Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies

Valentina Presutti and Aldo Gangemi

ISTC-CNR, Semantic Technology Lab, Italy
{valentina.presutti,aldo.gangemi}@istc.cnr.it

Abstract. In this paper we present how to extract and describe emerging content ontology design patterns, and how to compose, specialize and expand them for ontology design, with particular focus on Semantic Web technologies. We exemplify the described techniques with respect to the extraction of two content ontology design patterns from the DOLCE+DnS Ultra Lite ontology, and by showing the design of a simplified ontology for the music industry domain.

1 Introduction

Computational ontologies in the context of information systems are artifacts that encode a description of some world (actual, possible, counterfactual, impossible, desired, etc.), for some purpose. They have a (primarily logical) structure, and must match both domain and task: they allow the description of entities whose attributes and relations are of concern because of their relevance in a domain for some purpose, e.g. query, search, integration, matching, explanation, etc.

Like any artifact, ontologies have a lifecycle: they are designed, implemented, evaluated, fixed, exploited, reused, etc. In this paper, we focus on patterns for ontology design [9,11].

Today, one of the most challenging and neglected areas of ontology design is *reusability*. The possible reasons include at least: *size* and *complexity* of the major reusable ontologies, *opacity* of design rationales in most ontologies, *lack of criteria* in the way existing knowledge resources (e.g. thesauri, database schemata, lexica) can be reengineered, and *brittleness* of tools that should assist ontology designers.
On this situation, an average user that is trying to build or reuse an ontology, or an existing knowledge resource, is typically left with limited assistance in using unfriendly logical structures, some large, hardly comprehensible ontologies, and a bunch of good practices that must be discovered from the literature. On the other hand, the success of very simple and small ontologies like FOAF [5] and SKOS [18] shows the potential of really portable, or "sustainable" ontologies. The lesson learnt supports the new approach to ontology design, which is sketched here.

Under the assumption that there exist classes of problems that can be solved by applying common solutions (as it has been experienced in software engineering), we propose to support reusability on the design side specifically. We envision small ontologies with explicit documentation of design rationales, and best reengineering practices.

Q. Li et al. (Eds.): ER 2008, LNCS 5231, pp. 128–141, 2008.
© Springer-Verlag Berlin Heidelberg 2008

These components need specific functionalities in order to be implemented in repositories, registries, catalogues, open discussion and evaluation forums, and ultimately in new-generation ontology design tools. In this paper, which is a result of the evolution of work described in [9], we describe small, motivated ontologies that can be used as practical *building blocks* in ontology design. A formal framework for (collaborative) ontology design that justifies the use of building blocks with explicit rationales is presented in [11].

We call the practical building blocks to be used in ontology design *Content Ontology Design Patterns* (CP, [9]). CPs encode *conceptual*, rather than *logical* design patterns. In other words, while Logical OPs [23] (like those investigated by [22]) solve design problems independently of a particular conceptualization, CPs propose patterns for solving design problems for the domain classes and properties that populate an ontology, therefore addressing *content* problems [9]. CPs exemplify Logical OPs (or compositions of Logical OPs), featuring a non-empty signature. Hence, they have an explicit non-logical vocabulary for a specific domain of interest (i.e. they are content-dependent). For example, a simple **participation** pattern (including objects taking part in events) emerges in domain ontologies as different as enterprise models [13], sofware management [20], and biochemical pathways [10]. Other, more complex patterns have also emerged in the same disparate domains.

CPs are strictly related to small use cases i.e., each of them is built out of a domain task that can be captured by means of competency questions [13]. A competency question is a typical query that an expert might want to submit to a knowledge base of its target domain, for a certain task. Moreover, CPs are transparent with respect to the rationales applied to the design of a certain ontology. They are therefore an additional tool to achieve tasks such as ontology evaluation, matching, modularization, etc.

For example, an ontology can be evaluated against the presence of certain patterns (which act as *unit tests* for ontologies, cf. [28]) that are typical of the tasks addressed by a designer. Furthermore, mapping and composition of patterns can facilitate ontology mapping: two ontologies drafted according to CPs can be mapped in an easier way: CP hierarchies will be more stable and well-maintained than local, partial, scattered ontologies. Finally, CPs can be also used in training and educational contexts for ontology engineers.

The paper is organized as follows: section 1.1 gives some background notions; section 2 defines the notion of CP, and briefly describes the online repository and Web portal; section 3 provides methodological guidelines for creating and reusing CPs, presents two of them, and an example of reuse. Finally, section 4 provides some conclusions and remarks.

