Weaving Ontology Aspects Using a Catalog of Structural Ontology Design Patterns

Ralph Schäfermeier and Adrian Paschke

Corporate Semantic Web Group, Institute of Computer Science,
Freie Universität Berlin, Germany
{schaef,paschke}@inf.fu-berlin.de
http://www.csw.inf.fu-berlin.de

Abstract. Modular development of ontologies proves beneficial at different stages of the ontology lifecycle. In our previous work, we proposed aspect-oriented ontology development as a flexible approach to modular ontology development and a-posteriori modularization of existing monolithic ontologies, inspired by aspect-oriented programming and based on so called cross-cutting concerns. Similar to other formalisms for modular ontologies (e.g. $\mathcal{E}$-Connections), aspect-oriented ontologies rely on an extension of the used ontology language. This derivation from the standard in turn requires specially adapted tools in order to use these ontologies in applications. In this paper, we present an approach to the recombination of aspect-oriented ontology modules to standard OWL 2 ontologies by using an aspect-weaving service. The weaving service relies on a preconfigured catalog of structural ontology design patterns. We show that the use of the weaving service yields syntactically correct and semantically complete ontologies while still allowing ontology developers to fully benefit from modular ontology development.

Keywords: Aspect-Oriented Ontology Development, Aspect-Oriented Programming, Ontology Modularization, Ontology Design Patterns

1 Introduction

Modular ontology development has proved beneficial when it comes to improvement of reasoning and query result retrieval performance, scalability for ontology evolution and maintenance, complexity management, amelioration of understandability, reuse, context-awareness, and personalization [17]. A significant amount of research work has been dedicated in the field of ontology modularization, and various kinds of approaches tackle the problem from different perspectives. One kind of approaches provides algorithmic solutions for the problem of modularizing existing large and monolithic ontologies, while others provide methodological and formal means for contsructing ontologies in a modular fashion from scratch.

Aspect-Oriented Ontology Development (AOOD) [18, 19] is an approach to modular ontology development inspired by the Aspect-Oriented Programming

paradigm (AOP) [12]. As its counterpart from the programming domain, AOOD is based on cross-cutting concerns. Aspect-Oriented Ontology Development makes use of meta-modeling in order to combine a main ontology module with other modules each of which provides additional knowledge about a particular concern. To this end, an additional syntactic category had to be added to the OWL 2 language[1], requiring specialized tools for processing aspect-oriented ontologies that make use of this syntactic extension.

In Aspect-Oriented Programming, which also requires an extension of the programming language at hand for the representation of software aspects, a special software utility named *aspect weaver* is responsible for recombining the modules using the extra information contained in the aspect extension and generating executable code that conforms to the standards of the programming language.

In this paper, we present an aspect weaver for aspect-oriented ontologies, which, in a manner analogous to that of a software aspect weaver, combines ontology modules represented as ontology aspects to a valid OWL 2 ontology. The weaver uses a catalog of structural ontology design patterns (ODPs) [8, 9][2], and a mapping from what we consider typical aspect constructs to ODP patterns. The rest of the paper is structured as follows: Section 2 provides an introduction to Aspect-Oriented Ontology Development. Section 3 presents the approach for recombining aspect-orientation-based ontology modules using structured ODPs, which constitutes the core contribution of this paper. Section 4 summarizes the relevant related work in the field of modular ontology development and use of ontology modules. Section 5 provides the conclusion and an outlook on future work.

2 Aspect-Oriented Ontology Development

As mentioned in the introduction, the approach of Aspect-Oriented Ontology Development has drawn inspiration from the Aspect-Oriented Programming paradigm [7] (also referred to as *Aspect-Oriented Software Development*). The following section provides a brief introduction to the essential concepts of Aspect-Oriented Programming and then explains how the principle has been adopted for modular ontology development.

2.1 Aspect-Oriented Programming

The main goal of Aspect-Oriented Programming is the decomposition of software systems into concerns which cross-cut the system. A code module covering a particular concern is referred to as an aspect. Concerns may be functional concerns, which are directly related to the systems's domain of interest and business logic and non-functional concerns, such as security, logging/auditing and performance.

[1] http://www.w3.org/TR/owl2-overview/
[2] http://ontologydesignpatterns.org/

The decomposition is accomplished by introducing extensions to existing programming languages (such as AspectJ[3] for Java) that allow the decomposition of code into modules, each of them dealing with a concern, as well as a mechanism for recombining the modules at compile or runtime into a complete and coherent system. Programming languages without aspect-orientation have no means for separating those concerns, which leads to undesired code tangling and hinders system decomposition.

