eXtreme Design with Content Ontology Design Patterns

Valentina Presutti and Enrico Daga and Aldo Gangemi and Eva Blomqvist

Semantic Technology Laboratory, ISTC-CNR

Abstract. In this paper, we present eXtreme Design with Content Ontology Design Patterns (XD): a collaborative, incremental, iterative method for pattern-based ontology design. We also describe the first version of a supporting tool that has been implemented and is available as a plugin for the NeOn Toolkit. XD is defined in the context of a general approach to ontology design based on patterns, which is also briefly introduce in this work.

1 Introduction

Ontology design patterns (ODPs) [7] are an emerging technology that favors the reuse of encoded experiences and good practices. ODPs are modeling solutions to solve recurrent ontology design problems. They can be of different types[1] including: *logical*, which typically provide solutions for solving problems of expressivity e.g., expressing n-ary relations in OWL; *architectural*, which describe the overall shape of the ontology (either internal or external) that is convenient with respect to a specific ontology-based task or application e.g. a certain DL family; *content*, which are small ontologies that address a specific modeling issue, and can be directly reused by importing them in the ontology under development e.g., representing roles that people can play during certain time periods; *presentation*, which provide good practices for e.g. naming conventions; etc.
With the name **eXtreme Design (XD)** we identify an approach, a family of methods and associated tools, based on the application, exploitation, and definition of ontology design patterns (ODPs) for solving ontology development issues. In this paper, we describe XD and go into details of its guidelines for ontology development with Content ODPs (CPs). Also we briefly describe the prototype of a supporting tool i.e. the XD plugin for the NeOn Toolkit.
XD adopts the notion of ontology project, a development project characterized by two main sets: (i) the *problem space*, which is composed of the actual modeling issues, here referred to as the *local problems*, that have to be addressed during the project e.g., to transform a set of microformats to an RDF dataset, to model roles that can be played by people during certain time periods; (ii) the *solution space*, which is made up of reusable modeling solutions e.g. a reengineering practice for associating microformats' attributes to a certain RDF vocabulary's

[1] http://ontologydesignpatterns.org/wiki/OPTypes

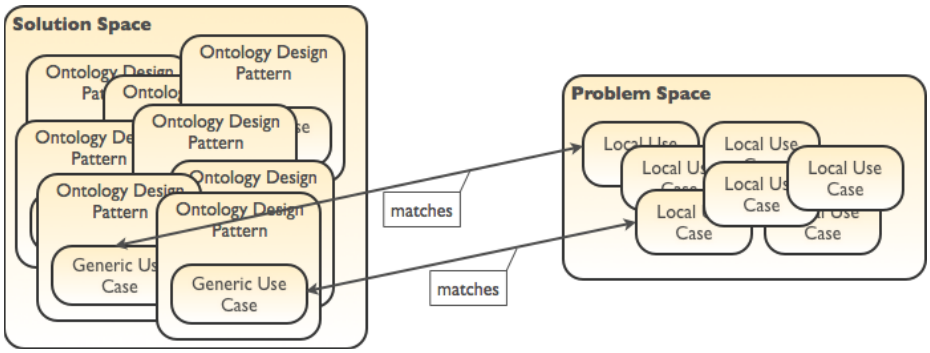


Fig. 1. The eXtreme Design approach. ODPs are associated with Generic Use Cases and compose the ontology project's *solution space*, which is used as the main knowledge source for addressing ontology design issues e.g. reengineering, evaluation, construction, etc., the ontology project's *problem space* provides descriptions of the actual issues called "Local Use Cases".

relations, a piece of an ontology that models time-indexed roles i.e. a CP.

The general approach is schematized in Figure 1. Each element in the solution space is an ODP associated with a Generic Use Case (GUC), the latter representing the problem that the ODP provides a solution for, as introduced by [6]. The elements of the problem space are called "Local Use Case" (LUC), they define the actual modeling issues that need to be addressed in order to work out the ontology project, they represent the ontology project's requirements. Under the assumption that GUCs and LUCs are represented in a compatible way e.g., both in the form of competency questions or sentences, it is possible to compare LUCs to GUCs, and if they match, the ODPs associated with the matching GUCs are selected and reused for building the final solution. Informally, a GUC matches a LUC, if the latter can be completely or partly described exactly in terms of the GUC, or as a more specific case of it; or if the LUC can be described in terms of part of the GUC.

All matching ODPs are selected and used according to specific guidelines and possibly with some tool support.