1.1 Background

Ontology engineering literature has tackled the notion of design pattern at least since [6], while in the context of Semantic Web research and application, where ontology design patterns (OPs) are now a hot topic, the notion has been introduced by [10,24,26]. In particular, [10,26] take a foundational approach that anticipates that presented in [9]. Some work [4] has also attempted a learning approach (by using case-based reasoning) to derive and rank patterns with respect to user requirements. The research has also

addressed domain-oriented patterns, e.g. for content objects and multimedia [2], software components [20], business modelling and interaction [12,15], relevance [17] etc.

Throughout experiences in ontology engineering projects[1] in our Laboratory, as well as in other ongoing international projects that have experimented with these ideas, typical conceptual patterns have emerged out of different domains, for different tasks, and while working with experts having heterogeneous backgrounds. For an historical perspective and a more detailed survey, the reader can refer to [1,9,12,14,16]

2 Content Ontology Design Patterns (CPs)

Content ontology design patterns (CPs) are reusable solutions to recurrent content modelling problems. In analogy to conceptual modeling (cf. the difference between class and use case diagrams in the Unified Modeling Language (UML) [21]) and knowledge engineering (cf. the distinction between domain and task ontologies in the Unified Problem-solving Method Development Language (UPML) [19]), these problems have two components: a domain and a use case (or task). A same domain can have many use cases (e.g. different scenarios in a clinical information context), and a same use case can be found in different domains (e.g. different domains with a same "competence finding" scenario). A typical way of capturing use cases is by means of *competency questions* [13]. A competency question is a typical query that an expert might want to submit to a knowledge base of its target domain, for a certain task. In principle, an accurate domain ontology should specify *all and only* the conceptualizations required in order to answer the competency questions formulated by, or acquired from, experts.

Based on the above assumptions, we define what a Content Ontology Design Pattern (CP) is:

> *CPs are distinguished ontologies. They address a specific set of competency questions, which represent the problem they provide a solution for. Furthermore, CPs show certain characteristics i.e., they are: computational, small and autonomous, hierarchical, cognitively relevant, linguistically relevant, and best practices.*

According to [9], such characteristics can be described as follows:

- *Computational components.* CPs are language-independent, and should be encoded in a higher-order representation language.[2] Nevertheless, their (sample) representation in OWL is needed in order to (re)use them as building blocks over the Semantic Web.
- *Small, autonomous components.* Regardless of the particular way a CP has been created (section 3.1 describes how to create a CP), it is a *small, autonomous* ontology.

[1] For example, in the projects *FOS*: http://www.fao.org/agris/aos/, *WonderWeb*: http://wonderweb.semanticweb.org, *Metokis*: http://metokis.salzburgresearch.at, and *NeOn*: http://www.neon-project.org

[2] Common Logic (see http://cl.tamu.edu/) is a good candidate because of its expressivity and computationally-sound syntax.

Smallness (typically two to ten classes with relations defined between them) and autonomy of CPs facilitate ontology designers. Smallness also allows diagrammatical visualizations that are aesthetically acceptable and easily memorizable.
- *Hierarchical components.* A CP can be an element in a partial order, where the ordering relation requires that at least one of the classes or properties in the pattern is specialized.
- *Inference-enabling components.* A CP allows some form of inference e.g. a taxonomy with two sibling disjoint classes, a property with explicit domain and range set, a property and a class with a universal restriction on that property, etc.
- *Cognitively relevant components.* CP visualization must be intuitive and compact, and should catch relevant, "core" notions of a domain. [9]
- *Linguistically relevant components.* Many CPs nicely match linguistic patterns called *frames*. A frame can be described as a lexically founded ontology design pattern; The richest repository of frames is FrameNet [3].
- *Best practice components* A CP should be used to describe a "best practice" of modelling. Best practices are intended here as *local*, thus derived from experts, emerging from real applications.

A Catalogue and Repository of CPs. The above definition provides ontology designers with the necessary means to identify CPs within existing ontologies. However, we believe it is important for reuse purpose, to have a repository of CPs and related services, where CPs can be added and retrieved, and to guarantee that published CPs have a high level of quality.

With the above principles in mind, we have set up the Ontology Design Patterns Web portal[3] (ODPWeb), where CPs are collected, classified, described with a specific template, and available for download. They respond to a common specification (which extends the above CP definition), and are described in terms of a template, which is inspired by the well known one used for Software Engineering design patterns [7]. The Web portal is open to contribution from any user, who is only required to register in order to have authoring rights. ODPWeb is intended as a space where ontology designers, practitioners, and Semantic Web users can discuss about web ontology design issues, find information about good practices, and download reusable components for building web ontologies. Moreover, the ODPWeb is associated with a lightweight (peer reviewing) workflow, which guarantees both quality of the published CPs and openness of the community.

