. All developments from the personal spaces are pushed into the shared space and reviewed and discussed by all participants. Provided ontologies are compared, reviewed, evaluated and furnished with critical comments. This feedback is used to adapt ontologies from others to the own purposes, agree on common understandings of conceptualisations for a common

shared ontology, and propose new ontology versions by incorporating suggested changes in the feedback.

HCOME is a development process in a hugely collaborative manner, requiring profound agreement and understanding, intended for ontology users (who don't usually have a lot of development experience). Thus it relies more than any other methodology presented in this chapter on appropriate tool support.

2.2.6. Unified Process for ONtology building (UPON)

As the meaning of the acronym suggests, UPON ([9], [10]) tries to adapt the Unified Process [19], that is widely known in the field of software engineering, to the area of ontology engineering. As such, it relies completely upon UML as modelling language (up to, but not including, the final implementation in the target ontology language) and thus can make full use of a wide array of UML tools.
The main objectives of UPON (and also of reusing an established developing methodology) are

- reducing time and costs in (large scale) ontology developments
- enhancing ontology quality
- identifying and define the use of knowledge engineer and domain expert expertise explicitly
- identifying activities, roles and responsibilities
- producing intermediate results

The process itself is use-case driven and thus not intended to be used for developing generic domain ontologies; it is heavily iterative and incremental. Figure 2.10 depicts a simple sketch of UPON. The process consists of several development cycles; each cycle produces a new major version of the ontology. A cycle itself is separated into four phases: **Inception**, **Elaboration**, **Construction** and **Transition**. Each phase is subdivided into iterations, and for each iteration there are five workflows that have do be followed: **Requirements**, **Analysis**, **Design**, **Implementation** and **Test**. Figure 2.11 provides some insight into this possibly confusing matter. It also shows the amount of involvement of the two kinds of experts (domain expert and knowledge engineer) within the process. As one can see, not all workflows are relevant in each phase. For example, the inception phase deals only with capturing the requirements and some conceptual analysis, whereas in the transition phase the requirements won't be changed anymore, and testing is the main activity. The workflows themselves comply with what one would expect from the names of these workflows, and as they are standard software engineering terminology it is not befitting to describe them thoroughly. However, it is worth mentioning that [10] provides some useful details for these five workflows for those who are interested (not only for the sake of interest, but also especially when wanting to apply this development methodology in a project). Finally, Figure 2.12 provides a complete overview over the combined workflows in an UML diagram.

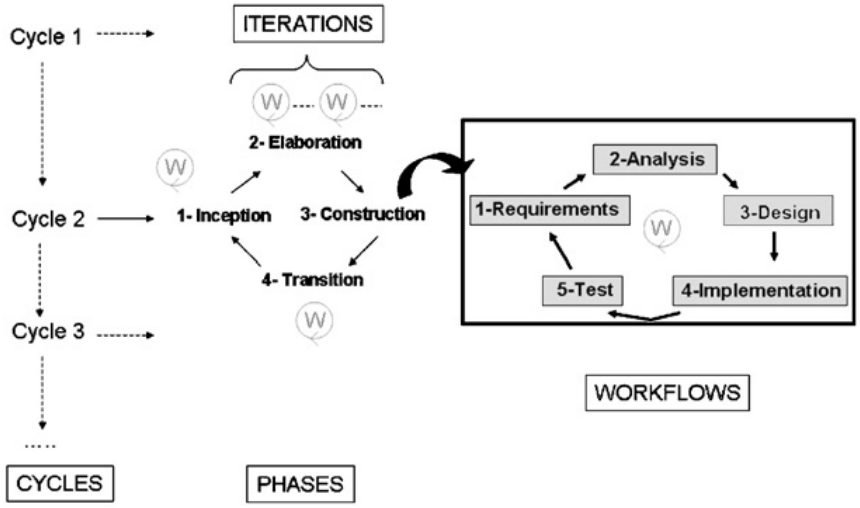


Figure 2.10.: UPON: simple sketch [10]

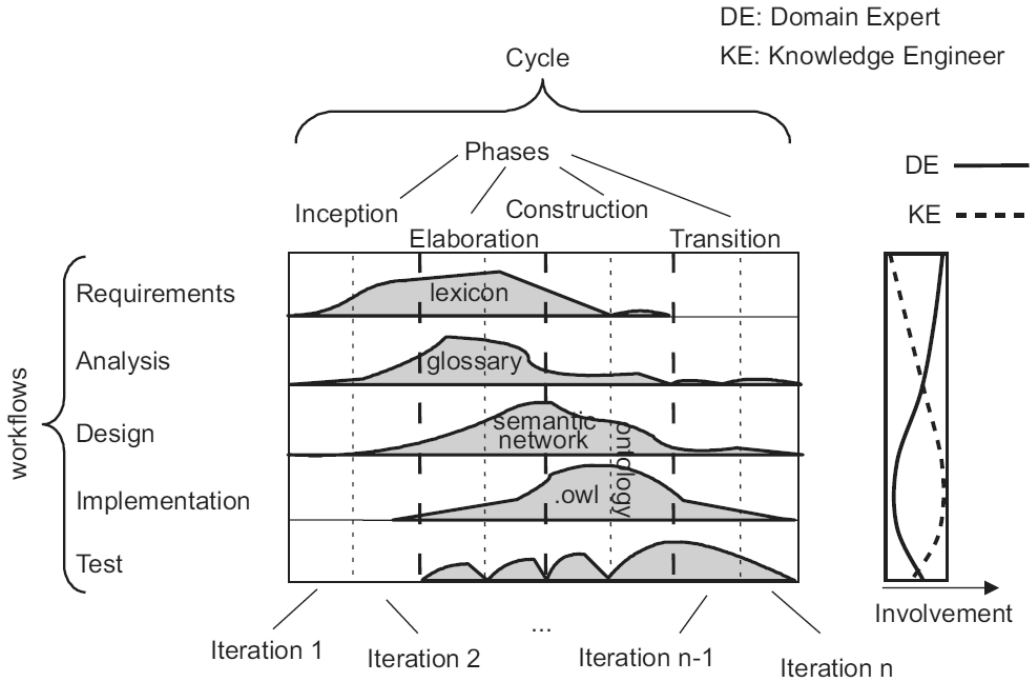


Figure 2.11.: The UPON framework and experts' involvement [10]

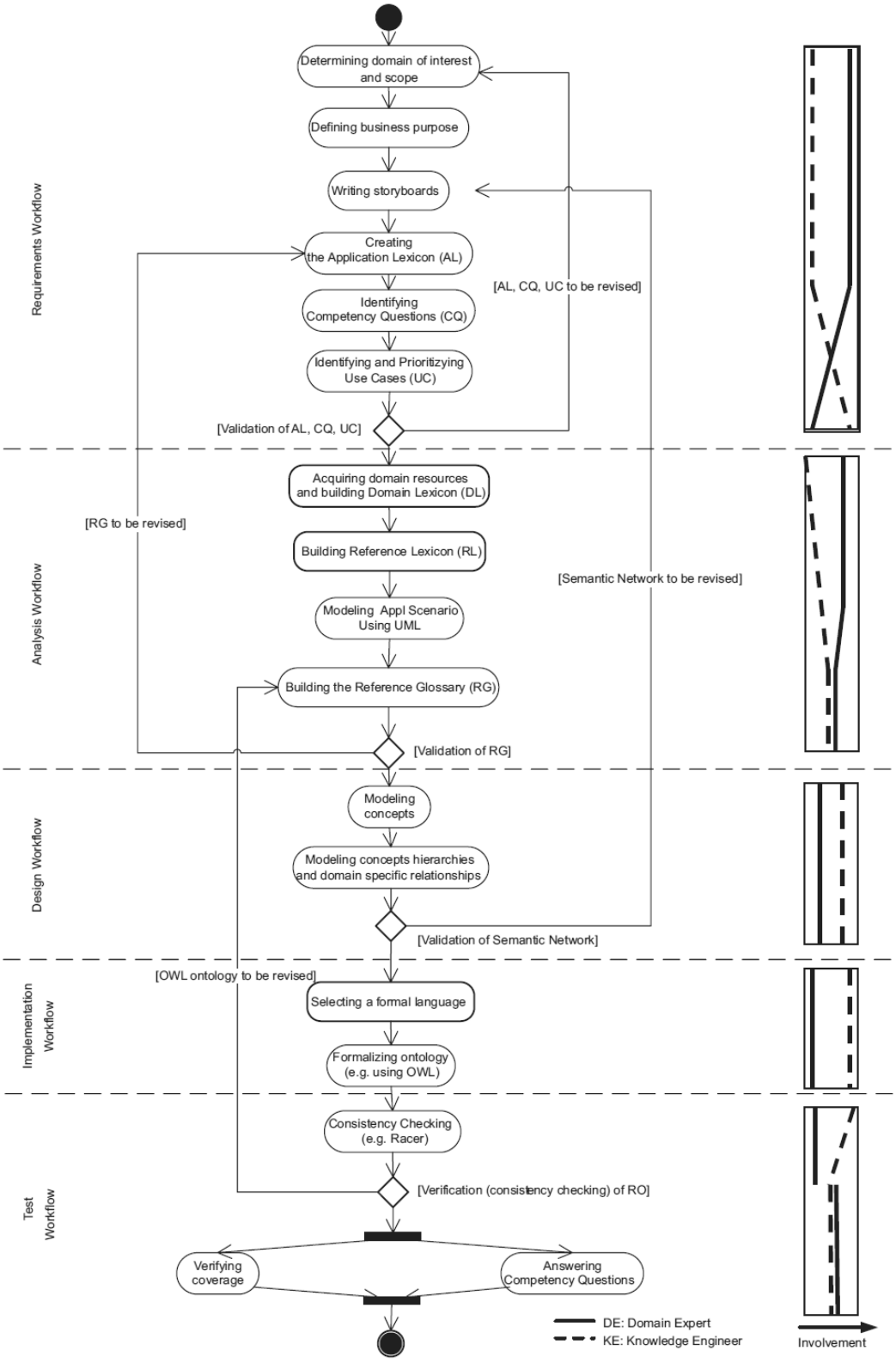


Figure 2.12.: UPON process [10]

2.2.7. NeOn Methodology for Building Ontology Networks (NeOn)

The NeOn Methodology for Building Ontology Networks (short: NeOn) was developed within the NeOn-Project [1], a five year long project with 14 European partners, whose aim was to develop a methodology for the development of ontologies, capable of handling the requirements of large scale semantic application, in a networked environment (i.e. with interconnected ontologies that exist in different versions and use certain aspects of one another).
Currently (as of February 2012), the in-depth documentation for the methodology is available only as collection of the NeOn Project deliverables, where in particular [34, 35, 36, 37, 38] contain the main methodological content. Newer deliverables (Dx.y.z+1) update or even replace the content in older versions (e.g. deliverable D5.4.1 is updated by D5.4.2), which hinders to get a quick and concise insight into the details. The Book "Ontological Engineering in a Networked World" [33], which shall be published in spring 2012, should rectify this problem, as it is meant to be a reference documentation for the NeOn Methodology.

NeOn captures the possible different approaches when developing an ontology in, as NeOn calls it, 'scenarios'. A scenario defines a set of activities that has to be carried out when that scenario is applicable. Different scenarios can be combined with one another, and thus one can obtain an individual instance of the NeOn Methodology for the needs at hand. A graphical overview over the different scenarios can be seen in Figure 2.13.

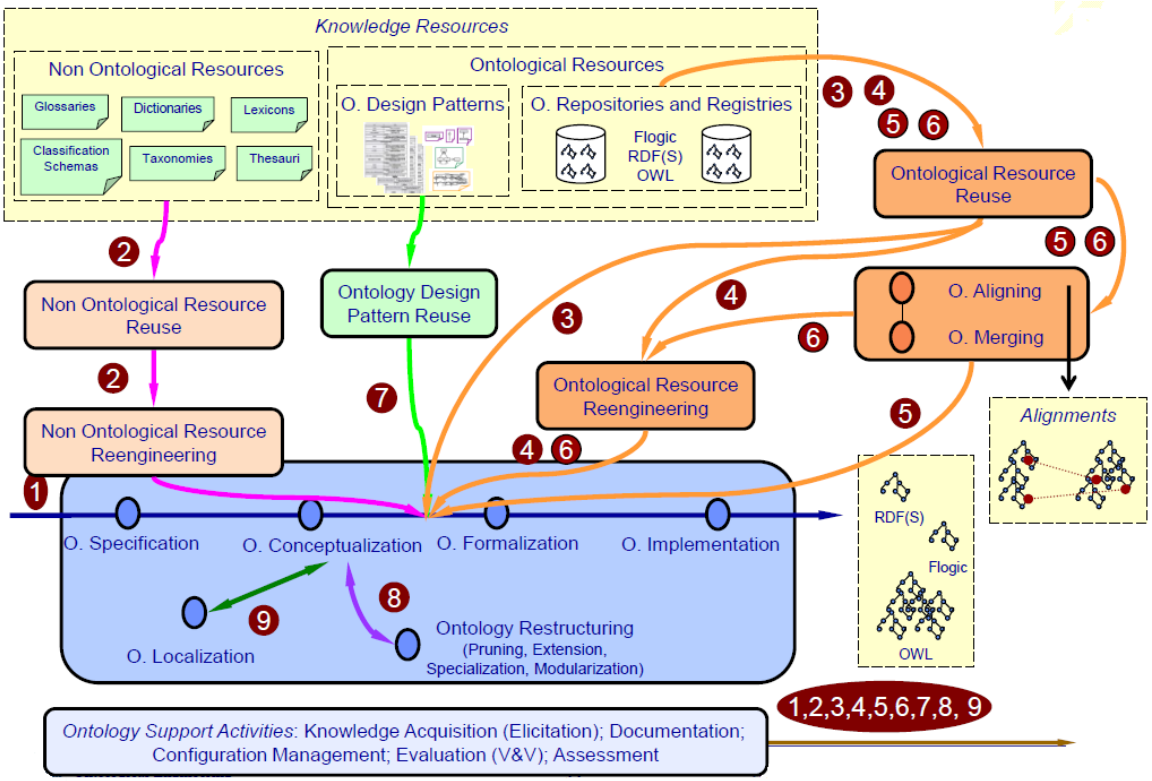


Figure 2.13.: Overview of the NeOn Scenarios, adapted from [37] [sic]

The scenarios capture different aspects in the ontology development.

1. Scenario 1 deals with the basic case, developing an ontology from scratch, i.e. from specification to implementation.
2. Scenario 2 deals with the reuse and reengineering of non-ontological resources during the ontology development.
3. Scenario 3 deals with the reuse of other ontologies for the development.
4. Scenario 4 is an extension of scenario 3 and deals with the reuse and the re-engineering of ontological resources.
5. Scenario 5 extends scenario 3 by also regarding the merging of the selected ontological resources for reuse.
6. Scenario 6 combines scenarios 4 and 5 and thus deals with the reuse, merging and re-engineering of ontological resources in the development of an ontology.
7. Scenario 7 deals with the application of ontological design patterns during the development.
8. Scenario 8 deals with all kinds of restructuring of an ontology, i.e. pruning, extension, specialization and modularization.
9. Scenario 9 deals with the localization of an ontology in different languages.

As one can see, all scenarios interact with the baseline scenario (scenario 1) and add various activities to it. Regardless of the selected scenario, several support activities (i.e. Knowledge Acquisition, Documentation, Evaluation, etc.) accompany the whole development process and are to be carried out at each phase of the ontology development. To attain a concrete flow of activities for the ontology development, the scenario activities have to be mapped to an ontology life cycle. Here, NeOn poses two basic alternatives. For a straightforward execution of a scenario, where all requirements are known beforehand, a waterfall model determines the order of the activities to be carried out. Alternatively, when the requirements are not set in stone at the beginning of the ontology development, NeOn proposes an iterative model, where, after project initiation, the development is carried out in two or more development iterations, each of which follows a waterfall model according to the applied scenario. Thus, with short iterations, one can achieve an evolutionary like ontology development. An earlier version of NeOn differentiated between several more life cycle models than just a waterfall and an iterative model [35], but due to the difficulties that many casestudy participants had with them and the confusion it lead to, it was decided to reduce the number of life cycle models and restrict them to a waterfall and an iterative model. The length of the waterfall is decided by the selected scenarios; a waterfall of maximum length is shown in Figure 2.14, the iterative model is illustrated in Figure 2.15.

Aside from the scenarios and life cycle for the ontology development, NeOn also provides a glossary for all relevant development activities. This is a novelty since it clearly defines what a named activity covers, and due to the involvement of this large number of partners, it can be regarded as a suggestion for a naming standard. Applying these conventions

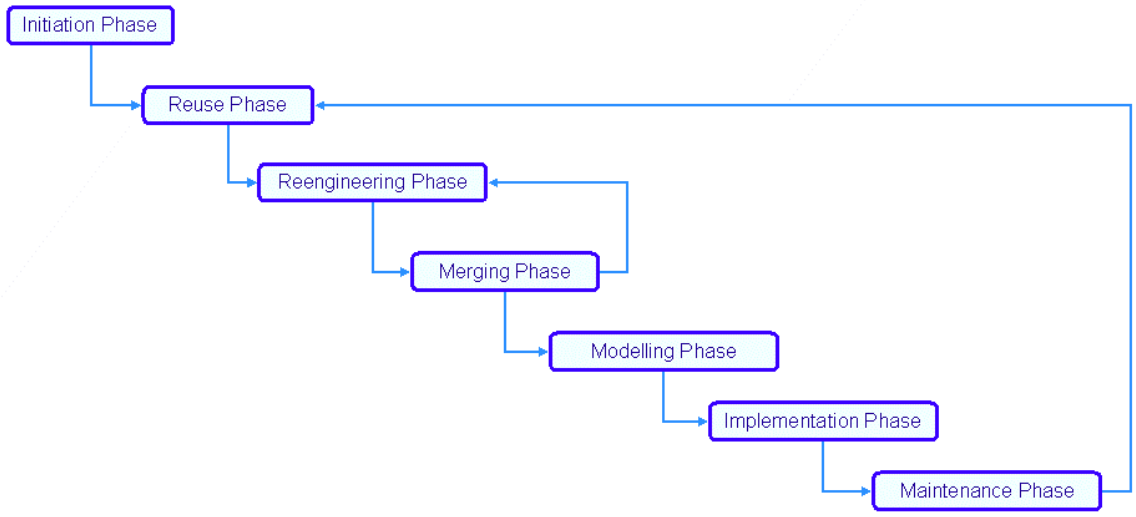


Figure 2.14.: NeOn: maximum waterfall life cycle model [37]

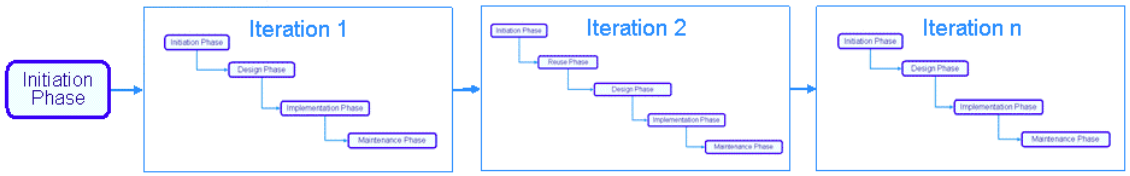


Figure 2.15.: NeOn: iterative life cycle model [37]

reduces the ambiguity of activity names that was present in previous literature.
Generally, NeOn portrays certain aspects of the ontology development process in far more detail than the previous methodologies from earlier subchapters.