In this paper, we focus on XD guidelines for CPs, where GUCs and LUCs are expressed in the form of natural language competency questions, and ODPs are CPs. In the rest of the paper XD is used for referring to XD with CPs.

XD is partly inspired by software engineering eXtreme Programming (XP) [11], and experience factory[12, 1]. The former is an agile software development methodology the aim of which is to minimize the impact of changes at any stage of the development, and producing incremental releases based on customer requirements and their prioritization. The latter is an organizational and process approach for improving life cycles and products based on the exploitation of past experience know-how.

Although XD has similarities with the two approaches, its focus is different:

where XP diminishes the value of careful design, this is exactly where XD has its main focus. XD is test-driven, and applies the divide-and-conquer approach as well as XP does. Also, XD adopts pair design, as opposed to pair programming. The intensive use of CPs, modular design, and collaboration are the main principles of the method. While a rigorous evaluation of the whole methodology is still in our future plans, the effectiveness of CPs in ontology design has been rigorously evaluated in [5], where XD has been used as reference development guidelines. Furthermore, initial questionnaires and informal discussions made emerge that the perception of the trainees with respect to the method is positive. The contribution of this paper is twofold: (i) a collaborative, incremental, and iterative method for pattern-based ontology design, called eXtreme Design (XD); (ii) a first version of the XD tool, a NeOn Toolkit plugin that currently supports CP repository browsing and selection, a good practice assistant, and a wizard for CPs' specialization.
The paper is structured as follows: in Section 1.1 we briefly discuss the state of art on ontology design methodologies and ontology design patterns. Section 2 discusses the principles of the method and details the XD workflow with the help of a simplified scenario taken from a real case study. Section 3 describes XD tool, a NeOn Toolkit[2] plugin for pattern-based design support.

1.1 State of the art and related work

The notion of "pattern" has proved useful in the context of design within many areas, such as architecture, software engineering, etc. So far, very few purely pattern-based methodologies have been proposed. In ontology engineering, pattern-based methods are present primarily on the logical level, where patterns support methods for ontology learning, enrichment and similar tasks like in [2]. In these methods patterns are used more or less automatically, e.g. lexico-syntactic patterns to identify ontological elements in a natural language text or to extract relations between ontology concepts. In [3], a method for constructing ontologies based on patterns is proposed, the difference between this method and XD is that the former do not consider collaboration, and that the patterns were assumed to be a non-evolving set mostly defined with a top-down approach. Another related approach is that described in [8], where competency questions have been introduced. XD takes inspiration from this work, specially for the use of competency questions as reference source for the requirement analysis. However, the methodology described in [8] do not consider modular nor pattern-based design approach, and do not address collaborative development. Many other ontology development methodologies have been proposed, an example is the DILIGENT methodology [9], which takes into account collaborative aspects, but do not consider the use of patterns and is not test-oriented. From the software engineering field we can mention the eXtreme Programming methodology, which XD is inspired by. However, the focus of XD is completely different from that of XP, although they share some base principles such as pair

[2] http://www.neon-toolkit.org

design as opposed to pair programming, test-driven development, and customer involvement.

2 Guidelines for XD with Content Ontology Design Patterns

In this section, we describe the methodological guidelines for applying XD with Content ODPs (CPs), through the definition of an iterative workflow and a case example showing an actual iteration.
The XD method with CPs is the result of the observation and consequent description of the way we (in our lab) use to develop ontologies with CPs. Since 2005, we have been developing CPs, teaching pattern-based ontology design in conference tutorials and PhD courses, and for much longer we have been using and refining this approach for our professional work.
In order to teach pattern-based design to PhD students and practitioners, we needed to provide trainees with guidelines to follow. This requirement provided us with a good occasion for defining the XD method with CPs, and also with a context for running the method with different teams, and applying possible refinement/adjustment. So far, we have identified the main principles and setting of the method, defined the iterative workflow, identified a set of requirements for tool support, started supporting tools development, and identified and started investigation of open issues.