3 CP Creation and Usage

Content Ontology Design Pattern (CP) creation and usage rely on a common set of operations.

- *import:* as with any ontology, it consists of including a CP in the ontology under development. This is the basic mechanism for CP reuse. Elements of a CP cannot be modified.

[3] http://www.ontologydesignpatterns.org.

- *clone:* consists of duplicating an ontology element i.e., a class and a property, which is used as a prototype[4]. We can distinguish among three kinds of clones:
  - *shallow clone*: consists of creating a new ontology element O' by duplicating an existing ontology element O. Axioms of O and O' will refer to the same ontology elements.
  - *deep clone*: consists of creating a new ontology element O' by duplicating an existing ontology element O, and by creating a new ontology element for each one that is referred in O's axiomatization, recursively.
  - *partial clone*: consists of deep cloning an ontology element, by keeping only a subset of its axioms.
- *specialization:* can be referred to ontology elements or to CPs. Specialization between ontology elements of a CP consists of creating sub-classes of some CP's class and/or sub-properties of some CP's properties. A CP c' is a specialization of a CP c, if at least one ontology element of c' specializes an ontology elements of c, and all ontology elements of c' are either a specialization of ontology elements of c, or clones of them.
- *generalization:* is the reverse of the specialization operation.
- *composition:* consists of associating classes (properties) of one CP with classes (properties) of other CPs by subsumption, by creating new owl restrictions, or by creating new properties.
- *expansion:* consists of enriching an ontology with ontology elements and axioms, which do not identify any CP or composition of them.

3.1 CP Creation

CPs mainly emerge either from ontologies (i.e., *foundational*, *core*, and *domain* ontologies)[5] or by reengineering other types of conceptual models (e.g. E-R models, UML models, linguistic frames, thesauri, etc.) to ontologies. CPs can be defined in four main ways:

- *Reengineering from other data models* A CP can be the result of a reengineering process applied to different conceptual modeling languages, primitives, and styles. [12] describes a reengineering approach for creating CPs starting from UML diagrams [21], workflow patterns [27], and data model patterns [16].
  Other knowledge resources that can be reengineered to produce candidate CPs are database schemas, knowledge organization systems (e.g. thesauri), and lexica for reengineering techniques on these resources). The reader can refer to [12] for more references.
- *Specialization/Composition of other CPs* A CP can be created either by composition of other CPs or by specialization of another CP, (both composition and specialization can be combined with expansion).
- *Extraction from reference ontologies* A CP can be *extracted from* an existing ontology, which acts as the "source" ontology. Extraction of a CP is a process consisting of (partial) cloning the ontology elements of interest from the source ontology.
- *Creation by combining the above techniques.*

[4] There is a strong analogy between the clone operation in OO software programming and the ontology element clone operation.

[5] see [9] for references.

Figure 1 shows the typical process that is performed by an ontology engineer for creating a CP by extraction from a reference ontology, possibly including specialization and expansion. The creation of a CP starts with the creation of a new ontology to which a suitable namespace is assigned. Each pattern has its own namespace that does not depend on that of the source ontology. The source ontology(ies) is(are) then imported. Elements of the source ontology must not be modified. Some tools allow designers to modify imported ontologies, when they are locally stored and writable. In such a case, it is a good practice to lock the imported ontologies in order to set access permissions to read-only.

The creation proceeds with the partial cloning of the ontology elements i.e., classes and properties, of interest. Some ontology design tools support the shallow clone operation [6], while deep clone and partial clone are not yet supported by any existing tool. Currently, in order to obtain a partial or deep clone of an ontology element we can either start from a shallow clone (when supported), or we can write a SPARQL `CONSTRUCT` query, and then manually update the results. For example, the SPARQL expression (1) allows us to extract the class `DUL:Agent` and its associated axioms from the source ontology, and to create the class `Agent` as a shallow clone of it. Within the results provided by the SPARQL engine, we can choose which axioms we want to keep. With this procedure, the selected axioms will still contain ontology elements from the source ontology. Therefore, we have to manually update such axioms in order to substitute those elements with new cloned ones.

$$\begin{aligned} &CONSTRUCT\,\{\,:Agent\ ?r\ ?y\,\} \\ &WHERE\,\{\,DUL{:}Agent\ ?r\ ?y\,\} \end{aligned} \tag{1}$$

After all elements of interest have been cloned and updated, optional specialization and/or expansion is performed. At this point, possible disjointness axioms are introduced before launching the reasoner for consistency checking, and for inferences, some of which might be explicitly asserted. Finally, the imports are removed and the CP and its elements are annotated.