2.2.8. Methodology Comparison

After having given a short presentation of current ontology development methodologies, in order to select a fitting methodology for this project, we need to compare the methodologies to one another. There are several formal comparisons of these methodologies already found in publications, which are put together in Appendix A.1 for reference. They provide a good basis for comparing the methodologies from a purely technical viewpoint. However, they fail to provide insights on "soft" matters which are also important for selecting an appropriate methodology, such as "what is the target audience?", "is special tool support provided or even necessary?", "how detailed are given descriptions?", etc.
Thus, to gain an appropriate understanding, while reviewing the methodologies several soft factors were regarded and collected, to provide for a wider basis to make a decision as to which ontology development methodology to select for the development of the aircraft design ontology. Table 2.2 through 2.5 summarize those findings.

Criterion	**METHONTOLOGY**	**DILIGENT**	**HCOME**
Target Audience	ontology engineers	diversity of roles: domain experts, ontology engineers, knowledge engineers, users	knowledge workers (users)
Special Tool Support?	no (ODE used in case studies)	no (OntoEdit OEE used in case studies)	yes: HCONE (needed!)
Target domain / derived from?	chemicals	general distributed development	usage and evolution of ontologies "on the fly" by those working with them
Generality / domain dependent?	generic	generic	generic
Year	1997/1999	2005	2005
Impression	basic process, semi-formalized	Truly evolutionary approach, fine grained guidance; neglects some details for the "personal use"-phase; feels slightly like some kind of meta-methodology for "evolutionizing" standard developments; the single activities are not described into every detail, they're rather just some guideline or reference other ontology development tasks	toolset dependent; made for the "daily working crew"; total absence of specialists, no centralized coordination except discussions; was originally not backed up by a case study; somewhat similar to DILIGENT; process is not very detailed; tool dependent
Addresses particularly?	define standard activities and systematics facilitate engineering instead of just crafting; define products for each process phase, to use for documentation; introduction of intermediate representations	decentralization, partial autonomy, iteration, non-expert builders	empowers knowledge workers to participate during the entire life cycle; distributed and collaborative development

Table 2.2.: Semi-formal comparison of ontology development methodologies (part 1/4)

Criterion	**METHONTOLOGY**	**DILIGENT**	**HCOME**
Applicability for this project (features / stoppers)	partial to full (some details stem from the chemical background, but should also otherwise be applicable; no real stoppers detected)	none to partial; intended for distributed, evolutionary development; activities not involved with the distribution aspect are provided better in other methodologies	none (depends on specific tool set for abstraction from formal details; intended for use during daily work by normal users)
Ontology language specific	no	no	no, but depends on the tool framework
Process details (life cycle etc.)	dev. Process defines activities without order of execution; life cycle defines the "order" of these activities; "evolving prototype" life cycle as an adaption of a waterfall model	initial core development, after that cycles of distribution → feedback and update gathering → consolidation and evolution → etc.	life "cycle", i.e. the iteration aspect, is not directly expressed in this methodology, but only implied
Support of mind-mapping	not mentioned	not mentioned	not mentioned
Human issues?	(domain) experts need to understand the (intermediate) representations for evaluation and feedback	active contribution from users required for ontology evolution (resp. in most other methodologies users are not intended to directly contribute to the development of the ontology), else the methodology becomes something similar to earlier methodologies like OTK	no expert involvement, hence totally dependent on user interaction; on the other hand this may back up user acceptance (or not, depending on "discussion culture"); higher abstraction level necessary since users are no experts

Table 2.3.: Semi-formal comparison of ontology development methodologies (part 2/4)

Criterion	**OTK**	**UPON**	**NeOn**
Target Audience	ontology engineers in cooperation with domain experts	ontology engineers in cooperation with domain experts	ontology engineers in cooperation with domain experts
Special Tool Support?	yes: OntoEdit+Plugins	indirectly (→ UML tools)	yes: NeOn Toolkit
Target domain / derived from?	(enterprise) knowledge management	general development for specific usage (→ use case driven); B2B eBusiness	general development in a networked environment
Generality / domain dependent?	generic	generic	generic
Year	2002	2005/2008	2006/2010
Impression	ideal standard compared to software engineering process models; sequential, iterations within single phases; agility lies mainly within evolution	detailed process that bears the closest resemblance to software engineering processes; makes intensive use of UML	modular; indicates some sort of tailoring (→ compare V-Modell XT); detailed description of activities; closest to state of the art
Addresses particularly?	proposes a generic ontology engineering methodology for the establishment of a knowledge management application	presents an ontology development process based on an established software engineering process	networked ontologies, i.e. ontologies in the context of others, with different relations or dependencies

Table 2.4.: Semi-formal comparison of ontology development methodologies (part 3/4)

Criterion	**OTK**	**UPON**	**NeOn**
Applicability for this project (features / stoppers)	partial; embedded in the context of a knowledge management application (includes knowledge in form of an ontology), software engineering and human issues)	none to partial (use case driven; not intended for generic domain ontologies, but for knowledge application development)	partial to full (no stoppers detected)
Ontology language specific	no	no	no
Process details (lifcycle etc.)	step by step process with iterations in between; process embedded into an organization ($\rightarrow$ therefore the first phase, feasibility study)	cyclic development, divided into phases which may each require several iterations, with a workflow for each iteration	flexible life cycle, depending on project requirements
Support of mind-mapping	explicitly mentioned	not mentioned	indirectly mentioned (via a reference to OTK for the design and implementation phases)
Human issues?	explicit acknowledgement of human issues (acceptance, organisation structure etc.)	easier transition for and possibly higher acceptance from people with familiarity with UML	documentation currently only available distributed over several project deliverables

Table 2.5.: Semi-formal comparison of ontology development methodologies (part 4/4)

2.3. Aircraft Basics

This section provides a short and rough overview of the domain of aircraft design. This is a very extensive domain, and since the aircraft design ontology in a first version will only scratch the surface of this domain, this appropriate overview also won't go into deeper details. For those interested in an in-depth discussion of this matter, please refer to [26] and [41], which also provide the basis for this overview.

Aircraft design is a highly iterative process. Certain design decisions can have an influence upon previous decisions, so each phase of the aircraft design process is normally iterated several times. The design process itself can be grouped into three major phases:

1. Conceptual design
2. Preliminary design
3. Detail design

Conceptual design deals with the basic questions, such as will the intended aircraft work at all, what will it look like, what are the requirements and how do they influence the design, what will it cost, etc. This is followed by the preliminary design, where the rough configuration is frozen (although extreme and from a project planning standpoint unsatisfactory, the term 'frozen' is actually used in this context) and further designed and extended in detail. Finally, the detail design is already close to the production process, as here the actual parts and pieces that will be built are designed, major items are thoroughly tested, support items (e.g. tools) for the production are designed, and any open performance and weight issues are finalized. After (or even partially parallel to) this phase, the production of the aircraft begins.

For this thesis, only very basic aspects of this wide domain will be outlined, which are mainly associated with the conceptual design. And since this domain is very extensive, this will provide only a rough draft.

An example for a rough process can be seen in Figure 2.16, which also shows the iterative nature of the aircraft design process. Starting from the requirements, one can first establish some estimates as rough boundaries for the design: what is the transport capacity, what is the intended cruise speed and the desired range, what could typical missions look like, etc. From this data, a first sketch of the airplane arrangement can be made in order to obtain an initial concept for the aircraft. Questions here, amongst others, concern:

- the basic shape and size of the **fuselage** and its possible layout
- size, shape, installation and positioning of the **wing** (vertically, that is high-wing, mid-wing or low-wing, i.e. is the wing mounted on top of the fuselage, in the middle, or below; horizontally, that is relative to the length of the fuselage)
- the **engine** layout, positioning and installation (how many engines, where are they located, what kind of engines)

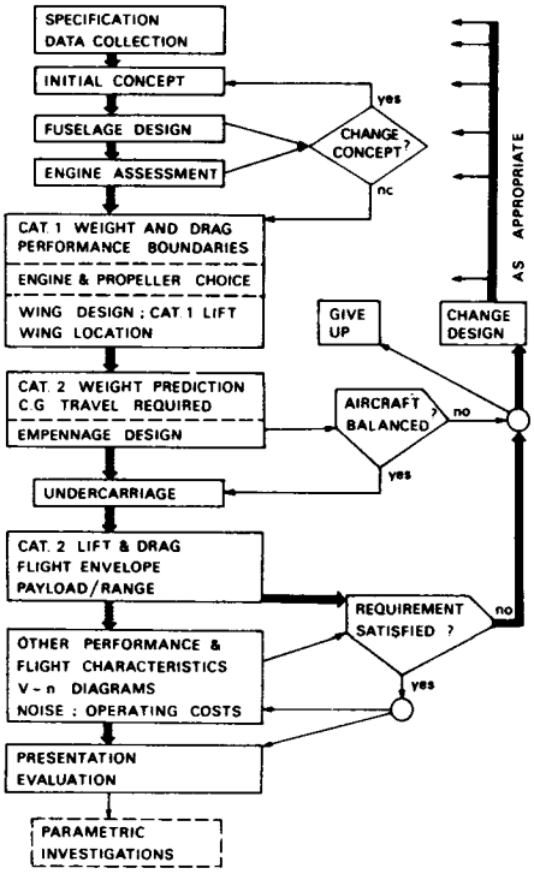


Figure 2.16.: Aircraft design process sketch [41]

- the size and layout of the **tail**
- the configuration, layout and position of the **undercarriage**
- **weight** estimates that affect the design (e.g. operating weight empty, maximum take-off- or maximum landing weight, etc.)
- **performance** estimates and boundaries

For each of these aspects, there are lots of details that have to be decided and designed, and for each aspect specialists and special design tools are required. As soon as the configuration is set, the design process can continue with the next phase.

3. Ontology Development Process

In this chapter, the development process for the aircraft design ontology is described. In chapter 3.1 the project preliminaries are mentioned, as well as why according decisions were made. Chapter 3.2 deals with the initiation, or better referred to as 'instantiation', of the selected NeOn methodology for the ontology creation, i.e. the tailoring of the methodology to this project and creation of a specification and project plan. Chapters 3.3 through 3.5 follow the methodology phases of our concrete methodology instance, starting with the search for reusable ontological and non-ontological resources in chapter 3.3, followed by the re-engineering of found resources in chapter 3.4 and finally the details about the implementation in chapter 3.5, where the completed ontology is described. The evaluation of the completed ontology and its results are shown in the last chapter, 3.6.
Please notice that this chapter only contains the details about the development process; the resulting ontology is described in detail in chapter 4.

3.1. Preliminary Project Decisions

Some preliminaries for the project were almost clear at the announcement of this thesis, others became clear during the preparation and the theoretical part of the thesis. As we have seen in chapter 2.1.3, there exist more than one language capable of capturing knowledge for the purpose of building an ontology. However, it was fairly clear from the start, that the language the ontology will be built in, will be the Web Ontology Language, also known as OWL. First of all, it was a requirement from the people who will be using the ontology, since they already had some experience in OWL, and changing to another language would have resulted in investments needed to learn the specifics of another language, as well as the investments needed to make use of another language, since there was already a basic set of code able to work with OWL, as well as the experience of working with some OWL processing APIs like Jena[1] or the OWL API[2]. Furthermore, the decision could be justified by the fact that OWL (with its whole language stack) is a World Wide Web Consortium (W3C) recommendation and thus widely used in various products. It is also capable of a rich expressiveness, whose scope is determined by several different profiles, and is supported by a wide range of reasoners and development tools, as was already shown in chapter 2.1.3 and 2.1.4.

Regarding the development tools, a representative selection was shown in chapter 2.1.4. The choice for a development environment fell on Protégé, for several reasons. First of all, due to its long history, Protégé has a lot of "ease of use features" that support and accelerate the development of an ontology; lots of these features are missing in other development tools

[1] http://incubator.apache.org/jena/
[2] http://owlapi.sourceforge.net/

(e.g. the pragmatic bulk generation of class hierarchies, or the support for transforming primitive classes to defined classes and vice versa, etc.). Another important aspect is the extendabilitiy via plug-ins; also, many other development tools are extendable via plug-ins, but again due to its long history, Protégé has a very large base of downloadable plug-ins available. Furthermore, there are several reasoners available that can be integrated into Protégé, whereas other tools often have only one kind of reasoner available. And last but not least, Protégé restricts itself to pure OWL, meaning that no RDFS modifications or variations are supported via the user interface; this may be regarded as a restriction, but it also furthers the development of pure OWL ontologies, staying true to the OWL definition without individual extensions which might affect the usability of the ontology in other tools.

With respect to the ontology development methodologies presented in chapter 2.2, it was decided to use the NeOn methodology for the development of this aircraft design ontology. The reasons for this decision were twofold. First of all, the methodology is currently the latest state of the art in ontology engineering methodologies (the NeOn project ended in 2010). It also has a very large member base that was participating in the project: 14 European partners, some of which took part in the development of earlier methodologies like METHONTOLOGY or OTK [1]. With this sophisticated background, one could regard the methodology as a de-facto standard, or at least as a serious proposition. But the second and even more important aspect is that in many regards NeOn provides far more details on what to to and how to do it than other methodologies, thus giving more sophisticated development guidelines. The NeOn Glossary of Activities furthermore provides a summary and description of all relevant development activities, reducing possible ambiguity that could occur without such a glossary. All together, NeOn fitted our requirements, and it also has the advantage of being customizable to the needs at hand, so we instantiated the process accordingly, which is the customized NeOn process that is described in the following subchapters.

To set up the project initiation, the NeOn Toolkit together with the gOntt plug-in were used to set up the initial project plan. The Toolkit and said plug-in were only used for a first hand initiation, the results of the plan were manually transformed into a “Microsoft Project” project. The main reason for this decision were the instability issues that were experienced with the NeOn Toolkit in connection with the gOnnt plug-in, but also Microsoft Project allows a more sophisticated project planning with diverse restrictions, compared to the rather basic gOntt plug-in, whose main feature is to create a NeOn conforming project plan (the advertised support of the NeOn methodology with explanation cards and the interlocking of NeOn features in the NeOn Toolkit would have provided a substantial additional value that would have made its use worthwhile, but it seems like these advertised features were mainly present in earlier versions of the Toolkit and plug-in), since the current versions (NeOn Toolkit 2.5.2 and gOntt 1.6.5) only offer a rather incomplete experience of said features.

3.2. Initiation Phase

In this section, the Initiation of the NeOn methodology will be presented. At first, the methodological guidelines will be described, and afterwards the application of these guidelines within this project will be illustrated.

3.2.1. Initiation Phase: Methodology Guidelines

For the initiation, NeOn provides rather detailed guidelines. Prior to the beginning of the development, an Ontology Environment Study and an Ontology Feasibility Study shall determine if, or under what circumstances, the development of a planed ontology can be done [35]. After this initial assessment, and assuming it has come to a positive conclusion, the actual development can commence.
First of all, NeOn requires formulation of the needs and general framework into an Ontology Requirements Specification Document (ORSD) [36]. For this purpose, it provides a task flow that is shown in figure 3.1.

Furthermore NeOn also provides a document template to capture the results of the requirements specification tasks, that is shown in figure 3.2. With the content from the ORSD, one can roughly estimate what to expect during the development process and use this knowledge as input for the scheduling tasks. Figure 3.3 provides a compact overview. Depending on whether the requirements are completely known at the beginning of the development, or whether the requirements have been prioritized, the basic life cycle model will be either a waterfall life cycle model, or an iterative incremental life cycle model (meaning that the development will consist of several iterations, where each iteration follows a waterfall life cycle model). From the OSRD one can foresee what kind of scenarios (see chapter 2.2.7) will be applicable in the ontology development. Each scenario defines a set of activities that has to be carried out, and the combination of the applicable scenarios, i.e. the union of the activities, defines the complete set of relevant activities for the ontology development at hand. With the knowledge of what activities have to be carried out, these activities must then be mapped to the life cycle model and put in an appropriate order, which results in a basic activity flow, that can be enhanced by time- and resource-estimates and -restrictions, which will finally result in the concrete schedule for the ontology development. Thus, one can also say that the development methodology has been properly instantiated.

A useful tool during this instantiation process for gaining the final schedule is the NeOn Toolkit in combination with the gOntt plug-in. This fully supports the scheduling process with appropriate automatic mappings and ordering of the activities, making it unnecessary to perform these tasks manually.