XD principles. XD principles are inspired by those of the agile software methodology called eXtreme Programming (XP) [11]. The main idea of agile software development is to be able to incorporate changes easily, in any stage of the development. Instead of using a waterfall-like method, where you first do all the analysis, then the design, the implementation and finally the testing, the idea is to cut this process into small pieces, each containing all those elements but only for a very small subset of the problem. The solution will grow almost organically and there is no "grand plan" that can be ruined by a big change request from the customer.
The XD method is inspired by XP in many ways but its focus is different: where XP diminishes the value of careful design, this is exactly where XD has its main focus. Of course, designing software and designing ontologies is inherently different, but still there are many lessons to be learnt from programming. XD is test-driven, and applies the divide-and-conquer approach as well as XP does. Also, XD adopts pair design, as opposed to pair programming. Although we did not perform yet a formal evaluation of pair design's effectiveness, we have collected trainees feedback through informal discussions and questionnaires after the executions of XD with different trainees teams. Most of them feel to take benefit from on-the-fly brainstorming, and perceive to improve the effect of learning-by-doing with this setting. We have planned to conduct more rigorous evaluation of the method which also involves the analysis of this aspect.
The intensive use of CPs, modular design, and collaboration are the main prin-

ciples of the method. The effectiveness of CPs in ontology design has been rigorously evaluated in [5], where XD has been used as reference development guidelines.
The main principles of the XD method can be summarized as follows:

- **Customer involvement and feedback.** The development of an ontology is part of a bigger picture, where typically a software project is under development. Ideally, the customer should be involved in the ontology development and its representative should be a team, whose members are aware of all parts and needs of the project. For example, the roles that should be represented include: domain experts i.e. persons with deep knowledge of the domain to be described by the ontology; those who are in charge of maintaining the knowledge/data bases i.e. persons who know the views over the data that are usually required by users; those who control/coordinate organization processes i.e. persons who have an overall view on the entire flow of data; etc. Depending on the project characteristics, and on the complexity of the organization, the customer representative can be one person or a team. It is important that the team of designers is able to easily interact with the customer representative in order to minimize the possible number of assumptions that they have to make on the incomplete requirement descriprions i.e. assumptions on the implicit knowledge, without discussing/validating them first. Interaction with the customer representative is key for favoring the explicit expression of knowledge that is usually implicit in requirement documents, including competency questions. Furthermore, the customer representative should be able to describe what tasks the application involving the ontology is expected to solve.
- **Customer stories, Competency Questions (CQs), and contextual statements.** The ontology requirements and its tasks are described in terms of small stories by the customer representative. Designers work on those small stories and, together with the customer, transform them in the form of CQs and contextual statements. Contextual statements are accompanying assertions that explicit knowledge that is typically implicit in CQs. CQs and contextual statements will be used through the whole development, and their definition is a key phase as the designers have the challenge to help the customer in making explicit as much implicit knowledge as possible.
- **CP reuse and modular design.** If there is a CP's GUC that matches a LUC it has to be reused, otherwise a new module i.e. a CP with its GUC, is defined based on the LUC under development and shared with the team (and ideally on the Web[3]).
- **Collaboration and Integration.** Integration is a key aspect of XD as the ontology is developed in a modular way. Collaboration and constant sharing of knowledge is needed in a XD setting, in fact similar or even the same CQs and sentences can be defined for different stories. When this happens, it is important e.g. that the same CP is reused.

[3] For example on http://ontologydesignpatterns.org

– **Task-oriented design** The focus of the design is on that part of the domain of knowledge under investigation that is needed in order to address the user stories, and more generally, the tasks that the ontology is expected to address. This is opposed to the more philosophical approach of formal ontology design where the aim is to be comprehensive with respect to a certain domain.
– **Test-driven design.** Stories, CQs, and contextual statements are used in order to develop unit tests. A new story can be treated only when all unit tests associated with it have been passed. This aspect enforces the task-oriented approach of the method. It has to be noticed that in this context, "unit tests" have a completely different meaning with respect to software engineering unit tests. An ontology module developed for addressing a certain user story associated to a certain competency question, is tested e.g. (i) by encoding in the ontology[4] a sample set of facts based on the user story, (ii) defining one or a set of SPARQL queries that formally encode the competency question, (iii) associating each SPARQL query with the expected result, and (i) running the SPARQL queries against the ontology and compare actual with expected results. Unit tests for ontologies have been analyzed already in [13], where the focus is more on purely logical structures. We leave the investigation of unit test types, and their employment in XD, at future developments.
– **Pair design.** The team of designers is organized in pairs. At least one pair is in charge of integrating ontology modules.

The next section shows details of the XD iterative workflow.

2.1 XD iterative workflow

Figure 2 shows the workflow of XD with CPs. In this section we will describe the single tasks with the help of a simplified scenario coming from a real case study in the Fishery domain. XD is an incremental, iterative method for pattern-based ontology development. Before entering the details of each single task, it is worth to make few premises. The team of designers is organized in pairs that work in parallel. At least one pair is in charge of integrating the modules produced by the other pairs, in order to obtain incremental releases of the ontology. A wiki for the project is set up with a basic structure able to collect customer stories and their associated modeling choices, testing documentation, and contextual statements. The wiki will be used in order to build incrementally the project documentation. During the development, and in particular for testing purposes, an ontology module containing instances according to the customer stories is created and shared. This module is used in order to run unit tests against the ontology.