CPs that are published on ODPWeb are annotated by means of the *cp annotation schema*[7].

3.2 The Information Realization CP

In this section we describe a CP that is named **information realization**. It is created by extraction from the Dolce Ultra Lite ontology[8], and represents the relations between information objects like poems, songs, formulas, etc., and their physical realizations like printed books, registered tracks, physical files, etc.. We also show how it is extracted, and provide the main information according to that contained in its associated catalogue entry.

Figure 2 depicts some screenshots of the ontology editor while we extract the information realization CP. The arrows indicates the ontology element that we clone i.e.,

[6] e.g., TopBraid Composer available at http://www.topbraidcomposer.com/

[7] http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl

[8] http://www.loa-cnr.it/ontologies/DUL.owl

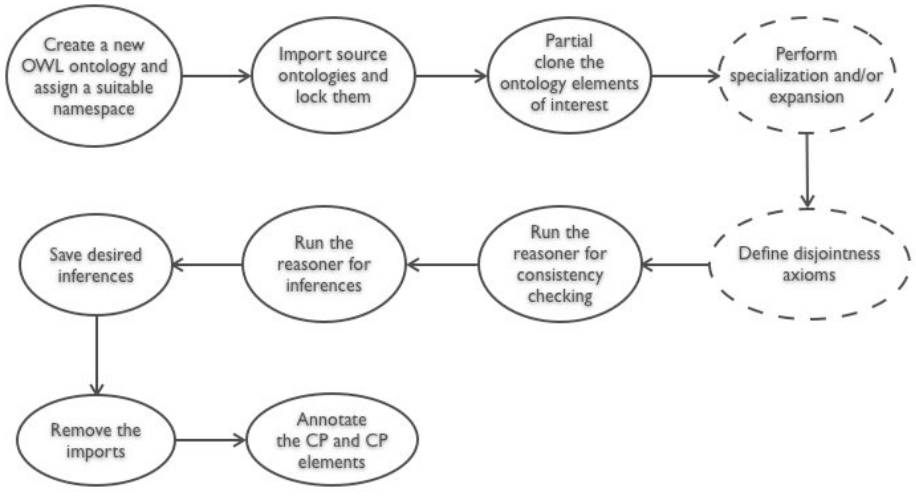

Fig. 1. The CP creation by extraction process. Circles with dashed lines indicates steps that can be skipped.

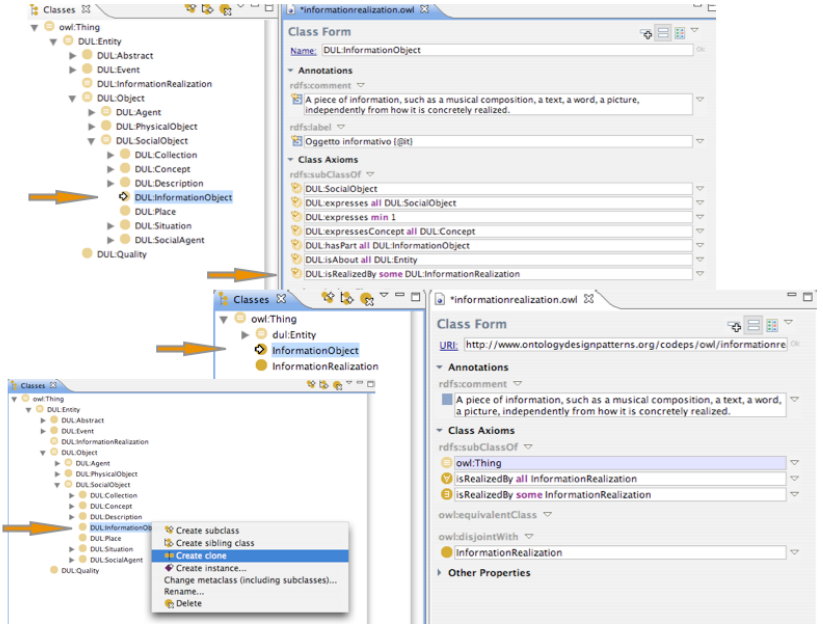

Fig. 2. The information realization CP extraction from Dolce+DnS Ultra Lite ontology. The arrows identify the class `DUL:InformationObject`, the result of its cloning, which is the class `InformationObject`, and the axiom kept and updated from the source class definition.

