Porting XDtools from NeOn Toolkit to Protégé

by

Ylva Hecktor

LIU-IDA/LITH-EX-A--13/069--SE

2014-01-16
Final thesis

Porting XDtools from NeOn Toolkit to Protégé

by

Ylva Hecktor

LIU-IDA/LITH-EX-A--13/069--SE

2014-01-16

Supervisor: Eva Blomqvist
Examiner: Henrik Eriksson
Abstract

XDtools is a plugin originally used in the NeOn Toolkit ontology editor. XDtools is used as a tool for the ontology engineer when designing new ontologies or to add new parts to existing ontologies. The XDtools plugin was partially ported to another ontology editor, Protégé. This thesis describes the implementation and design choices made during the work with porting the plugin. A description of the parts that have been ported can be found in the report. It also contains the result of the testing and user study made after the implementation. Conclusions of those results and proposed changes to the plugin are also found in the report.
## Contents

1 Introduction ........................................... 1
   1.1 Background .................................... 1
   1.2 Purpose ....................................... 2
   1.3 Scope .......................................... 2
   1.4 Structure ..................................... 3
   1.5 Glossary ....................................... 4

2 Theoretical background ............................... 5
   2.1 Ontologies ..................................... 5
   2.2 NeOn Toolkit and Protégé ....................... 6
   2.3 Content Ontology Design Patterns .............. 9
   2.4 Ontology Design with CPs ...................... 10
   2.5 eXtreme Design ................................ 11
   2.6 XDtools in NeOn Toolkit ....................... 12
      2.6.1 the ODP Registry View ................. 12
      2.6.2 the ODP Selector View ................. 12
      2.6.3 the Specialization Wizard ............. 12
      2.6.4 the Annotation Dialog ................. 13
      2.6.5 the XD Analyzer View ................. 14
      2.6.6 Additional Features ................... 14

3 Method .................................................. 16
   3.1 Methods for the Development ................. 16
   3.2 Methods for the Testing ...................... 16
   3.3 Methods for the User Study ................... 17

4 Development and Design Choices .................. 19
   4.1 the Plugin Development ....................... 19
   4.2 Porting the Annotation Dialog ............... 20
   4.3 Porting the ODP Registry View ............... 22
   4.4 Porting the Specialization Wizard .......... 22
      4.4.1 WPageInputOutput ....................... 24
      4.4.2 WPageSelectEntities ................... 24
      4.4.3 WPageSpecialized ....................... 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.4 WPageAxioms</td>
<td>27</td>
</tr>
<tr>
<td>4.4.5 WPageOverview</td>
<td>28</td>
</tr>
<tr>
<td>4.4.6 After the Specialization Wizard</td>
<td>29</td>
</tr>
<tr>
<td>4.5 Porting Additional Features</td>
<td>29</td>
</tr>
<tr>
<td>5 XDtools in Protégé</td>
<td>30</td>
</tr>
<tr>
<td>5.1 The XDtools Menu and Tab</td>
<td>30</td>
</tr>
<tr>
<td>5.2 the Annotation Dialog</td>
<td>30</td>
</tr>
<tr>
<td>5.3 The ODP Registry View</td>
<td>31</td>
</tr>
<tr>
<td>5.4 The Specialization Wizard</td>
<td>32</td>
</tr>
<tr>
<td>5.5 Example Use</td>
<td>33</td>
</tr>
<tr>
<td>6 Evaluation</td>
<td>38</td>
</tr>
<tr>
<td>6.1 Acceptance Testing</td>
<td>38</td>
</tr>
<tr>
<td>6.2 User Study</td>
<td>39</td>
</tr>
<tr>
<td>6.2.1 the ODP Registry View</td>
<td>39</td>
</tr>
<tr>
<td>6.2.2 the Specialization Wizard</td>
<td>40</td>
</tr>
<tr>
<td>6.2.3 the Annotation Dialog</td>
<td>43</td>
</tr>
<tr>
<td>6.2.4 In General About the Whole Plugin</td>
<td>43</td>
</tr>
<tr>
<td>7 Discussion about future work</td>
<td>44</td>
</tr>
<tr>
<td>7.1 Conclusions from the Development</td>
<td>44</td>
</tr>
<tr>
<td>7.2 Conclusions of the Acceptance Testing</td>
<td>45</td>
</tr>
<tr>
<td>7.3 Conclusions of the User Study</td>
<td>46</td>
</tr>
<tr>
<td>8 Discussion</td>
<td>49</td>
</tr>
<tr>
<td>9 Conclusions</td>
<td>51</td>
</tr>
<tr>
<td>A Testing</td>
<td>55</td>
</tr>
<tr>
<td>A.1 Testing Instructions</td>
<td>55</td>
</tr>
<tr>
<td>B User Study</td>
<td>57</td>
</tr>
<tr>
<td>B.1 User Study Instructions</td>
<td>57</td>
</tr>
<tr>
<td>B.2 User Study Form</td>
<td>60</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter introduces the background, purpose, scope, method and structure of this master thesis.

1.1 Background

An ontology is a model of a real world domain. It can be used to model and solve problems. There are some changes that can be made to the process of creating ontologies to make the process easier. Traditionally when starting a new ontology project you had to design all the axioms for the ontology from scratch or start from some existing ontology. A way to solve this problem is to use reusable ontology design patterns (ODPs).

ODPs are modeling solutions for solving recurring ontology design problems. There are several different types of ODPs, one is Content Ontology Design Patterns (CPs). CPs are small ontologies which are solutions to recurring ontology design problems. They address a specific modeling issue, and solve design problems for specific domain classes and properties that populate an ontology. CPs are used as parts when designing a new ontology. Instead of designing it from scratch the idea is to take different CPs, combine them, maybe change some of them and add more logical constraints to the resulting ontology. The result will be a new tailor-made ontology.

The eXtreme Design plugin (the XDtools plugin) contains tools to be used with the eXtreme Design (XD) methodology and supports the use of CPs. XD is an agile method for ontology design, one of the main characteristics of XD is the use of CPs. NeOn Toolkit and Protégé are ontology editors, they are used to manage ontologies.
1.2 Purpose

This thesis consists of two parts. In the first part the aim was to port the XDtools plugin for NeOn Toolkit version 2.4 to Protégé version 4.1. The expectation was that by porting XDtools to the more widely used Protégé the number of users of the plugin would increase. Thereby also increasing the use of CPs and XD. In the second part the resulting plugin was evaluated. The plugin was tested and the GUI was evaluated in a user study.

This thesis strives to answer the question of how the XDtools plugin for NeOn Toolkit can be ported to Protégé 4.1. This question has been broken down in to more detailed research questions, that are answered in the thesis:

- What are the major differences in the implementations in NeOn Toolkit and Protégé for each of the components?
- What changes need to be made to the user interface of XDtools in order to be able to port the plugin to Protégé?
- How will the resulting XDtools plugin look and work?
- Does the Protégé plugin offer the user the same possibilities in terms of choices, when working with CPs, as the NeOn Toolkit plugin?
- Does the Protégé plugin produce the same results as the NeOn Toolkit plugin?
- Are there any changes that could be made to make the new plugin easier to understand and more user friendly?

1.3 Scope

The main focus of this work has been to get a working XDtools plugin for Protégé. Due to time constraints only three of the five components of the plugin were ported to Protégé. These are the Annotation Dialog, the ODP Registry View, and the ODP Specialization. The ”cheat sheets” and help sections were removed from the tool since these features are specific to a plugin that inherits its GUI parts from Eclipses GUI Framework. To evaluate the Protégé plugin, acceptance testing and a user study of the plugin have been conducted. The focus of the testing has not been to find all bugs, but to find out if the XDtools plugin for Protégé is working in the same way as the XDtools plugin for NeOn Toolkit, and if the result that is produced by the components is the same. The aim of the user study has been focused more broadly on getting to know how the users feel about the current user interface and if there are any changes that should be made to improve it.
1.4 Structure

This thesis report begins with a look at the theoretical background in chapter 2. The theoretical background chapter begins with information about ontologies in general and CPs in particular and how these are used. The rest of the chapter contains an explanation of why and how XDtools is used, and then lists and explains the different components of the plugin.

The next chapter, chapter 4, contains information about the development of the Protégé XDtools plugin and which design choices that were made. Chapter 5 provides an overview of how the finished plugin look and work. In chapter 6 the result of the acceptance testing and the user study can be found.

A discussion about possible improvements to the Protégé plugin can be found in chapter 7. Chapter 8 contains a discussion about the result of the work and Chapter 9 contains the final conclusions including what ought to be done in order to get a fully working Protégé XDtools plugin.
1.5 Glossary

Acceptance testing Black box testing performed by a user familiar with the system, in order to simulate how the system will be used. The test cases should emulate real world scenarios that a future user might use the system for.

Class An object class in Java.

CP Content Ontology Design Patterns

Eclipse A development environment.

GUI Graphical User Interface

OWL-class An OWL entity. The OWL-class represents some kind of underlying concept. A set of individuals (different things) can be linked to the class in order to show that they belong to that concept, they are called the instances of that class. For instance an OWL-class could be flowers, and the instances could be Daisies and Tulips.

Reasoner Software for inferring logical consequences from a set of asserted axioms.

Test user Refers to the person that performed the acceptance testing on the Protégé plugin.

the NeOn Toolkit plugin The old XDtools plugin for NeOn Toolkit.

the Protégé plugin The partial XDtools plugin for Protégé, that was implemented during this thesis work.

the XDtools plugin the eXtreme Design plugin

Vocabulary A special kind of ontology that contains simpler definitions. These vocabularies can be imported and used by other ontologies in order not to have to define simple things multiple times.

XD eXtreme Design
Chapter 2

Theoretical background

This chapter introduces the background of this thesis, it contains information about ontologies, ontology editors, Content Ontology Design Patterns, eXtreme Design, and the eXtreme Design plugin. There is also a chapter about the two ontology editors NeOn Toolkit and Protégé.

2.1 Ontologies

Ontologies are used to model domains and reason about entities. Ontologies represent the relations between different concepts, and can be used to model real world problems. Important characteristics of ontologies are the possibility to share and reuse them [6]. This means sharing the available information in an existing ontology, and reusing the structures that an existing ontology provides to model identical or similar domains.