[4] According to the common way of using the term "ontology", in this context we do not distinguish between TBox and ABox, or ontology and knowledge base. Here, an ontology includes also facts.

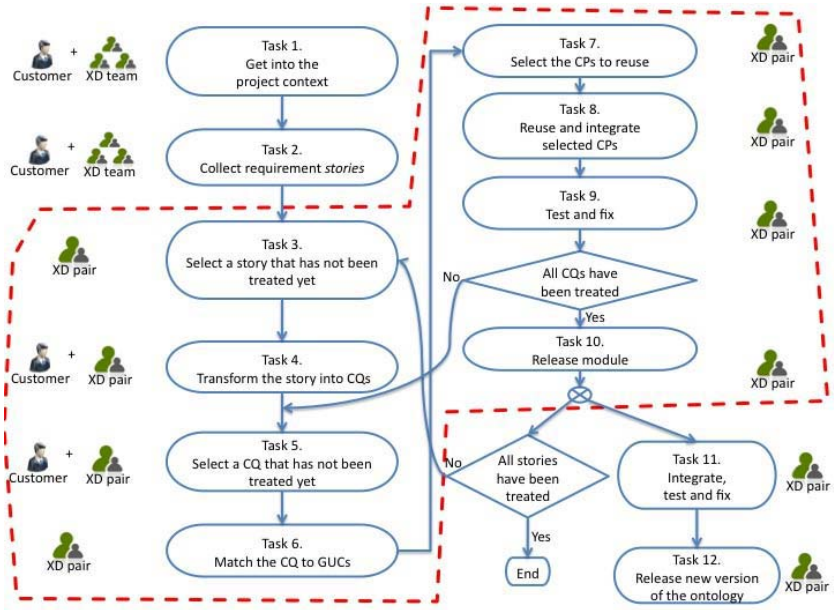


Fig. 2. The XD iterative workflow.

Task 1. Get into the project context. The development starts with a group of designers and a group of domain experts i.e. the customer representative. In principle they do not know much about each other, do not have a precise idea of what will be the result of the project, are used at different terminology, and have a different background. This task has a twofold objective: (i) make the customer representative aware of the method and tools that will be applied during the project, (ii) provide the designer team with an overview of the problem, its scope, and initial terminology.
The result of this task is the setting up of a collaborative environment where customer and designers will share documentation and argument about modeling issues, including terminology i.e. a wiki for the project.

Task 2. Collect requirement stories. The customer representative is invited to write stories, possibly from real, documented scenarios, that samples the typical facts that should be stored in the resulting ontology[5]. All stories are organized in terms of priority, and possible dependencies between them are identified and made explicit[6]. Each story is described by means of a small card,

[5] We do not distinguish between TBox and ABox, ontology and knowledge base. With the term ontology we encompass both according with the current trend in the Semantic Web community.

[6] E.g., a story can be modeled only if another story already has been successfully addressed

like the one depicted in Table 1, which includes the story's title, a list of other stories which it depends on, a description in natural language, and a priority value[7]. It is important to notice that this task is not intended to be performed only once during the project. Stories can be added by the customer during the whole project life cycle. For example, if a new requirement emerges new stories can be written.

Task 3. Select a story that has not been treated yet. Each pair of designers selects a story that will be the focus of their work for the next iteration. A new wiki page for the story is created: the name of the page is the title of the story, and its content is set up based on the information that are in the card. By performing this task a pair enter a development iteration. For example, consider that a pair has selected the story described by the card in Table 1.

Table 1. A requirement story card. It includes the story's title, a list of other stories which the story depends on, a natural language description, and a priority value.

Title	Tuna observation
Depends on	Exploitation values, Tuna areas
Description	In 2004 the resource of species "Tuna" in water area 24 was observed to be fully exploited in the tropical zone at pelagic depth.
Priority	High

Task 4. Transform the story into CQs. The pair process the story and from it derive a set of CQs. In order to do that, designers could involve the customer for having feedback/clarifications. First the story is split into simple sentences, meaning that complex example sentences may be broken up into shorter sentences to increase clarity. The sentences are abstracted so that they describe a class of facts instead of a specific one. The sentences are then transformed into CQs. For example, the story "Tuna observation" is transformed to the following CQs[8], which are added in the story wiki page:

- CQ_1: What are the exploitation state and vertical distance observed in a given climatic zone for a certain resource?
- CQ_2: What resources have been observed during a certain period in a certain water area?