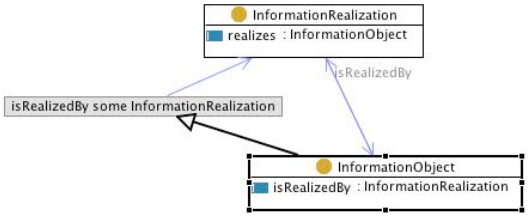

Fig. 3. The information realization CP UML graphical representation

`DUL:InformationObject`[9] The upper part of the picture depicts the axiomatization of `DUL:InformationObject`, in the bottom left part, the (shallow) clone functionality is applied to `DUL:InformationObject`. The clone operation result is a new class belonging to the information realization CP and namespace, named `InformationObject`. We also clone `DUL:InformationRealization` and create `InformationRealization`, and clone the two object properties `DUL:realizes`, and `DUL:isRealizedBy` and create `realizes`, and `isRealizedBy`, object properties. We remove the axioms we do not want to keep, and update the kept ones with the new ontology elements. In the bottom right of the picture is shown the resulting definition of `InformationObject`. It can be noticed that we keep the comment, and the restrictions on the `DUL:isRealizedBy` object property, and update the restricted property to the local cloned one i.e., `isRealizedBy`. We use the same approach for all the other ontology elements. Finally we remove the import and obtain the information realization CP. Figure 3 shows a UML diagram of the information realization CP. The **information realization CP** is associated with information according to the catalogue entry fields reported below:

- **Name:** Information Realization
- **Intent:** Which physical object realizes a certain information object? Which information object is realized by a certain physical object?
- **Extracted from:** The Dolce Ultra Lite ontology available at http://www.loa-cnr.it/ontologies/DUL.owl
- **Examples:** That CD is the recording of *The Dark Side of the Moon*
- **Diagram:** See Figure 3
- **Elements:**
  - `InformationObject:` A piece of information, such as a musical composition, a text, a word, a picture, independently from how it is concretely realized.
  - `InformationRealization:` A concrete realization of an InformationObject, e.g. the written document containing the text of a law.
  - `realizes:` A relation between an information realization and an information object, e.g. the paper copy of the Italian Constitution realizes the text of the Constitution.
  - `isRealizedBy:` A relation between an information object and an information realization, e.g. the text of the Constitution is realized by the paper copy of the Italian Constitution.

[9] DUL is the prefix for the Dolce+DnS Ultra Lite ontology namespace.

- **Consequences:** The CP allows to distinguish between information encoded in an object and the possible physical representations of it .
- **Known uses:** The Multimedia ontology, available at http://multimedia.sematicweb.org/COMM/multimedia-ontology.owl[10] used this CP.
- **Building block:** The CP is available at http://wiki.loa-cnr.it/index.php/LoaWiki:informationrealization

With reference to the complete set of fields that compose the template, here we are missing: the **Also Known as** field, which provides alternative names for the CP; and the **Related CPs** field, which indicates other CPs (if any) that e.g., specialize, generalize, include, are components of, or are typically used with, etc. the CP.

3.3 The Time Indexed Person Role Pattern

The **time indexed person role** is a CP that represents time indexing for the relation between persons and roles they play. This CP is created by combining extraction and specialization. According to its associated catalogue entry, the main information associated with this CP are the following:

- **Name:** Time Indexed Person Role
- **Intent:** Who was playing a certain roles during a given time interval? When did a certain person play a specific role?
- **Extracted from:** The Dolce Ultra Lite ontology available at http://www.loa-cnr.it/ontologies/DUL.owl
- **Examples:** George W. Bush was the president of the United States in 2007.
- **Diagram:** See Figure 4, the elements which compose the CP are described in the **Elements** field.
- **Elements:**
  - `Entity`: Anything: real, possible, or imaginary, which some modeller wants to talk about for some purpose.
  - `Person`: Persons in commonsense intuition, i.e. either as physical agents (humans) or social persons.
  - `Role`: A Concept that classifies a Person
  - `TimeInterval`: Any region in a dimensional space that aims at representing time.
  - `TimeIndexedPersonRole`: A situation that expresses time indexing for the relation between persons and roles they play.
  - `hasRole`: A relation between a Role and an Entity, e.g. 'John is considered a typical rude man'; your last concert constitutes the achievement of a lifetime; '20-year-old means she's mature enough'.
  - `isRoleOf`: A relation between a Role and an Entity, e.g. the Role 'student' classifies a Person 'John'.

[10] Actually the multimedia ontology used a simplified version of Dolce Ultra Lite including classes and properties we have extracted (from the same source ontology) in order to define the CP.

- `isSettingFor`: A relation between time indexed role situations and related entities, e.g. 'I was the director between 2000 and 2005 ', i.e.: the situation in which I was a director is the setting for a the role of director, me, and the time interval.
- `hasSetting`: The inverse relation of `isSettingFor`.