3.2.2. Initiation Phase: Methodology Application and Results

This subsection describes the application of the above presented initiation for the development of the aircraft design ontology. The pre-studies were not an issue in this case, since the ontology development was given with the announcement of the thesis. So it was not an issue whether the ontology can be built, but rather to build the ontology and gather experience in this field, that is currently in the syllabus of students of computer science

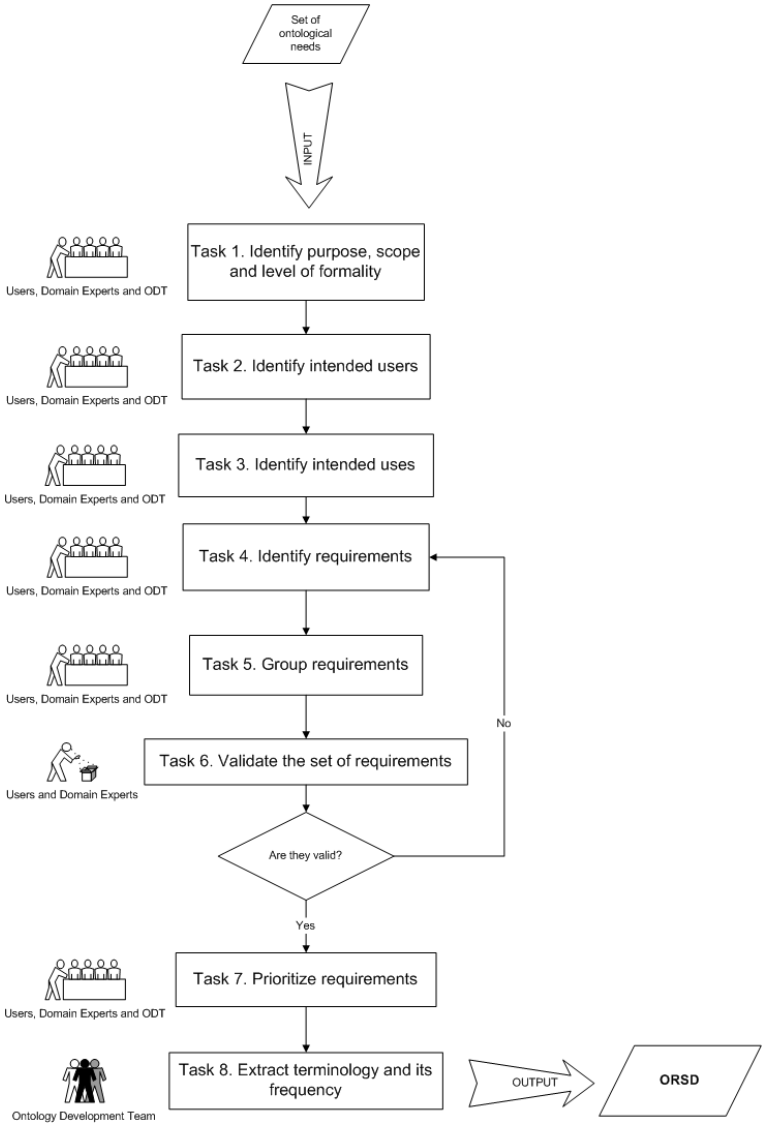


Figure 3.1.: Tasks for Ontology Specification [36]

	Ontology Requirements Specification Document Template
1	**Purpose**
	"Software developers and ontology practitioners should include in this slot the purpose of the ontology"
2	**Scope**
	"Software developers and ontology practitioners should include in this slot the scope of the ontology"
3	**Level of Formality**
	"Software developers and ontology practitioners should include in this slot the level of formality of the ontology"
4	**Intended Users**
	"Software developers and ontology practitioners should include in this slot the intended users of the ontology"
5	**Intended Uses**
	"Software developers and ontology practitioners should include in this slot the intended uses of the ontology"
6	**Groups of Competency Questions**
	"Software developers and ontology practitioners should include in this slot the groups of competency questions and their answers, including priorities for each group"
7	**Pre-Glossary of Terms**
	Terms
	"Software developers and ontology practitioners should include in this slot the list of terms included in the CQs and their frequencies"
	Objects
	"Software developers and ontology practitioners should include in this slot a list of objects and their frequencies"

Figure 3.2.: Template for the ORSD [36] [sic]

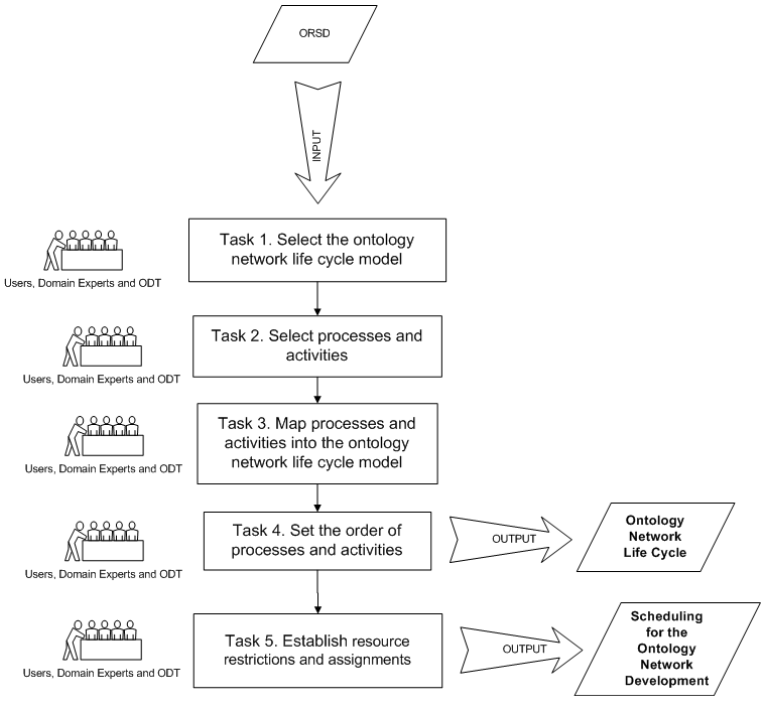


Figure 3.3.: Scheduling Tasks [37]

still more of a side topic, compared to classic data modelling for example in the context of classic relational databases.

Specification

The ontology that shall be built, as was presented in chapter 1, shall be a first version of a domain ontology for the domain of aircraft design, which shall be used as a central data model to further the ease of data exchange between various programs and tools that are used during the process of aircraft design. The main focus for this first ontology version should lie on the structural aspects of an aircraft. For this purpose, models from three programs were given in XMI format as a reference for the aspects and parameters to be modeled. The content of these models was restricted, i.e. content that was regarded as irrelevant for this first version was excluded.
To obtain the ontology specification, the task flow that NeOn proposes was followed (see 3.1). For the requirements identification, NeOn proposes the use of competency questions, natural language questions that shall be answerable by the completed ontology; these competency questions are used to define the scope of the ontology, and furthermore they can be used as a completion test to check the built ontology. In our case, the formulation of all relevant competency questions would have been a huge and very repetitive task; the models contain structural descriptions where components can have several dozen parameters (like the span of a wing, the length of the fuselage, etc.). We didn't see any added value in formulating hundreds of competency questions like "what is the length of an aircraft's fuselage?" or "what is the height of an aircraft's fuselage?" etc., thus we decided to skip these questions

and regard them as given implicitly through the models (those models are used to limit and define the focus of the ontology); any further re-engineering of the models would also rather fit into the re-engineering phase (see chapter 3.4). The competency questions formulated in the specification cover more general or non-primitive aspects. The competency questions were grouped and prioritized as suggested by NeOn. However, the pre-glossary of terms that NeOn suggests to set up was omitted, due to the lack of fine-grained integration of the model contents into the competency questions; as stated before this would have anticipated the re-engineering task of these models. With the completed ORSD (which can be seen in appendix A.2) we could now establish a project plan and schedule for the project.

Scheduling

In order to acquire the project schedule it was chosen to use the functionality of the NeOn toolkit in combination with the gOntt plug-in for this task (NeOn Toolkit v2.5.2 and gOntt v1.6.5). At first it was intended to perform at least two iterations for the development cycle, but due to time constraints it was only possible to plan for one iteration. Therefore, scenario 1 (ontology implementation; this is usually the baseline scenario), scenario 2 (non-ontological resource reuse; processing of the given models and maybe other resources), and scenario 3 (ontological resource reuse; looking for and possibly reusing ontologies that model relevant aspects for the planned ontology, e.g. units and measures) were selected for the plan creation (see chapter 2.2.7). The resulting plan was then manually transformed into a Microsoft Project plan, due to several issues, mainly instability. The transformation was done manually since, unfortunately, the gOntt plug-in uses a proprietary format to store its information and it does not provide any kind of export possibilities into common file formats. The time estimates were adjusted on a 'best guess' basis to fit into the remaining thesis schedule. The resulting project plan can be seen in appendix A.3.

3.2.3. Initiation Phase: Discussion

The start of a NeOn project does not feel totally well conceived, since there are some slight discontinuities. The initiation phase already requires a lot of work and involvement, which at this point is not yet planned or scheduled. It would be smoother if there was some kind of pre-scheduling at the very beginning that also encompasses the pre-development studies (which lead to the decision whether the intended project will be started) and the requirements specification task (if the project is okay to "go") up to the development-scheduling, since these activities normally already require some effort. Thus, these activities would not only be planned afterwards, as is now the case with the scheduling activity. Another thing that comes to mind are the competency questions. While these are a very valuable tool, formulating them to the very last becomes very tedious for large ontologies like in this project, without offering any real added value (when regarding the myriads of parameters for each aircraft component). Also it would have meant processing the given models beforehand, foreclosing a lot of work that was normally intended for the re-engineering phase. Thus one ought to think about an equivalent alternative that captures the same information but without the unnecessary effort. In this project we chose a pragmatic approach and used placeholders / wildcards / variables for those kind of questions; the items themselves were given via the models, so we regarded this as sufficient, and thus only have, for example, one

competency question "what is the [(structural) parameter] of [a (structural) component]?" instead of a single competency question for each parameter in each component.

3.3. Reuse Phase

After the initiation phase, the next phase that follows according to our NeOn project plan is the reuse phase. As the name suggests, it deals with the reuse of ontological or non-ontological resources (see [36] and [38]). This encompasses the search for appropriate ontological resource candidates, assessing their features and comparing them to one another in order to possibly find suitable ontological resources that can be selected for integration into the planned ontology.
First, the guidelines that NeOn proposes will be shown, which will then be followed by demonstrating the application of these guidelines in this project.

3.3.1. Reuse Phase: Methodology Guidelines

Regarding Ontological reuse, NeOn distinguishes between several settings, depending on the kind of ontology and the granularity to be reused. Figure 3.4 provides an overview over these different types.

For each type of ontological reuse, NeOn proposes a flow of activities that details the reuse process. This activity flow differs from type to type where each has different focus points regarding the specific needs, but a core concept that appears in all these is the basic activity flow

1. search
2. assessment
3. comparison
4. selection
5. integration.

First of all, one has to search for appropriate ontological resources that cover concepts that are mentioned in the ORSD. This is usually done via a general internet search, or by using specific ontology collections, for example, Swoogle [12] or Ontology Lookup Service (OLS, [13]). The thus found ontological resources then have to be assessed as to whether it is useful to reuse them in the ontology development, regarding the requirements from the ORSD. All resources that have been found to be suitable candidates will then be compared to one another, in order to select the resource that best fits the requirements at hand. Besides hard requirements, like for example the ontology language or specific functional requirements, it is also important to take basic quality concerns into consideration, like how easy it is to understand the ontological resource (i.e. if the documentation is sufficient) and how much time that would take, how much effort it would take to integrate the resource into the development, whether the resource follows good design principles, and how reliable it is (has it been evaluated, has it been reused in other projects, etc.). This comparison yields a (some) resource(s) that will be selected for the integration into the development project.

Figure 3.4.: Different Types of Ontological Resource Reuse [36]

This whole basic process is done for each aspect from the ORSD that might possibly be covered by reusing ontological resources (with variations depending on the different ontological resource types).
Generally, one has to be careful to find an acceptable compromise between reusing some ontological resources (if possible) and reusing too many ontological resources, in order not to unnecessarily waste important development resources like manpower or time.

Regarding Non-Ontological Resource Reuse, the process to select appropriate resources is basically similar to that for ontological resources, only more condensed:

1. search
2. assessment
3. selection

After searching for appropriate resources, each resource has to be assessed regarding the ORSD requirements, as well as qualitative properties like documentation, trustworthiness of the information, etc. The selected non-ontological resources are then processed in the upcoming re-engineering phase (which could also be regarded as a final step for the above list, but since re-engineering is a separate phase from the reuse phase, this would not be exactly correct).

3.3.2. Reuse Phase: Methodology Application and Results

Now that we have shown the rough NeOn guidelines for reuse, it is time to present their application within this project.

Non-Ontological Resources

Regarding non-ontological resources, there were several different possibilities available. On the one hand, there were the reference models that were provided for this thesis. Where two of these models (SIMCAD[3] and PACE[4]) contain a useful content, with component names and values, the APA[5] model unfortunately was missing any names at all (names that are associated to variables and structures, in order to properly identify them) and consisted only of a collection of unnamed arrays and their values. Thus, only these two useful models were selected as resources for the ontology development. Later on during the development, however, it became clear that the APA model given was faulty and that a more current version not only redeemed those faults but also contained valuable annotations. Unfortunately, this only became clear in the advanced stages of the development of the ontology, and due to time constraints it was not possible anymore to make use of this change.
Beside these models, there was already a semantic wiki in use at Bauhaus Luftfahrt whose contents might have been exploitable. However, this wiki mainly contained organizational topics, and not a systematic description of aircraft or aircraft-design topics. Due to this shortcoming, we relinquished to use the Bauhaus wiki as a resource for the ontology.

[3]Provided by Bauhaus Luftfahrt. Commercial software.
[4]Provided by Bauhaus Luftfahrt. Commercial software. http://www.pace.de/
[5]Provided by Bauhaus Luftfahrt. Internally developed modelling tool.

A third and valuable resource at hand were the on-site experts from Bauhaus Luftfahrt. Among them are researchers and engineers who design new aircraft concepts to improve and adapt mass passenger air transport to upcoming challenges and requirements in the aircraft industry. These experts have a deep knowledge and understanding of aircraft- and aircraft design concepts. Tapping into this knowledge was intended to be done via interviews, especially for answering specific questions, and for the evaluation of modeled concepts in the ontology. Unfortunately, due to time constraints, we were only able to interview two experts for the final evaluation of the ontology.
Finally, via libraries and the internet, there exists huge amount of textual resources, as books or electronic resources, that one may use. Resources like these were used primarily for obtaining domain knowledge with respect to understanding the domain and answering contextual questions that came up during the development. An approach to automatically use and transform those kind of resources via NLP (Natural Language Processing) techniques didn't seem useful, as the results of some case studies from the ontology development methodologies in chapter 2.2 suggest.

Ontological Resources

Regarding ontological resources, there were two main aspects that were striking from the ORSD. First of all, are there any ontological resources that model aircrafts or some aircraft relevant aspects that can be used for this ontology? A thorough search via Google and appropriate ontology search engines and collections (in November/December 2011) came up with no results that were useful for this project. One project that is easily found is the JFACC Air Campaign Planning Ontology[6], but it has a heavy focus on military air campaigns, and aircrafts are only modeled as a whole in this context, which is insufficient for our project. As a side note, the ontology is also over a decade old. Other resources may distinguish between certain kinds and versions of aircrafts, but also do not provide any detail about the aircraft specifics and build-up; the aircraft is at most only regarded as a whole. Finally, there is a Chinese paper with an abstract available in English, but the paper is only available in Mandarin from a Chinese website, which makes it unusable due to (our) lacking Mandarin skills. All in all there were no aircraft-specific ontological resources to be found that could be put to use in this project, but this was already expected, otherwise this thesis would not have been proposed.

The second aspect that was deemed worthwhile to look for was the modelling of quantities and units. Since this is a rather fundamental topic, and there exist standards like the SI units [6] and an International Vocabulary of Metrology [20], it was anticipated we would find a standard ontology for this topic. The search came up with various results, however there was no standard ontology to be found. The found ontologies are listed in Table 3.1. From those found ontologies, those that were not available (or easily convertible) into proper OWL2-DL could already be disregarded, since that is a requirement in the ORSD. From the remaining 5 ontologies (MUO, QUDV, UO, UCUM and QU), the QU ontology was the one that was best fitting for this project: the UCUM ontology is rather simplistic and does not have a rich semantic content; UO lacks, for example, the modeling of unit symbols,

[6] http://www.isi.edu/isd/JFACC/loom-jfacc-final-report.pdf

and it draws its use mainly through the connection with a rdf reference database (PATO), making its handling in Protégé extremely unwieldy; the typical OWL reasoners FaCT++, HermiT and Pellet could not properly process the XML literals in MUO; QUDV serves as a basis for QU and is soundly modeled, but it lacks individuals for the units; QU uses QUDV as a sound foundation, it is, however, enriched by a vast set of all kinds of quantities and units, furthermore it is still easy to understand and to integrate.

Name	Full Name	Dialect	URL
QUDT	Quantities, Units, Dimensions and Data Types	OWL 2 Full	www.qudt.org
MUO	Measurement Units Ontology	**OWL 2 DL**	http://forge.morfeo-project.org/wiki_en/index.php/Units_of_measurement_ontology#Measurement_Units_Ontology_.28MUO.29
QUDV	Quantities, Units, Dimensions and Values	**OWL 2 DL**	http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-qudv:quantities_units_dimensions_values_qudv
W3C units	'Unit Ontology published on the W3C Website'	OWL 1 Full	http://www.w3.org/2007/ont/unit
UO	Ontology of Units of Measurement	**OWL 2 DL**	http://code.google.com/p/unit-ontology/
SWEET (sub-part)	Semantic Web for Earth and Environmental Terminology	OWL 2 Full	http://sweet.jpl.nasa.gov/
OM	Ontology of Units of Measure and Related Concepts	OWL 2 Full	http://www.wurvoc.org/vocabularies/om-1.8/ & http://www.semantic-web-journal.net/sites/default/files/swj177_3.pdf
UCUM	Unified Codes for Units of Measure	**OWL 2 DL**	http://marinemetadata.org/ucum & http://marinemetadata.org/references/ucumunits
QU	Library for Quantity Kinds and Units: schema, based on QUDV model OMG SysML(TM), Version 1.2	**OWL 2 DL**	http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu.owl & http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu-rec20.owl

Table 3.1.: Unit ontology search results

3.4. Re-Engineering Phase

This section describes details of the re-engineering phase, the guidelines NeOn provides, as well as the application in this project.

3.4.1. Re-Engineering Phase: Methodology Guidelines

The guidelines for the re-engineering phase can be found in the NeOn deliverables [36] and [34]. Regarding ontological resources, there was nothing to be done in our case, since the selected QU ontology for units could easily be integrated without modifications.
For the purpose of non-ontological resource re-engineering, NeOn suggests a three step approach:

1. reverse engineering
2. resource transformation
3. forward engineering

The reverse engineering deals with identifying and grasping the underlying concepts of the resource, that are then transformed into a conceptual model in the resource transformation step. Finally, in the forward engineering step, the generated conceptual model is then formalized and implemented into an ontology.

3.4.2. Re-Engineering Phase: Methodology Application and Results

As described before, the non-ontological resources that were relevant and selected in this project were the given reference models from SIMCAD and PACE. These UML models were given in a hierarchical XML format (XMI). To understand and grasp the underlying concepts, each model was transformed into a mind map. This was a practical way to get to know the modeled concepts and their connections, and it was a quick way to get a graphical overview which supports this understanding process. These two mind maps were restricted according to the assumptions made during the specification (i.e. certain concepts were excluded) and then were used as a reference to model the appropriate entities within the aircraft design ontology. This final forward engineering step in terms of content overlaps with the design and implementation phase, thus it was decided not to perform this as a separate step, but integrated within the design and implementation phase.

An interesting alternative method to use and transform the given models might have been an automatic model transformation from UML to an OWL ontology. However, this is no simple task. A general UML to OWL transformation has to deal with the problem that each UML model tool often uses a manufacturer dependent UML dialect [17]. For the ATL[7] framework (ATL Transformation Language), there exists a model transformation from UML to OWL [8], but since this example is from 2007 the transformation is only to OWL 1, and furthermore, the transformation is also incomplete, since the participants in this project could not agree on how to transform some specific UML aspects. Getting this

[7] http://www.eclipse.org/atl/
[8] http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/

example to run is also a difficult endeavour, since the ATL framework has since experienced some overhaul and there are a lot of dependencies between the needed plug-ins which have incompatibilities to their earlier versions. As such it requires a tremendous effort and a lot of patience to even test this example.
The alternative would be to develop such a transformation oneself, but since such a development is a time consuming matter in itself, especially if it is to be done properly, it was out of scope for this thesis.