Additionally, the following contextual statement is derived from the discussion with the customer representative[9]:

- A resource contains one or more species.

[7] Priority values are assigned by designers based on interaction with the customer representative. The values can vary and depends on project's conventions.

[8] The elaboration of the story is a complex cognitive procedure which is not explained here. It would deserve a dedicated discussion which is out of scope of this paper.

[9] There would be additional ones, we have simplified for the sake of brevity.

– Species are associated to vertical distances. As a consequence, the vertical distance of a resource is inferred through the vertical distance of the species.

Contextual statements are listed in a dedicated wiki page, and are handled by the pair in charge of the integration task.

Task 5. Select a CQ that has not been treated yet. The iteration continues by selecting one of the CQs. For example, CQ_1.

Task 6. Match the CQ to GUCs. This task has the aim of identifying candidate CPs based on the CQ, which express part of the LUC. The matching procedure can be done either with some tool support e.g., keyword based searching, or manually e.g., if the designers have a good knowledge of available CPs. We here assume that designer manually perform the matching against the ontologydesignpatterns.org repository [4] of CPs. In our example, candidate CPs are: *situation*, and *time interval*All of them are available on http://ontologydesignpatterns.org. The competency question of *situation "What entities are in the setting of a certain situation?"* can be said to match the observation, the resource, and the parameters that are in the setting of that observation. Additionally, the *time interval* CP may be seen as partially matching the question of what period a certain observation was made, although this could also be solve with just a simple datatype property. The CP contains CQs such as: *"What is the end time of this interval?, What is the starting time of this interval?, What is the date of this time interval?"*. The result of this task is then two matching CPs.

Task 7. Select the CPs to reuse. The goal of this task is to select which of those patterns should be used for solving the modeling problem. We may decide that *time interval* adds too much extra effort, besides the needed year of observation, in which case we will only select *situation*.

Task 8. Reuse and integrate selected CPs. The term "reuse" here refers to the application of typical operation that can be applied to CPs i.e. import, specialization, and composition [7]. In our example, we specialize *situation* in order to address CQ_1. The particular situation is in our case the observation, and the thing observed is the resource. Additionally, the exploitation state, climatic zone, and vertical distance of the observation, are also involved in the setting. Thereby, we add a subclass of `situation:Situation`[10] named `AquaticResourceObservation`, and add the other entities as subclasses of `owl:Thing`. In addition, we construct subproperties of the `situation:isSettingFor` and its inverse `situation:hasSetting`, for connecting the observations to the resources and the different parameters. The result is shown with a UML diagram in Figure 3. In this case we have shown a simplified example where only one CP has been reused and specialized. In other cases, we might reuse more CPs. Each of them would be first specialized then integrated with the others. The process

[10] The prefix "situation:" is for http://ontologydesignpatterns.org/cp/owl/situation.owl, while "Situation" is a class defined in the *situation* CP.

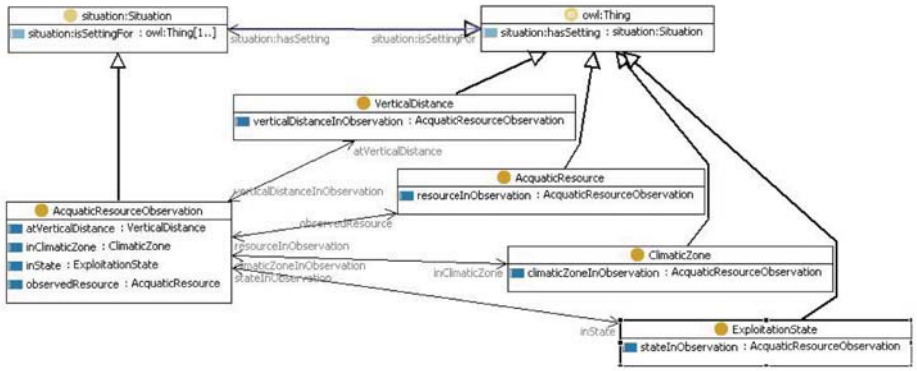


Fig. 3. The *acquatic resource observation* ontology module that specializes the *situation* CP.

that is typically performed during this task is sketched in Figure 4.