Ontologies are usually modeled by OWL (Web Ontology Language). To make it possible to efficiently model and solve problems OWL is based on Description Logics. Description Logics provides the possibility to construct complex concepts from simpler ones and is useful for solving decision problems [7]. OWL is used to publish and share ontologies on the Internet, as a way of providing the information in the ontology to computers [6]. OWL can be written in several different syntaxes.

A simple example ontology can be seen in code segment 2.1. The example ontology represents a car register. The example data for the ontology can be found in code segment 2.2. The data contains information about cars and their owners.
2.2 NeOn Toolkit and Protégé

OWL ontologies can be managed with an ontology editor. Ontology editors are used to create new ontologies and edit existing ontologies. Another use is knowledge acquisition, to get information from an existing ontology via queries. NeOn Toolkit\(^1\) and Protégé\(^2\) are two ontology editors.

Protégé is a free, open source ontology editor developed at Stanford University. There are several different mailing lists for Protégé, to support

\(^1\)[http://neon-toolkit.org/]
\(^2\)[http://protege.stanford.edu/]
users and developers. The Protégé site also contain pages with information for developers and documentation of the Protégé API. Protégé supports multiple different types of plugins\(^3\), it uses the OSGi\(^4\) framework to support the use of plugins.

Figure 2.1 shows NeOn Toolkit’s and Protégé’s user interfaces. To the left in figure 2.1a, that is showing an example of NeOn Toolkit’s user interface, is a tree containing two projects. The second project contains an ontology, people.owl. The “Classes” node of that ontology is expanded, to show the ontology’s classes. To the right in the figure more detailed information, such as in this instance the annotations for the chosen class, are shown.

At the top of figure 2.1b, that shows an example of Protégé’s user interface, the name and URI of the active ontology can be found. Each of Protégé’s tabs show detailed information about different parts of the active ontology. In the open “Active ontology” tab the annotations for the ontology can be found. At the bottom of the tab, the user can see the ontologies that are directly and indirectly imported into the active ontology. When an ontology is directly imported, Protégé imports the ontology direct into the active ontology. This is done in order for the active ontology to be able to reuse the imported ontology’s parts, instead of having to redefine those parts in the active ontology. An indirect import is not imported into the active ontology, but instead it is imported by Protégé for some other use.

As can be seen in figure 2.1 the two programs Neon Toolkit and Protégé show ontologies in different ways. In Neon Toolkit it is possible for the user to have multiple projects each with multiple ontologies in it. In Protégé there is only one active ontology that the user can import other ontologies into.

Another difference is that Neon Toolkit and Protégé use different GUI frameworks. Neon Toolkit uses SWT\(^5\) and Protégé uses AWT and Swing to make its own GUI elements. NeOn Toolkit and Protégé both use the OWL API\(^6\). The OWL API is a Java implementation for creating, manipulating and serializing OWL Ontologies.

\(^3\)http://protegewiki.stanford.edu/wiki/PluginAnatomy
\(^4\)http://www.osgi.org
\(^5\)SWT is Eclipse’s GUI framework.
\(^6\)http://owlapi.sourceforge.net/
2.2. NEON TOOLKIT AND PROTEGÉ

THEORETICAL BACKGROUND

(a) NeOn Toolkit

(b) Protégé

Figure 2.1: Ontology editor user interfaces.
2.3 Content Ontology Design Patterns

Ontology design patterns are used when designing ontologies to solve recurring modeling problems [1]. Content Ontology Design Patterns (CPs) are one type of ontology design patterns. The problems that CPs are used to solve have two components, a domain and a use case [2]. A domain can have many different use cases and a use case can be relevant to many different domains [2, 1]. CPs must explicitly encode both the domain and the use case, to be reused as solutions to modeling problems. Ontologies are usually considered as models capturing a domain. A way of capturing the use cases is by using competency questions (CQs) [2, 5]. A CQ is written as an ordinary question using natural language. A CQ corresponds to a certain task and is a typical query that an expert might formulate and submit to a knowledge base of a domain [2, 1]. A CQ, capturing a use case for one part of the example ontology in section 2.1, is “Who owns a certain vehicle?”. A domain ontology should be formulated not only to be necessary but also sufficient to represent all the CQs an expert might formulate for the domain [5]. A CP corresponds to a set of CQs, which addresses the problem that the CP provides a solution to. These CQs can be used to find a CP that matches a modeling problem.

CPs show certain characteristics, according to Presutti and Gangemi[1, 2] they are computational, small, autonomous, hierarchical, cognitively relevant, linguistically relevant, and use best practices. CPs are used as building blocks to form parts of an ontology by applying different operations to the CPs. They only affect the parts of the ontology that relates to the domain modeling problem that they address. Presutti and Gangemi[1, 2] list a set of operations that are important for creation and usage of CPs, these are:

- **Import** consists of including a CP in the ontology that is under development. The elements of the imported CP are not changed.

- **Specialization** is when an ontology sub-classes or sub-properties at least some OWL-class or property of a CP. An ontology is specialized from a CP by importing it and then specializing at least one element from the original CP.

- **Generalization** is the reverse of specialization. Instead of specializing at least one element from the original CP it is instead generalized.

- **Composition** is the operation of associating an OWL-class or property from one CP with an OWL-class or property from another CP. This is done by using some OWL axiom. The result can then be used in the ontology under development.

- **Expansion** consists of importing a CP into an ontology, and then adding further OWL-classes, properties and axioms to it, with the aim of covering requirements not met by the original CP.
These operations can be used by an ontology engineer to make and import a new CP and then use it in an ontology that is being developed or use it as the basis for a new ontology.

2.4 Ontology Design with CPs

In order to be able to reuse CPs, an ODP portal\textsuperscript{7} has been setup that contains a collection of different CPs. When reusing a CP there are two main steps, selection and application \cite{1}. Selection includes searching and evaluating possible CPs, to find the most appropriate CP for the modeling problem.

To find a suitable CP the intent of the CP must match the local modeling problem. Then the resulting CP is applied to the target ontology’s domain, by means of the operations described in section 2.3.

Several CPs can match the modeling problem fully or partially, in different ways. These matches can be divided into some categories described by Presutti and Gangemi\cite{1, 2}. Note that the description of how to select CPs is based on the assumption that the work is done by hand, and not with any tools such as XDtools. The possible types of matches are:

**Precise or redundant matching** means that it is possible to use the CP directly, it only has to be imported into the target ontology.

**Broader matching** is a too general match. Then the “Generalization of” field in the CP’s catalog entry may contain references to less general CPs that specialize it. If one of them is appropriate, it should be imported and specialized to fit the domain part it is being used to model.

**Narrower matching** is a too narrow match, and the “specialization of” field in the CP’s catalog entry may contain references to more general CPs. If there are no CPs that are more general, then the narrow CP can be imported instead, and then be generalized in order to fit the domain part it is going to model.

**Partial matching** means that it is a match for some parts of the problem but not for other parts. The “is component of” in the CP’s catalog entry may contain CPs suitable for the domain problem of the target ontology. But there might not be any such CPs, then the domain problem of the target ontology has to be divided into smaller pieces. The current CP can then be a solution to one of those pieces, and further CPs to use for the other pieces have to be found. All the CPs corresponding to the problem should be imported and composed to form a solution to the problem.

\textsuperscript{7}ontologydesignpatterns.org
2.5 eXtreme Design

eXtreme Design (XD) is an agile methodology for designing ontologies [3]. The method uses ontology design patterns as reusable solutions for problems. XD is inspired by eXtreme Programming (XP) and experience factory [4]. Experience factory is an organizational and process approach, used to improve life cycles and products based on past experience and the know-how derived from them [4]. XD has several similarities to the two approaches but the aim is different, contrary to XP the main focus of XD is on careful design. XD is test-driven and applies the divide-and-conquer approach. It also uses pair designing and emphasizes the need for customer involvement and feedback [4]. Below is a brief simplified walk through of the XD process. The process is covered in more details by Presutti, Blomqvist, Daga and Gangemi [3, 4].

The XD process begins with a meeting between the design team and the domain experts that represents the customer. This meeting will make the customer aware of the methods and tools that will be used during the project, and the design team will get an opportunity to get to know the details of the problem.

The customer representatives are then invited to write down stories that describe possible scenarios, these stories should exemplify the typical kind of information that will be stored in the resulting ontology.

When there is a sufficient number of stories, each design pair will pick one story to focus on for the rest of the current iteration. The design pair will then translate that story into CQs.

Then one, or a small set that represents a coherent modeling problem, of the story’s CQs is selected, and the work begins with finding relevant CPs for it. This is done by matching the selected CQ, or set of CQs, against the CQs of the CPs.

The matching CPs that were found should then be evaluated, to find the best matching and most suitable ones. As can be seen in section 2.4 there can be several different types of matches, some better than others. In some cases it is obvious what match is the best, maybe one is a precise match and the other is a partial match. In some cases the type of match will be the same, but the possible CPs will give differences in the modeling solutions. Then it is the design pair’s task to evaluate them and decide which modeling solution that is the best. The result of this process can be one or many CPs. The chosen CPs are then reused by the operations described in section 2.3. A tool such as the XDtools plugin can be used to find matching CPs and reuse them.

The resulting CPs are tested and potential bugs are fixed. If there are any other CPs that belong to the story, they are also tested and any discovered bugs are fixed. All CPs corresponding to a story together make up a new module.
If there are CQs that belong to the story that have not been addressed, then the design pair will go back and choose a new CQ that belongs to the story but have yet not been addressed. If all the story’s CQs have been addressed then the design pair will release the module for integration. At least one design pair is responsible for the integration of the new modules with the rest of the modules that make up the current version of the ontology. Once all modules are integrated the resulting ontology will be tested, bugs will be fixed and a new version of the ontology will be released. All this is done by the design pair that is responsible for the integration.

2.6 XDtools in NeOn Toolkit

The NeOn Toolkit plugin supports the XD process with CPs. The current NeOn Toolkit version of the plugin implements support for some of the tasks of the methodology such as finding relevant CQs, importing and specializing CPs. XDtools consists of five components, the ODP Registry View, the ODP Selector View, the XD Analyzer View, the Specialization Wizard, and the Annotation Dialog [3]. Below is a short description of how the XDtools plugin for NeOn Toolkit works. It describes the five main components and some additional features.