Quantification and Obliviousness Two fundamental properties of Aspect-Oriented Programming are *quantification* and *obliviousness* [7]. *Obliviousness* refers to the fact that all information necessary to determine the execution points where the application should make a call into an aspect module are contained within the aspect itself rather than in the application code. A developer of one module does not, and need not, have knowledge about other modules that might potentially be called.

This information may be provided in the form of an exhaustive list of signatures or in terms of *quantified statements* over signatures, called a *pointcut*. Each single matching signature is called a *join point*.

Formally, Aspect-Oriented Programming uses quantified statements of the following form [22]:

$$\forall m(p_1,\ldots,p_n) \in M : s(sig(m(p_1,\ldots,p_n))) \\ \rightarrow (m(p_1,\ldots,p_n) \rightarrow a(p_1,\ldots,p_n)), \quad (1)$$

where M is the set of all methods defined in the software system, s a predicate specifying a matching criterion, $m(p_1,\ldots,p_n) \in M$ a method matching the signature $sig(m(p_1,\ldots,p_n))$, and $a(p_1,\ldots,p_n)$ the execution of the aspect with all the parameters of each method, respectively. The code in the aspect, which is executed at each joint point, is referred to as *advice*. In APO terminology, an aspect *advices* the main code.

The idea behind Aspect-Oriented Ontology Development is to use pointcuts in order to describe ontology modules and aspects in order to attach additional knowledge (advice) to each of these modules.

2.2 Aspect-Oriented Ontologies

As in software, cross-cutting concerns can be observed in ontologies. Consider for example Abox facts that are constrained to be valid only during a certain period of time. Figure 1 shows a concrete example of a time-constrained fact, namely the recognition of the Kosovo as a self-governing entitiy, using concepts from the geopolitical ontology of the Food and Agriculture Organization of the United Nations[4]. The time period is modeled using the W3C time ontology[5].

[3] `https://eclipse.org/aspectj/`
[4] `http://www.fao.org/countryprofiles/geoinfo/en/`
[5] `www.w3.org/TR/owl-time/`

The intention is to reify the first fact recognizedBy(Kosovo, United_Kingdom) with the open time interval individual Interval_1 using a validDuring relationship. This, however, is not permissible due to limitations in the expressivity of OWL and the underlying Description Logics. What is instead recommended by the W3C is to introduce a surrogate individual to represent the ternary relationship between Kosovo, the UK and the time interval[6]. The right hand side of Figure 1 shows the combined facts with the new introduced Recognition_1 surrogate individual. In addition to the individual, two new object properties need to be introduced. The existing recognizedBy property now has the new surrogate individual in its range instead of the recognizing country. The two new properties connect the surrogate with the recognizing country and the time interval, respectively.

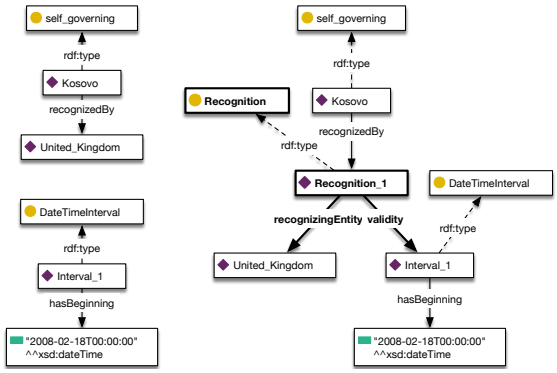

Fig. 1. Axioms from the FAO and the W3C time ontology (left) and necessary refactoring of the reused ontology in order to allow for the extension, following the W3C n-ary relations pattern (right)

The recommended pattern for n-ary relationships leads to a high degree of entanglement of different concerns (in this case different domains, namely the domain of self-governing political entities and the domain of time), which brings the following disadvantage: After the introduction of the surrogate individual, the representation of the fact recognizedBy(Kosovo, United_Kingdom) as a simple binary relation is lost. An ontology engineer, however, might be interested in reusing knowledge about self-governing entities from this ontology but without the temporal information. Due to the entanglement it is not trivial anymore to separate these parts from each other and reuse them individually.

Therefore, we introduce new syntactic category *Aspect*, which is used in order to establish a relationship between OWL 2 classes and axioms. Figure 2 depicts the representation of the ternary relationship from the above example using an aspect. Note that the figure contains two ternary relationships: The one from

[6] `http://www.w3.org/TR/swbp-n-aryRelations/`

the example and the class assertion axiom Kosovo rdf:type self_governing which is also supposed to be valid only during the given time interval.

