Ontology Design Patterns for Semantic Web Content

Aldo Gangemi

Laboratory for Applied Ontology, ISTC-CNR, Rome, Italy
a.gangemi@istc.cnr.it

Abstract. The paper presents a framework for introducing design patterns that facilitate or improve the techniques used during ontology lifecycle. Some distinctions are drawn between kinds of ontology design patterns. Some content-oriented patterns are presented in order to illustrate their utility at different degrees of abstraction, and how they can be specialized or composed. The proposed framework and the initial set of patterns are designed in order to function as a pipeline connecting domain modelling, user requirements, and ontology-driven tasks/queries to be executed.

1 Introduction

The lifecycle of ontologies over the Semantic Web involves different techniques, ranging from manual to automatic building, refinement, merging, mapping, annotation, etc. Each technique involves the specification of core concepts for the population of an ontology, or for its annotation, manipulation, or management [7][9][10][11][14][19]. For example, an OWL ontology of gene expression for bioinformatics can be manually built by encoding experts' conceptual patterns [20], or can be automatically learnt e.g. out of a textual corpus by encoding natural language patterns, then refined according to conceptual patterns provided by experts [3], and finally annotated with meta-level concepts for e.g. confidence measurement, argumentation, etc.

Throughout experiences in ontology engineering projects[1,2] at the Laboratory for Applied Ontology (LOA)[3], typical conceptual patterns have emerged out of different domains, for different tasks, and while working with experts having heterogeneous backgrounds. For example, a simple *participation* pattern (including objects taking part in events) emerges in domain ontologies as different as enterprise models [11], legal norms [30], sofware management [17], biochemical pathways [9], and fishery techniques [10]. Other, more complex patterns have also emerged in the same disparate domains: the *role*<->*task* pattern, the *information*<->*realization* pattern, the *description*<->*situation* pattern, the *design*<->*object* pattern, the *attribute parametrization* pattern, etc.

Those emerging patterns are extremely useful in order to acquire, develop, and refine the ontologies from either experts or documents. Often it's even the case that a community of expertise develops its own conceptual pattern, usually of an informal

[1] For example, in the projects *IKF*: http://www.ikfproject.com/About.htm, *FOS*: http://www.fao.org/agris/aos/, and *WonderWeb*: http://wonderweb.semanticweb.org.
[2] http://www.loa-cnr.it
[3] http://dolce.semanticweb.org

Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 262 – 276, 2005.

diagrammatic sort, which can be reengineered as a specialization of the mentioned patterns, for the sake of an ontology project. In some situations, experts do not grasp the utility of ontologies until they realize that an ontology can encode effectively a domain conceptual pattern. Once experts realize it, they usually start discussions on how to improve their own rational procedures by means of ontology engineering techniques!

Following this evidence, for two years a set of conceptual patterns has been used for practical, domain ontology design while still being based on a full-fledged, richly axiomatic ontology (currently DOLCE and its extension s[4] [14][15]). A major attention has been devoted to patterns that are expressible in OWL [18], and are therefore easily applicable to the Semantic Web community.

Independently, in 2004 the W3C has started a working group on Semantic Web Best Practices and Deployment, including a task force on Ontology Engineering Patterns (OEP) [21], which has produced some interesting OWL design patterns that are close, from the logical viewpoint, to some of the ontology design patterns that the LOA has been developing.

In this paper a notion of pattern for ontology design is firstly introduced, contrasting it to other sibling notions. Then a template to present ontology design patterns that are usable to assist or improve Semantic Web ontology engineering is sketched, focusing on patterns that can be encoded in OWL(DL). Some distinctions are drawn between patterns oriented to individuals, to classes or properties, to logical primitives, and to argumentation. Some content-oriented patterns are discussed in order to illustrate that notion at different degrees of abstraction, and how they can be composed. Finally, some conclusions are provided.

2 Some Bits of History

The term "pattern" appears in English in the 14th century and derives from Middle Latin "patronus" (meaning "patron", and, metonymically, "exemplar", something proposed for imitation).[5] As Webster's puts it, a pattern has a set of senses that show a reasonable degree of similarity (see my italics): «a) a *form* or *model* proposed for *imitation*, b) something *designed* or used as a *model* for *making things*, c) a *model* for making a *mold*, d) an artistic, musical, literary, or mechanical *design* or *form*, e) a natural or chance *configuration*, etc., and, f) a *discernible coherent system* based on the *intended interrelationship* of component parts».

In the seventies, the architect and mathematician Christopher Alexander introduced the term "design pattern" for shared guidelines that help solve design problems. In [1] he argues that a good design can be achieved by means of a set of rules that are "packaged" in the form of patterns, such as "courtyards which live", "windows place", or "entrance room". Design patterns are assumed as archetypal solutions to design problems in a certain context.