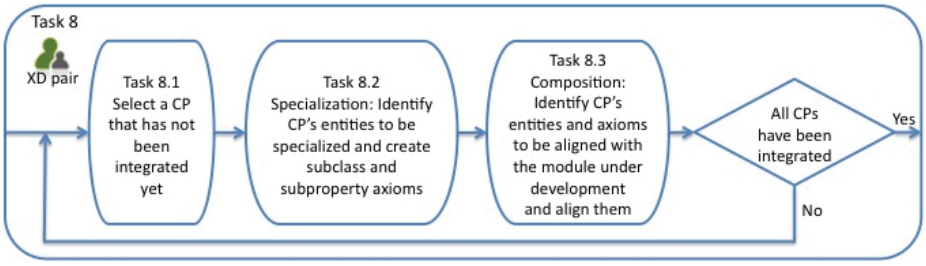


Fig. 4. The process performed in order to execute Task 8 "Reuse and integrate selected CPs".

Task 9. Test and fix. The goal of this task is to validate the resulting module with respect to the CQ just modeled. To this aim, the task is executed through the following steps: (i) the CQ is elaborated in order to derive a unit test e.g., SPARQL query; (ii) the instance module is fed with sample facts based on the story; (iii) the unit test is ran against the ontology module. If the result is not the expected one i.e. the test is not passed, the module is revised in order to fix it, and the unit test ran again until the test is passed; (iv) run all other unit tests associated with the story so far until they all pass. Notice that all unit tests are described in dedicated wiki pages that are properly linked to the associated story. If all CQs associated to the story have been addressed, the pair can pass to Task 10, otherwise they "go back" to Task 5. In our example, the unit test associated to CQ_1 is the following:

```
SELECT ?exp ?dist ?resource ?zone
WHERE {
?obs a :AquaticResourceObservation .
?obs observedResource ?resource .
?obs inClimaticZone ?zone
?obs inState ?exp .
?obs atVerticalDistance ?dist
}
```

Task 10. Release module. This task identifies the end of an iteration for a pair and its result is an ontology module. Once a whole story has been addressed, and the resulting module has been successfully tested, the new module can be released. The module is assigned with a URI and published in order to be shared by the whole team. If the module can be publicly shared, it can be published in open Web repositories such as ontologydesignpatterns.org. The module is then passed to the pair in charge of the integration. The pair of designer selects a new story if there are still some unaddressed.

Task 11. Integrate, test and fix. Once a new module is released, it has to be integrated with all the others that constitute the current version of the ontology. At least one pair is in charge of performing integration and related tests: new unit tests are defined for the integration, and all existing ones are again executed as regression tests before moving to next task. In this task, all contextual statements are taken into account and all necessary alignment axioms are defined. The module is now under the complete control and editing of the pair in charge of the integration. The products of this tasks are new unit tests and alignment axioms, all properly documented in the wiki.

Task 12. Release new version of the ontology. Once all unit tests have been passed, a new version of the ontology can be released.

3 XD tools for the NeOn toolkit

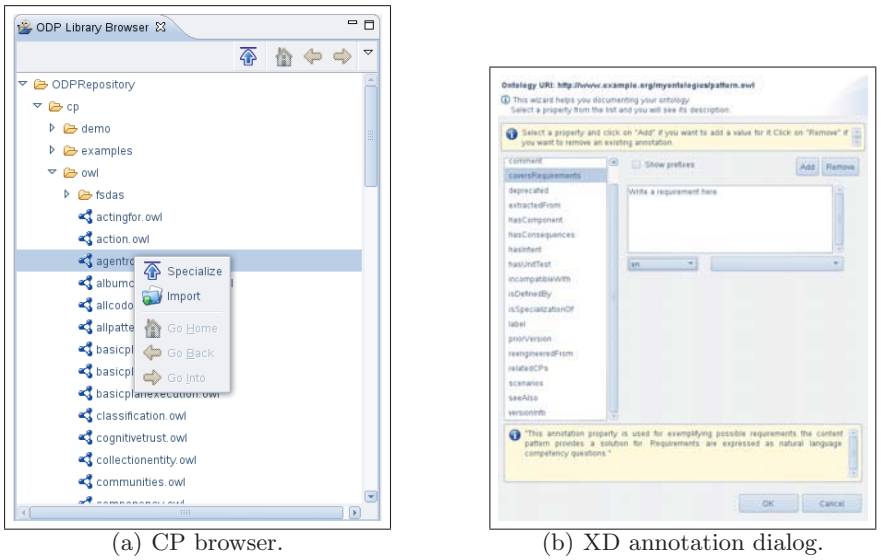
The **eXtreme Design** plugin for NeOn Toolkit[11] (XD tool) supports pattern-based ontology design. Its current version[12], focusses on XD with CPs, and supports some of the tasks described in Section 2. The main goal of XD tool is to improve the user experience with ontology design editors by providing them with a new interaction approach to ontology design. Instead of performing language-oriented operations e.g. instantiate a class, define a subclass, etc., the designer handles "pieces" of ontologies i.e. CPs, and is helped in performing modular design and applying design good practices.
The tool provide an Eclipse perspective, named "eXtreme Design", that includes