3.5. Design and Implementation Phase

This section describes the design and implementation phase within NeOn, and the application in this project.

3.5.1. Design and Implementation Phase: Methodology Guidelines

Regarding the design and implementation, NeOn surprisingly does not provide more detail than other methodologies. In fact, for the baseline scenario 1, as is mentioned in [35] NeOn suggests to follow the steps and activities that are described in METHONTOLOGY ([14, 23]) or OTK ([31], [28]). We chose to follow the process described in OTK.
OTK proposes three kinds of approaches for the conceptualization: Top-down, bottom-up, and middle-out. The top-down approach starts by modelling concepts and relations on a generic level, and then extend these further into more detailed levels, until the necessary level of detail has been reached. This is typically done manually. The bottom-up approach is practically the opposite, as it usually involves machine learning methods. Relevant concepts and relations are automatically extracted from available documents, and from these fine-grained structures common concepts on higher hierarchy levels are identified. The middle-out approach is, as the name implies, a compromise between those two extremes. Here, the most important concepts are first identified, and then, depending on the needs at hand, further broken down into more details, or combined into common concepts at a higher level.
Furthermore, OTK also recommends the use of competency questions as guidelines to identify important and relevant concepts for the development. It also encourages the use of mind maps as a form of conceptualization for the ontology.
With this conceptualization, the ontology can finally formalized into the target language.

3.5.2. Design and Implementation Phase: Methodology Application and Results

Mind maps from the reference models were already at hand from the previous phase, and thus only needed to be slightly refined. Concepts from these conceptual models in the mind maps were iteratively added to the ontology in a classic top-down approach (whereas the inclusion was marked via a color scheme in the mind map, which also provides an overview of what and how many concepts from the models were actually modeled; remember that some parts of the models were excluded beforehand) and enriched by relational concepts that were required according to the ORSD.

Certain guidelines, especially naming conventions, were established (or assimilated from other sources) to assert a certain amount of quality regarding the effort to read or better understand the ontology. All names had to be written fully, no acronyms. All class names had to start with a capital letter, as well as any sub-composing names of that class (e.g. "NoseLandingGear"). All property names had to start with a small letter, contain a describing verb, and sub-composing words again start with a capital letter (e.g. "hasPart" and "isPartOf" instead of just "part" and "partOf". Also empty spaces, that could easily be achieved by setting a string in single quotes (e.g. " 'has part' ") were prohibited. Although this led to longer names, it made reading the ontology more comfortable and less confusing (and writing long names is not an issue with an auto complete feature like in Protégé), which is why such kind of conventions are commonly used for programming, where coherence and non-confusability is even more important for readability.
For classes or properties, where the need to make clarifications or explanations was given, appropriate annotation comments have been added.

A description of the final ontology can be found in chapter 4.

3.5.3. Design and Implementation Phase: Discussion

One thing that comes to mind for this phase is the lack of detail NeOn provides for this phase, compared to the other phases. It simply references other (older) methodologies, but provides nothing more on its own, which is not completely satisfactory; a full citation or small adaption would have been useful, so that the NeOn methodology could be complete in that respect. What would also have been a very valuable addition are development guidelines, in the sense of suggestions for coding conventions (that are extremely common and often can be automatically enforced in development environments for the usual programming languages) or a summary of "do"s and "don't"s, to further the quality and readability of ontologies (like "don't use spaces in names!", which are found surprisingly often).
The implementation itself can get really frustrating, when it comes to finding errors within the ontology. The reasoners usually only provide rather vague hints as to where an error might originate (with Pellet being the most informative reasoner), if they even provide any hint at all, and even then, these hints can lead in a wrong direction; in theory, Protégé provides an explanation tab in case of inconsistencies, but is has to rely on the information and explanations the reasoners provide, which is often so insufficient that the explanation tab contains no hints at all as to where the inconsistency might originate (during the course of this thesis, the explanation tab proved to be useful only on a very limited number of occasions). A good example for such an inconsistency is a falsely set disjoint clause, that can cause a lot of trouble. This can make searching for and finding errors in ontologies really difficult. So one lesson learned is to always synchronize the ontology with the reasoner in very small intervals where one can easily backtrack the last steps made.
Another thing that became apparent is that huge class hierarchies (some hundred classes with deep hierarchies) have a high performance impact on the reasoners. In an early (still unfinished) version of the ontology, each component-parameter (over 300) was modeled as a single class in a deeply hierarchic structure (with the intended usage of mapping models to the ontology in mind), which lead to a dramatic drop in performance, eventually even leading to timeouts so that the reasoner cancelled its task. Restructuring the ontology, so that there are only few general parameter classes, and adding the factual parameters through

object properties, led to a huge increase in performance (from almost one minute runtime with pellet before the change, to a runtime of five seconds after the change). Depending on the usage scenario, and the semantic content to be modeled and especially the expected size of the ontology, it might be advisable to choose another OWL dialect than OWL2-DL that can cope better with large ontologies / class hierarchies. Finally, a rather natural observation: different reasoners have different performance. On the given development computer, synchronizing the final ontology took about 16 seconds with Pellet, about 6,5 seconds with HermiT, and 1,5 seconds with FaCT++. But, as was said above, while being the slowest reasoner of those three, Pellet usually provides the most useful error messages in case of an inconsistency, so using FaCT++ is nice to see whether there is an inconsistency or not, but when an inconsistency is found, switching to Pellet until the inconsistency is resolved is advisable.

3.6. Final Ontology Evaluation

The NeOn methodology itself does not suggest one single evaluation at the end, but rather regards evaluation as an accompanying activity that is to be performed after each activity and phase. However in our case, we felt the need to at least have experts involved in some part of the development, for which a final evaluation seemed most appropriate. This is important in so far, as an ontology is a shared conceptualization of a domain; since experts were, due to time constraints, only scarcely involved in the development, this provided the opportunity to check the acceptance of the ontology by several experts.

First of all, the methodological guidelines NeOn provides for evaluations will be shortly outlined, followed by the setup and execution of the final evaluation of the aircraft design ontology.

3.6.1. Evaluation: Methodology Guidelines

NeOn provides details for evaluation in an earlier deliverable [27]. However, these are not all-encompassing guidelines, but merely suggestions. It differentiates between three dimension groups for evaluation:

- structural
- functional
- usability-related

The structural dimension applies to the pure structure of the ontology, independent from its context. In essence, the ontology is regarded as a graph (information object). As such there are, among others, the following measures one can analyze: breadth, depth, tangledness, fan-outness, differentia specifica, density, modularity, consistency, complexity.
The functional dimension relates to the intended use of the ontology, its function in a context, and essentially regards the ontology as a language (information object + conceptualization). Relevant quality measures are, for example, precision, recall, accuracy, adequacy and various qualified types of these dimensions.
Finally, the usability-related dimension regards an ontology as a meta language (information object + conceptualization + semiotic context) and covers the ease of use and the level

of annotation. Examples for quality measures are presence, completeness, and reliability.
NeOn focuses on the functional dimension for evaluation. It regards evaluation as an accompanying task for the whole development, over all phases, and also assumes expert involvement at the appropriate phases and thus also for evaluation. Since we didn't have the opportunity for extensive expert involvement during the development, this is where we adapt the methodology to our situation and add a final ontology interview for evaluation purposes at the end of the development phase. Regarding interviews, NeOn states that this does not compare an ontology to a formal corpus, but to the knowledge of an expert, who can have his own viewpoints. As such, a consensus of several experts is needed (which also goes along with the ontology being a shared conceptualization that is agreed on). NeOn also suggests using gold standards as a means for evaluating an ontology. A gold standard is a reference implementation that the ontology will be compared with. Details about the gold standards would lead too far for this chapter (especially since they are of no real use in this project, see 3.6.2); please refer to [27] for further reading.

There are also more sources of knowledge for ontology evaluation, which are interesting to take a look at because the NeOn deliverable that handles evaluation is not meant as an extensive summary. Some of these sources are, for example, [7], [18] and [44]. Mainly, they also describe various metrics and formal methods; please refer to those sources for more details on this topic.
One interesting method, that is mentioned in those sources and also in NeOn, is OntoClean[9] (see also [28]). It is not necessarily really an evaluation method, but a method to transform an ontology into an ontology that has eliminated redundancies and unnecessary concepts. Effectively, one can compare it to the normalization of classic data bases. However, it is an extremely formal method, has to be carried out mostly by hand, and itself also leaves room for debate regarding certain assignments and taggings that have to be made during the course of the method application.
Finally, here is a list of the relevant tools that support evaluation, which are mentioned in the above mentioned sources:

- OntoAnalyser
- OntoGenerator
- OntoClean in WebODE
- ONE-T
- S-OntoEval
- ODEval
- OntoManager
- AEON Automatic Evaluation of Ontologies → automatic tagging for OntoClean (that means, a first step for automation)

[9] http://www.ontoclean.org/

3.6.2. Evaluation: Methodology Application and Results

The above mentioned evaluation methods concerning various metrics usually have one huge shortcoming: the missing tool support (tools that, for example, automatically apply some metrics to the ontology and display the metric results, provide maybe a reference database for these metrics to compare the results with, possibly pointing directly to the parts in the ontology that are responsible for an unfavourable result in a metric or even provide suggestions for improvements). The tools these metrics were introduced with are outdated, i.e. were developed in the early to mid years of the last decade, and mostly not really available anymore, often the links they were published under are simply dead. Besides, since all of these tools were rather old, none of them would have had the capability to work with OWL2. Appropriate plug-ins for e.g. Protégé, that would fill this gap, are not to be found (even the current version of the NeOn toolkit is missing appropriate plug-ins), and the only metrics that are available are the simple metrics that the OWL API [42] can output. But besides all that, the proposed metrics would have only been useful if there was an appropriate measure to compare their results with, which often only comes from analyzing other (similar) ontologies that may already have proven to be useful.

A similar problem arises for the proposal of gold standards. Without an available gold standard for comparison, the propositions for using a gold standard fail to be of use. This leaves only manual expert interviews of the constructed ontology as a viable evaluation method, as well as checking the competency questions from the ontology specification against the built ontology. Thus we will use these two evaluation methods, beginning with the interviews and concluding with the coverage of the competency questions.

Evaluation Interview: Approach

There were a total of two interviews, each with a time frame of one hour. This arose from the fact that the available experts were deeply involved in current projects at Bauhaus Luftfahrt which drastically restricted their available time for other things, in this case this ontology project. The focus for these interviews lay in the question whether a component, with respect to a traditional passenger aircraft like an A320, is modeled correctly, i.e in the correctness of the given statements. Out of scope was the completeness, since the pretense in this thesis was not to provide a model that can represent any aircraft one can possibly think of, as well as not to provide a model that models an aircraft in every last detail up to the single bolts and screws used in construction. The detail only has a depth “as needed”, which in this case was determined by the given reference models. This aspect of the modelling detail of the knowledge, only as detailed as needed, is, as has already been stated in chapters 1 and 2, a central aspect of OWL ontologies; if further detail is needed, the appropriate knowledge can be modeled into the ontology at any time, without invalidating any statements that were made prior to that, thanks to the open world assumption. Thus, completeness was disregarded in the expert interviews.

The interviews were intended to begin with an introduction of about five minutes, in which a very basic intro about how OWL ontologies work is given, as well as an overview

about the structure of the built ontology, the composition and partition into a structure describing part and a part that contains the parametrical content that details the structural components.

The next 30-40 minutes were intended to be used to evaluate the structural description of an aircraft in the ontology. The modeled concepts were put in an order regarding their complexity, so that the most complex and thus error prone components were to be reviewed at the beginning, which left the possibility to skip some of the least complex components if time ran out. Each component was to be presented in natural language, first by giving a complete overview of the component and its sub-components, which was followed by a statement by statement eradication. For this purpose, all relevant classes and their composition were printed out on paper, and presented one at a time, where each statement that constitutes that class was presented and explained, for which the interviewed expert could either assert the correctness of the statement or declare an error.

The remaining time after the structure evaluation, which was intended to be between 15 to 25 minutes, should be used for the evaluation of the parameters that are associated with their appropriate components. The focus here also lay in correctness rather than completeness. Each component has a set of parameters associated with itself; the order these parameter sets were to be evaluated in was the same as the order of the evaluation structural part, which provides a complete review for the most complex (and thus most error prone) components.

Evaluation Interview: Results

Followinf the outline of the interviews, the actual results of the two interview sessions will now be described.

In the first interview session, we were able to cover the complete structural description, as well as the parametrical descriptions of aircraft components. All in all, there were 343 superclass-statements (a statement entry in the superclass-description part of a class in Protégé) to be checked, 314 of them were checked, 29 had to be omitted due to time constraints. The interview lasted 75 minutes, which is 15 minutes longer than was initially planned, but the extra 15 minutes were a welcome addition.
Several kinds of errors were identified during the interview. One type of error was using a wrong name for a concept, since this concept is actually named otherwise or the name refers to another concept. Another error type, weaker than the one just mentioned, was an ambiguous naming of a concept that may lead to confusion since the name may have more than just one meaning, or may imply other factors that are not part of the modeled concept. The third error type also refers to naming, in that a concept is modeled more than once, but each time under a different name. The fourth error type was found in the modeled parameters. For some parameters, a reference is needed to properly interpret them; parameters where such a reference is missing are, in fact, not useful. The last error type that was encountered was technically a follow-up error from one of the above. Due to naming misconceptions and wrong assumptions, it became obvious that some parameters,

Type	Code	Count
Used name is **W**rong	W	4
Used name or modeled concept is **A**mbiguous	A	11
Double entry (concept already available under another name)	D	8
Reference for describing parameter is missing	R	3
Missing parameter (due to wrong assumptions)	M	2
Total		28

Table 3.2.: Overview about identified Errors

that were intended to be in the ontology, were in fact missing. Table 3.2 provides a summary of all those error types.

Here follow some examples for the error types from table 3.2:

- W: The "horizontal stabilizer" was called "tailplane", which is in fact a more general term that describes any stabilizing plane at the tail of an aircraft
- A: The parameter "isDescribedByRatioVolume some DimensionlessParameter" is somewhat vague regarding its name, since it is a ratio, but is calculated by multiplying an area with a length, which one would assume to be a volume
- D: Some components had a parameter "centerline chord", as well as a "mean aerodynamic chord", which is essentially the same
- R: The parameter "isDescribedByYKink some DistanceParameter" for a wing is as it is not clearly defined, since it is not clear and definite from where (and in what direction) this distance resp. length is measured
- M: due to the misconception of a parameter "lever arm" for the vertical stabilizer, its "span" was omitted

The second interview session went pretty different, which was expected to some degree, but not to the extent that was actually the case. Hence the results can not be summarized in the same way as in the first interview session. The interview was not as focused as the first one and there were more deviations, which was the one hand due to the good intentions of the interviewed expert to be very thorough and exhaustive, which in combination with him being a very knowledgeable expert produced a lot of detours from the planned route for the interview, but on the other hand it was also due to not defining the boundaries for the interview clearly enough. Nevertheless, this interview also produced worthwhile results. One fact that became clear in the interview, was that there is a distinction in the nomenclature between British English and American English. Where an American engineer would call the rear end of an aircraft with its stabilizing surfaces simply "tail", a British engineer might call it "empennage" instead. Another fact that became clear was that it is difficult to describe all possible configurations of a concept, and that there have to be clear definitions and boundaries to delimit a concept from what it does not encompass. Furthermore, details become more and more complicated as one gets more and more detailed into structural concepts; as a rather simple example, regard the landing gear compartment within the fuselage that stows away the landing gear when it is retracted: how do you define the fuselage

(shape) exactly? Where does this compartment really belong to? Does it have a stronger attachment to the fuselage, or to the landing gear? There are more things to consider when the detail level rises, and there are also more inter-dependencies. One last finding from this interview session was the fact that there are some terms that are an established term for itself in the area of aircraft design which were not properly identified as such. A propulsion group for example is such an established term, encompassing all power plants of an aircraft with all their sub-components, i.e. it is a set of components and sub-components that may or may not be directly connected to each other and perform in the task of propulsion. Similarly, the word "system" is generally used for an integrated component and all parts it encompasses, e.g. a single landing gear system (which, in the case of an A320, would consist of a strut, a damper, an axle and two wheels, among other parts). This distinction between a "group" and a "system" should be kept clear for all further possible developments. What finally was also made clear is that this ontology, with its focus mainly on the structure of an aircraft, just regards one aspect of many that are relevant for an aircraft.
All in all, this second interview brought some clarity regarding the nomenclature, but in general mostly brought some basic insights and ideas as to where to direct possible future developments.

Evaluation of Competency Questions

Besides these two interviews, there were also the competency questions from the ORSD (see A.2) to evaluate the ontology. From the competency questions in the ORSD, the final ontology is able to answer CQ01, CQ02, CQ03b, CQ03c, CQ04 (with respect to the actually modeled parameters), CQ05, CQ07 and CQ08. This makes a total of 8/14 answerable competency questions, with 5/5 answered questions with priority 1 (most important), 3/5 answered questions from priority 2, and 0/4 answered questions from priority 3 (least important).

3.6.3. Evaluation: Discussion

Something that was clear already before the evaluation: earlier and more expert involvement would have been of great benefit. Thus, especially in the second interview, it became apparent that there was too little time for too much (unchanneled) knowledge from the expert; the expert tried to convey as much of this knowledge as he could (with good intentions), which was a little overwhelming and led to a lot of jumping around between ontology concepts, as well as some (albeit informative, but at this point unnecessary) excursions. Earlier and more expert involvement would have led to a better focus of what the ontology is actually intended to be used for, and thus led to more productive interviews. Also having a prototype knowledge application at hand that demonstrated the use of the ontology would have also benefited that cause instead of having just a domain ontology, and it might also have led to a greater acceptance on the side of the experts, enabling them to better see a profit in the endeavor.
Another important insight came from the second interview. With the intended purpose of eventually serving as a tool to enable parameter exchange between different aircraft design tools, components or concepts will become massively over-defined (one tool describes a con-

cept with parameters A, B and C, whereas another tool described the same concept with parameters C, D and E, which leads to the fact that a unified model needs to include all five parameters A, B, C, D, and E, thus allowing to describe a component with more than just the smallest number of necessary parameters). It should be investigated whether this really poses a problem, and if it does, how to deal with this.
Regarding the interview approach itself there is room for improvement, in order to get a better focus and not to stray from the intended path. For once, the interviewer needs to be prepared to cut-off discussions where they are not needed or when they have been informative enough for the purpose at hand. To reduce the likelyhood for unnecessary distractions or discussions, the interview should also contain details about the assumptions and boundaries of the modeled ontology and its concepts, once at the start of the interview for the whole ontology, and later on for each relevant concept. This does cost some of the valuable interview time, but especially consecutive interviews with the same expert would greatly benefit from such a course of action, and also inexperienced interviewers would certainly profit from this stronger guidance for the interviewed expert.