Taking seriously the architectural metaphor, the notion has been eagerly endorsed by software engineering [2][6][13], where it is used as a general term for formatted guidelines in software reuse, and, more recently, has also appeared in requirements

[4] Cf. Online Etymology Dictionary: http://www.etymonline.com)

[5] In software engineering, formal approaches to design patterns, based on dedicated ontologies, are being investigated, e.g. in so-called *semantic middleware* [17].

analysis, conceptual modelling, and ontology engineering [12][20][21][24][29]. Traditional desing patterns appear more like a collection of shortcuts and suggestions related to a class of context-bound problems and success stories. In recent work, there seems to be a tendency towards a more formal encoding of design patterns (notably in [2][12][13][19]). [24] also addresses the issue of ontology design patterns for the Semantic Web, taking a foundational approach that is complementary with that presented here.

2.1 The Elements of a Design Pattern from Software to Ontology Engineering

For space reasons, a review of the existing literature, and how this proposal differs from it, is not attempted here. Instead, the typical structure of design patterns in software engineering is presented, and contrasted with typical patterns in ontology engineering and with the so-called *content* patterns.

The mainstream approach in Software Engineering (SE) patterns is to use a template that can be similar to the following one (adapted from [22]), used to address a problem of form design in user interfaces:

Slot	Value
Type	UI form
Examples	• Tax forms • Job application forms • Ordering merchandise through a catalog
Context	The user has to provide preformatted information, usually short (non-narrative) answers to questions
Problem	How should the artifact indicate what kind of information should be supplied, and the extent of it?
Forces	• The user needs to know what kind of information to provide. • It should be clear what the user is supposed to read, and what to fill in. • The user needs to know what is required, and what is optional. • Users almost never read directions. • Users generally do not enjoy supplying information this way, and are satisfied by efficiency, clarity, and a lack of mistakes.
Solution	Provide appropriate "blanks" to be filled in, which clearly and correctly indicate what information should be provided. Visually indicate those editable blanks consistently, such as with subtle changes in background color, so that a user can see at a glance what needs to be filled in. Label them with clear, short labels that use terminology familiar to the user; place the labels as close to the blanks as is reasonable. Arrange them all in an order that makes sense semantically, rather than simply grouping things by visual appearance

The slots used here follow quite closely those suggested by Alexander: given an *artifact type*, the pattern provides *examples* of it, its *context*, the *problem* addressed by the pattern, the involved "*forces*" (requirements and constraints), and a *solution*.

In ontology engineering, the nature of the artifact (ontologies) requires a more formal presentation of patterns.[5] For example, the pattern for "classes as property values" [16] produced by the OEP task force [21] can be sketched as follows (only an excerpt of the pattern is shown here):

Slot	Value
General issue	It is often convenient to put a class (e.g., Animal) as a property value (e.g., topic or book subject) when building an ontology. While OWL Full and RDF Schema do not put any restriction on using classes as property values, in OWL DL and OWL Lite most properties cannot have classes as their values.
Use case example	Suppose we have a set of books about animals, and a catalog of these books. We want to annotate each catalog entry with its subject, which is a particular species or class of animal that the book is about. Further, we want to be able to infer that a book about African lions is also a book about lions. For example, when retrieving all books about lions from a repository, we want books that are annotated as books about African lions to be included in the results.
Notation	In all the figures below, ovals represent classes and rectangles represent individuals. The orange color signifies classes or individuals that are specific to a particular approach. Green arrows with green labels are OWL annotation properties. We use N3 syntax to represent the examples.
Approaches	Approach 1: Use classes directly as property values In the first approach, we can simply use classes from the subject hierarchy as values for properties (in our example, as values for the dc:subject property). We can define a class Book to represent all books.
Considerations	• The resulting ontology is compatible with RDF Schema and OWL Full, but it is outside OWL DL and OWL Lite. • This approach is probably the most succinct and intuitive among all the approaches proposed here. • Applications using this representation can directly access the information needed to infer that Lion (the subject of the LionsLifeInThePrideBook individual) is a subclass of Animal and that AfricanLion (the subject of the TheAfricanLionBook individual) is a subclass of Lion.
OWL code (N3 syntax)	default:BookAboutAnimals a owl:Class ; rdfs:subClassOf owl:Thing ; rdfs:subClassOf [a owl:Class ; owl:unionOf ([a owl:Restriction ;

Slot	Value
	owl:onProperty dc:subject ; owl:someValuesFrom default:Animal] [a owl:Restriction ; owl:onProperty dc:subject ; owl:someValuesFrom [a owl:Restriction ; owl:hasValue default:Animal ; owl:onProperty rdfs:subClassOf]])]

As evidenced from the examples, an ontology engineering pattern includes some formal encoding, due to the nature of ontological artifacts. OEP slots seem to "merge" some SE slots: examples and context are merged in the "use case", while the slot "forces" is missing, except for some "considerations" related to the "solution" slot (called "approach" in OEP).