2.6.1 the ODP Registry View

The ODP browser, in the ODP Registry View, allows the user to see different CPs in a tree-like view, as can be seen in the lower left corner of figure 2.2. By default the user will see all the CPs available in the ODP portal. When a CP is selected from the ODP browser the ODP details view, at the lower right corner in figure 2.2, will show a description of it based on all its annotations. The CP can both be directly imported, or first specialized and then imported into a local ontology [3, 4].

2.6.2 the ODP Selector View

The ODP Selector View will help the user to find an appropriate CP to a particular modeling issue. When typing keywords or a competency question, the algorithm will return a selection of candidate CPs [3, 4].

2.6.3 the Specialization Wizard

The Specialization Wizard helps the user with the process of specializing a CP. The Specialization Wizard can be accessed by clicking on a local ontology as well as clicking on a CP in the ODP Registry View.

The wizard starts by letting the user choose between three alternatives that affects how and where the specialized CP will be imported. A specialized CP can either be imported into an existing ontology or make up a new
ontology by itself with its own URI. If the specialized CP is imported into an existing ontology there are two different choices, it can have its own URI or the statements of the specialized CP can be imported into the ontology as locally defined statements. These three choices can be seen in figure 2.3, under “select task outcome”.

It then asks the user to choose which of the CPs entities to specialize and in the next step lets the user specialize each of those entities. The wizard suggests possible axioms that can be added to the resulting ontology, the user chooses the axioms that are relevant to the new ontology.

Last, the user is shown an overview of the selections that were made, to let the user see if there is anything that should be changed before importing the specialized CP. If there are such things the user can go back and change them at any time during the specialization process.

XDtools will open the Annotation Dialog after the specialization process has been finished, in order to let the user annotate the new ontology [3, 4].

2.6.4 the Annotation Dialog

The Annotation Dialog, seen in figure 2.4, is used to document an ontology by using annotation properties. Multilingual annotation of ontologies is
supported by the Annotation Dialog. The annotation properties are loaded from vocabularies. Currently the Annotation Dialog contains a number of default vocabularies, and additional ones can be added. One of the available vocabularies is the CP annotation schema, which is useful for annotating CPs [3, 4].

2.6.5 the XD Analyzer View

The XD Analyzer View provides the user with feedback concerning possible mistakes and suggestions of best practices. A message indicates to the user if there are any problems. There are three different levels of messages; errors, warnings and suggestions [3, 4].

2.6.6 Additional Features

XDtools provides several help features. Both as inline info boxes, ”cheat sheets” and help sections in the Eclipse help center [3]. Figure 2.5 shows the black inline info box at the bottom and the Eclipse help center can be found at the right.
2.6. XDTOOLS IN NEON TOOLKIT

THEORETICAL BACKGROUND

Figure 2.4: the Annotation Dialog

Figure 2.5: Help feature in the NeOn Toolkit plugin.
Chapter 3

Method

To be able to answer the research questions three main tasks were performed. The first task was the development of the Protégé plugin. Then the evaluation of the plugin followed, with testing and a user study. This chapter contains a section for each of the tasks, each with information of the methods used for that task.

3.1 Methods for the Development

The development followed an agile methodology, where each component of XDtools was ported separately. Different design choices were analyzed during the work to find a solution to occurring problems. The work was performed in an iterative way where a component was finished and then tested by the developer. Then the bugs were fixed and the component was tested again. The iteration proceeded until no more bugs were found in the component. Then the work proceeded with the next component.

3.2 Methods for the Testing

The work started with testing the old NeOn Toolkit plugin, in order to get to know the XDtools plugin before starting with the implementation phase. Then requirements and test cases were written for that plugin before starting the work, in order to prevent the knowledge of how the code of the Protégé plugin worked from affecting how the test cases were written. However, it was discovered that it was not possible to write test cases before the implementation, as the understanding of the XDtools plugin was not sufficient. Another problem was the differences between NeOn Toolkit and Protégé that meant that test cases written for NeOn Toolkit did not work in Protégé. These realizations lead to the work with writing test cases being postponed, as it was not possible to write sufficient test cases at that time.
The Protégé plugin was instead acceptance tested by the developer of the original NeOn Toolkit plugin. Acceptance testing is black box testing performed by the test user. The test user should test the system by using test cases that emulates typical tasks that a user might use the system for [10]. Acceptance testing is performed to make sure that the system meets the requirements, works correctly, and is usable [10]. In this case the acceptance testing was performed to find out if the components were working in the same way in the Protégé plugin as in the NeOn Toolkit plugin, and if the same results were produced by the components.

The instructions for the user testing were deliberately not too strictly defined, as can be seen in appendix A.1. This was done to make it possible for the test user to test with test cases that reflects how the tool is normally used. According to Watkins[10] it is particularly important in user acceptance testing that the testing process is monitored by an independent test observer in order to make sure that test personnel from the developing team are not meddling in the test process. As this is a small project with one developer, it was not possible to have a dedicated person for that task. But particular consideration was put to the fact that the developer should not influence the process, by letting the person testing the plugin work as freely as possible. If the test instructions would have been too detailed there would be a risk that the test user only tested the plugin in the same way as the developer, and some ways that the plugin is normally used might have been missed. As the test user was quite familiar with the NeOn Toolkit plugin he should be able to test the plugin without step by step instructions.

3.3 Methods for the User Study

A qualitative user study was carried out to evaluate the user interface of the Protégé plugin. Instructions and assignments to test the Protégé plugin and a questionnaire was sent out to a group of test users familiar with ontologies. The instructions and questionnaire can be found in appendix B.1 and B.2.

The reason for choosing a questionnaire, instead of for instance conducting interviews with the participants of the study, was that the user study was conducted via email. The study was conducted via email to get a sufficient number of participants with sufficient knowledge about ontologies. Knowledge about ontologies was important to be able to test the plugin without first having to learn about ontologies. This knowledge was also important when answering the questions in the questionnaire, to be able to give relevant feedback about the plugin. For instance to know what additional information that could be shown about the CPs the respondent had to know what information an ontology can contain.

The questionnaire begun with questions to find out background information about the users, and then continued with questions for the System Usability Scale. The System Usability Scale (SUS) is a simple usability scale developed to get a quick overall measurement of a systems usability [8]. The
result of the SUS is collected by letting the respondents state their immediate response to ten statements. Each item is accompanied by a scale where the user can express agreement or disagreement with the statement. The selected statements cover a variety of different aspects about system usability, this will give a good overall measurement of the system’s usability [8]. To get a measurement that is easier to interpret, Bangor, Kortum, and Miller[9] introduces a way of determining what individual SUS scores mean. The SUS score is placed on a scale with different ratings of the usability, to get a description of the usability in ordinary words.

After the SUS questions, questions more particular to the Protégé plugin were added, to find out more about how the users felt about the plugin. A lot of free-text questions were added to the questionnaire, to let the respondents describe in more detail how they felt about the plugin. According to Bisantz[11], questions are either closed-form (e.g. multiple choice, ratings) or open-ended (e.g. free-text questions). A closed-form question is easier to analyze, but restricts the respondents responses to the choices added by the designer of the questionnaire. The open-ended are more difficult to analyze, but offer the respondents greater possibilities to state their opinion.

The following general guidelines were used when constructing the questionnaire. The questions in a questionnaire should be simple and straightforward in order to prevent leading the respondents to a specific response or confusing them [12], which will result in a faulty response or no response at all. All types of leading questions should be avoided. It is also important to remember that the set of response answers to a question will contribute to the result. Respondents are generally reluctant to answer with an answer that is not within that set [13].

Questions addressing more than one issue should be split into multiple questions [12]. If you ask several questions at the same time the respondent is more likely to just answer one of them or the answer might be vague and you might not know what questions he has answered. Terms or expressions that are unfamiliar to the respondents should be avoided [12]. All terms that could be unfamiliar to the respondents were explained in either the user study instructions or when they were used in the questionnaire.

The questions in the questionnaire should be tested, in order to ensure that they will be correctly understood by the respondents, to get feedback, critique and suggestions on how to improve the questionnaire. Ideally it should be tested on a sample group of participants [12]. The questionnaire was evaluated by a person that had first tested the user study instructions, in order to get feedback about the questionnaire. Changes were made to the questionnaire in response to that feedback.

It is important to think of the data analysis already at the point when the questions are formulated, in order to make it easy to collect relevant data [12]. Improper wording or design of questions can make it hard to know what the respondent is really answering, is it the intended meaning of the question or has he interpreted the question in another way.
Chapter 4

Development and Design Choices

This chapter contains information about the implementation of the Protégé plugin. Including information about the different design choices that were made and explanations of why those choices were made. The chapter starts with a section containing more general information on the development of the plugin, and it is then followed by a section about each component that has been ported.

4.1 the Plugin Development

The main difference between the Neon Toolkit and the Protégé plugin is that they use different GUI frameworks. This meant that the whole user interface and all its underlying mechanisms had to be rewritten.

NeOn Toolkit and Protégé uses the same OWL API. With the exception of some methods having become deprecated and removed, most of the code for handling ontologies was possible to reuse. Some minor changes were made due to the fact that Protégé does not have projects. OWLOntologyManager is the class in the OWL API that is used to get access to and work with ontologies. In Protégé there is only one OWLOntologyManager for the active ontology. In NeOn Toolkit every project has its own OWLOntologyManager, to separate the ontology data of the different projects. This means that in NeOn Toolkit it is necessary to keep track on which project the ontology belongs to, to get the right OWLOntologyManager to use when operating on the ontology. In Protégé you only have to get the OWLOntologyManager for the active ontology, and there is no need to keep track of which OWLOntologyManager that belongs to which ontology as there is only one. This meant some changes in the implementation removing all mentions of projects from the code, the code was in some ways simplified by this.
In the Protégé plugin the locations of the plugin components are different than their locations in NeOn Toolkit. Unlike in NeOn Toolkit, there is a special menu for XDtools and a XDtools tab in the Protégé plugin. The XDtools tab contains the ODP Registry View, and the Annotation Dialog can be found in the XDtools menu. The menu item and tab were added to the GUI by adding plugin.xml files containing instructions on what to add and by letting delegate classes subclass certain Protégé classes, to get access to Protégé’s internal state for the ontologies.

There are generally no exceptions or other error messages if something is wrong either with the plugin.xml files or in some of the packages that contains the data for the component, if something else than the java code is wrong. The corresponding menu item, tab or view is simply not added during runtime. This was a problem at multiple times during the development, as menu items or tabs did not show up and there was no clue to what could be wrong. There seems to be no detailed documentation on how to add plugins either, only some examples. A way to solve these problems was to move all the content of the affected packages to new packages. Then the tab or menu would start working again, probably due to some difference between the old and new setups.