4. Aircraft Design Ontology

Having demonstrated how the development of the aircraft design ontology via application of the NeOn methodology was carried out, this chapter will describe the first version of the aircraft design ontology in detail. We will take a look at the class hierarchy, show how object-properties relate these classes to one another, and see how the chosen unit ontology is used in this project. Remember that the ontology focuses on a structural description of an aircraft on a rather high level. Also, the ontology does not claim to be able to model all different aspects of aircraft one can possibly think of. Its main intention as of now is to be able to model standard fixed wing passenger aircrafts, for example an Airbus A320 or a Boeing 747, on a basic level; there are a lot of other aircraft kinds and configuration this ontology currently probably cannot model adequately. On the first level of the class hierarchy are four basic classes: *"Aircraft"*, *"AircraftSubComponent"*, *"AircraftAspect"* and *"AircraftParameter"*. The following subsections describe these different parts of the aircraft design ontology.

4.1. Aircraft Structure

The structure of an aircraft is modeled in the class *"Aircraft"*, using the sub-classes of *"AircraftSubComponent"* as building blocks. This is a pattern that continues within the aircraft sub components: an object is built up of object-sub-components that are comprised under a class "ObjectSubComponent" that exists on the same level as the class "Object" in the general class hierarchy. This pattern is repeated, until the necessary level of depth (specified by the given reference models) has been reached. These components are built up via a *"hasPart"* (and its inverse *"isPartOf"*) object property, forming a classic part-of-relationship. The transitive *"hasPart"* object property is specialized by an object property *"hasDirectPart"*, which is used to declare a direct has-part relationship between two components (which is inverse functional, asymmetric and irreflexive). This, however, is still only a generalized object property, which is again specialized for all relevant direct-part relationships between components. These specialized object properties finally have a concrete domain and range defined, which specify what components take part in this specialized relationship (take, for example, the object property *"hasCabin"*, which relates a *"Fuselage"* with a *"Cabin"*). Figure 4.1 shows the appropriate classes and object properties, and Figure 4.2 shows, for example, how these are put together in the class *"Aircraft"*.

There is another object property that is used to define an aircraft's physical structure. *"isConnectedTo"* is a symmetric relationship, that denotes a connection between two objects. It is, for example, used to define where the power plants of the aircraft reside: they can be connected to the wing (as seen on an Airbus A320), the fuselage (as often seen on business jets like a Learjet 45), or the fin (as seen on a McDonnell Douglas DC-10).

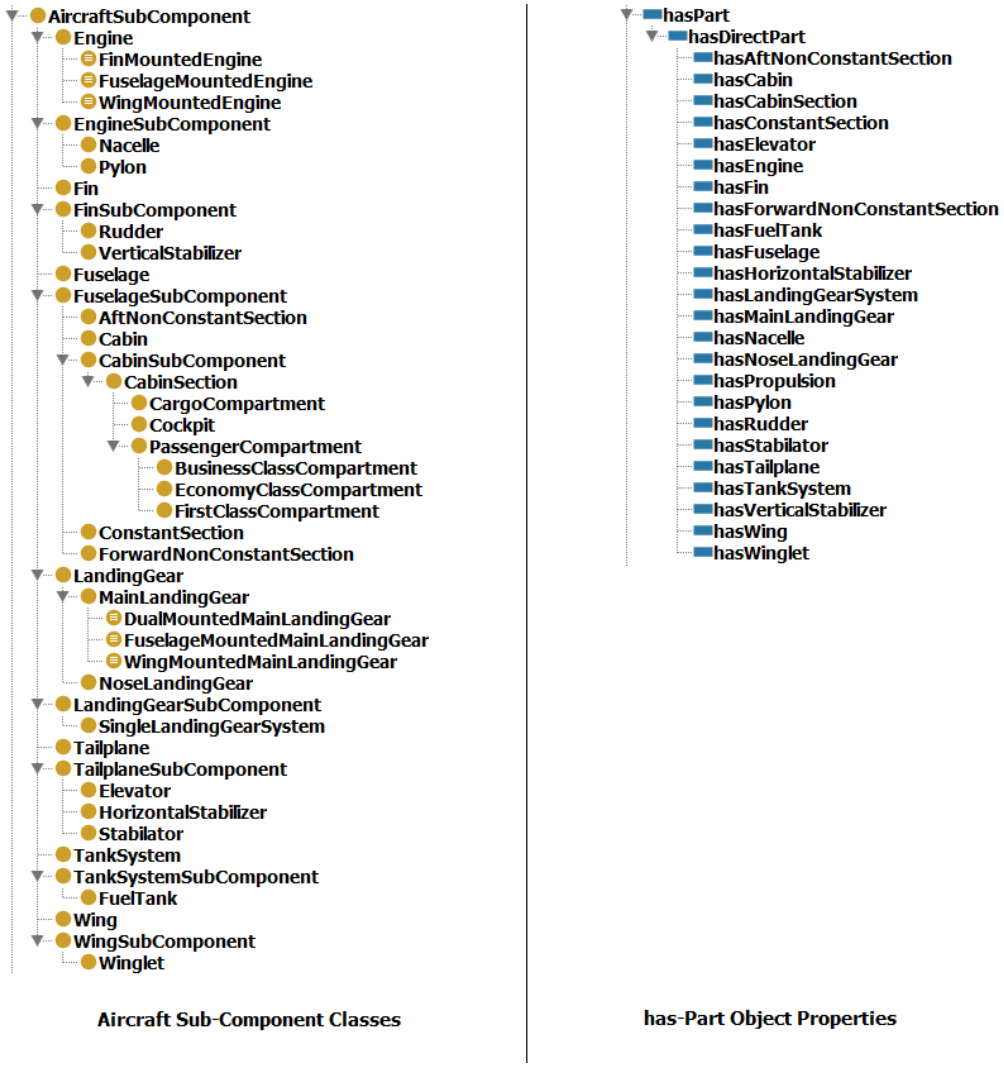


Figure 4.1.: Structural classes and object properties

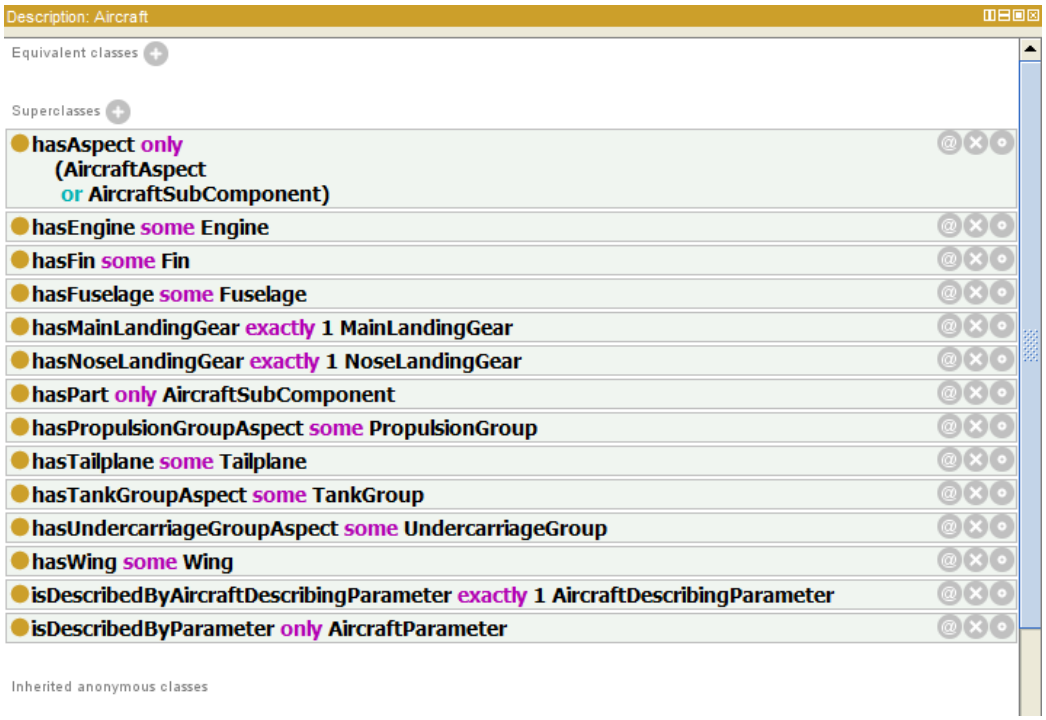


Figure 4.2.: Build-up of the class *"Aircraft"*

4.2. Aircraft Aspects

Aside from viewing an aircraft as a whole, there are certain aspects of an aircraft that are often regarded in more detail, but that also do not form a whole interconnected component or system, but are rather a group of components or systems, not directly connected to one another, often spatially separated. These different aspects of an aircraft, one could also call them views, are consolidated under the class *"AircraftAspect"*. An example for a specialized *"AircraftAspect"* is the *"PropulsionGroup"*, which encompasses all power plants (engines) of an aircraft and defines the propulsion boundary characteristics for an aircraft. These aspects are incorporated and defined via the *"hasAspect"* object property, that is, analogous to the *"hasPart"* object property, transitive, and again specialized by a *"hasDirectAspect"* object property, that is itself also analogously specialized into object properties that define appropriate domains and ranges for the relationships. There is just one more distinction as to what a partner in the relationship is. An aspect can be comprised of other aspects, or of an aircraft sub-component (which can then be called a member of that aspect); when this transitive relationship is, in its whole, interpreted as a tree or graph, the leaves will always be these kind of members. So, for example, an aircraft has an aspect *"UndercarriageGroup"*, which is defined in the class *"Aircraft"* via the object property *"hasUndercarriageGroupAspect"*. The *"UndercarriageGroup"* itself comprises all objects belonging to the undercarriage of an aircraft, in our case nose- and main-landing-gear. These are incorporated within the *"UndercarriageGroup"* via the specialized aspect object properties *"hasNoseLandingGearMember"* and *"hasMainLandingGearMember"*. Figure 4.3 shows the classes and object properties pertaining to these aspects.

Figure 4.3.: Aspect Classes and Object Properties

4.3. Aircraft Parameters

What now remains to be shown is how parameters for the components are modeled in this ontology; parameters are, for example, the span of a wing, or the length of a fuselage. For that, let's first take a look at the ontology that was chosen to be used for modelling units. As stated in chapter 3.3, the Library for Quantity Kinds and Units (QU) (see Table 3.1) was chosen for this project. It defines types of quantity kinds and units as classes, and implements concrete instances as individuals. These unit individuals have a unit defined via a data property, and are either a reference unit (see SI units), or derived from a reference unit. If they are a derived unit, they have a reference unit assigned via an object property (e.g. "metre" as reference unit for "millimetre", as well as maybe a designated prefix (like "milli" for "millimetre"), and via data properties these units also have conversion values to convert them to their reference unit, as well as their unit symbols as a string. If a unit is a reference unit itself, instead of a reference it is assigned a quantity kind (for example "distance" in case of a metre). Thus, units become comparable and distinguishable, concerning values or quantities. Figure 4.4 gives a graphical overview.

What this unit ontology is missing is the combination of a unit together with a value, but this is easily rectified, as will be shown shortly. Parameters for components, aspects or other artifacts are modeled under the class *"AircraftParameter"*. Its sub-class *"SingleAircraftParameter"* models all primitive parameters, where a simple unit kind is assigned a numerical value, using the object property *"unit"* and the data type property *"numericalValue"* from the unit ontology to bring both together. So, for example, a *"MassParameter"* has a double as numerical value, and any kind of *"MassUnit"* as unit. These simple parameters are then grouped together into compound parameters (which are thus in fact a parameter set), each describing a certain component, aspect or artifact, which are then attached to the concept they describe. Thus, the parameters are separated from the structural description in the component classes, making maintaining these classes much easier, since each component is normally described by a lot of single parameters, which would otherwise clutter the class description. Thus, the parameters are now incorporated via a single object property, that assigns a compound parameter to a component.
The object properties that are used to form the aircraft parameters are found as specialization of *"isDescribedByParameter"*. There are two direct specializations. The first one,

[1] http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#Units_of_measurement_and_quantity_ontologies

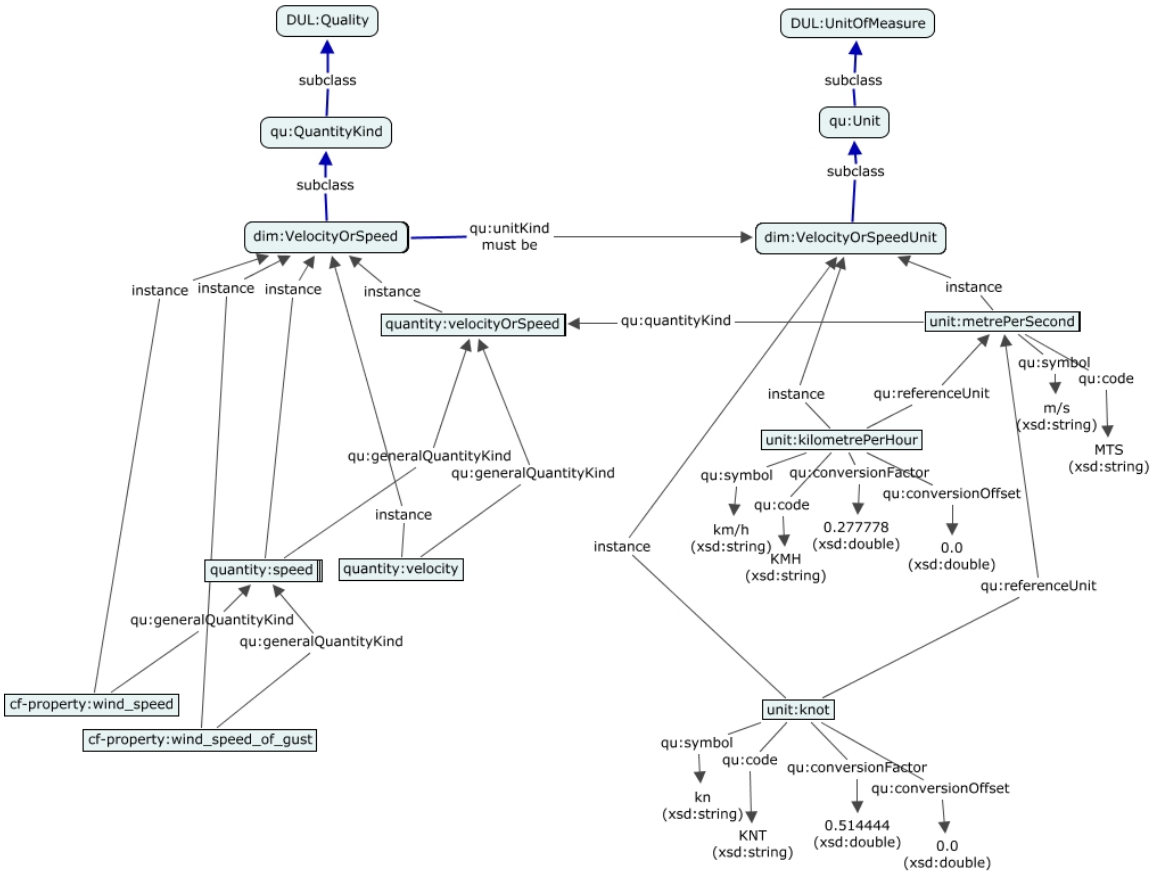


Figure 4.4.: QU Ontology Overview[1]

"isDescribedByCompoundParameter" is used to assign the compound parameter classes to their appropriate concepts (with domain and range set accordingly), e.g. in the class *"Wing"*, the object property *"isDescribedByWingDescribingParameter"* assigns a *"WingDescribingParameter"* to it. As previously stated, these compound parameters are comprised of single parameters, and the assignment is done via specializations of the object property *"isDescribedBySingleParameter"*. With the intended usage of the ontology in mind (the mapping of data exchange variables from various tools within the aircraft design process to the ontology as common data model), here each single parameter is assigned though a named specialization of *"isDescribedBySingleParameter"* which incorporate the appropriate *"SingleAircraftParameter"*s. These distinctively named object properties are needed in order to differentiate between two parameter of the same kind for a component. For example, a wing is described by several chords, which are in essence distances, but nevertheless not the same, as the chord at the root of a wing certainly has another value than the chord at the wing tip (note that this mapping problem could have also been solved in other fashions). So, as an example, the class *"WingDescribingParameter"* bundles all single parameters that can be used to describe a wing, among others e.g. the reference area, which is incorporated into the class via the object property *"isDescribedByReferenceArea"* that has *"AreaParameter"* as range.
Figure 4.5 shows an example for one aircraft compound parameter, together with a single parameter that is used in this parameter set. Furthermore, Figure 4.6 shows the parameter classes under *"AircraftParameter"*, and Figure 4.7 shows the object properties to relate the parameters to their appropriate classes.