In this paper, a step towards the encoding of *conceptual*, rather than *logical* design patterns, is made. In other words, while OEP is proposing patterns for solving design problems *for OWL*, independently of a particular conceptualization, this paper proposes patterns for solving (*in OWL* or another logical language) design problems for the domain classes and properties that populate an ontology, therefore addressing *content* problems.

3 Conceptual Ontology Design Patterns

3.1 Generic Use Cases

The first move towards conceptual ontology design patterns requires the notion of a "Generic Use Case" (GUC), i.e. a generalization of use cases that can be provided as examples for an issue of domain modelling. Differently from the "artifact type" slot in SE patterns and from the "issue" slot in OEP patterns, a GUC should be the expression of a recurrent issue in many domain modelling projects, independently of the particular logical language adopted. For example, this is a partial list of the recurrent questions that arise in the modelling practice during an ontology project:

- Who does what, when and where?
- Which objects take part in a certain event?
- What are the parts of something?
- What's an object made of?
- What's the place of something?
- What's the time frame of something?
- What technique, method, practice is being used?
- Which tasks should be executed in order to achieve a certain goal?
- Does this behaviour conform to a certain rule?
- What's the function of that artifact?
- How is that object built?

- What's the design of that artifact?
- How did that phenomenon happen?
- What's your role in that transaction?
- What that information is about? How is it realized?
- What argumentation model are you adopting for negotiating an agreement?
- What's the degree of confidence that you give to this axiom?

Being generic at the use case level allows us to decouple, or to refactor the design problems of a use case, by composing different GUCs. Ideally, a library of GUCs should include a hierarchy from the most generic to the most specific ones, and from the "purest" (like most of the examples above) to the most articulated and applied ones (e.g.: "what protein is involved in the Jack/Stat biochemical pathway?").

The intuition underlying GUC hierarchies is based on a methodological observation: ontologies must be built out of domain tasks that can be captured by means of *competency questions* [11]. 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 all the competency questions formulated by, or acquired from, experts.

A GUC can thus be seen as the preliminary motivation to build the pipeline connecting modelling requirements, expected queries (semantic services), and ontology population. Following the distinction between tasks, problem-solving methods, and ontologies that underlies recent architectures for Semantic Web Services [26], GUCs can be used to access at a macroscopic level (partly similar to "use-case diagrams" in UML) the profile (or registries) for a service, the available ontology design patterns (see next section), as well as existing ontologies and knowledge bases. GUC taxonomy is not addressed here for space reasons.

3.2 Features of Conceptual Ontology Design Patterns

A GUC cannot do much as a guideline, unless we are able to find formal patterns that encode it. A formal pattern that encodes a GUC is called here a *Conceptual* (or *Content*) *Ontology Design Pattern* (CODeP).

CODePs are characterized here in a twofold way. Firstly, through an intuitive set of features that a CODeP should have; secondly, through a minimal semantic characterization, and its formal encoding, with the help of some examples.

- A CODeP is a template to represent, and possibly solve, a modelling problem.
- A CODeP "extracts" a fragment of either a *foundational* [14] or *core* [8] ontology, which constitutes its background. For example, a connected path of two relations and three classes ($Ax \wedge By \wedge Cz \wedge Rxy \wedge Syz$) can be extracted because of its domain relevance. Thus, a CODeP lives in a reference ontology, which provides its taxonomic and axiomatic context. A CODeP is axiomatized according to the fragment it extracts. Since it depends on its background, a CODeP inherits the axiomatization (and the related reasoning service) that is already in place.
- Mapping and composition of patterns require a reference ontology, in order to check the consistency of the composition, or to compare the sets of axioms that are to be mapped. Operations on CODePs depend on operations on the reference ontologies. However, for a pattern user, these operations should be (almost) invisible.