It might appear awkward to represent aspects as classes. In the following subsection where we describe the semantics of aspects, we provide a justification of that choice.

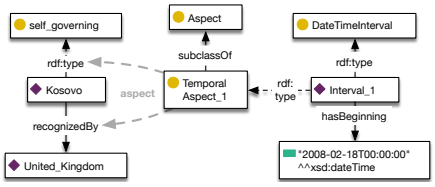

Fig. 2. Representation of the ternary relationships using an aspect. Note that the pointcut relation points to axioms, not individuals.

As can be seen, this way of representing the above relationships keeps the original structure of the ontology intact. In particular, it allows to connect additional (time) knowledge to an existing binary relation (about countries) while keeping both domains separated from each other, allowing partial reuse and independent maintenance and evolution.

Syntax As mentioned above, we extend the OWL 2 language by a syntactic category *Aspect* which is used to represent a relationship between classes and axioms.

The definition of the abstract syntax in OWL functional-style syntax is as follows:

```
AspectAssertion ::= 'AspectAssertion' '(' AxiomAnnotationSet JoinPoint Advice ')'
AxiomAspectSet ::= Aspect*
JoinPoint ::= IRI | AnonymousIndividual
Advice ::= ClassExpression
Pointcut ::= SPARQLPointcut | ModulePointcut | DLQueryPointcut
SPARQLPointcut ::= 'SPARQLPointcut' '(' AxiomAnnotationSet Aspect
  '"' ConstructQuery '"' ')'
ModulePointcut ::= 'ModulePointcut' '(' AxiomAnnotationSet Aspect Signature ')'
DLQueryPointcut ::= 'DLQueryPointcut' '(' AxiomAnnotationSet Aspect
  ClassExpression ')'
Signature ::= EntityIRI*
```

The definitions of the categories *Annotation, AxiomAnnotationSet, IRI, AnonymousIndividual, ClassExpression* and *Axiom* are provided in the OWL 2 Structural Specification[7]. The defintion of *ConstructQuery* is provided in the SPARQL 1.1 Query Language Specification[8].

An *AxiomAspectSet* can be added to each axiom that can contain an *AxiomAnnotationSet*, for example:

[7] http://www.w3.org/TR/owl2-syntax/

[8] http://www.w3.org/TR/sparql11-query/#rConstructQuery

```
EquivalentClasses ::= 'EquivalentClasses' '(' AxiomAnnotationSet
  AxiomAspectSet ClassExpressionSet ')'
```

Semantics We define the semantics of ontology aspects in correspondence with the possible-world semantics of multi-modal logics:

- Aspects correspond to sets of axioms or facts that are true in certain possible worlds.
- Aspects are modeled as classes.
- Possible worlds are modeled as individuals.
- Accessibility relations are modeled as object properties.
- The semantics of aspects depend on the choice of conditions on frames (axioms on accessibility properties).

The rationales behind that choice are:

- (Multi-)modal logics are a syntactic variant of and thereby semantically equivalent to Description Logics [20, 1].
- Aspects are a sort of modality in that there is a function that determines in which situations an aspect is active and in which it is not. That corresponds to possible worlds in modal logics where a truth-functional valuation determines whether a fact is valid in a possible world or not.
- The kind of modal logic is determined by conditions on Kripke frames, which (to a certain extent) may be controlled by fixing the characteristics of the accessibility relations. This allows the representation of e.g., temporal logic (as in our running example), simple views, agent beliefs, etc.
- Using classes as aspects allows to use abstract class definitions using constraints with quantifiers.

Figure 3 depicts a more complex example using a temporal aspect on an Abox fact capitalOf(Bonn, Germany), which was true between 1949 and 1990. We used the W3C time ontology again to model time instances. We interpret each time instance as a possible world, and *after* (and *before*) are accessibility relations, which are reflexive and transitive. We thereby obtain the conditions on the Kripke frames for a temporal logic:

- $(M) : \Box A \rightarrow A$
- $(4) : \Box A \rightarrow \Box\Box A$

The temporal aspect is then the class expression
after value 1949 and before value 1990,
which includes the values 1949 and 1990 due to the reflexivity of the before and after relations.

Likewise, we can obtain a simple Logic K by just setting the accessibility relation reflexive. We can use this logic to model simple views, which are manually assigned to axioms.