– **Consequences:** The CP allows to assign a time interval to roles played by people.
– **Building block:** The CP is available at http://wiki.loa-cnr.it/index.php/LoaWiki:timeindexedpersonrole

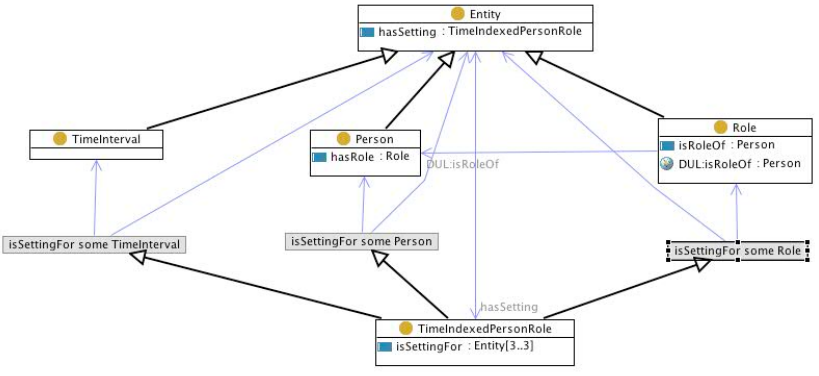

Fig. 4. The time indexed person role CP UML graphical representation

The **time indexed person role** CP is created by combining extraction, specialization, and expansion. The class `TimeIndexedPersonRole` is derived by specializing the Dolce Ultra Lite class `DUL:Classification` (pointed by the blue arrow), while the other elements are partially cloned with the same approach we use for classes and properties of the **information realization** CP.

3.4 CP Usage

Supporting reuse and alleviating difficulties in ontology design activities are the main goals of setting up a catalogue of CPs. In order to be able to reuse CPs, two main functionalities must be ensured: *selection* and *application*.

Selection of CPs corresponds to finding the most appropriate CP for the actual domain modeling problem. Hence, selection includes search and evaluation of available CPs. This task can be performed by applying typical procedures for ontology selection e.g., [25] and evaluation [8].

Informally, *intent* of the CP must match the actual local modeling problem. Once a CP has been selected, it has to be applied to the domain ontology. Typically, application is performed by means of import, specialization, composition, or expansion (see section 3). In realistic design projects, such operations are usually combined.

Several situations of matching between intent of CPs and local domain problem can occur, each associated with a different approach to using CPs. The following summary

assumes a *manual* (re)use of CPs. An automatic support to CP selection and usage should take into account the principles informally explained in the summary below.

- *Precise or redundant matching*. The CP intent perfectly or redundantly matches the local domain problem. The CP is directly usable to describe the local domain problem: the CP only has to be *imported* in the domain ontology.
- *Broader matching*. The CP intent is more general than the local domain problem: the **Generalization Of** field of the CP's catalogue entry, may contain references to less general CPs that specialize it. If none of them is appropriate, the CP has firstly to be *imported*, then it has to be *specialized* in order to cover the domain part to be represented.
- *Narrower matching*. The CP intent matches is more specific than the local domain problem: the `odpschema:specializationOf`[11] property of the CP annotation schema may contain references i.e., URIs, to more general CPs it is the specialization of, the same information is reported in the **Specialization Of** field of the CP's catalogue entry. If none of them is appropriate, the selected CP has firstly to be *imported*, then it has to be *generalized* in order to cover the domain part to be represented.
- *Partial matching*. The CP intent partly matches the local domain problem: the **is Component Of** field of the CP's catalogue entry may contain CPs it is a component of. If none of such compound CPs is appropriate, the local domain problem has to be partitioned into smaller pieces. One of these pieces will be possibly covered by the selected CP. For the other pieces, other CPs have to be selected. All selected CPs have to be *imported* and *composed*. If the local domain problem is not too big, it is worth to propose a new entry to the catalogue of CPs for the resulting composed CP.