- A CODeP can be represented in any ontology representation language whatsoever (depending on its reference ontology), but its intuitive and compact visualization seems an essential requirement. It requires a critical size, so that its diagrammatical visulization is aesthetically acceptable and easily memorizable.
- A CODeP can be an element in a partial order, where the ordering relation requires that at least one of the classes or relations in the pattern is specialized. A hierarchy of CODePs can be built by specializing or generalizing some of the elements (either classes or relations). For example, the *participation* pattern can be specialized to the *taking part in a public enterprise* pattern.
- A CODeP should be intuitively exemplified, and should catch relevant, "core" notions of a domain. Independently of the generality at which a CODeP is singled out, it must contain the central notions that "make rational thinking move" for an expert in a given domain for a given task.
- A CODeP can be often built from informal or simplified schemata used by domain experts, together with the support of other reusable CODePs or reference ontologies, and a methodology for domain ontology analysis. Typically, experts spontaneously develop schemata to improve their business, and to store relevant know-how. These schemata can be reengineered with appropriate methods (e.g. [10]).
- A CODeP can/should be used to describe a "best practice" of modelling.
- A CODeP can be similar to a database schema, but a pattern is defined wrt to a reference ontology, and has a general character, independent of system design.

4 Examples of CODePs

4.1 Some Foundational and Core Patterns

Some examples of CODePs are shown here, but many others have been built or are being investigated. Due to space restrictions, the presentation is necessarily sketchy.

As proposed in the previous section, a CODeP emerges out of an existing or dedicated reference ontology (or ontologies), since it needs a context that facilitates its use, mapping, specialization, and composition.

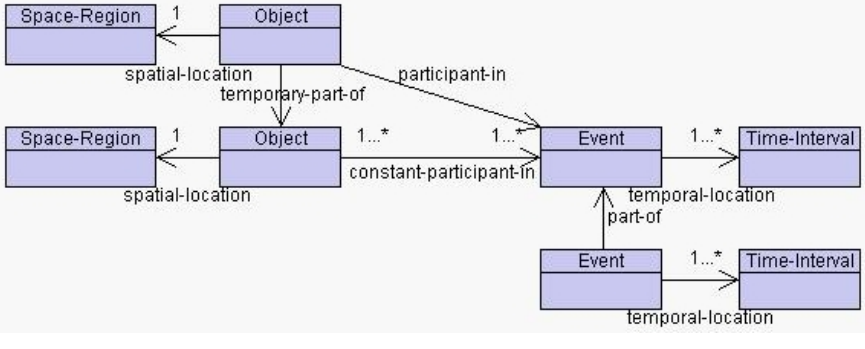


Fig. 1. The basic DOLCE design pattern: participation at spatio-temporal location

A first, basic example (Fig. 1) is provided by the *participation* pattern, extracted from the DOLCE [14] foundational ontology, developed within the WonderWeb Project [5]. It consists of a "participant-in" relation between *objects* and *events*, and assumes a time indexing for it. Time indexing is provided by the temporal location of the event at a *time interval*, while the respective spatial location at a *space region* is provided by the participating object.

Some inferences are automatically drawn when composing the participation CODeP with the *part* CODeP (not shown here, see [14]). For example, if an object *constantly participates in* an event, a temporary part of that object (a part that can be detached), will simply *participate in* that event, because we cannot be sure that the part will be a part at all times the whole participates. For example, we cannot infer for each member of a gang that she participated in a crime, just because she is a member.

An alternative CODeP (Fig. 2) for time-indexed participation can be given by reifying the participation relation (in OWL a ternary relation cannot be expressed conveniently). The *reified participation* pattern features a kind of "situation" (see next example), called *time-indexed-participation*, which is a setting for exactly one object, one event, and one time interval. This simple reification pattern can be made as complex as needed, by adding parameters, more participants, places, etc.

A third, more complex example, is the *Role*<->*Task* CODeP (Fig. 3). This CODeP is based on an extension of DOLCE, called D&S (Descriptions and Situations) [9][15], partly developed within the Metokis Project [4]. D&S provides a vocabulary and an axiomatization to type-reified [27] classes and relations ("concepts" and "descriptions"), and to token-reified [27] tuples ("situations"; for a semantics of D&S, see [28]).

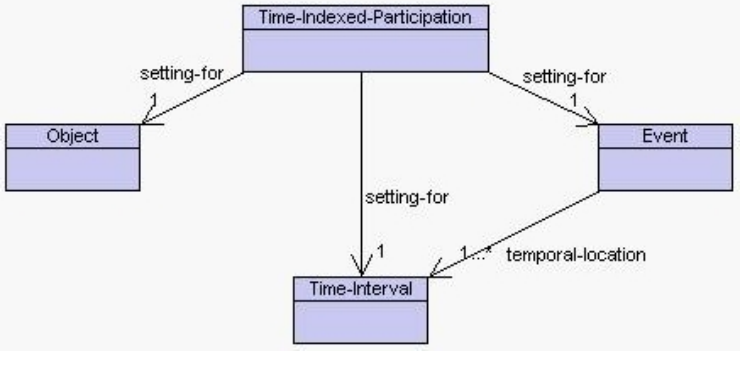


Fig. 2. A pattern for reification of time-indexed relations (in this case, *participation*): a situation (like *time-indexed participation*) is a *setting for* an event, the entities participating in that event, and the time interval at which the event occurs