## 4.2 Porting the Annotation Dialog

In Protégé there is a dialog window, seen in figure 4.1, similar to the Annotation Dialog. It is different to the Annotation Dialog in the way that it only works on one annotation at a time. It can be opened from several places in Protégé, one place is the ontology header view that can be seen in figure 4.2. The ontology header view is created by the OWLOntologyAnnotationViewComponent. It inherits the abstract class AbstractOWLObjectEditor in order to show the field for annotating an ontology. To show the list with annotations the OWLOntologyAnnotationList class is used.

An effort was made to inherit AbstractOWLObjectEditor and OWLOntologyAnnotationList as a first step towards building the Annotation Dialog with the help of Protégé’s classes. This approach was subsequently abandoned and the plugin was built from scratch, not inheriting any parts when building the GUI. This was done because those Protégé classes were not written to be subclassed, and it would have been tricky to get it to work as multiple variables that were needed were private to those classes. A benefit of inheriting from Protégé’s classes would have been that the GUI gets the same look as Protégé.

The order the vocabularies are loaded is important in Protégé, it is not possible to load an ontology with the same namespace into Protégé several times. This will happen when a vocabulary is loaded both from file and from a URL, then an exception will be thrown. This problem was solved by simply removing the duplicate vocabularies that were loaded from file. The reason for solving the problem by removing the duplicates was that the
Figure 4.1: the Annotation Editor Dialog

Figure 4.2: The Annotation Editor Dialog can be opened in the ontology header view by clicking the plus sign in the area showing the annotations.
4.3 Porting the ODP Registry View

In Protégé a tab consists of one or many views that contain the elements that make up the GUI of that tab. The views are separate from each other. This lead to some problems with the ODPRegistry and ODPDetails classes that contains the GUI for the ODP Registry View component. When the user selects a CP in the table in the ODPRegistry GUI the information about that CP should be shown by the ODPDetails GUI. There are two different methods to get this to work, either put both GUI parts in one view or make them communicate with each other. Since they are put in two different GUI parts in NeOn Toolkit and this also seems to be the way Protégé tabs are usually made, the decision was made to put them in two different views. This also makes it easier if someone wants to make changes to one of the components in the future. As the implementation of the two components are separated the risk that changes in one will affect the other is lowered.

In order to make ODPDetails update the information when a selection is made in ODPRegistry, ODPRegistry had to have the possibility to retrieve only one and the same instance of ODPDetails. This however meant that there could not be multiple different instances of ODPDetails. This did not seem as a big trade-off, as this will not be a problem as long as you do not want to look at the details of two or more different CPs at the same time. Implementing the possibility to look at multiple details at the same time would mean much more work than just changing this. There would still have to be other changes in how the instances of the class displaying the details was stored and shown to the user.

4.4 Porting the Specialization Wizard

ODP Specialization has been created by inheriting Protégé’s GUI wizard classes\(^1\). This results in ODP Specialization getting Protégé’s standard GUI look. Which is positive because it gets the same unified look as Protégé and there is no need to implement a new wizard. The wizard contains multiple specialization wizard panels, each containing the data for a separate panel in the Specialization Wizard. Whenever the user clicks next in the wizard a new panel will appear.

The Protégé wizard uses the chain of responsibility design pattern, and this means that the ODP Specialization component in the Protégé plugin

\(^1\)http://protegewiki.stanford.edu/wiki/P4UiComponentSummary#Wizards
also does that, as the SpecializationWizard class inherits Protégé’s wizard classes. The Specialization Wizard in NeOn Toolkit inherits Eclipse’s wizard. In the NeOn Toolkit wizard all the panels are added in the order they are supposed to appear in the SpecializationWizard class, and then their positions are fixed when going from one panel to another. This meant some changes in the SpecializationWizard class and the specialization wizard panels, with respect to adding the panels and going from one panel to the next, as those implementation details are handled in different ways in the different wizards.

Figure 4.3: UML diagram explaining the inheritance between the wizard’s classes.

Figure 4.3 illustrates how the specialization wizard panels inherit Protégé’s classes to make up the Specialization Wizard. SpecializationWizardPage is the super class of all the wizard panels, it inherits Protégé’s AbstractWizardPanel class. AbstractWizardPanel inherits another Protégé class, WizardPanel. The createUI(JComponent parent) method is inherited from the super class AbstractWizardPanel, and is overloaded in the specialization wizard panels. The createUI(JComponent parent) method is called by the method createUI() in AbstractWizardPanel.

The constructor of the SpecializationWizardPage has to start by calling the constructor of its super class AbstractWizardPanel. In this constructor the method CreateUI() is called. This means that createUI(JComponent parent) will be called before the constructor of the specialization wizard panel class is finished. An effect of this is that the variables that are assigned values in the specialization wizard panels’ constructors can not
be used in `createUI(JComponent parent)` as their values have not been assigned yet.

The method `displayingPanel` is inherited from the super class `wizardPanel` and overloaded in the specialization wizard Panels. `displayingPanel` is called after the GUI is being displayed, and is used to update the GUI. The values of some class global variables, mainly containing parts of the GUI, were assigned values in `createUI()` and then updated in `displayingPanel`, however this did not work and the variables had to be made static.

This means that in order to update the GUI with data that is assigned in the constructor of the specialization wizard panel the data needs to be updated in `displayingPanel`, and in order for that to be possible the variable being assigned with the data needs to be static. If the data is not saved statically the consequence would be a strange behavior where the data have disappeared when you try to update the variable later.

An example of such a variable is `displayPrefixesCheckBox`. `displayPrefixesCheckBox` contains a checkbox indicating if namespace prefixes should be shown or not. The `SpecializationWizard` class stores the state of the checkbox from one panel to the next. The current instance of the `SpecializationWizard` is passed as an argument to the specialization wizard panels constructors. To make the checkbox and its current state available all the time both before and after showing the GUI to the user, the checkbox first needs to be added to the GUI in `createUI(JComponent parent)`. It then needs to be updated in `displayingPanel` with the current state passed from the previous panel with the help of `SpecializationWizard`. In order for this to work `displayPrefixesCheckBox` needs to be static or else it will not contain the checkbox assigned to it in `createUI(JComponent parent)`.

### 4.4.1 WPageInputOutput

The first panel `WPageInputOutput` contains the choice of how the CP should be imported. Since there is only one active ontology in Protégé, unlike NeOn Toolkit, where the user can have several projects open at one time, there is no point in choosing the ontology for the CP to be imported in the XDtools Protégé plugin, it is instead automatically set to the active ontology. The choice of whether to import the CP with its own URI or import its statements as locally defined statements in the ontology was kept, as those choices were still seen as relevant in the Protégé plugin. The result of these changes can be seen in figure 4.4.

### 4.4.2 WPageSelectEntities

The implementation of the trees in the panels `WPageSelectEntities` and `WPageSpecialized`, has changed from the NeOn Toolkit plugin to the Protégé plugin. The tree shows the entities to be selected in `WPageSelectEntities` and to be specialized in `WPageSpecialized`, just as in the NeOn
4.4 PORTING THE SPECIALIZATION WIZARD

DEVELOPMENT AND DESIGN CHOICES

Figure 4.4: There are two different choices of how to import the specialized CP in the Protégé plugin.

Toolkit plugin. The trees are implemented as tabletrees\(^2\) in the NeOn Toolkit plugin. There is no standard component to implement tabletrees in java, a tree was used instead. It was decided to wait with a more time consuming implementation until the user study was finished in order to first find out if any changes were needed. In the Protégé plugin the trees are instead implemented as regular trees containing a panel that contains a representation of the data. In the case of \texttt{WPageSelectEntities} each panel contains a checkbox and a label with the entity name and label, as can be seen in figure 4.5. In \texttt{WPageSpecialized} there is no checkbox, the label contains the same information as in \texttt{WPageSelectEntities}.

4.4.3 \texttt{WPageSpecialized}

In \texttt{WPageSpecialized} there are the choices to add, edit and remove a specialization of an entity. When the user wants to add or edit an entity, the corresponding button is pressed and a dialog will appear. In NeOn Toolkit this dialog is built on the same wizard as the ODP Specialization dialog, adding one or two identical panels after one another to the wizard.

In Protégé this could not be done using the Protégé wizard. All the

\(^2\)A tree in a table, every item in the tree has its own row.
data that will be updated after instantiation of the class needs to be saved statically, if the Protége wizard is used. This means that if there are two instances of that class the same data object will show the same data. This will inhibit using the same class twice to show different data with the same variable in two different panels.

Instead the panels are added to a dialog window. If there are two panels then they will be added after each other. This dialog was first made by inheriting Protége’s [VerifiedInputEditor](http://protegewiki.stanford.edu/wiki/P4UiComponentSummary#Dialogs) class. It is a class that provide a dialog window with buttons to save the data or close the window. This implementation was also changed as there was no way to get access to the dialog instance in [VerifiedInputEditor](http://protegewiki.stanford.edu/wiki/P4UiComponentSummary#Dialogs) and the outer panel was placed directly into the dialog window. This meant that there was no way to place a scroll bar in such a way that it would cover a majority of the window. This was needed in order to let the user resize the window and still be able to reach all the GUI components. Since the window was rather large it would not fit into smaller screens, and adding a scroll bar was important to make sure that all future users could use the plugin. To make this possible the dialog was instead implemented as a regular [JDialog](http://protegewiki.stanford.edu/wiki/P4UiComponentSummary#Dialogs), the final dialog can be seen in figure 4.6.

---

**Figure 4.5:** The GUI of WPageSelectEntities.
4.4.4 WPageAxioms

The implementation of WPageAxioms was rather straightforward and there are no major differences in neither the implementation or in how the panel looks. The most time-consuming part of the implementation of WPageAxioms was in how the GUI components are created. The content of the drop down boxes and the table are added directly one by one to the components in the NeOn Toolkit plugin. This is not possible to do in the same way in the Protégé plugin as Swing uses a model to store the data of the components. In the Protégé plugin the elements are instead added one by one to a new model and then it is added to the component.