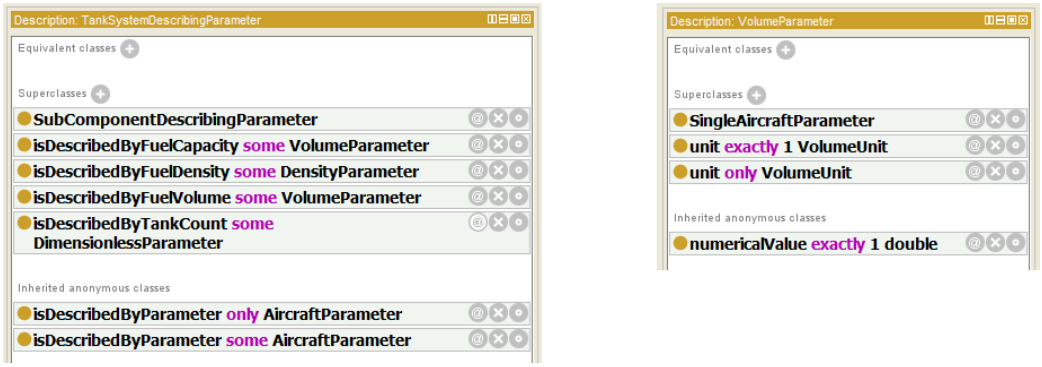


Figure 4.5.: Examples for a compound parameter and a single parameter

4.4. Ontology Application

In conclusion, this section describes how the concepts within the aircraft design ontology are instantiated by individuals. This is done via a straightforward example, where one specific cut of an aircraft is modeled via individuals, from the aircraft as a whole to a specific parameter in a specific component. This example will not contain a "complete" aircraft (complete meaning that it uses all capabilities that this ontology provides), but due to the open world assumption, this does not lead to an inconsistent ontology; as long as those parts that are not included are not explicitly excluded, this just states that there is not yet any knowledge

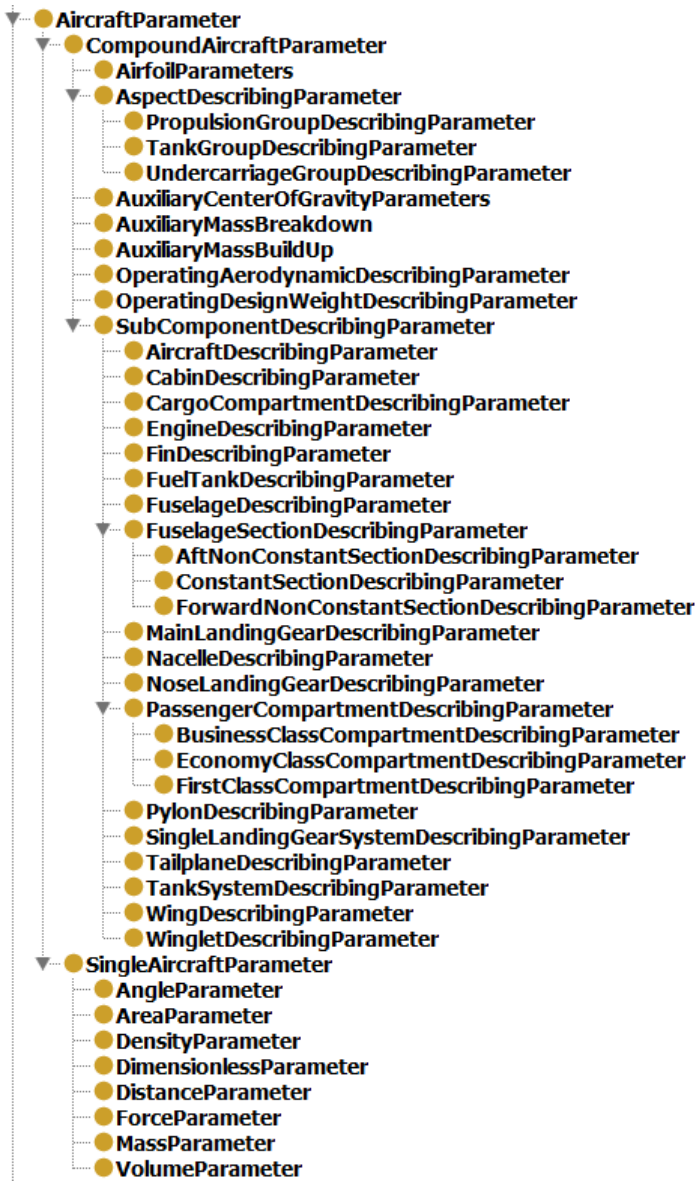


Figure 4.6.: Sub-Classes of *"AircraftParameter"*

Figure 4.7.: Object Properties for assigning parameters (excerpt; not all parameters are fully expanded)

about them, but this knowledge may always be added as soon as it is gained. This is also the explanation for the fact, that using an ontology for checking integrity constraints is only viable in a limited way, since "lower boundaries" are usually hard to check (for example, an aircraft individual that doesn't have a wing as its part is still a valid aircraft individual, although the definition of an aircraft states that it must have at least one wing; this is true as long as there is no explicit statement that this aircraft individual doesn't have any wings at all, which would produce an inconsistency; without such an explicit exclusion, however, it is always possible to add a wing to the aircraft, as soon as one knows the details).

The addition of individuals in Protégé is done via the Individuals-tab. First, select the class *"Aircraft"* and add an individual as member, e.g. "ExampleA320". Then, select the aircraft sub component class *"Wing"* and also add an individual here, e.g. "ExampleWing". To connect the wing to the aircraft, add an object property assertion *"isWingOf"* to the wing and select the aircraft individual "ExampleA320" as target. Now, the "ExampleWing" is a part of the "ExampleA320". Alternatively, an equivalent assertion could have been made with an object property assertion of *"hasWing"* for the "ExampleA320" individual, which would then have had "ExampleWing" as target.

To be able to add parameters to the wing, an instance of the class *"WingDescribingParameter"* is needed, call it "ExampleWingDescribingParameter". To assign this individual to the wing, select the "ExampleWing", and add an object property assertion *"isDescribedByWingDescribingParameter"* with the just created "ExampleWingDescribingParameter" as target. Now, this still empty parameter set is associated to the aircraft wing.

To complete the example, a parameter is added to the wing. Select the class *"AreaParameter"* and add an individual "ExampleWingReferenceArea". Add a data property assertion *"numericalValue"* with a double value of 122.6 as target. Additionally, add an object property assertion *"unit"* and select the individual " 'square metre' " as its target (the previous datatype property and this object property originate from the imported unit ontology). To associate this parameter with the "ExampleWing", select the "ExampleWingDescribingParameter" and add an object property assertion *"isDescribedByReferenceArea"* with the target "ExampleWingReferenceArea".

As an interesting alternative, don't create the "ExampleWing" as a member of *"Wing"*, but create it as a direct member of *"AircraftSubComponent"* or even *"Thing"*, with the rest of the associations as described above. After running the reasoner, one can now see that "ExampleWing" has been classified as a *"Wing"* (which is deduced from the domain and range of the *"hasWing"* object property).

This concludes the example, which has demonstrated how modeled concepts within the aircraft design ontology can be utilized and instantiated.

5. Conclusion and Outlook

This final chapter concludes the thesis and provides some comments and ideas for possible follow-up developments.

First of all, the built ontology fulfills the expectation to be able to model the physical structure of a typical passenger aircraft like an Airbus A320 (on a high abstraction level). This abstraction level can, at any time, be extended with more details as the need arises. However, as said before, structural concerns are but one of several views that are relevant for an aircraft. Further development iterations could add behavioural aspects of an aircraft, together with aircraft states, to be able to model various aircraft missions (a mission is a flight of an aircraft beginning from the start at the airport right after boarding through the different climb, flight and landing phases until the disembarking begins) under different conditions.

Also, for further development iterations, more expert involvement, especially earlier on in the development, would greatly benefit the ontology development. An ontology is a shared conceptualization that is agreed on by all participants in the development, thus more expert involvement can only enhance the quality of the ontology. Also, it would enable those experts to set the focus of the development more appropriate to their current needs, increasing their acceptance of the developed ontology, and maybe also helping to set the corresponding focus for the knowledge application that will make use of the ontology.
To this same purpose, further development should always be done in conjunction with the development of said knowledge application, as is suggested by many ontology development methodologies. For this thesis, this was not within the scope, since the development of this ontology was decoupled from the knowledge application, which was (and still is) being developed by other staff members. But the experience gained during this thesis suggest that a stronger focus of an ontology for (or together with) a specific (software) application is to be recommended. Ideally, the development of the application and the ontology would be embedded in an agile process with short iterations, where for each iteration a (prototype) application is available that makes full use of the appropriate ontology version. With an application at hand, the development could be directed in a way that most benefits the applications future users, possibly providing a more appropriate solution for the model data exchange problem mentioned in chapter 1.

Regarding the development itself, it might have been an interesting solution to automatically transform the given data models into an ontology (taking care of a large portion of the development). However, model transformation is not a simple task. In our case, we would have wanted to convert UML in XMI format into an ontology. However, as [17] states, such a general conversion is not easily possible, since often different UML modelling tools depend on a manufacturer dependent dialect, which poses a problem for a general solution.

For early UML versions, there was an example conversion available for the ATL framework. However, that was no general solution, as the developers were unable to agree upon the conversion for more specific UML aspects, thus it remained incomplete. Furthermore, employing this example is rather difficult due do certain dependencies of various needed plug-ins and incompatibilities between different plug-in versions. The most viable way of making use of ATL would be installing a current version, and developing a conversion oneself. This, however, is no simple task, and would probably be fitting for another complete thesis.

Concerning the ontology evaluation, having access to tools that enable the application of some proposed metrics, as well as some reference data for some common ontologies, would provide a more thorough and especially more formal result. Developing appropriate software that can also cope with the current OWL 2 version and its dialects would certainly be a rewarding endeavour.

One purpose for which an ontology is not as viable as one might hope, is the purpose of checking integrity constraints. The reason for this is the open world assumption, which states that knowledge that is not explicity given (or implicitly inferable from the explicilty given knowledge) is simply currently not known, but it might be possible to gain that knowledge in the future and add it then. This result "not known" is a fundamental difference from classic data models, where referring to not explicitly given knowledge normally results in a negative answer. This behaviour leads to some unexpected results until one has become accustomed to this concept, as, for instance, the example from chapter 4.4 does not lead to an inconsistent ontology, although the instance model is far from complete.
There are, however, endeavours to somehow rectify this problem, as the producers of reasoners have noticed according demands by the users of their software. Pellet ICV[1] is an approach that makes some restrictions on the open world assumption (and the non-unique naming assumption too) to enable a more intuitive checking of integrity constraints. The example from chapter 4.4 would then result in an inconsistent ontology, since the modeled aircraft instance is missing a lot of required components. However, Pellet ICV is not yet released, and there are currently also no preview versions available for download.
Other ways of trying to enforce integrity constraints could involve the use of SPARQL queries, or SPIN[2] rules (which require the SPIN API[3] from TopBraid).

Finally, one aspect that remains open, as further development leads to bigger and bigger ontology versions (overall, all imports considered, since further development would also probably lead to more distinction into separate ontologies that depend on one another; an ontology network, as NeON states), is the performance of the reasoners. In chapter 3.5 it was already mentioned that certain modelling concepts can have an impact on reasoner performance. It remains to be seen, whether using OWL 2 DL with appropriate reasoners remains viable for a growing ontology with lots of closely connected concepts, or if it might be necessary to switch to another OWL 2 dialect, which, however, would also lead to a

[1] http://clarkparsia.com/pellet/icv/
[2] http://spinrdf.org/
[3] http://topbraid.org/spin/api/

loss of some knowledge, since those other dialects pose further restrictions on usable OWL statements. But, depending on the circumstances, this might be an acceptable tradeoff.

Appendix

A. Detailed Descriptions

This appendix contains further information that was too extensive for the main text. On the following pages, more comparisons for ontology development methodologies can be found, as well as the ontology requirements specification document, and finally the initial project plan for the ontology development project.

A.1. More Ontology Development Methodology Comparisons

Feature			Cyc	Usdhold and King	Grüninger and Fox	KACTUS	METHONTOLOGY	SENSUS	On-To-Knowledge
Project management processes	Project initiation		Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Proposed
	Project monitoring and control		Not proposed	Not proposed	Not proposed	Not proposed	Proposed	Not proposed	Proposed
	Ontology quality management		Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Proposed
Ontology development-oriented processes	Pre-development processes	Concept exploration	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Proposed
		System allocation	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed
	Development processes	Requirements	Not proposed	Proposed	Proposed	Proposed	Described in detail	Proposed	Proposed
		Design	Not proposed	Not proposed	Described	Described	Described in detail	Not proposed	Proposed
		Implementation	Proposed	Proposed	Described	Proposed	Described in detail	Described	Proposed
	Post-development processes	Installation	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed
		Operation	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed
		Support	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed
		Maintenance	Not proposed	Not proposed	Not proposed	Not proposed	Proposed	Not proposed	Proposed
		Retirement	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed
Integral processes	Knowledge acquisition		Proposed	Proposed	Proposed	Not proposed	Described in detail	Not proposed	Proposed
	Verification and validation		Not proposed	Proposed	Proposed	Not proposed	Described in detail	Not proposed	Proposed
	Ontology configuration management		Not proposed	Not proposed	Not proposed	Not proposed	Described in detail	Not proposed	Proposed
	Documentation		Proposed	Proposed	Proposed	Not proposed	Described in detail	Not proposed	Proposed
	Training		Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed

Figure A.1.: Ontology development methodology comparison from [8]

Feature			Cyc	Uschold & King	Grüniger & Fox	KACTUS	METHONTOLOGY	SENSUS	On-To-Knowledge	HCOME
Ontology management activities	Scheduling		NP	NP	NP	NP	Proposed	NP	Described	NP
	Control		NP	NP	NP	NP	Proposed	NP	Described	NP
	Quality assurance		NP	NP	NP	NP	NP	NP	Described	NP
Ontology development oriented activities	Pre development processes	Environment study	NP	NP	NP	NP	NP	NP	Proposed	NP
		Feasibility study	NP	NP	NP	NP	NP	NP	Described	NP
	Development processes	Specification	NP	Proposed	Described in detail	Proposed	Described in detail	Proposed	Described in detail	Proposed
		Conceptualization	NP	NP	Described in detail	Proposed	Described in detail	NP	Proposed	Proposed
		Formalization	NP	NP	Described in detail	Described	Described	NP	Described	Proposed
		Implementation	Proposed	Proposed	Described	Proposed	Described in detail	Described	Described	Proposed
	Post development processes	Maintenance	NP	NP	NP	NP	Proposed	NP	Proposed	Described
		Use	NP	NP	NP	NP	NP	NP	Proposed	Described
		Evolution	NP	NP	NP	NP	NP	NP	NP	Proposed
Ontology support activities	Knowledge acquisition		Proposed	Proposed	Proposed	NP	Described in detail	NP	Described	NP
	■ Distributed know. acquisition									Proposed
	■ Onto. Learning integration									
	Evaluation		NP	Proposed	Described in detail	NP	Described in detail	NP	Proposed	NP
	Integration		Proposed	Proposed	Proposed	Proposed	Proposed	NP	Proposed	NP
	Configuration management		NP	NP	NP	NP	Described	NP	Described	NP
	Documentation		Proposed	Proposed	Proposed	NP	Described in detail	NP	Described	NP
	■ Results									
	■ Decision process									Proposed
	Merging and Alignment		NP	NP	NP	NP	NP	NP	NP	Proposed

Figure A.2.: Ontology development methodology comparison from [32]

A comparison of different processes with respect to the IEEE 1074–1995 standard

IEEE 1074-1995 standard processes			Uschold and King	Grüninger and Fox	METHONTOLOGY	On-To-Knowledge	UPON
Project management processes	Project initiation		–	P	–	+	–
	Monitoring and control		–	P	P	+	–
	Quality management		–	–	P	–	–
Ontology development-oriented processes	Pre-development	Environment study	–	–	–	–	P
		Feasibility study	–	–	–	+	–
	Development	Requirements	P	+	+	+	+
		Design	–	+	+	P	+
		Implementation	+	+	+	+	+
	Post-development	Installation	–	–	–	–	–
		Operation	–	–	–	–	–
		Support	–	–	–	–	–
		Maintenance	–	–	P	P	P
		Retirement	–	–	–	–	–
Integral processes	Knowledge acquisition		+	P	+	–	+
	Evaluation		+	–	+	+	+
	Configuration management		–	–	+	–	–
	Documentation		+	–	P	–	+
	Training		–	–	–	–	P

–, unsupported; +, supported; P, partially supported.

Figure A.3.: Ontology development methodology comparison from [10]

	METHONTOLOGY	On-To-Knowledge	DILIGENT	NeOn Methodology (Version1)
NeOn Dimensions				
Collaboration	Not mentioned	Not mentioned	Treated	*Mentioned, but not treated in detail*
Context	Not mentioned	Not mentioned	Not mentioned	*Not mentioned*
Dynamic	Mentioned, but not treated	Mentioned, but not treated	Mentioned, but not treated	*Not mentioned*
Detailed Guidelines for Processes and Activities				
Ontology Specification	Not provided Only Competency Questions are proposed	Not provided Only Competency Questions are proposed	Not provided In fact, this activity is not proposed by the methodology	*Provided*
Reusing Non Ontological Resources	Not provided, neither explicitly mentioned	Not provided, neither explicitly mentioned	Not provided, neither explicitly mentioned	*Provided*
Reengineering Non Ontological Resources	Not provided, neither explicitly mentioned	Not provided, neither explicitly mentioned	Not provided, neither explicitly mentioned	*Provided in a preliminar manner*
Reusing Ontologies	Not provided Only a list of activities to be carried out is proposed	Not provided Only recommendation of identifying ontologies to be reused is given	Not provided, neither explicitly mentioned	*Provided*
Reusing Ontology Design Patterns	Not provided, neither explicitly mentioned	Not provided, neither explicitly mentioned	Not provided, neither explicitly mentioned	*Provided in a preliminar manner*
Audience				
Targeted to Software Developers and Ontology Practitioners	Targeted to ontolgy engineers and researchers	Not targeted to ontolgy engineers and researchers	Intended to domain experts and users	*Targeted to ontolgy engineers and researchers*

Figure A.4.: Ontology development methodology comparison from [36]

A.2. Aircraft Design Ontology Requirements Specification Document

This page and the next one shows the aircraft design ontology requirements specification document (ORSD) that was created during the application of the NeOn methodology.

Ontology Requirements Specification Document	
1	**Purpose**
	"Provide a reference ontology for aircraft design which can be used as a semantic reference for models from various disciplines in aircraft design."
2	**Scope**
	"The focus of the ontology is the domain of conceptual aircraft design. Its focus lies primarily in the aircraft structure and only secondary on the aircraft behaviour. The level of granularity is given by certain reference models whose contents have to be integrated as well as the competency questions and thus identified terms."
3	**Level of Formality**
	"The ontology has to be implemented in OWL2 (OWL-DL) [and must be usable in Protégé 4.x and with the JENA Framework]."
4	**Intended Users**
	- CDT developers: focus on model integration (intelligent user interface) *- Aircraft design engineers: focus on semantic annotation of aircraft models*
5	**Intended Uses**
	1. *semantic annotation; connection of own models to concepts in the ontology* 2. *context retrieval; identifying sub- and super structures of a model element (i.e "part-of")* 3. *semantic model verification: is the modeled content consistent?*

Table A.1.: Aircraft design ontology requirements specification document (part 1/2)

Ontology Requirements Specification Document (continued)			
6	**Example Competency Questions**		
	Code	Prio	Question
			Structure
	CQ01	1	*What is the structure of a model component (from the given models)?*
	CQ02	1	*Of what part(s) is part X a part of?*
	CQ03a	2	*What is the wing configuration of the airplane?*
	CQ03b	2	*What is the undercarriage configuration of the airplane?*
	CQ03c	2	*What is the engine configuration of the airplane?*
	CQ04	1	*What is the [(structural) parameter] of [a (structural) component]?*
	CQ05	1	*To which component is [another component] attached? (geometry details do not have to be included, i.e. a coordinate system for attachment points etc.)*
	CQ06	2	*What parameters are relevant for aerodynamics?*
			Units
	CQ07	1	*In what unit is a parameter measured? (→ model units!)*
	CQ08	2	*Is a certain length in foot equal to another length in meter? (→ conversion factors!)*
	CQ09	3	*On which quantities does the speed of sound depend on? (→ medium (i.e. air), pressure, temperature*
			Behaviour
	CQ10	3	*What is the operational radius of the airplane?*
	CQ11	3	*On which parameters does the operational radius depend on?*
	CQ12	3	*Can an airplane go directly from take-off to cruise?*
7	**Pre-Glossary of Terms**		
	"N/A" (see chapter 3.2)		

Table A.2.: Aircraft design ontology requirements specification document (part 2/2)

A.3. Aircraft Design Ontology Project Plan

The initial project plan for the aircraft ontology design project can be found on the following pages.