In practice, the *Role*<->*Task* pattern allows the expression, in OWL(DL), of the temporary *roles* that objects can play, and of the *tasks* that events/actions allow to execute. The reified relation specifying roles and tasks is a *description*, the reified tuple that satisfies the relation for certain individual objects and events is called *situation*. Roles can have assigned tasks as *modal targets*. This CODeP is very expressive, and can be specialized in many domains, solving design issues that are quite hard without reification. For example, the assignments of tasks to role-players in a workflow can be easily expressed, as well as plan models [28].

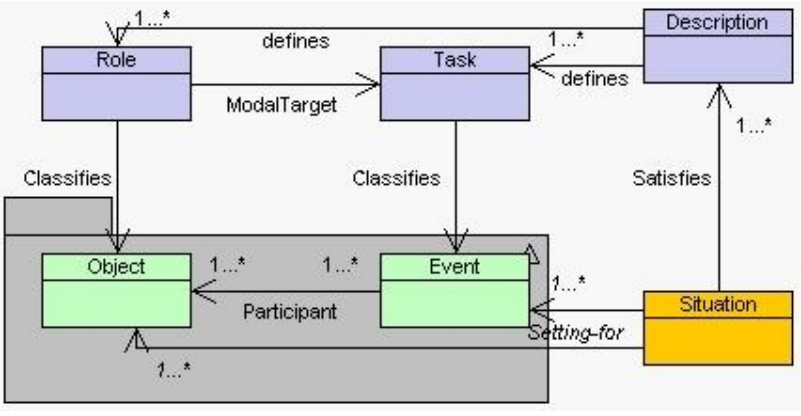


Fig. 3. A pattern for roles and tasks defined by descriptions and executed within situations

By composing the *Role*<->*Task* pattern with the *Collection*<->*Role* pattern (not shown here), and specializing such composition to the domain of material design, we obtain the so-called *Design*<->*Artifact* CODeP (Fig. 4). This pattern is very expressive and quite complex. Starting from *Role*<->*Task*, and *Collection*<->*Role,* and specializing objects to *material artifacts*, descriptions to *designs, situations to design materialization*, and substituting tasks with *functions,* we can conceive of a *functional unification* relation holding between a design model and a material artifact. The internal axiomatization operates by unifying the collection of "relevant" components ("proper parts") of the material artifact within a "collection", where each component plays a functional role defined by the design model.

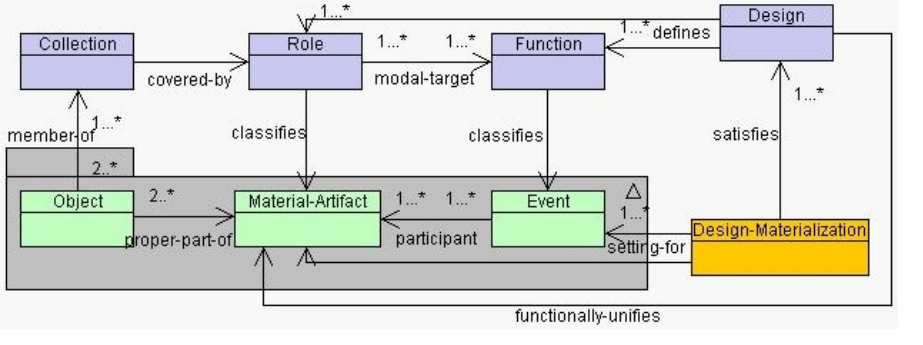


Fig. 4. A pattern for talking about relations between design models and material artifacts

The design materialization keeps together the actual physical components of an individual material artifact. This CODeP can be easily specialized for manufacturing, commercial warehouses, etc.

The previous CODePs are *foundational*. An example of a *core* CODeP is instead provided here with reference to the NCI ontology of cancer research and treatment [23] (Fig. 5). It specializes the foundational *Role*<->*Task* CODeP (Fig. 3).