Due to lack of time the coloring of the axioms in WPageAxioms and WPageOverview has not been added to the Protégé XDtools plugin. These different colors are used to mark different types of axioms, to make it simpler for the user to find the axioms that are useful for the CP being created.
4.4.5 WPageOverview

The last wizard panel WPageOverview shows an overview of the choices made in the wizard. During the testing of the component it was realized that some of the data was missing from this part. The data was missing because the secondary ontologies that should have been indirectly imported by the main ontology were not imported. To import the missing ontologies a reasoner was needed, the pellet reasoner\(^4\) was therefore added to the XD Specialization component. This reasoner is needed by the class loading the ontologies.

When XDtools is being developed the reasoner has to be directly packaged with the plugin, since other plugins cannot be loaded when testing the plugin directly in Eclipse. However if the plugin is distributed using Protégé’s built in tool to download plugins, the reasoner can instead be a requirement when installing the XDtools plugin. If the user already has installed the reasoner, it will then not be installed several times.

A bug was found in the Protégé plugin. When the user first specialized one CP and then tried to specialize another CP that both imported the same ontology, an exception was thrown. For example if the user was first to specialize actingFor which imports the cpannotationschema and then try to import airline which also imports the cpannotationschema, an exception would be thrown. In Protégé an ontology cannot be imported more than one time, but in this case the main ontology can not be imported because it imports another ontology that is already imported. This seemed to be a bug in Protégé 4.1, the method loadOntologyFromOntologyDocument\(^5\) (in the class OWLOntologyManager) that throws this exception should handle this exception and load the CP.

An alternative way of solving this problem would be to catch the OWLOntologyRenameException\(^6\) that loadOntologyFromOntologyDocument should throw according to the documentation. Then use the getOntologyID() in OWLOntologyRenameException to get the id of the ontology that is already imported and then remove that ontology and try loading the CP again. This was however not possible since an OWLOntologyRenameException was not thrown, instead a java.lang.RuntimeException was thrown, with the following information:

```
java.lang.RuntimeException: java.lang.RuntimeException:org.\semicaweb.owlapi.model.OWLOntologyRenameException: Could not rename ontology. An ontology with this ID already exists:
\ OntologyID(OntologyIRI(<http://www.ontologydesignpatterns.org/schema/cpannotationschema.owl>))
```

\(^4\)http://clarkparsia.com/pellet
\(^5\)http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/model/OWLOntologyManager.html#loadOntologyFromOntologyDocument%28org.semanticweb.owlapi.model.IRI%29
\(^6\)http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/model/OWLOntologyRenameException.html
This would however merely be a temporary workaround to get it to work until the bug is fixed. As this solution would mean some possible trouble if the user had a changed copy of the ontology that is being removed and replaced by a new unchanged copy. It would mean that the user’s changes were thrown away, and this could potentially lead to strange behaviors.

The plugin was also tested in Protégé 4.3, but the problem remained. When contacting the Protégé developers it was found out that this was a bug with no known workaround\(^7\), the bug probably comes from the OWL API\(^8\).

Currently the RuntimeException is caught and an error message is being showed. The specialization process is then aborted, but the problem remains and the second ontology can not be imported into the same active ontology.

### 4.4.6 After the Specialization Wizard

Contrary to the XDtools NeOn Toolkit plugin the Annotation Dialog is not opened after the specialization process is finished. When the annotation component was first being developed it was thought that the component only needed to annotate the active ontology. The implementation was done in such a way that only the active ontology could be annotated with the XD Annotation component. As the specialization component was developed it was discovered that this was not the case as it should be possible to annotate the ontology resulting from the specialization as the last step of the specialization process. As this was discovered at the end of the implementation there was no time to change this implementation.

### 4.5 Porting Additional Features

XDtools’ inline info boxes are kept as a help for the user when using the plugin. The ”cheat sheets” and help sections are naturally inherited from Eclipse in a plugin built with Eclipse’s classes. They do not offer the user the immediate help that the info boxes provide, and are consequently not implemented in the Protégé plugin. Protégé have no built-in help features, the alternative would be to implement some especially for XDtools.

\(^7\)https://github.com/protegeproject/protege/issues/12
\(^8\)https://github.com/owlcs/owlapi/issues/53
Chapter 5

XDtools in Protégé

Below is a description of the parts of the Protégé plugin that have been ported. It describes how they look and how they can be used by a user.

5.1 The XDtools Menu and Tab

In the Protégé plugin there is a XDtools menu and a XDtools tab. The XDtools tab contains the ODP Registry View. The Annotation Dialog and ODP Specialization components can be found in the XDtools menu.

5.2 the Annotation Dialog

The Annotation Dialog is a tool helping the user to annotate ontologies. A screen shot of the Annotation Dialog’s user interface can be seen in figure 5.1. The Annotation Dialog supports multiple default vocabularies, both from the Internet and local sources. Additional vocabularies can be added, by modifying the list of vocabularies in the code. One available vocabulary, the CP annotation schema, is especially useful for the eXtreme Design method, as it is used for annotating CPs.

When starting the Annotation Dialog all the possible annotation properties will be loaded from the vocabularies. All of the active ontology’s annotations that use an annotation property in one of the vocabularies will have its data loaded into the component too. Annotations can be added, removed or changed in the tool. It is possible for the user to change the language, data type and property for an annotation.

The black info box can be seen in figure 5.1, at the top of the dialog window. The info box helps the user to understand how to use the component.
5.3 The ODP Registry View

When the XDtools tab is opened it shows two Protégé views, a view of the registry and one of the CP details, as can be seen in figure 5.2. The registry view, to the left, will show a tree containing different CPs, by default the tree contains all the CPs in the ODP portal. When a CP is selected in the tree the details view, to the right, will show all of its annotations. The content of each annotation will contain the annotation property, value and language. The CPs can be imported or specialized by clicking on the CP in the tree. There are two ways to import a CP. A right click will let the user either import or specialize the CP. If the user double clicks on a CP then a dialog will ask the user if he is sure that he wants to import the chosen CP.
5.4 The Specialization Wizard

The Specialization Wizard is used to specialize a CP. The wizard inherits its looks from Protégé. It can be accessed by right clicking on a CP in the ODP Registry View and then choosing “specialize”.

The wizard consists of different panels, each letting the user specialize different parts of the CP. Black inline info boxes can be found throughout the Specialization Wizard. These boxes contain information on how to use the different panels.

The first panel of the wizard lets the user choose between creating a CP with its own URI and importing it into the active ontology or adding the statements resulting from the specialization as locally defined statements in the active ontology.

Then the user is asked to select the entities that are going to be specialized in the following steps. The user can then specialize those entities in the next panel, by adding at least one name, label and comment for each of the selected entities. In the following panel there is a list of suggested axioms. The user is asked to choose the axioms relevant to the resulting ontology or skip the step.

In the last panel the user is shown an overview of the selections that were made. If there is anything that the user wishes to change then it is possible to go back and change it at any time during the process. If the user is satisfied with his choices, then he can click finish and the result will be imported into the active ontology in Protégé.
5.5 Example Use

If for example the user wants to describe a house, with rooms as parts of that house, then he could go to the XDtools tab and look for a suitable CP to specialize. The details of the CPs are shown in the details view. After looking at different CPs he might decide PartOf is suitable for the intended purpose. He chooses to specialize PartOf, by right clicking on the CP in the registry, as can be seen in figure 5.3a.

In the first panel he selects the choice that the result of the specialization process should be inserted as locally defined statements in the active ontology. This will incorporate the result from the specialization process into the active ontology, just as if they had been added to the active ontology directly from Protégé. The first panel is shown in figure 5.3b.

In the next panel the relevant entities are selected, as seen in figure 5.2c. The user selects Thing, hasPart and isPartOf. Then in the specialization panel each of the selected entities are specialized in a dialog window. The user specializes Thing, with new classes for House and Room. The properties hasPart and isPartOf are specialized with houseHasPart and isPartOfHouse. The result of the specialization of the entities can be seen in figure 5.2d.

In the next step the axioms relevant to the result is chosen. Such as for instance “hasPart cannot_be_a isPartOf” or “The relation isPartOfHouse is_applicable_to House”. The panel for selecting the axioms can be seen in figure 5.1e.

Last, the result of the specialization is shown in the overview panel, seen in figure 5.1f. After checking that the specialization of the CP is right, the user clicks “finish” and the result is imported into the active ontology.

After the specialization process the user could go to the XDtools menu and select the Annotation Dialog component. Then he could add some annotations to the ontology. For instance he could select creationDate and add an annotation containing the current date. He could also add a description of the ontology by selecting comment. The creation of the second annotation can be seen in figure 5.2.
5.5. EXAMPLE USE

(a) Selecting PartOf in the registry.

(b) Deciding where the result of the specialization will be located.
5.5. EXAMPLE USE

(c) Selecting the entities.

(d) Specializing the selected entities.
5.5. EXAMPLE USE

(e) Selecting the axioms.

(f) Overview of the result of the specialization process.

Figure 5.1: The process of specializing PartOf.
5.5. EXAMPLE USE

Figure 5.2: Commenting the ontology.
Chapter 6

Evaluation

The XDtools Protégé plugin was evaluated to find out if it produces the same result as the old NeOn Toolkit plugin. This was done by performing an acceptance test. The test was not meant to be used to find all bugs in the system, but to let a person familiar with the old NeOn Toolkit plugin compare the two plugins to find out if the Protégé plugin meets the requirements. A user study was then done with people familiar with ontologies to find out how a user would experience the XDtools Protégé plugin’s GUI.

6.1 Acceptance Testing

The instructions for the testing can be found in appendix A.1. The main conclusions of the acceptance testing were that the results produced by the Specialization Wizard and the Annotation Dialog components of the Protégé plugin were the same as in the original NeOn Toolkit plugin. The same data, as in the old plugin, was showed in the last page of the Specialization Wizard. The test user however remarked on the fact that whether the representation of the axioms make sense or not should be decided by the users. He was not sure that the expression cannot_be_a is appropriate when comparing properties instead of types, but the functionality was alright as it always corresponded to “propertyX is owl:disjointPropertyWith propertyY”. The test user checked if there were any choices that were missing in the plugin, he could not find any that were missing.

The annotations were correctly imported from the Annotation Dialog component into Protégé. The test user remarked that it was annoying that the changes to the annotations, that were done in the Annotation Dialog, are performed outside of Protégé’s current in-memory version of the ontology. The changes to the ontology are detected by the system from a file change, and results in a question from Protégé whether you want to import the changes into Protégé.