ID	Task Mode	Task Name	Duration	Start	Finish	Notes	Predecessors	Actual Start
1		*Project Iteration 1 Start*	*0 days*	*Tue 20.12.11*	*Tue 20.12.11*			*NA*
2		**Iteration 1: Aircraft Structure & Behaviour**	**44 days**	**Tue 20.12.11**	**Fri 17.02.12**	**1st priority: structure**	**1**	**Tue 20.12.11**
3		**Initiation Phase**	**7 days**	**Tue 20.12.11**	**Wed 28.12.11**			**Tue 20.12.11**
4		Ontology Environment Study	0 days	Tue 20.12.11	Tue 20.12.11	not needed; decision is already made	1	NA
5		Ontology Feasibility Study	0 days	Tue 20.12.11	Tue 20.12.11	not needed; decision is already made	1	NA
6		Ontology Requirements Specification	7 days	Tue 20.12.11	Wed 28.12.11		4;5	Tue 20.12.11
7		Scheduling	0 days	Wed 28.12.11	Wed 28.12.11	already done before for rough estimates	6	NA
8		**Reuse Phase**	**6 days**	**Wed 28.12.11**	**Thu 05.01.12**		**3**	**NA**
9		Ontology Search	0 days	Wed 28.12.11	Wed 28.12.11	already done: qudt, muo or w3c for units and measures	7	NA
10		Ontology Assessment	2 days	Thu 29.12.11	Fri 30.12.11		9	NA
11		Ontology Comparison	2 days	Mon 02.01.12	Tue 03.01.12		10	NA
12		Ontology Selection	1 day	Wed 04.01.12	Wed 04.01.12		11	NA
13		Ontology Integration	1 day	Thu 05.01.12	Thu 05.01.12	in our case there is not yet an ontology into which any reused one can be integrated => part of conceptualization for the new ontology	12	NA
14		Non Ontological Resource Reuse	5 days	Thu 29.12.11	Wed 04.01.12	(manually?) extracting info from bhl models	7	NA

Project: NeOn Instance
Date: Sun 11.03.12

Task	External Milestone	Manual Summary Rollup	
Split	Inactive Task	Manual Summary	
Milestone	Inactive Milestone	Start-only	
Summary	Inactive Summary	Finish-only	
Project Summary	Manual Task	Deadline	
External Tasks	Duration-only	Progress	

Page 1

Figure A.5.: NeOn aircraft design ontology project plan (page 1/9)

ID	Task Mode	Task Name	Duration	Start	Finish	Notes	Predecessors	Actual Start
15		**Reengineering Phase**	**5 days**	**Fri 06.01.12**	**Thu 12.01.12**		**8**	**NA**
16		Non-Ontological Resource Reverse Engineering	1 day	Fri 06.01.12	Fri 06.01.12		14	NA
17		Non-Ontological Resource Transformation	1 day	Mon 09.01.12	Mon 09.01.12		16	NA
18		Ontology Forward Engineering	3 days	Tue 10.01.12	Thu 12.01.12		17	NA
19		**Design Phase**	**21 days**	**Fri 13.01.12**	**Fri 10.02.12**		**15**	**NA**
20		Ontology Conceptualization	14 days	Fri 13.01.12	Wed 01.02.12		18	NA
21		Ontology Formalization	7 days	Thu 02.02.12	Fri 10.02.12		20	NA
22		**Implementation Phase**	**5 days**	**Mon 13.02.12**	**Fri 17.02.12**		**19**	**NA**
23		Ontology Implementation	5 days	Mon 13.02.12	Fri 17.02.12		21	NA
24		**Maintenance Phase**	**0 days**	**Fri 17.02.12**	**Fri 17.02.12**		**22**	**NA**
25		Ontology Upgrade	0 days	Fri 17.02.12	Fri 17.02.12	there is no existing ontology that will be replaced by a new version	23	NA
26		Ontology Versioning	0 days	Fri 17.02.12	Fri 17.02.12	there are not yet any different versions of the ontology available => applicable for further iterations, if there is a need to use more than one version	25	NA
27		**Support Activities**	**44 days**	**Tue 20.12.11**	**Fri 17.02.12**	**duration of all activities equals the duration of the whole iteration, i.e. project**	**1**	**NA**
28		Control	44 days	Tue 20.12.11	Fri 17.02.12		1	NA

Project: NeOn Instance
Date: Sun 11.03.12

Task		External Milestone		Manual Summary Rollup	
Split		Inactive Task		Manual Summary	
Milestone		Inactive Milestone		Start-only	
Summary		Inactive Summary		Finish-only	
Project Summary		Manual Task		Deadline	
External Tasks		Duration-only		Progress	

Page 2

Figure A.6.: NeOn aircraft design ontology project plan (page 2/9)

ID	Task Mode	Task Name	Duration	Start	Finish	Notes	Predecessors	Actual Start
29		Ontology Quality Assurance	44 days	Tue 20.12.11	Fri 17.02.12		1	NA
30		Ontology Configuration Management	44 days	Tue 20.12.11	Fri 17.02.12		1	NA
31		Ontology Elicitation	44 days	Tue 20.12.11	Fri 17.02.12		1	NA
32		Ontology Documentation	44 days	Tue 20.12.11	Fri 17.02.12		1	NA
33		Ontology Evaluation	44 days	Tue 20.12.11	Fri 17.02.12		1	NA
34		Ontology Assessment	44 days	Tue 20.12.11	Fri 17.02.12		1	NA
35		*Project Iteration 1 Finish*	*0 days*	*Fri 17.02.12*	*Fri 17.02.12*		*26*	*NA*

Project: NeOn Instance
Date: Sun 11.03.12

Task		External Milestone		Manual Summary Rollup	
Split		Inactive Task		Manual Summary	
Milestone		Inactive Milestone		Start-only	
Summary		Inactive Summary		Finish-only	
Project Summary		Manual Task		Deadline	
External Tasks		Duration-only		Progress	

Page 3

Figure A.7.: NeOn aircraft design ontology project plan (page 3/9)

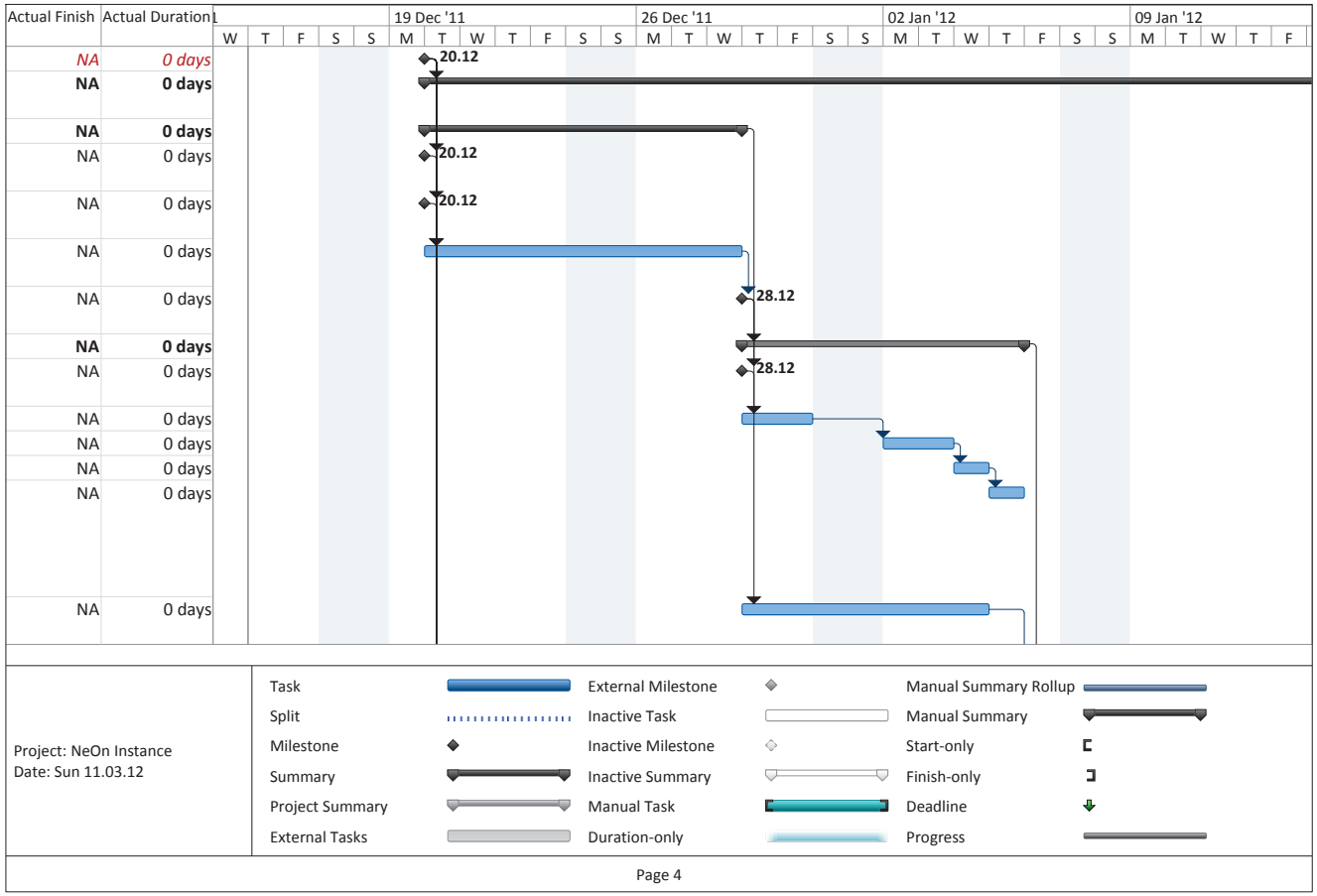


Figure A.8.: NeOn aircraft design ontology project plan (page 4/9)

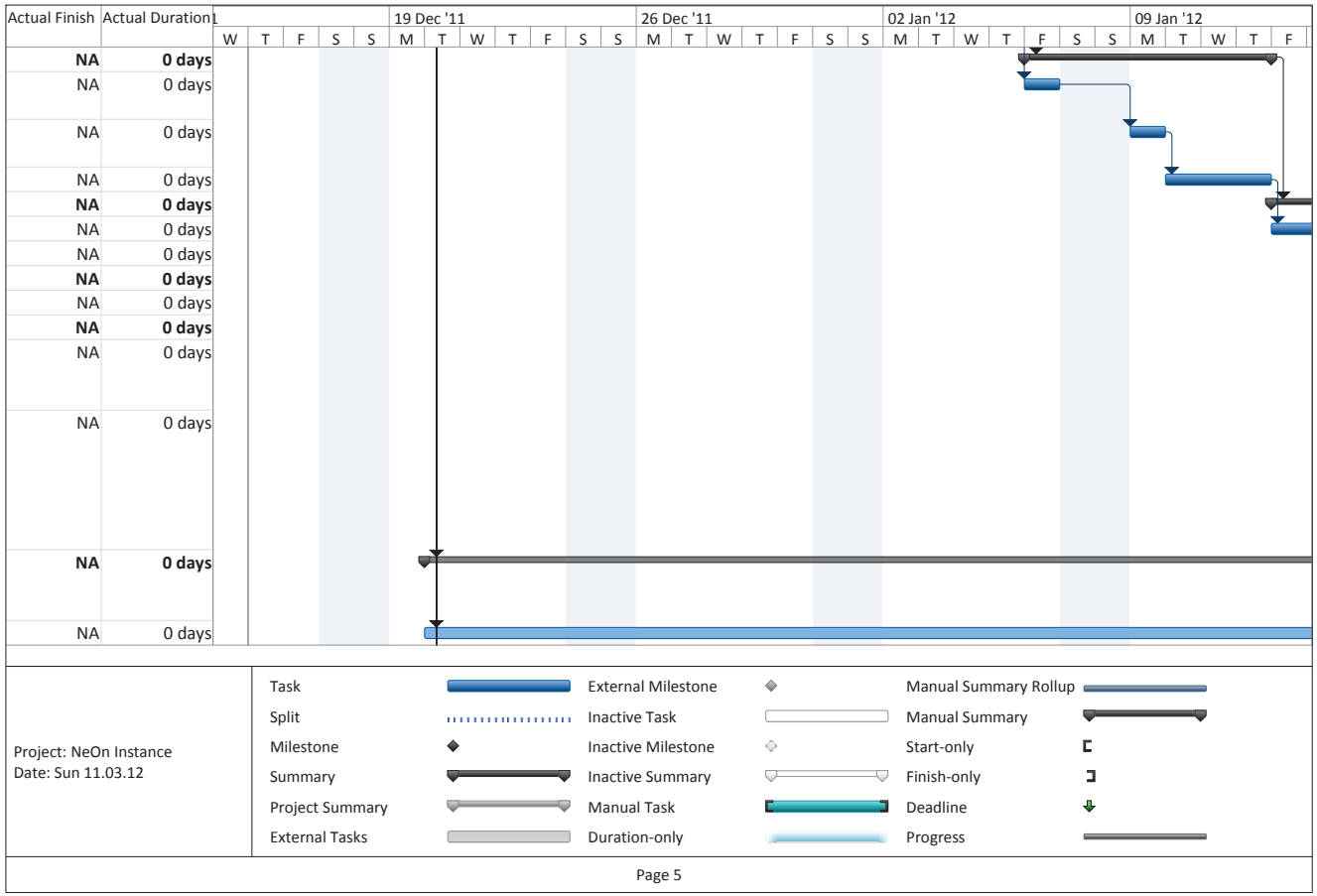


Figure A.9.: NeOn aircraft design ontology project plan (page 5/9)

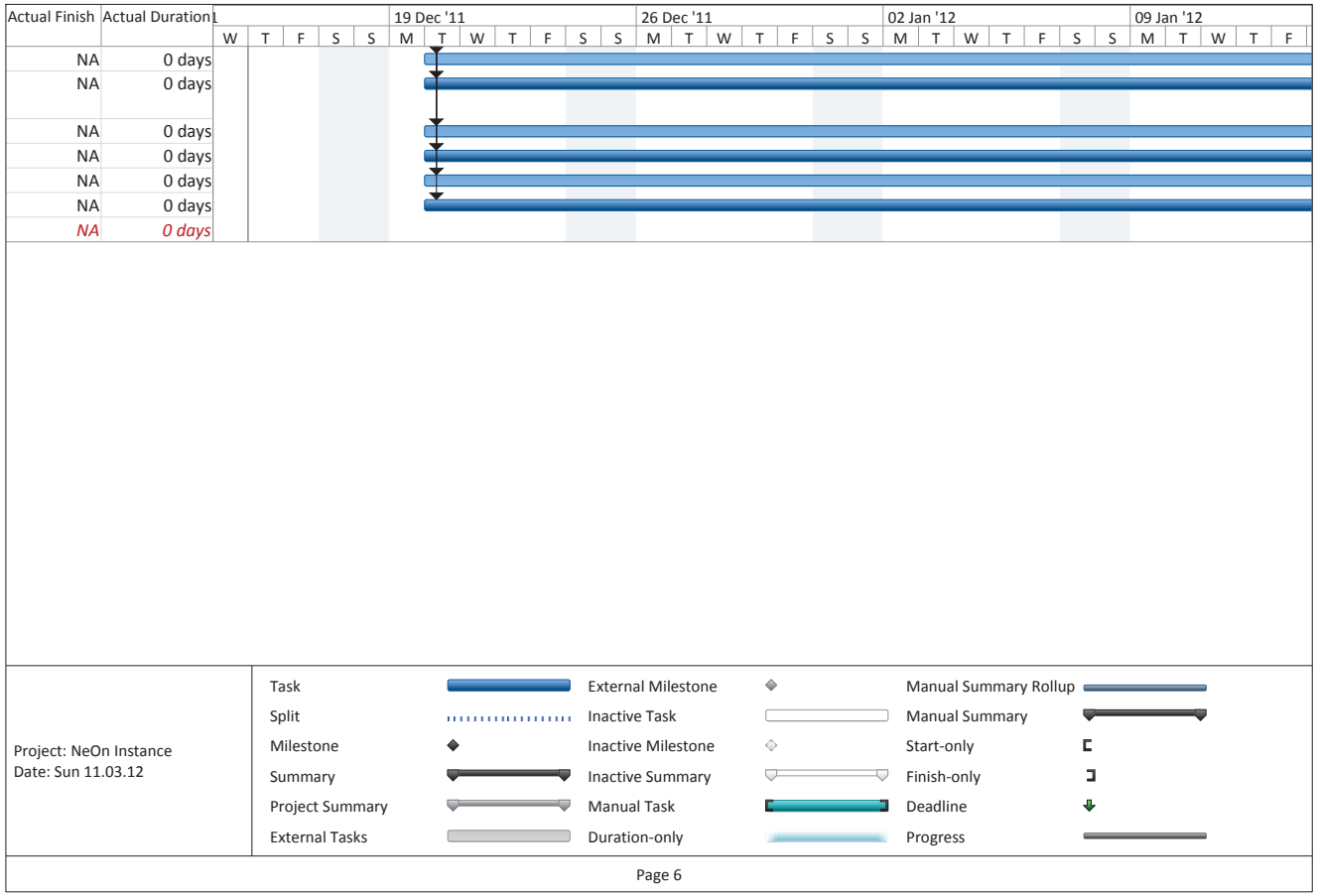


Figure A.10.: NeOn aircraft design ontology project plan (page 6/9)