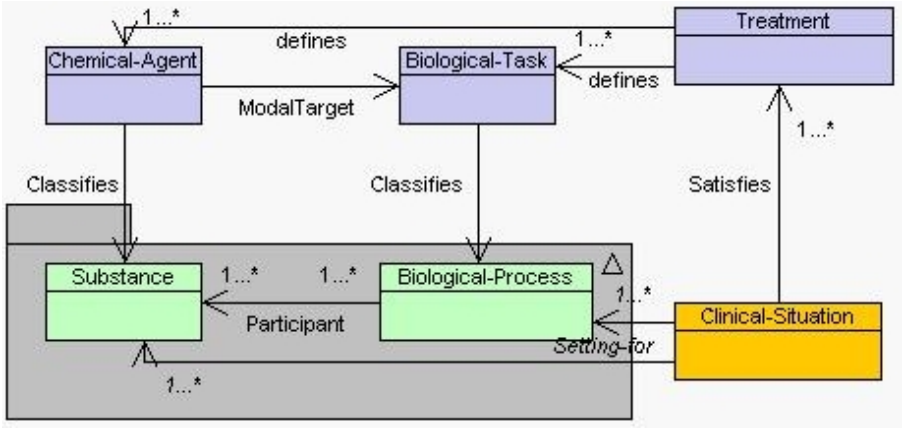


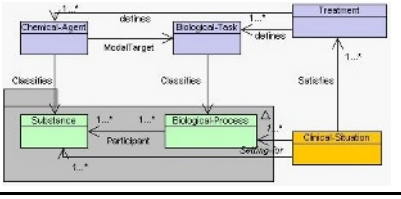
Fig. 5. A *core* pattern for chemotherapy, specializing the *Role*<->*Task* CODeP

4.2 How to Introduce a CODeP

A template can be used to annotate CODePs, to share them in pre-formatted documents, to contextualize them appropriately, etc. Here the following frame is proposed, and presented through the previous example from the NCI ontology [23]:

Slot	Value
Generic use case (GUC)	Chemicals playing roles in biological processes for chemotherapy.
Local use case(s)	Various chemical agents, mostly drugs, are used to control biological processes within a chemotherapeutical treatment. When talking about drugs and processes, there is a network of senses implying a dependence on roles and functions (or tasks) within a clinical treatment. Intended meanings include the *possible* functional roles played by certain substances, as well as the actual administration of amounts of drugs for controlling actually occurring biological processes. Therefore, both class- and instance-variables are present in the maximal relation for this pattern.
Logic addressed	OWL, DL species
Reference ontologies	DOLCE-Lite-Plus, NCI Ontology

Slot	Value
Specialized CODeP	Role<->Task
Composed CODePs	Time-Indexed-Participation, Concept<->Description, Description<->Situation
Formal relation	rChemical_or_Drug_Plays_Role_in_Biological_Process(ϕ,ψ,x,y,t, c_1,c_2,d,s), where ϕ(x) is a chemical agent class, ψ(y) is a biological process class, t is a time interval, c_1 and c_2 are two reified intensional concepts, d is a reified intensional relation, and s is a reified extensional relation.
Sensitive axioms	rChemical_or_Drug_Plays_Role_in_Biological_Process(ϕ,ψ) $=_{df}$ ∀x,y,t(ϕ(x) ∧ ψ(y) ∧ participates-in(x,y,t) ∧ Chemical-Agent(x) ∧ Biological-Process(y) ∧ Time-Interval(t)) ↔ ∃c_1,c_2,d(CF(x,c_1,t) ∧ MT(c_1,c_2) ∧ CF(y,c_2,t) ∧ DF(d,c_1) ∧ DF(d,c_2) ∧ ∀s(SAT(s,d)) ↔ (SETF(s,x) ∧ SETF(s,y) ∧ SETF(s,t))
Explanation	Since OWL(DL) does not support relations with >2 arity, reification is required. The Description<->Situation pattern provides typing for such reification. Since OWL(DL) does not support classes in variable position, we need reification for class-variables. The Concept<->Description pattern provides typing for such reification. Similarly, since participation is time-indexed, we need the time-indexed-participation pattern, which is here composed with the previous two patterns (time indexing appears in the setting of the general treatment situation).
OWL(DL) encoding (abstract syntax)	Class(Chemical_Plays_Role_in_Bio_Process complete Description restriction(defines someValuesFrom(Chemical-Agent)) restriction(defines someValuesFrom(Biological-Task))) Class(Chemical-Agent complete Role restriction(defined-by someValuesFrom(Chemical_Plays_Role_in_Bio_Process)) restriction(classifies allValuesFrom(Substance)) restriction(modal-target someValuesFrom(Biological-Task))) Class(Biological-Task complete Task restriction(classifies allValuesFrom(Biological-Process)) restriction(modal-target-of someValuesFrom(Chemical-Agent))) Class(Chemical-in-Biological-Process_Situation complete

Slot	Value
	Situation restriction(satisfies someValuesFrom(Chemical_Plays_Role_in_Bio_Process)) restriction(setting-for someValuesFrom(Substance)) restriction(setting-for someValuesFrom(Biological-Process)) restriction(setting-for someValuesFrom(Time-Interval)))
Class diagram	

The CODeP frame consists of:

- Two slots for the *generic use case*, and the *local use cases*, which includes a description of context, problem, and constraints/requirements.
- Two slots for the addressed *logic*, and the *reference ontologies* used as a background for the pattern.
- Two slots for -if any- the *specialized* pattern and the *composed* patterns.
- Two slots for the *maximal relation* that encodes the case space, and its intended *axiomatization*: a full first-order logic with meta-level is assumed here, but the slot can be empty without affecting the functionality of a CODeP frame.
- Two slots for *explanation* of the approach, and its *encoding* in the logic of choice.
- A last slot for a *class diagram* that visually reproduces the approach.