As a third example, we can use multi standard deontic logic for modelling access permissions over axioms for different agents by having a serial accessibility relation a_i for each agent i in order to obtain the axiom

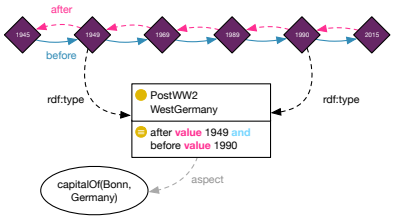

Fig. 3. A more complex temporal aspect using temporal logic

– $(D) : \Box_i A \rightarrow \Diamond_i A$

The intuition behind this is that an aspect describes a (syntactic) module in an ontology (which technically consists of a set of axioms) and adds second-order information to it (as for example a temporal validity restriction, as in the above example). The purpose of this approach is to permit to extract modules depending on the outcome of some reasoning process. We could, for example, extract a module with axioms that are valid only during the 1950s (which would include the fact that Bonn is capital of Germany) and at the same time are accessible to some agent.

3 Use of Structural Ontology Design Patterns for Aspect Weaving

The Aspect-Oriented Ontology Development approach outlined in Section 2 has the advantage of keeping concerns separated from each other, so that individual development, evolution, and maintenance of each concern is easier. One obvious disadvantage, however, is the fact that a non-standard (syntactic and semantic) extension to the ontology language at hand is necessary in order to represent aspects in an ontology. This requires adapted tools that can read, represent, and perform reasoning with ontologies that use this extension.

As mentioned in the introduction, Aspect-Oriented Programming makes use of *aspect weavers*, which collect and combine all relevant aspects in a software project and generate standard-conformant code that contains all the necessary modules and calls to them.

In what follows, we introduce a weaving facility for ontologies, which converts aspect-oriented ontologies into standard-conformant OWL 2 ontologies, preserving the information conveyed by the aspects. We use ontology design patterns (ODPs) in order to automate the conversion process. ODPs are structural (e.g., logical or architectural), conceptual, or lexico-syntactic templates that abstract from typical ontology modeling problems and serve as a recipe for ontology developers to solve the corresponding problem.

3.1 Approach

We used the collection of categorized ODPs mentioned in Section 1 as the source of potentially suitable ODPs for the aspect weaver. As a first step, we selected the appropriate ODPs from the collection. The criteria we applied for the selection follow from the nature of ontology aspects: Since aspects are a way of metamodeling in order to convey contextual information to existing parts of an ontology, we selected all patterns related to metamodeling and contextual knowledge. The selection process resulted in the ODPs *View Inheritance, Context Slices,* and *N-ary Relation Pattern.*

We used an extended version of the Ontology Pre-Processor Language (OPPL)[9] for formulating the patterns and the necessary refactoring operations. We extended OPPL by syntactical features for aspects, in correspondence to the aspect-oriented extensions of OWL 2 described in Section 2.

The *View Inheritance ODP*[10] is an architectural pattern that provides a way for modeling multifacted classification schemes or multiple class inheritance hierarchies. It does this by introducing intermediate classes which represent the different classifiers, referred to as *criteria.* The actual target domain concepts are made subclasses of the classes representing the classification criteria. The pattern has two disadvantages, a semantic and a structural one. The semantic disadvantage consists in the fact that the classifier classes are directly introduced into the inheritance hierarchy. Subclasses are now subclasses of the classifier, which is not the intended meaning but merely a way to circumvent the expressive restrictions of DL which do not allow object relations between classes. The structural disadvantage is similar to the one discussed in Section 2: Once introduced, the classifier cannot easily be eliminated from the hierarchy. A typical use case for multifaceted classification is to specify a classifier and hide the other hierarchies with different classifiers. Since the classifier is now part of the ontology, this is not easily possible. In an aspect-oriented ontology, this kind of classifier is represented as an aspect class which is attached to the owl:SubClassOf axioms that correspond to this specific classifier. Since it expresses simple views, Logic K is the appropriate type of modal logic, and therefore, this aspect has a reflexive accessibility relation.

Figure 4 shows how the weaver transforms the aspects into an ontology by applying the pattern.

The *context slices* pattern[11] may be used for the expression of agents' beliefs about Abox facts, or, more precisely, object property assertions. Each agent's conception of a part of the universe (i.e., the axioms valid in the part of the universe accessible to the agent) is referred to as a context. A context is represented by an individual of type Context, and the subject and object of a contextualized object property assertion are connected to this context via additional object