An example in the music domain As an example of usage we design a small fragment of an ontology for the music industry domain. The ontology fragment has to address the following competency questions:

- *Which recordings of a certain song do exist in our archive?*
- *Who did play a certain musician role in a given band during a certain period?*

The first competency question requires to distinguish between a song and its recording, while the second competency question highlights the issue of assigning a given musician role e.g., singer, guitar player, etc., to a person who is member of a certain band, at a given period of time. The intent of the **information realization** is related to the first competency question with a *broader matching*. The intent of the **time indexed person role** partially and broadly matches the second competency question. The second requirement also requires to represent membership relation between a person and a band[12]. Let's consider the case that we cannot find more specialized CPs for reusing. We proceed by following the above guidelines. Figure 5 shows a screenshot

[11] odpschema is a prefix for http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl

[12] The **collection entity** CP is about membership relations.

of the resulting ontology fragment. In the bottom part of the screenshot we find the import tab where the **information realization**[13] and **time indexed person role**[14] CPs are imported. Additionally, we import the **time interval** CP that allows us to assign a date to the time interval[15] In order to complete our ontology fragment we create: the class `Song` that specializes `ir:InformationObject`, the class `Recording` that specializes `ir:InformationRealization`, the class `MusicianRole` that specializes `tipr:Role`, the class `Band`, and the object property `memberOf` (and its inverse) with explicit domain i.e., `tipr:Person`, and range i.e., `Band`.

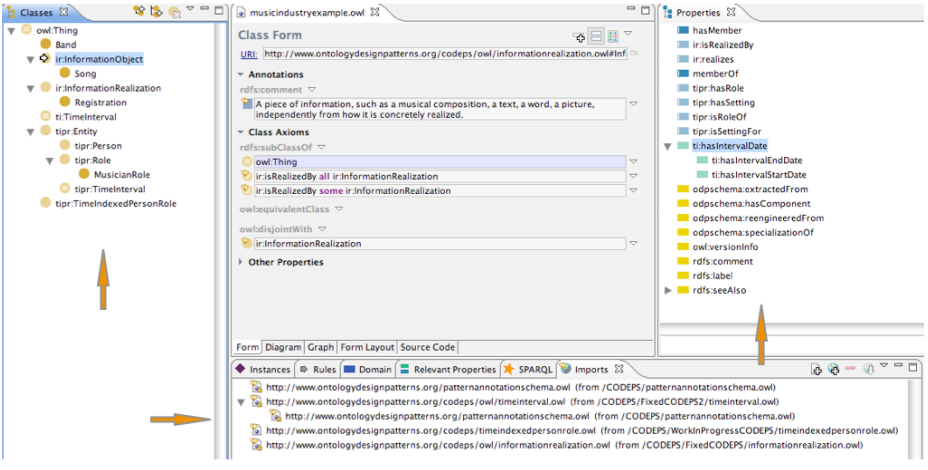

Fig. 5. The music industry example. The arrows indicate the imported CPs (bottom of the figure), and the ontology elements we have specialized (left and right side of the figure).

4 Conclusion and Remarks

Ontology design is a crucial research area for semantic technologies. Many bottlenecks in the wide adoption of semantic technologies depend on the difficulty of understanding ontologies and on the scarcity of tools supporting their lifecycle, from creation to adaptation, reuse, and management. The lessons learnt until now, either from the early adoption of semantic web solutions or from local, organizational applications, put a lot of emphasis on the need for simple, modular ontologies that are accessible and understandable by typical computer scientist and field experts, and on the dependability of these ontologies on existing knowledge resources.[16] In this paper, we have described content ontology design patterns, which are beneficial to ontology design in terms of

[13] We use the prefix `ir` for this CP.

[14] We use the prefix `tipr` for this CP.

[15] The **time interval** CP also defines two additional sub-properties of the `hasIntervalDate` for expressing a start and an end date to the time interval.

[16] References to review work of evaluation, selection and reuse methods in ontology engineering can be found in [12].

their relation to requirement analysis, definition, communication means, related work beyond ontology engineering, exemplification, creation, and usage principles.

We have shown how CPs can be created and reused, and presented two of them as sample entries from a larger catalogue, with an example in the design of a small ontology in the music domain. Finally, we have briefly described our ongoing work a Web portal where designers, practitioners, and users can discuss about, propose, and download content ontology design patterns.