[11] http://www.neon-toolkit.org
[12] Available at http://stlab.istc.cnr.it/stlab/Download

the following components: *CP browser and CP details view*, *XD annotation dialog*, *XD selector*, *XD assistant*, *XD wizards*. In the following sections, these components are described in detail.

3.1 The CP browser and CP details view

The CP browser relies on a remote connection to registries of CPs. By default, XD tool allows to browse all CPs available at ontologydesignpatterns.org. The repository of CPs is visualized according to a semantic description based on the *codolight*[13] ontology. This approach makes the XD tool able to easily add new repositories to the browser, once it is provided with a codolight-based OWL description of them. This view allows the user to browse and select CPs as shown in Figure 5(a). The CP details view shows all available annotations of the selected CP. From the CP browser, a CP can be specialized and imported in the ontology under development.



(a) CP browser. (b) XD annotation dialog.

Fig. 5. XD browser and annotation dialog.

3.2 XD annotation dialog

The XD annotation dialog, shown in Figure 5, supports multilingual annotation of ontology modules or CPs. This dialog supports a number of default vocabularies, and custom ones can be added. One of the vocabularies available by

[13] http://ontologydesignpatterns.org/cpont/codo/codolight.owl

default is the CP annotation schema[14], an OWL ontology particularly suited for annotating CPs.

3.3 XD selector: pattern selection support

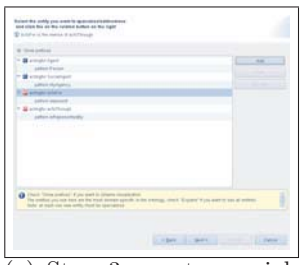
The XD selector provides the infrastructure for plugging into XD a component implementing a matching/searching algorithm that starting from a CQ gives as output a selection of candidate CPs. Currently only the APIs have been implemented, we are working at two proofs of concept components: an instance of Watson [10] specific for CPs, and a new version of OntoCase [2].

3.4 The XD assistant: support for design good practices

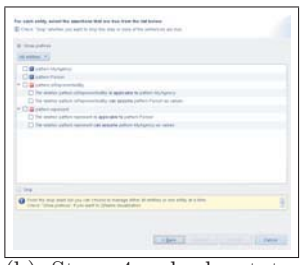
This component is able to provide the user with feedback related to possible mistakes and suggestions on good practices in a certain modeling situation. The XD assistant has a plugin-based architecture that make it easy to extend the help elements based on new emerging good/bad practices.
The XD assistant communicates two types of messages to the user: (i) **Warnings**: these messages are visualized when there is a strong suspect of wrong design. E.g. there is an anti-pattern in the module; (ii) **Suggestions**: these messages are visualized in order to suggest axioms currently missing in the module, that could improve the design.

3.5 XD wizards

XD also features a set of wizards. At the moment it includes a wizard for supporting CP specialization. The wizard can be accessed in the CP browser view,



(a) Step 3: create specialized entities.



(b) Step 4: check statements that are true.

Fig. 6. Some steps of XD specialization wizard

by the command "specialized" included in the contextual menu that can be activated by right.clicking on the CP to be specialized (selected from the repository).

[14] http://ontologydesignpatterns.org/schemas/cpannotationschema.owl

The wizard guides the user through the following steps:

Step 1. The user selects the project and the target ontology (if any). Additionally, the user has to check one of three possible results that can be produced by the wizard:(i) *Create a new module/CP and import it in the target ontology*; (ii) *Create a new module/CP and store it in the indicated project*; (iii) *Merge the resulting entities in the target ontology.*

Step 2. All entity leaves are displayed to the user that is invited to select the entities to be specialized.

Step 3. For each selected entity the user creates the new specific one as shown in Figure 6(a).

Step 4. The wizard suggests possible axioms that can be added to the ontology by means of natural language statements. The users can automatically produce these additional axioms by selecting the assertions that they consider "true" in their context. This step of the wizard is depicted in Figure 6(b).