He noticed that the styles of most parts of the plugin could be improved,
such as distance between objects, positioning of labels, and the black background of the help sections.

6.2 User Study

The instructions and questionnaire for the user study can be found in appendix B.1 and B.2. 7 persons participated in the user study. All of the respondents stated that they were very familiar with ontologies, as can be seen in figure 6.1. Most of the users were also previously familiar with CPs. Some had used the NeOn Toolkit plugin before, and all of the respondents had used Protégé before.

![Image of self-evaluation](image_url)

Figure 6.1: Self evaluation, by the respondents, of their knowledge about ontologies.

The SUS gave an overall rating of 76.4 to the plugin. The result of the SUS can range from 0 to 100, the rating 76.4 implies that there is some room for improvement. According to Bangor, Kortum and Miller [9] a number between 71.4 and 85.5 can be interpreted as good usability.

Brooke [8] notes, when describing how to score SUS, that scores of the individual statements are not meaningful on their own. The questions of the SUS have not been constructed to be interesting on their own, but as a means for getting an overall rating. Thus the scores of the individual statements can not give any accurate information about the different areas that the separate questions concern.

6.2.1 the ODP Registry View

The respondents felt that the ODP Registry View component, seen in figure 5.2, was easy to understand. Some of the respondents commented that right clicking to “get” or “specialize” functions might not be an ideal solution. The addition of buttons for those tasks was proposed. It was suggested to put them close to the list of CPs, maybe at the bottom. One of the respondents said that a search field to let the user filter out relevant CPs would be a desirable feature.
The respondents had many different ideas about what information they were missing in the component. One respondent wanted the possibility to configure the URLs used for the registry. Another wanted the possibility to see information about selected folders in the details view. One respondent had many different ideas about specific features and information that he missed. He wanted either a graphical representation of the CP or an example instantiation of the CP. He mentioned the possibility to also add other information about the CPs, such as size, number of properties and depth. Two of the respondents said that the table in the description view was not ideal for showing the details of the CP. They thought that the long text strings in the value field were hard to read. One of them suggested adding a link to a page with a visual example of the usage of the CP.

6.2.2 the Specialization Wizard

All of the panels in the Specialization Wizard can be seen in figure 5.1. The respondents found the WPageInputOutput panel rather easy to understand, and they seemed to have no problems with any of the choices in the panel.

They also found the WPageSelectEntities panel rather easy to understand. One respondent pointed out that the URL in the tooltip would be more useful if it was clickable. The respondents thought that the NeOn Toolkit plugin represented the entities better in the tree, they liked the separation of the name and the label in NeOn Toolkit. One of the respondents suggested that the label could be put first in bold and then the name in plain text. Another suggested the possibility to let the user decide what should be shown. Then the user could decide if he wanted to see the name, the label or both.

The respondents found the WPageSpecialized panel easy to understand. They had the same feedback with respect to the WPageSpecialized panel as for the WPageSelectEntities panel. One of the respondents wanted the information of the inverse property, if there was one, to be added to the tooltip.

Most respondents felt that the Entity Dialog, seen in figure 4.6, was easy to understand. But one of the users found it unclear why there were two properties to specialize in the dialog when only one was selected in the panel. The respondents found it harder to use the dialog when two properties were shown, data of this can be seen in figure 6.2. The majority however did not feel that it would be more intuitive to specialize one entity at a time, as can be seen in figure 6.3. Some of the respondents had the idea that it might be better to show the properties side by side instead of one after another. One of the respondents noted that only the last added field could be removed, he thought that each field should have its own remove button.

The respondents felt that the WPageAxioms was harder to understand than the rest of the panels. All of them thought that a color coding similar to the one in the NeOn Toolkit plugin would be useful, as can be seen in
6.2 USER STUDY

EVALUATION

Figure 6.2: Data from the user study about the Entity Dialog.

Figure 6.3: Data from the user study about specialization of properties in the Entity Dialog.
6.2. USER STUDY  

EVALUATION

figure 6.4. The color coding of the axioms in the $\text{WPageAxioms}$ panel in NeOn Toolkit can be seen in figure 7.1. The respondents that found the panel hard to understand stated the lack of color coding as one of the reasons why it was hard to understand the panel. They also thought that underlining of the text should be removed. One of the respondents stated that the terminology used was not immediately intuitive, especially with the term $\text{is\_applicable\_to}$. Another respondent stated that he thought that the axioms could be changed in order to read “Room applies to $\text{isPartOf}$” instead of “$\text{isPartOf}$ is $\text{is\_applicable\_to}$ Room”.

![Figure 6.4: Data from the user study about the color coding of the axioms.](image)

Most of the respondents thought that the $\text{WPageOverview}$ panel was easy to understand. One of them found it hard to understand, and the motivation was that too much text was shown in the panel without separators. Other changes, such as removing the underlining of the text, left aligning and color coding, was suggested as possible change to make the panel easier to understand.

There seem to be no overwhelming need for more help in the Specialization Wizard. The overall feeling was that the highlighting of the help sections was useful, but the coloring of them should be changed. This can be seen in figure 6.5. The placements of the help sections were fine according to the respondents.

![Figure 6.5: Data from the user study about the highlighting of the help sections.](image)

42
6.2.3 the Annotation Dialog

The respondents felt that the Annotation Dialog component, seen in figure 5.1, was quite easy to understand. One of the respondents felt that it was unclear when the annotations were saved. He felt that if a button press was needed to create the annotation, then it would be logical for another button press to save that individual annotation. This was linked to the fact that he did not realize that the meaning of the Cancel and OK button was “Close without saving” and “Save and close”.

The black background used in the help section and property information field is too dark, according to the respondents.

6.2.4 In General About the Whole Plugin

One user recommended that the GUI could be improved by following Protégé’s look and feel. The respondents found some bugs when they used the Protégé plugin. The bugs were not described in enough detail to make it possible to draw any real conclusions from them, other than that there are still bugs left. One respondent noticed that the “.xd” directory, where the plugin saves the local content, is saved in the working directory of the program. He noted that this could potentially lead to problems, for instance it can lead to failure in opening the registry in an environment where the program directory cannot be modified by the user.
Chapter 7

Discussion about future work

This chapter discusses different possibilities in the future work with the Protégé plugin. Conclusions are drawn from the development as well as the evaluation of the plugin.

7.1 Conclusions from the Development

In multiple parts of the plugin, such as for instance in the details view of the ODP Registry View component, there are “junk characters” that should be removed. The ODP Registry View component with the “junk characters” can be seen in figure 5.2. In the class XDLabelsBuilder there are methods for removing these kinds of characters. They are already partially removed, but further work will be needed to remove them all. It is important to remember that those methods are used at several places in the code, and that the data they remove the “junk characters” from is different. In order to assure that none of the data is removed by mistake, it should not be assumed that the data is structured in a certain way.

The implementation of the Annotation Dialog should be changed in order to make it possible to open the Annotation Dialog after the specialization process of the selected CP has finished. As it is now the ontology which the Annotation Dialog works on is hard-coded to be the active ontology. This should be changed in order to make it possible for the Annotation Dialog to work on other ontologies, such as the ontology resulting from the specialization process.

In NeOn Toolkit there is a possibility to specialize local ontologies, by clicking on them and selecting specialize. Protégé does not show multiple ontologies, as NeOn Toolkit does in its project structure. Because of this, and time constraints, the possibility to specialize local ontologies was not
ported. This feature can be added by making it possible for the user to specialize ontologies that are fetched locally from the disk.

Due to lack of time, the expansion of the tree in `WPageSelectEntities` was not finished. The result of this is that when the user selects “show complete hierarchy” all of the entities are not expanded. The tree still works, but the user has to manually expand the entities to see their children.

Some variables that are used at multiple places in a component are passed around between different classes. This is not ideal, a better solution would have been to save them in some central class instead, dedicated to storing variables used at multiple places in the component. This is partially done in the Specialization Wizards implementation in the `SpecializationData` class. Examples of such variables, that are being passed around a lot, are `OWL.OntologyManager` and `OWLEditorKit`. This is also relevant to the ontology that the Annotation Dialog works on, instead of hard-coding it to be the active ontology it should be saved in a class, dedicated for saving such variables. Then the Annotation Dialog could have been opened after the specialization process, and the ontology resulting from that specialization could have been annotated via the Annotation Dialog.

There are some bugs left in the plugin. Most of the remaining bugs are probably of a more complex nature. They will only appear if the user makes a sequence of things, for example makes some choices in one of the specialization wizard panels and then goes back one panel and then forward to the previous panel. It will take some time and thorough testing to find all of the bugs. It will also be best to do this when all of the implementation is done, in order to prevent new bugs from being introduced into the code.

The bug that prohibits the indirect import of an ontology several times, is making the porting of the plugin impossible, as it is not possible to import multiple CPs after one another. The bug has been reported to the developers of Protégé and OWL. If continuing the work with the plugin it is possible to fix the bug, if the bug has not already been fixed, as the code of both Protégé and OWL is open source. To fix the bug will however take some time, as the developer first would have to learn how the code of Protégé and OWL is working, in order to be able to fix that bug.

### 7.2 Conclusions of the Acceptance Testing

As the testing and user study were conducted in parallel it was not possible to draw any conclusions from the acceptance testing before the user study, and use those when making questions for the user study. Some questions that were relevant with respect to the conclusions of the testing were already added to the user study. Such as general questions about the GUI, and more specific questions concerning, for example, the black background of the help sections. However it might have been a good idea to add some questions about the representations of the axioms, and how the axioms were formulated.
If it is possible, the changes to the annotations that were made in the Annotation Dialog should be made to the current in-memory version of the active ontology, in order to make the dialog window with the question, of whether the user want to import the changes into Protégé or not, disappear.

### 7.3 Conclusions of the User Study

In order to make it easier for the users to understand the components of the Protégé plugin it would be a good idea to change the GUI of the components to get the same look and feel as Protégé. The overall perception of the GUI seemed to be that it was cluttered and that the style was inconsistent. This could also be improved by changing the style of the GUI to follow Protégé’s look and feel. The fields with black background, such as the help fields, should be changed to have another background color. They can preferably be changed to get a style similar to that of Protégé’s look and feel. As is discussed in section 6.2.4 the “.xd” directory, where the Protégé plugin saves its local content, should be saved somewhere else than in the program directory.