The frame for introducing CODePs can be easily encoded in XSD or in richer frameworks, like semantic web services (e.g. [25]) or knowledge content objects [26], for optimal exploitation within Semantic Web technologies. The high reusability of CODePs and their formal and pragmatic nature make them suitable not only for isolated ontology engineering practices, but ideally in distributed, collaborative environments like intranets, the Web or the Grid.

CODePs can also be used to generate intuitive, friendly UIs, which can present the user with only the relevant pattern diagram, avoiding the awkward, entangled graphs currently visualized for medium-to-large ontologies.

5 Conclusions

Conceptual Ontology Design Patterns (CODePs) have been introduced as a useful resource and design method for engineering ontology content over the Semantic Web. CODePs are distinguished from architectural, software engineering, and logic-oriented design patterns, and a template has been proposed to describe, visualize, and make operations over them.

The advantages of CODePs for ontology lifecycle over the Semantic Web are straightforward: firstly, patterns make ontology design easier for both knowledge engineers and domain experts (imagine having a menu of pre-built, formally consistent components, pro-actively suggested to the modeller); secondly, patterned design makes it easier ontology integration - perhaps the most difficult problem in ontology engineering. For example, the *time-indexed participation* presented in this paper requires non-trivial knowledge engineering ability to be optimally represented and adapted to a use case: a CODeP within an appropriate ontology management tool can greatly facilitate such representation.

The CODeP examples and the related frame and methods introduced in this paper have been applied for two years (some of them even before) in several administration, business and industrial projects, e.g. in fishery information systems [10], insurance CRM, biomedical ontology integration [9], anti-money-laundering systems for banks [30], service-level agreements for information systems, biomolecular ontology learning [3], legal norms formalization, and management of digital content [26].

Current work focuses on building a tool that assists development, discussion, retrieval, and interchange of CODePs over the Semantic Web, and towards establishing the model-theoretical and operational foundations of CODeP manipulation and reasoning. In particular, for CODePs to be a real advantage in ontology lifecycle, the following functionalities will be available:

- Categorization of CODePs, based either on the use cases they support, or on the concepts they encode.
- Pattern-matching algorithms for retrieving the pattern that best fits a set of requirements, e.g. from a natural language specification, or from a draft ontology.
- Support for specialization and composition of CODePs. A CODeP p_1 *specializes* another p_2 when at least one of the classes or properties from p_2 is a sub-class or a sub-property of some class resp. property from p_1, while the remainder of the CODeP is identical. A CODeP p_1 *expands* p_2 when p_1 contains p_2, while adding some other class, property, or axiom. A CODeP p_1 *composes* p_2 and p_3 when p_1 contains both p_2 and p_3. The formal semantics of these operations is ensured by the underlying (reference) ontology for the patterns, and will be given in an extended version of this paper.
- Interfacing of CODePs for visualization, discussion, and knowledge-base creation
- A rich set of metadata for CODeP manipulation and exploitation within applications.

References

1. Alexander, C.: The Timeless way of building. Oxford University Press, New York (1979).
2. Baker, N., A. Bazan, G. Chevenier, Z. Kovacs, T Le Flour, J-M Le Goff, R. McClatchey, S. Murray: Design Patterns for Description-Driven Systems.
3. Ciaramita, M., Gangemi, A., Ratsch, E., Rojas, I., Saric, J.: Unsupervised Learning of Semantic Relations between Concepts of a Molecular Biology Ontology. To appear in the proceedings of the Nineteenth IJCAI, Edimburgh, Scotland (2005).
4. EU FP6 Metokis Project: http://metokis.salzburgresearch.at
5. EU FP5 WonderWeb Project: http://wonderweb.semanticweb.org

6. Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA (1995).
7. Gangemi, A., Navigli, R., Velardi, P.: ML: The OntoWordNet Project: extension and axiomatisation of conceptual relations in WordNet. International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2003), Catania, (Italy), (2003).
8. Gangemi, A., Borgo, S. (eds.): Proceedings of the EKAW*04 Workshop on Core Ontologies in Ontology Engineering. Available from: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-118/ (2004).
9. Gangemi, A., Catenacci, C., Battaglia, M.: Inflammation Ontology Design Pattern: an Exercise in Building a Core Biomedical Ontology with Descriptions and Situations.D.M. Pisanelli (ed.) Ontologies in Medicine, IOS Press, Amsterdam (2004).
10. Gangemi, A., F. Fisseha, J. Keizer, J. Lehmann, A. Liang, I. Pettman, M. Sini, M. Taconet: A Core Ontology of Fishery and its Use in the FOS Project, in [8] (2004).
11. Gruninger, M., and Fox, M.S.: The Role of Competency Questions in Enterprise Engineering. Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice, Trondheim, Norway (1994).
12. Guizzardi, G., Wagner, G., Guarino, N., van Sinderen, M.: An Ontologically Well-Founded Profile for UML Conceptual Models. A. Persson, J. Stirna (eds.) Advanced Information Systems Engineering, Proceedings of16th CAiSE Conference, Riga, Springer (2004).
13. Maplesden, D., Hosking, J.G. and Grundy, J.C.: Design Pattern Modelling and Instantiation using DPML, Proceedings of the Tools Pacific 2002, Sydney, CRPIT Press (2002).
14. Masolo, C., A. Gangemi, N. Guarino, A. Oltramari and L. Schneider: WonderWeb Deliverable D18: The WonderWeb Library of Foundational Ontologies (2004).
15. Masolo, C., L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi and N. Guarino: Social Roles and their Descriptions. Procedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning, Whistler (2004).
16. Noy, N.: Representing Classes As Property Values on the Semantic Web. W3C Note, http://www.w3.org/2001/sw/BestPractices/OEP/ClassesAsValues-20050405/ (2005).
17. Oberle, D., Mika, P., Gangemi, A., Sabou, M.: Foundations for service ontologies: Aligning OWL-S to DOLCE. Staab S and Patel-Schneider P (eds.), Proceedings of the World Wide Web Conference (WWW2004), Semantic Web Track, (2004).
18. OWL Web Ontology Language Overview, D. L. McGuinness and F. van Harmelen, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-features-20040210/ (2004).
19. Rector, A.L., Rogers, J.: Patterns, Properties and Minimizing Commitment: Reconstruction of the GALEN Upper Ontology in OWL, in [8] (2004).
20. Reich, J.R.: Ontological Design Patterns: Modelling the Metadata of Molecular Biological Ontologies, Information and Knowledge. In DEXA 2000 (2000).
21. Semantic Web Best Practices and Deployment Working Group, Task Force on Ontology Engineering Patterns. Description of work, archives, W3C Notes and recommendations available from http://www.w3.org/2001/sw/BestPractices/OEP/ (2004-5).
22. Tidwell, J.: COMMON GROUND: A Pattern Language for Human-Computer Interface Design. http://www.mit.edu/%7Ejtidwell/interaction_patterns.html (1999).
23. Golbeck, J., G. Fragoso, F. Hartel, J. Hendler, B. Parsia, J. Oberthaler: The national cancer institute's thesaurus and ontology. *Journal of Web Semantics*, 1(1), (2003).
24. Svatek V.: Design Patterns for Semantic Web Ontologies: Motivation and Discussion. In: 7th Conference on Business Information Systems, Poznaň (2004).

25. Motta, E., Domingue, J., Cabral, L., Gaspari, M. (2003) IRS-II: A Framework and Infrastructure for Semantic Web Services. 2nd International Semantic Web Conference (ISWC2003) 20-23 October 2003, Sundial Resort, Sanibel Island, Florida, USA (2003).
26. Behrent, W., Gangemi, A., Maass, W., Westenthaler, R.: Towards an Ontology-based Distributed Architecture for Paid Content. To appear in A. Gomez-Perez (ed.), Proceedings of the Second European Semantic Web Conference, Heraklion, Greece (2005).
27. Galton, A.: Reified Temporal Theories and How To Unreify Them. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1991.
28. Gangemi, A., Borgo, S., Catenacci, C., Lehmann, J.: Task Taxonomies for Knowledge Content. Deliverable D07 of the Metokis Project. Available at http://www.loa-cnr.it.
29. Soshnikov, D.: Ontological Design Patterns in Distributed Frame Hierarchy. In Proceedings of the 5th International Workshop on Computer Science and Information Technologies, Ufa, Russia, 2003.
30. Gangemi A, Pisanelli DM, Steve G,: An Ontological Framework to Represent Norm Dynamics. In R Winkels (ed.), *Proceedings of the 2001 Jurix Conference, Workshop on Legal Ontologies*, University of Amsterdam, 2001.