References

1. Alexander, C.: The Timeless Way of Building. Oxford Press (1979)
2. Arndt, R., Troncy, R., Staab, S., Hardman, L., Vacura, M.: Comm: Designing a well-founded multimedia ontology for the web. In: Proceedings of the 4th European Semantic Web Conference (ISCW 2007), Busan Korea, November 2007. Springer, Heidelberg (2007)
3. Baker, C.F., Fillmore, C.J., Lowe, J.B.: The Berkeley FrameNet project. In: Boitet, C., Whitelock, P. (eds.) Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics, pp. 86–90. Morgan Kaufmann Publishers, San Francisco (1998)
4. Blomqvist, E.: Fully automatic construction of enterprise ontologies using design patterns: Initial method and first experiences. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3761, pp. 1314–1329. Springer, Heidelberg (2005)
5. Brickley, D., Miller, L.: Foaf vocabulary specification. Working draft (2005)
6. Clark, P., Thompson, J., Porter, B.: Knowledge patterns. In: Cohn, A.G., Giunchiglia, F., Selman, B. (eds.) KR2000: Principles of Knowledge Representation and Reasoning, pp. 591–600. Morgan Kaufmann, San Francisco (2000)
7. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
8. Gangemi, A., Catenacci, C., Ciaramita, M., Lehmann, J.: Modelling Ontology Evaluation and Validation. In: Proceedings of the Third European Semantic Web Conference. Springer, Heidelberg (2006)
9. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. In: Musen, M., et al. (eds.) Proceedings of the Fourth International Semantic Web Conference, Galway, Ireland. Springer, Heidelberg (2005)
10. Gangemi, A., Catenacci, C., Battaglia, M.: Inflammation ontology design pattern: an exercise in building a core biomedical ontology with descriptions and situations. In: Pisanelli, D.M. (ed.) Ontologies in Medicine. IOS Press, Amsterdam (2004)
11. Gangemi, A., Lehmann, J., Presutti, V., Nissim, M., Catenacci, C.: C-odo: an owl metamodel for collaborative ontology design. In: Alani, H., Noy, N., Stumme, G., Mika, P., Sure, Y., Vrandecic, D. (eds.) Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at WWW 2007, Banff, Canada (2007)
12. Gangemi, A., Presutti, V.: Ontology design for interaction in a reasonable enterprise. In: Rittgen, P. (ed.) Handbook of Ontologies for Business Interaction, IGI Global, Hershey, PA (November 2007)
13. Gruninger, M., Fox, M.: The role of competency questions in enterprise engineering (1994)
14. Guizzardi, G.: Ontological foundations for structural conceptual models. PhD thesis, University of Twente, Enschede, The Netherlands, Enschede (October 2005)
15. Guizzardi, G., Wagner, G.: A unified foundational ontology and some applications of it in business modeling. In: CAiSE Workshops (3), pp. 129–143 (2004)
16. Hay, D.C.: Data Model Patterns. Dorset House Publishing (1996)

17. Gomez-Romero, J., Bobillo, F., Delgado, M.: An ontology design pattern for representing relevance in owl. In: Aberer, K., Choi, K.-S., Noy, N. (eds.) The 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference 2007, Busan, Korea (November 2007)
18. Miles, A., Brickley, D.: SKOS Core Vocabulary Specification. Technical report, World Wide Web Consortium (W3C) (November 2005), http://www.w3.org/TR/2005/WD-swbp-skos-core-spec-20051102/
19. Motta, E., Lu, W.: A library of components for classification problem solving. ibrow project ist-1999-19005: An intelligent brokering service for knowledge-component reuse on the world- wide web. Technical report, KMI (2000)
20. Oberle, D.: Semantic Management of Middleware. The Semantic Web and Beyond, vol. I. Springer, New York (2006)
21. Object Management Group (OMG). Unified modeling language specification: Version 2, revised final adopted specification (ptc/04-10-02) (2004)
22. Semantic Web Best Practices and Deployment Working Group. Task force on ontology engineering patterns. description of work, archives, w3c notes and recommendations (2004), http://www.w3.org/2001/sw/BestPractices/OEP/
23. Presutti, V., Gangemi, A., Gomez-Perez, A., Figueroa, M.-C.S.: Library of design patterns for collaborative development of networked ontologies. Deliverable D2.5.1, NeOn project (2007)
24. Rector, A., Rogers, J.: Patterns, properties and minimizing commitment: Reconstruction of the galen upper ontology in owl. In: Gangemi, A., Borgo, S. (eds.) Proceedings of the EKAW 2004 Workshop on Core Ontologies in Ontology Engineering. CEUR (2004)
25. Sabou, M., Lopez, V., Motta, E.: Ontology selection for the real semantic web: How to cover the queen's birthday dinner? In: Staab, S., Svátek, V. (eds.) EKAW 2006. LNCS (LNAI), vol. 4248, pp. 96–111. Springer, Heidelberg (2006)
26. Svatek, V.: Design patterns for semantic web ontologies: Motivation and discussion. In: Proceedings of the 7th Conference on Business Information Systems, Poznan (2004)
27. Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14, 5–51 (2003)
28. Vrandecic, D., Gangemi, A.: Unit tests for ontologies. In: Jarrar, M., Ostyn, C., Ceusters, W., Persidis, A. (eds.) Proceedings of the 1st International Workshop on Ontology content and evaluation in Enterprise, Montpellier, France, October 2006. LNCS, Springer, Heidelberg (2006)