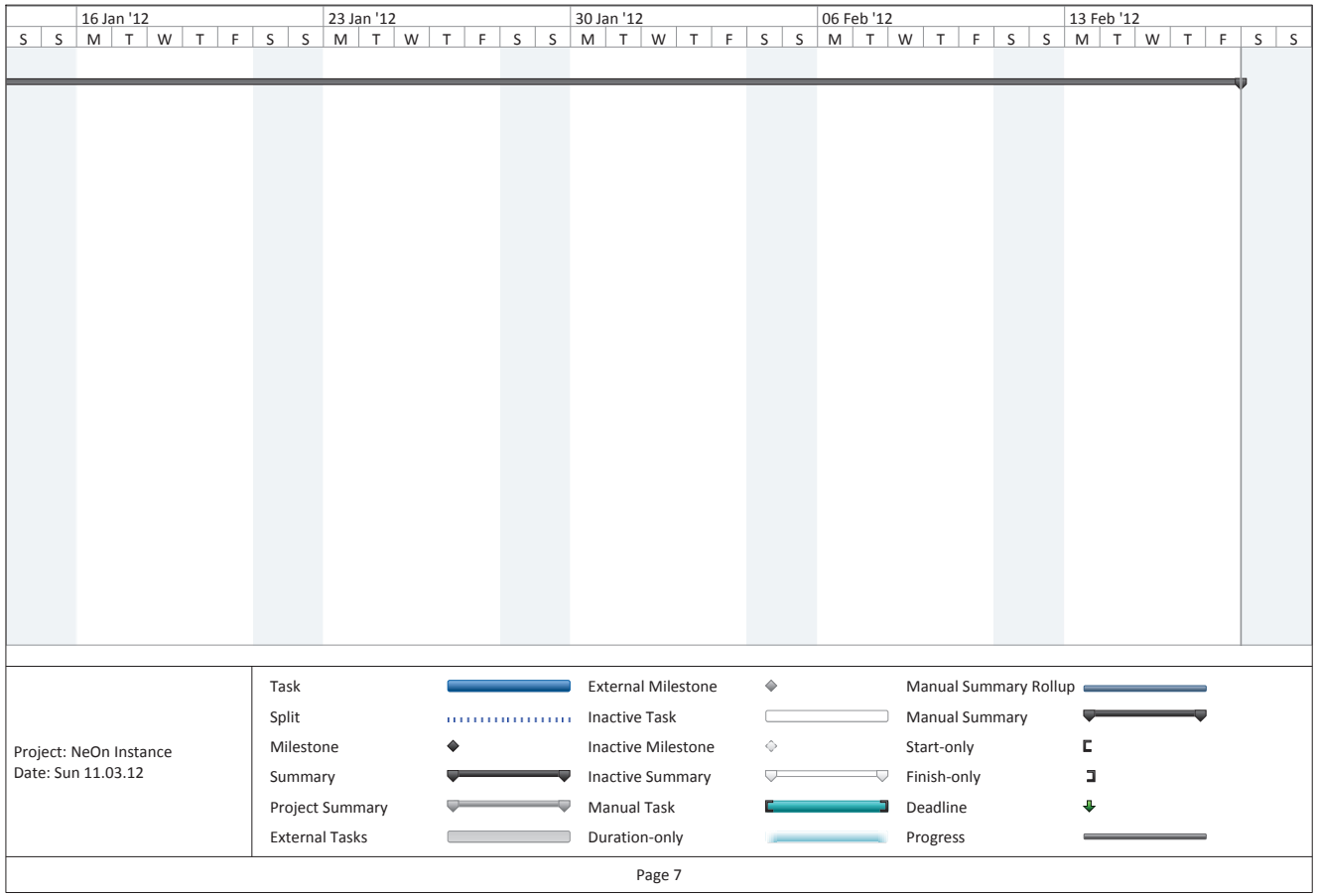


Figure A.11.: NeOn aircraft design ontology project plan (page 7/9)

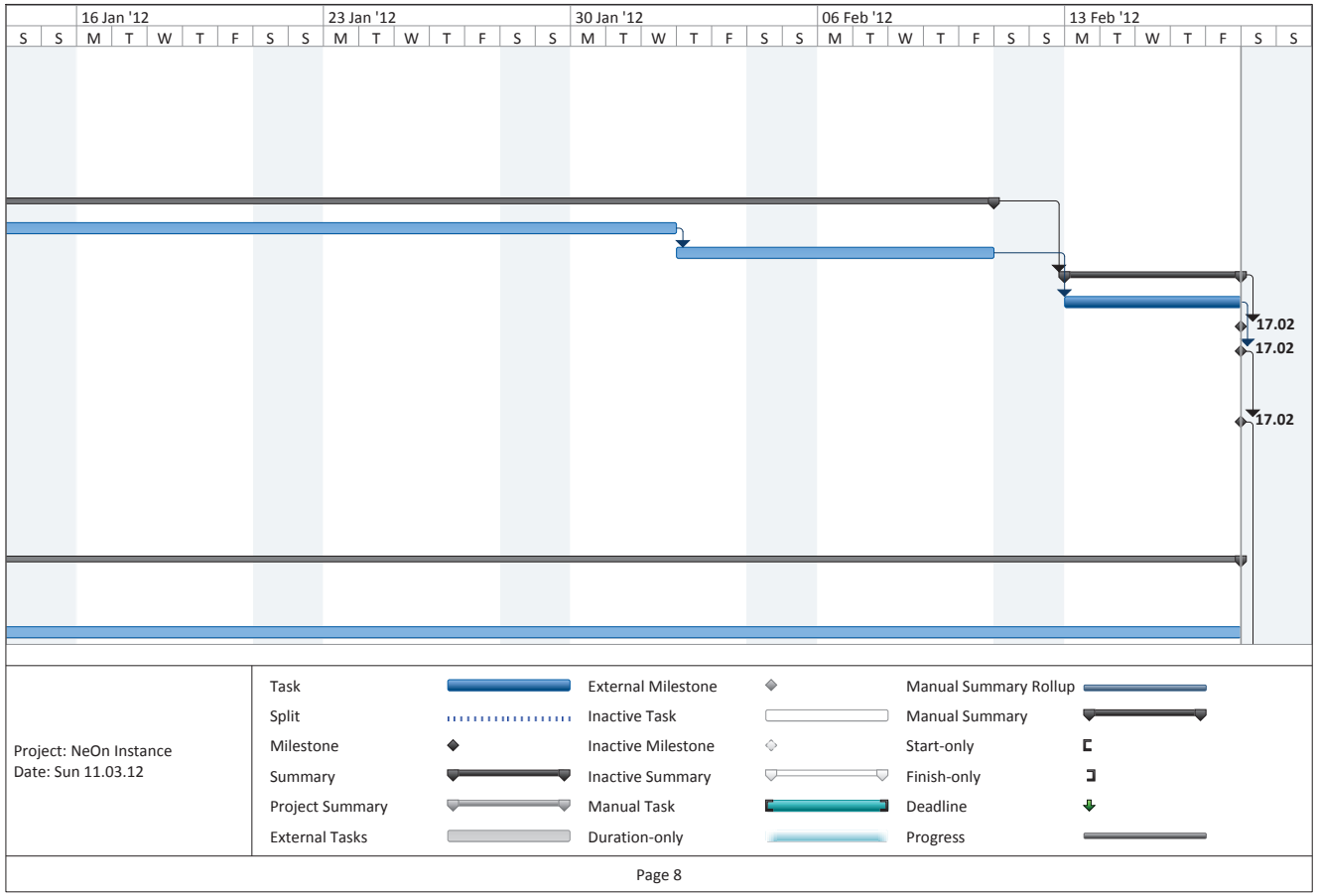


Figure A.12.: NeOn aircraft design ontology project plan (page 8/9)

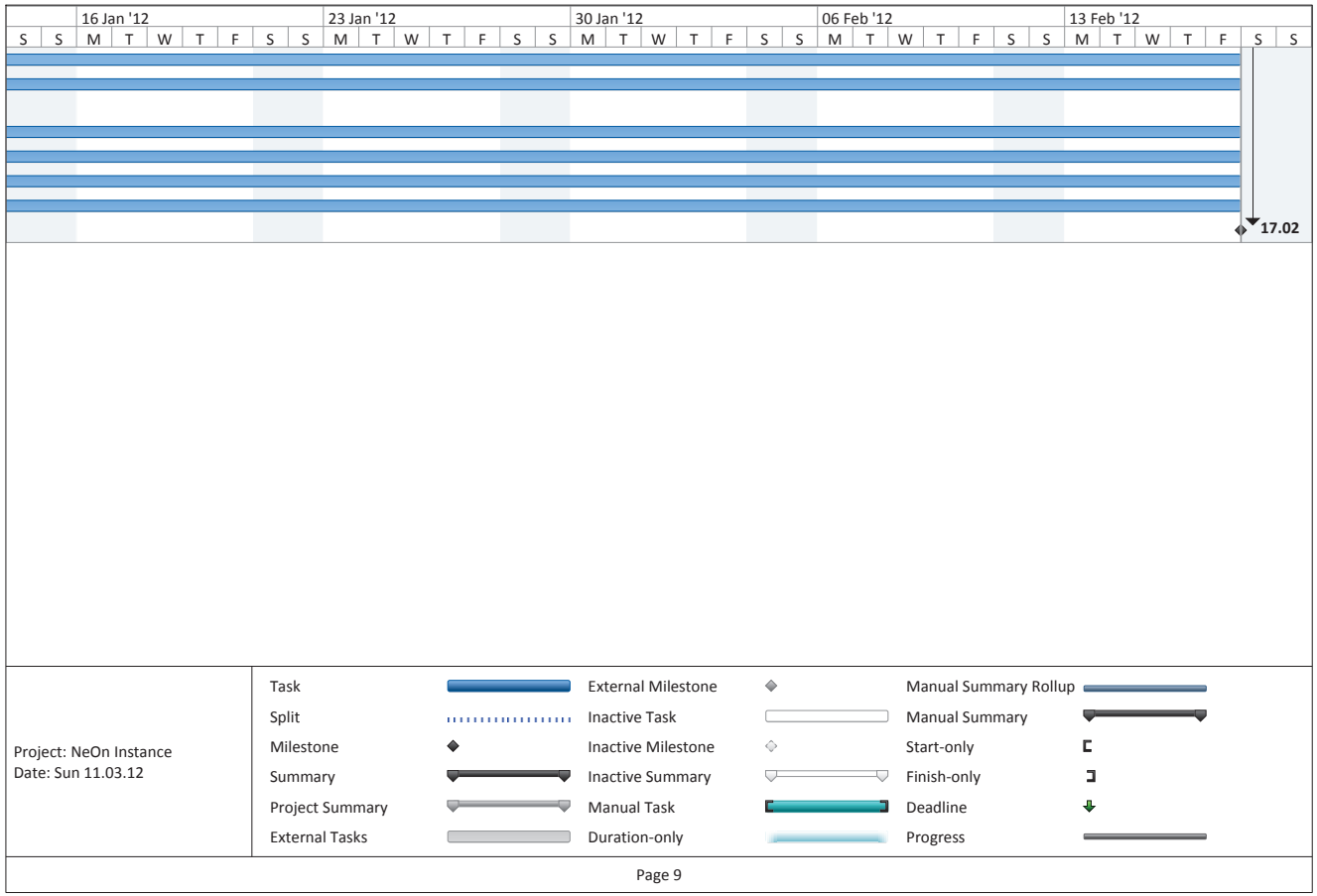


Figure A.13.: NeOn aircraft design ontology project plan (page 9/9)

Bibliography

[1] Neon project homepage. http://www.neon-project.org. last accessed: 02/2012.

[2] Antoniou, G., and Van Harmelen, F. *A semantic web primer*. The MIT Press, Cambridge, Massachusetts and London, England, 2004.

[3] Ath. Schreiber, B. Wielinga, W. J. The kactus view on the 'o' word. Tech. rep., University of Amsterdam, University of Amsterdam, The Netherlands, 1995. Technical Report, ESPRIT Project 8145 KACTUS.

[4] Baader, F., Diego, C., McGuinness, D., Nardi, D., and Patel-Schneider, P. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, 2003.

[5] Benjamin, P. C., Menzel, C. P., Mayer, R. J., Fillion, F., Futrell, M. T., deWitte, P. S., and Lingineni, M. Idef5 method report. Tech. rep., Knowledge Based Systems, Inc., Knowledge Based Systems, Inc., 1408 University Drive East, College Station, Texas 77840, 1994.

[6] BIPM. The international system of units (si). http://www.bipm.org/en/si/si_brochure/. Bureau Internationale des Poids et Mesures.

[7] Brank, J., Grobelnik, M., and Mladenić, D. A survey of ontology evaluation techniques. Tech. rep., Department of Knowledge Technologies, Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia, 2005.

[8] Corcho, O., Fernández-López, M., and Gómez-Pérez, A. Methodologies, tools and languages for building ontologies. where is their meeting point? *Data & knowledge engineering 46*, 1 (2003), 41–64.

[9] De Nicola, A., Missikoff, M., and Navigli, R. A proposal for a unified process for ontology building: Upon. In *Database and Expert Systems Applications*, K. Andersen, J. Debenham, and R. Wagner, Eds., vol. 3588 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 655–664. 10.1007/11546924_64.

[10] De Nicola, A., Missikoff, M., and Navigli, R. A software engineering approach to ontology building. *Information Systems 34*, 2 (2009), 258–275.

[11] Domingue, J., Fensel, D., and Hendler, J. *Handbook of semantic web technologies*. Springer, 2011.

[12] ebiquity group at UMBC. Swoogle. http://swoogle.umbc.edu, 2007. last visit: 15.03.2012.

[13] European Bioinformatics Institute. Ols - ontology lookup service. http://www.ebi.ac.uk/ontology-lookup/. last visit: 15.03.2012.

[14] Fernández-López, M., Gómez-Pérez, A., and Juristo, N. Methontology: from ontological art towards ontological engineering. In *Proceedings of the Ontological Engineering AAAI-97 Spring Symposium Series* (Palo Alto, California, March 24-25 1997), Stanford University, American Asociation for Artificial Intelligence.

[15] Gruber, T. R. A translation approach to portable ontologies. *Knowledge Acquisition 5(2)* (1993), 199–220.

[16] Grüninger, M., and Fox, M. Methodology for the design and evaluation of ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95* (Montreal, April 13 1995), University of Toronto.

[17] Grünwald, A. Evaluation of uml to owl approaches and implementation of a transformation tool for visual paradigm and ms visio. Bachelor thesis, IFS, Wien, IFS - Taubstummengasse 11, 1040 Wien, 2011.

[18] Hartmann, J., Spyns, P., Giboin, A., Maynard, D., Cuel, R., Suárez-Figueroa, M. C., and Sure, Y. D1.2.3 methods for ontology evaluation. internet, January 2005. Knowledge Web Consortium.

[19] Jacobson, I., Booch, G., and Rumbaugh, J. *The Unified Software Development Process*. Addison-Wesley Professional, 1999.

[20] JCGM. International vocabulary of metrology – basic and general concepts and associated terms (vim). http://www.bipm.org/utils/common/documents/jcgm/JCGM_200_2008.pdf, 2008. JCGM: Joint Committee for Guides in Metrology.

[21] Kotis, K., and Vouros, G. Human-centered ontology engineering: The hcome methodology. *Knowledge and Information Systems 10* (2006), 109–131. 10.1007/s10115-005-0227-4.

[22] Lenat, D., and Guha, R. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, Boston, 1990.

[23] López, M., Gómez-Pérez, A., Sierra, J., and Sierra, A. Building a chemical ontology using methontology and the ontology design environment. *Intelligent Systems and their Applications, IEEE 14*, 1 (1999), 37–46.

[24] Noy, N., McGuinness, D., et al. Ontology development 101: A guide to creating your first ontology, 2001.

[25] Pinto, H. S., Tempich, C., and Staab, S. Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engingeering of ontologies. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), August 22nd - 27th* (Valencia, Spain, August 2004), R. L. de Mantaras and L. Saitta, Eds., IOS Press, pp. 393–397.

[26] Raymer, D. P. *Aircraft Design: A Conceptual Approach (AIAA Education)*. American Institute of Aeronautics and Astronautics, 2006.

[27] Sabou, M., Angeletou, S., d'Aquin, M., Barrasa, J., Dellschaft, K., Gangemi, A., Lehmann, J., Lewen, H., Maynard, D., Mladenic, D., Nissim, M., Peters, W., Presutti, V., and Villazón, B. D2.2.1 methods for selection and integration of reusable components from formal or informal user specifications. Internet, May 2007. Deliverable 2.2.1 in the NeOn Project (www.neon-project.org).

[28] Staab, S., and Studer, R. *Handbook on Ontologies*. Springer-Verlag Berlin Heidelberg, 2009.

[29] Studer, R., B. V. F. D. Knowledge engineering: principles and methods. *IEEE Trans. Knowl. Data Eng. 25 (1-2)* (1998), 161–197.

[30] Studer, R., Grimm, S., and Abecker, A. *Semantic Web Services: Concepts, Technologies, and Applications*. Springer-Verlag New York, Inc., 2007.

[31] Sure, Y. *Methodology, Tools and Case Studies for Ontology based Knowledge Management*. Phdthesis, PhD thesis at the Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften, 2003.

[32] Sure, Y., Tempich, C., and Vrandečić, D. D7.1.1 sekt methodology: Survey and initial framework. Internet, December 29 2004. Project: SEKT EU-IST-2003-506826.

[33] Suárez-Figueroa, M., Gómez-Pérez, A., Motta, E., and Gangemi, A. Ontology engineering in a networked world. ISBN 978-3-642-24793-4, 02 2012.

[34] Suárez-Figueroa, M. C., Blomqvist, E., d'Aquin, M., Espinoza, M., Gómez-Pérez, A., Lewen, H., Mozetic, I., Palma, R., Poveda, M., Sini, M., Villazón-Terrazas, B., Zablith, F., and Dzbor, M. D5.4.2. revision and extension of the neon methodology for building contextualized ontology networks. Internet, February 2009. Deliverable 5.4.2 in the NeOn Project (www.neo-project.org).

[35] Suárez-Figueroa, M. C., de Cea, G. A., Buil, C., Caracciolo, C., Dzbor, M., Gómez-Pérez, A., Herrrero, G., Lewen, H., Montiel-Ponsoda, E., and Presutti, V. D5.3.1 neon development process and ontology life cycle. Internet, August 2007. Deliverable 5.3.1 in the NeOn Project (www.neon-project.org).

[36] Suárez-Figueroa, M. C., de Cea, G. A., Buil, C., Dellschaft, K., Fernández-López, M., García, A., Gómez-Pérez, A., Herrero, G., Montiel-Ponsoda, E., Sabou, M., Villazon-Terrazas, B., and Yufei, Z. D5.4.1. neon methodology for building contextualized ontology networks. Internet, February 2008. Deliverable 5.4.1 in the NeOn Project (www.neon-project.org).

[37] Suárez-Figueroa, M. C., Fernández-López, M., Gómez-Pérez, A., Dellschaft, K., Lewen, H., and Dzbor, M. D5.3.2 revision and extension of the neon development process and ontology life cycle. Internet, November 2008. Deliverable 5.3.2 in the NeOn Project (www.neon-project.org).

[38] Suárez-Figueroa, M. C., Gómez-Pérez, A., Poveda, M., Ramos, J. A., Euzenat, J., and Duc, C. L. D5.4.3. revision and extension of the neon methodology for building contextualized ontology networks. Internet, January 2010. Deliverable 5.4.3 in the NeOn Project (www.neon-project.org).

[39] Swartout, B., Ramesh, P., Knight, K., and Russ, T. Toward distributed use of large-scale ontologies. Tech. rep., AAAI Symposium on Ontological Engineering, Stanford, 1997.

[40] Tempich, C. *Ontology Engineering and Routing in Distributed Knowledge Management Applications*. Phdthesis, PhD thesis at the Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften, 2006.

[41] Torenbeek, E. *Synthesis of Subsonic Airplane Design*. Springer, 1982.

[42] University of Manchester. Owl api. http://owlapi.sourceforge.net/. last visit: march 2012.

[43] Uschold, M., and King, M. Towards a methodology for building ontologies. Tech. Rep. 183, AIAI, University of Edinburgh, July 1995.

[44] Vrandecic, D. *Ontology Evaluation*. Phdthesis, KIT, Fakultät für Wirtschaftswissenschaften, Karlsruhe, 2010.