A number of possible future changes to the ODP Registry View were proposed by the participants of the user study, these can be found in section 6.2.1. A search field for finding relevant CPs will be added to the XDtools tab if the ODP Selector View component, described in section 2.6, is ported. It might be a good idea to get rid of the right clicking in the tree or supplement it with buttons for the choices, as this option was not obvious to the respondents in the user study. The many other changes proposed in section 6.2.1 could be explored further in the future. But as most of the different changes were proposed by single persons, all of the users might not find the changes useful. An exception was the addition of some kind of more descriptive information about the CP, like a graphical representation or an example instantiation. It was proposed by two of the respondents. This could be an interesting feature, but if and how it should be added would have to be further investigated to make sure such a feature provides an improvement to the GUI, and that it is implemented in the way that benefits the users the most.

A number of changes to the Specialization Wizard are proposed in section 6.2.2. Some of them could be directly implemented in order to improve the user interface. The trees in the WPageSelectEntities and WPageSpecialized panels should be changed to make it easier for the user to find the relevant entities. One proposed improvement was to put the label first in bold and then the name in plain text. Letting the user decide if the name, label or both should be shown was also proposed. This might however confuse the user and clutter the GUI, but it could be a good addition if it is possible to avoid those problems. It might be a good idea to incorporate information about the inverse property in the tooltip for the trees. Making the URL in the tooltip clickable would be a nice feature, however this might
not be easy to implement. The tooltip for the entity disappears when the mouse is moved outside the entity, and in order to click the URL the mouse would need to be moved outside the current entity. The tooltip disappearing directly is a good thing, as it would be annoying for the user to have to wait for it to disappear.

The high number of respondents that felt that the entities, in the Entity Dialog seen in figure 4.6, should be put side by side instead of after one another suggested that this would be a useful change to the GUI. To allow the users to remove more than only the last added field would be an improvement to the Entity Dialog’s GUI. It would make it possible for the user to remove any of the labels or comments that he had added.

Adding color coding to the WPageAxioms panel, as in the NeOn Toolkit plugin seen in figure 7.1, would be an improvement to the GUI. Another proposed change was to remove the underlining of the terms in the axioms. This would probably make it easier to read the axioms. The respondents had some thoughts about how the terms in the axioms could be changed. In order be able to draw any sure conclusions about possible changes to the axioms, a more thorough evaluation would have to be done.
As with the rest of the Protégé plugin GUI the WPageOverview panel should be changed to get the look and feel of Protégé, and at the same time other changes could be made to make the information in the panel easier to read. By adding separators to the GUI, left aligning, as well as removing the underlining and adding color coding to the terms in the axioms, the readability of the GUI should improve. The color coding could be done in the same way as in the NeOn toolkit plugin, and as proposed for the axioms in the WPageAxioms panel.

How the annotations are saved, by the user, in the Annotation Dialog could be changed to make it work in the same way as in Protégé, to make it easier for users familiar with Protégé to use the dialog. This would mean rewriting most of the GUI for this component.
Chapter 8

Discussion

At the moment it is not possible to fully port the Protégé plugin as there is a bug that makes it impossible to import several different CPs into the same active ontology. This should be possible in order for the user to fully be able to use the tool.

The major difference between the plugin in NeOn Toolkit and Protégé is that the GUI frameworks are different. There are also some differences in how the two ontology editors are constructed, and this lead to some differences in the implementations. The different GUI frameworks of the two ontology editors lead to a number of differences in the user interfaces of the NeOn Toolkit and Protégé plugins. How the Protégé plugin looks and works has been shown in chapter 5 of this report.

The acceptance testing showed that the Protégé plugin works in the same way and produces the same result as the NeOn Toolkit plugin. The choices in the Protégé plugin offer the user the same possibilities as in the NeOn Toolkit plugin. This result is not a hundred percent reliable as there were only one test user, however it is unlikely that the test user would miss any larger flaws.

The conclusion of the results from the System Usability Scale was that the usability of the Protégé plugin is reasonably good, there is however still room for improvements. One proposed improvement was to change the style of the GUI to the look and feel of Protégé, to get a more consistent style throughout the components. The user study resulted in a number of possible changes, to improve the user interface, they were discussed in chapter 7.

A dilemma throughout the implementation has been whether to inherit GUI parts from Protégé or not. Perhaps an idea could have been to inherit smaller parts from Protégé, instead of trying to find solutions that covered a larger portion of the problem. Inheriting parts from Protégé have in most cases lead to problems, but there are benefits of components having a user interface that looks like the rest of Protégé. A good idea is to be careful when inheriting parts from Protégé, and only use the parts that are described in
the developer documentation of UI components\(^1\), as those should in theory be able to be subclassed. The documentation of Protégé is limited, this is also true for the OWL API. This means that there is no way of knowing for sure what a class or method really is supposed to do.

As mentioned in chapter 7 some variables were passed around a lot between the different classes. This could perhaps have been avoided if the porting of the components had been planned in more detail before starting to port them. This was however difficult, as the knowledge of the components before the work was limited. It was not until after the work had begun that the knowledge of the old NeOn Toolkit plugin was sufficient to make the planning possible.

If continuing the work with the already ported components, the first thing to do should be to fix the bug that makes it impossible to get a working plugin. That is the bug explained in section 4.4.5, that makes it impossible to specialize multiple CPs after another with the same active ontology. In section 7.1 remaining implementation details and things that should be changed in the implementation are mentioned. To get the components to work without any unwanted appearance and behaviors those issues should be addressed and corrected. Changes to the user interface that are mentioned in section 7.3, such as changing the appearance of the components to get the same style as Protégé, could be performed to improve the user experience. Those changes are however not that important as the usability of the plugin has already been shown to be good, in section 6.2.

\(^1\)http://protegewiki.stanford.edu/wiki/P4UiComponentSummary
Chapter 9
Conclusions

The result of this thesis work is a partial XDtools Prot´eg´e plugin and an evaluation of the plugin GUI. The Prot´eg´e plugin was also tested to ensure that it would work just as good as the NeOn Toolkit plugin, when working with CPs and annotating ontologies. The following research questions were answered in the report:

- What are the major differences in the implementations in NeOn Toolkit and Prot´eg´e for each of the components?
  Most of the changes in the implementation between the plugins, concerns parts of the code that is dealing with the GUI. The major differences in the implementations, for each of the components, can be found in chapter 4.

- What changes need to be made to the user interface of XDtools in order to be able to port the plugin to Prot´eg´e?
  Some changes were made in the user interface in order to port the plugin to Prot´eg´e. The changes that were made, why they were made, and how they were implemented can be found in chapter 4.

- How will the resulting XDtools plugin look and work?
  A description of how the plugin works in Prot´eg´e can be found in chapter 5. The chapter also contains an example of how the plugin can be used.

- Does the Prot´eg´e plugin offer the user the same possibilities in terms of choices, when working with CPs, as the NeOn Toolkit plugin?
  The Prot´eg´e plugin offer the user the same possibilities in terms of choices, as has been shown in section 6.1.

- Does the Prot´eg´e plugin produce the same results as the NeOn Toolkit plugin?
The components of the Protégé plugin that have been ported produce the same result as the NeOn Toolkit plugin. This conclusion has been reached in section 6.1.

- Are there any changes that could be made to make the new plugin easier to understand and more user friendly?

The conclusions of the user study can be found in section 7.3. Possible changes for improving the plugin, to make the plugin easier to understand and use, are proposed.

The ported components have some, mostly minor, work left on them. A description of what is left to do can be found in chapter 7. The conclusions of the user study have also been incorporated in the chapter, as possible changes and improvements to the current components. Chapter 7 was added to ease continued work with the plugin.

There is a need for testing the components after they have been fully ported, as there are remaining bugs. Those bugs are probably more complex, and a more thorough testing will need to be conducted in order to find them.

The parts of the XDtools plugin that have been ported so far can be used independently of the components not yet converted, when working with CPs. But to get a fully working XDtools plugin for Protégé the rest of the components of the original NeOn Toolkit plugin have to be converted. This means that it would be natural to continue the work, after fixing the remaining problems with the current components, by porting the remaining components.
Bibliography


Appendix A

Testing

A.1 Testing Instructions
**Instructions:**

If Protégé 4.1 is not installed, download and install it from http://protege.cim3.net/download/old-releases/Protege%204.x/4.1%20release%20candidates/4.1%20rc5/Web_Installers/

The XDtools plugin for Protégé plugin can be found at [http://www-und.ida.liu.se/~ylvhe819/](http://www-und.ida.liu.se/~ylvhe819/)

Unpack the zip file in Protégé 4.1 plugins folder. The XDtools plugin's components should show up automatically in Protégé now. But the XDtools tab will have to be added to the visible tabs. This is done by going to the window-> tabs menu and selecting XDtools.

Testing instructions:

Open a new ontology every time a CP is imported or specialized. This have to be done due to a bug in Protégé, when the bug is fixed multiple CPs should be able to be imported and specialized into the active ontology.

Import a CP into the ontology.

Specialize at least 2 CPs as locally defined statements in the active ontology. Specialize at least 2 CPs created as separate ontologies.

Annotate the ontology with at least 3 annotations.

Answer the following questions while working with the ontologies.

**Questions:**

* Is the right data showed in the last page of the specialization wizard?
* Is all the data from the specialization saved in the ontology?
* Are the annotations saved correctly?
* Are there any choices missing in any part of the plugin?
* Have you noticed any bugs or other strange behaviour?
* Is there anything else you have noticed or would like to add?
Appendix B

User Study

B.1 User Study Instructions
Thank for participating in this user study for the Xdtools plugin for Protégé 4.1. Below is some instructions followed by 3 assignments. Last are some information about where to find the questionnaire belonging to the user study.

**Instructions:**

If Protégé 4.1 is not installed, download and install it from http://protege.cim3.net/download/old-releases/Protege%204.x/4.1%20release%20candidates/4.1%20rc5/Web_Installers/

The XDT tools plugin for Protégé plugin can be found at [http://www-und.ida.liu.se/~ylvhe819/](http://www-und.ida.liu.se/~ylvhe819/)

Unpack the zip file and put the jar-files in the Protégé 4.1 plugins folder. The XDT tools plugin's components should show up automatically when you start Protégé now, but the XDT tools tab will have to be added to the visible tabs. This is done by going to the window-> tabs menu and selecting XDT tools.