Step 5. Finally, an overview of natural language assertions corresponding to the formal axioms stated in the developed module is shown. This step has the aim of giving the users the possibility to review the result before to produce the module. They can always go back to a certain step in order to fix possible mistakes.

4 Conclusion and future work

In this paper, we have discussed **eXtreme Design (XD)**, an approach to ontology design based on the application, exploitation, and definition of ontology design patterns (ODPs). In more detail, we have presented two main contributions: a collaborative, incremental, and iterative method for pattern-based ontology design; and a first version of the XD tool, a NeOn Toolkit plugin that currently supports CP repository browsing and selection, a good practice assistant, and a wizard for CPs' specialization.
XD main principles are collaboration, integration, testing, and the extensive use of CPs. XD has been inspired by good practices typically adopted by the eXtreme Programming (XP) [11] software development methodology, such as pair programming, and customer involvement. While a rigorous evaluation of the whole methodology is still in our future plans, the effectiveness of CPs in ontology design has been rigorously evaluated in [5] where, however, XD has been used as reference development guidelines. Furthermore, initial questionnaires and informal discussions made emerge that the perception of the trainees with respect to the method is positive.
In order to ease the execution of the method some automatic support is needed. For example, matching CQs with GUCs is a complex task and automatic support

for filtering candidate CPs is necessary in order to exploit at best the evolving repositories of CPs. For addressing this type of needs, we have developed a NeOn Toolkit plugin, named "XD tool", that currently support CP browsing, annotation, and specialization. Furthermore, it provides APIs for pluggin-in components that implement matching algorithms. A lot of work is still ongoing, including the development of two components supporting matching/selection of CPs. As future work, we have planned to continue with XD tool development, and to conduct frequent user studies in order to evaluate and improve the methodology as well as to collect feedback and suggestions on the XD tool.

References

1. V. R. Basili, G. Caldiera, and D. Rombach. *Experience Factory*, pages 469–476. Wiley & Sons, 1994.
2. E. Blomqvist. Ontocase - automatic ontology enrichment based on ontology design patterns. In A. Bernstein and D. Karger, editors, *To appear in Proceedings of the 8th International Semantic Web Conference (ISWC 2009)*, 2009.
3. P. Clark and B. Porter. Building concept representations from reusable components. In *Proceedings of AAAI'97*, pages 369–376. AAAI press, 1997.
4. E. Daga, V. Presutti, and A. Salvati. http: //ontologydesignpatterns.org and evaluation wikiflow. In A. Gangemi, J. Keizer, V. Presutti, and H. Stoermer, editors, *SWAP*, volume 426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
5. Eva Blomqvist and Aldo Gangemi and Valentina Presutti. Experiments on Pattern-based Ontology Design. In *The Fifth International Conference on Knowledge Capture*, 2009.
6. A. Gangemi. Ontology Design Patterns for Semantic Web Content. In *Proceedings of the 4th International Semantic Web Conference*, pages 262–276. Springer, 2005.
7. V. P. A. Gangemi. Ontology design patterns. In R. S. S. Staab, editor, *Handbook of Ontologies*, International Handbooks on Information Systems. Springer, 2nd edition, 2009.
8. M. Gruninger and M. Fox. The role of competency questions in enterprise engineering, 1994.
9. S. Pinto, S. Staab, and C. Tempich. DILIGENT: Towards a fine-grained methodology for Distributed Loosely-controllled and evolvInG Engineering of oNTologies. In *Proceedings of ECAI-2004*, 2004.
10. M. Sabou, C. Baldassarre, L. Gridinoc, S. Angeletou, E. Motta, M. d'Aquin, and M. Dzbor. Watson: A gateway for the semantic web. In *ESWC 2007 poster session*, June 2007-06.
11. J. Shore and S. Warden. *The Art of Agile Development*. O'Reilly, Sebastopol, CA, USA, 2007.
12. C. G. von Wangenheim, K.-D. Althoff, and R. M. Barcia. Goal-oriented and similarity-based retrieval of software engineering experienceware. In G. Ruhe and F. Bomarius, editors, *SEKE*, volume 1756 of *Lecture Notes in Computer Science*, pages 118–141. Springer, 1999.
13. D. Vrandečić and A. Gangemi. Unit tests for ontologies. In M. Jarrar, C. Ostyn, W. Ceusters, and A. Persidis, editors, *Proceedings of the 1st International Workshop on Ontology content and evaluation in Enterprise*, LNCS, Montpellier, France, October 2006. Springer.