XDT tools is a tool for importing small ontologies, Content Ontology Design Patterns (CPs), as building blocks when constructing an ontology.

The tool for annotating the ontology can be found in the XDT tools menu. XD registry, where you will find possible CPs to import into the ontology, can be found in the XDT tools tab. If you right click on one of the CPs in XD registry, you will find the option to specialize the CP.

The instructions below will ask you to open a new ontology every time a new CP is going to be specialized. This has to be done due to a bug in Protégé, when the bug is fixed multiple CPs should be possible to import and specialize in the active ontology.

**Assignments:**
The goal is to populate the new ontology with a couple of different CPs mainly describing houses and rooms, and then to annotate the ontology. The instructions are not to detailed, in order to let you more freely test the plugin. If some part of the instructions seem unclear, do what you think is intended or decide what to do yourself.

**part 1:**
Create a new ontology.

Go to the XDT tools tab. Select Place in the registry tree and choose to specialize it. In the specialization editor select to insert the result as locally defined statements in the active ontology. Click continue. Select all the entities. Click continue.

Give the entities the following specializations:

**hasLocation:**
Name: hasHouse
Labels: has House
comments: a room has a location which is a certain house

**isLocationOf:**
Name: isHouseOf
Labels: is House Of
comments: this house is the location of this room

**Place:**
Name: House
Labels: House
comments: A house
Place:
Name: Room
Labels: Room
comments: A room

Select the axioms that you feel are fitting or skip this step.
Look at the overview to see the result of the specialization and click finish.

part 2:
Create a new ontology.

In the XDTools tab, Select PartOf and specialize the CP.
In the specialization editor select to create a separate ontology module and import it into the active ontology. Use a URI of your own choice. Click continue.
Choose to show the complete hierarchy of the tree. Select Thing, hasPart, isPartOf. Click continue.
Give the entities the following specializations:

Thing:
Name: House
Labels: House
comments: A house

Thing:
Name: Room
Labels: Room
comments: A room

hasPart:
Name: houseHasPart
Labels: house Has Part
comments: a house has this as a part

isPartOf:
Name: isPartOfHouse
Labels: is Part Of House
comments: this is part of a house

Add some suitable axioms concerning the Room class by choosing the entity in the “show statements about:” dropdown list.
Look at the overview to see the if the result correlates with the instructions and click finish.

part 3:
Go to the XDTools menu and choose annotation. Add a creation date to the ontology. Also add a suitable description (with a property like hasIntent, rdfs:comment, or another property of your choice) and choose a language for that annotation.

After you have finished the assignments please answer the questions at https://docs.google.com/forms/d/1qaljXbNBboYBqo4OWBQV2jOsRc1axd-SZOTPgiBf8PR0/viewform
While answering some of the questions you may have to go back and look at parts of the XDtools components in Protégé again.
B.2 User Study Form
**XDtools for Protégé user study**

Welcome to the XDtools for Protégé user study. This user study is focused on evaluating the new XDtools plugin's user interface. The XDtools plugin was ported from NeOn toolkit to Protégé. Before answering the questions in this form make sure you have finished all of the assignments in the pdf.

*Required

**Background information**

1. *  
   How familiar are you with ontologies? I am ...
   
   *Mark only one oval.*
   
   1  2  3  4  5

   a beginner ☐ ☐ ☐ ☐ ☐ an expert

2. *  
   Were you familiar with Content Ontology Design Patterns before this user study?  
   
   *Mark only one oval.*
   
   ☐ yes  
   ☐ no

3. *  
   Have you used the "old" XDtools plugin for NeOn Toolkit?  
   
   *Mark only one oval.*
   
   ☐ yes  
   ☐ no

4. *  
   Have you used Prototégé 4 before?  
   
   *Mark only one oval.*
   
   ☐ yes  
   ☐ no

**Overall impression of XDtools**

Some of the questions below mentions 'the system', please consider the XDtools plugin as the system you are evaluating, not the entire Protégé tool.

How well does these statements coincide with your view of the XDtools plugin.
5. *  
I think that I would like to use this system frequently  
*Mark only one oval.*

1 2 3 4 5

| Strongly disagree | | | | | Strongly agree |

6. *  
I found the system unnecessarily complex  
*Mark only one oval.*

1 2 3 4 5

| Strongly disagree | | | | | Strongly agree |

7. *  
I thought the system was easy to use  
*Mark only one oval.*

1 2 3 4 5

| Strongly disagree | | | | | Strongly agree |

8. *  
I think that I would need the support of a technical person to be able to use this system  
*Mark only one oval.*

1 2 3 4 5

| Strongly disagree | | | | | Strongly agree |

9. *  
I found the various functions in this system were well integrated  
*Mark only one oval.*

1 2 3 4 5

| Strongly disagree | | | | | Strongly agree |
10. * I thought there was too much inconsistency in this system

Mark only one oval.

1 2 3 4 5

Strongly disagree 0 0 0 0 0  Strongly agree

11. * I would imagine that most people would learn to use this system very quickly

Mark only one oval.

1 2 3 4 5

Strongly disagree 0 0 0 0 0  Strongly agree

12. * I found the system very cumbersome to use

Mark only one oval.

1 2 3 4 5

Strongly disagree 0 0 0 0 0  Strongly agree

13. * I felt very confident using the system

Mark only one oval.

1 2 3 4 5

Strongly disagree 0 0 0 0 0  Strongly agree

14. * I needed to learn a lot of things before I could get going with this system

Mark only one oval.

1 2 3 4 5

Strongly disagree 0 0 0 0 0  Strongly agree

**XD Registry view**

The following questions concerns the XD Registry view that is the component contained in the XDtools tab. An image of XD Registry view can be seen below.
15. In your opinion how easy is it to understand the XDtools Registry view?

*Mark only one oval.*

1 2 3 4 5

very easy     very hard

16. Is there anything that can be changed to make it easier to understand the XDtools Registry view?

17. Is there any information you miss in the XDtools Registry view?
XD Specialization wizard
The questions in the following pages concern the different panels in the XD Specialization wizard.

The input and output panel

18. * In your opinion how easy is it to understand this panel?
Mark only one oval.

1 2 3 4 5

very easy very hard
19. If 4 or 5 was chosen, why was the panel hard to understand?

20. Is there anything that can be changed to make the panel easier to understand?

The select entities panel

[Image of the select entities panel from XDtools for Protégé user study]
21. In your opinion how easy is it to understand this panel?

*Mark only one oval.*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>very easy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>very hard</td>
</tr>
</tbody>
</table>

22. If 4 or 5 was chosen, why was the panel hard to understand?

23. In XDtools, place the mouse over one of the entities. Is there any more information from the tooltip that you think should be added to the tree?

24. Below is a picture from XDtools in NeOn Toolkit. In your opinion are the entities represented better or worse? Why?
The specialize entities panel
25. *  
In your opinion how easy is it to understand this panel?  
*Mark only one oval.*  

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>very easy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

26. If 4 or 5 was chosen, why was the panel hard to understand?

........................................................................................................................................................................

........................................................................................................................................................................

........................................................................................................................................................................

........................................................................................................................................................................
27. In XDtools, place the mouse over one of the entities. Is there any more information from the tooltip that you think should be added to the tree?

28. Below is a picture from XDtools in NeOn Toolkit. In your opinion are the entities represented better or worse? Why?
Entity Dialog
The questions in this page concern the entity dialog, seen below. The dialog is shown when add or edit is clicked in the specialize entities panel.

29. * In your opinion how easy is it to understand this dialog? 
Mark only one oval.

1 2 3 4 5

very easy   very hard
30. If 4 or 5 was chosen, why was the dialog hard to understand?

31. * 
   When the dialog contains two properties. Do you find the it easy to use the dialog? 
   Mark only one oval.

   1 2 3 4 5

   very easy  ○  ○  ○  ○  ○  very hard

32. When the dialog contains two properties, is there another way you think they should be shown in order to make it easier to use the dialog? In what way should they be shown?

33. Is there anything you would change with the dialog or its content? What?

34. * 
   Would it be more intuitive to specialize one property at a time, even though it has an inverse? 
   Mark only one oval.
   ○ yes  ○ no  ○ Other:
35. * 
In your opinion how easy is it to understand this panel? 
Mark only one oval.

1 2 3 4 5 

very easy □ □ □ □ □ very hard

36. If 4 or 5 was chosen, why was the panel hard to understand?
37. * Would a color coding of the axioms, similar to the one shown below, be useful? 
   *Mark only one oval.*

   - [ ] yes
   - [ ] no
   - [ ] Other:

The Overview panel
38. *In your opinion how easy is it to understand this panel? Mark only one oval.*

1 2 3 4 5

very easy  o  o  o  o  o  very hard

39. If 4 or 5 was chosen, why was the panel hard to understand?
Generally about the XD Specialization wizard

The questions below concern the whole of XD specialization wizard. The current help sections in XDtools are the fields with black background and white text. They are placed in the panels of the XD Specialization wizard.

40. Is there anything that can be changed to make it easier to understand this panel?

41. In your opinion, is more help or information needed in any of the panels in the wizard? Select the panels you think need more help or information below. *Tick all that apply.*
   - The input and output panel
   - The select entities panel
   - The specialize entities panel
   - The select Axioms panel
   - The Overview panel

43. If more help or information is needed in any panel, what kind of help or information would be useful to add?
44. * 
Is the highlighting of the help section useful?
Mark only one oval.

☐ yes
☐ no
☐ yes, but the colors should be changed.
☐ Other: 

45. Is the placement of the help sections good? Or should it be changed in some way?

XD Annotation

The following questions concerns XD Annotation, an image of the component can be seen below.
In your opinion how easy is it to understand the XD Annotation component? 
Mark only one oval.

1 2 3 4 5
very easy very hard

XDtools for Protégé user study
https://docs.google.com/forms/d/1qaljXbNboYBq...
47. If 4 or 5 was chosen, why was the XD Annotation component hard to understand?

48. Is there anything you would change about this component?

Generally about XDtools

49. Is there anything you would change in the XDtools plugin?

50. Is there anything else you have noticed or would like to change?

Thank you for answering these questions. Your answers are valuable for my work with evaluating the XDtools plugin.
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtidiga ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns detlösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsman- nens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet – or its possible replacement – for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

©Ylva Hecktor