[9] http://oppl2.sourceforge.net

[10] http://ontologydesignpatterns.org/wiki/Submissions:View_Inheritance

[11] http://ontologydesignpatterns.org/wiki/Submissions:Context_Slices

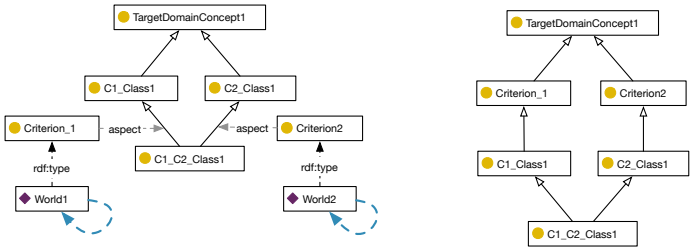

Fig. 4. A transformation by the weaver of multifaceted classification using the View Inheritance ontology design pattern. The aspect is expressed by a reflexive accessibility relation in a logic system K. The class expression representing the aspect is attached to the subclass axioms.

property assertions involving an object property hasContext with the context individual in the subject role and the two contextualized individuals in the object role.

In an aspect-oriented ontology the belief of an agent may be expressed using a doxastic logic with an object property $\mathsf{believes}_i$, representing the accessibility relation between possible worlds. The "actual" world is represented by an individual Agent_i and may be interpreted as the agent believing the axioms associated with it. Each possible world may be interpreted as a context. A fact in the ontology may be associated with a context by connecting it to a class expression. A context individual being of this type may be interpreted as the fact being true in this context. The agent being connected to at least one context where the fact is valid means that the agent beliefs the fact to be true.

Figure 5 shows how the weaver transforms the aspects into an ontology by applying the pattern.

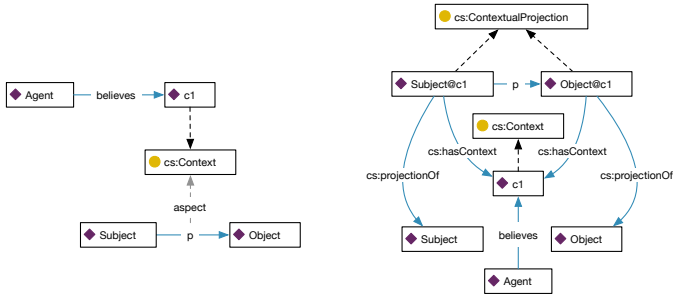

Fig. 5. A transformation by the weaver of a context aspect representing an agent's belief using the context slices pattern.

4 Related Work

Ontology modularization is an active research field, and there exists a rich body of related work. D'Aquin [4] distinguishes between different perspectives on the problem of which two different subfields have emerged. There exist approaches to *ontology partitioning*, where a monolithic ontology is decomposed into smaller fractions. The motivation for ontology partitioning comes from requirements concerning maintenance and reuse, thus constituting requirements rooted in an engineering point of view. The second class of approaches is referred to as *ontology module extraction*. The motivation for module extraction is mainly selective use and reuse [4]. In [11], the authors present a partitioning approach using so called $\mathcal{E}$-Connections [15, 3]. The criterion for the partitioning process is semantic relatedness. One drawback of this approach, however, is that concept subsumption or the use of roles across different modules is not possible. Schlicht et al. propose a semi-automated approach to ontology partitioning based on application-imposed requirements [21]. The method constructs a dependency graph of strongly interrelated ontology features, such as sub/super concept hierarchy, concepts using the same relations, or similarly labeled concepts. The method is parametrizable by the features taken into account for constructing the dependency graph and the size a module should have. Another class of partitioning approaches uses graph-based and social network metrics in order to determine central concepts and interrelated features which should be part of the same module [2]. Approaches to ontology module extraction comprise logic-based extraction methods, for example [10], [13], [24] and [14]. These approaches are automatic and aim at producing self-contained, consistent ontology modules. They make use of logical properties such as semantic locality and inseparability.

While the latter two classes comprise approaches for *a posteriori* modularization of existing ontologies, a third arising class of methodological approaches aim at modular construction of ontologies in an *a priori* manner. Related work in this area has been accomplished by [25], proposing a methodological framework for constructing modular ontologies driven by knowledge granularity. The proposed approach involves a separation into three levels: an upper ontology, modeling the theoretical framework, domain ontologies for reusable domain knowledge, and domain ontologies for application specific knowledge.

The shortcoming of existing modularization approaches is, as already mentioned in the introduction, their one-dimensionality, which is also acknowledged by [6] and [5]. The latter propose more unified approaches to the problem, however, they are restricted to the (graph-based) RDF model. Moreover, they lack formalisms of mapping modularizations to requirements, hindering relaying and re-use of module specifications. In [16], the authors have introduced the idea of microtheories in order to segment the increasingly large Cyc knowledge base into easier to handle modules. Microtheories are categories, laid out in a hierarchy, under which assertions of the knowledge base can be subsumed. An assertion is true in its associated microtheory and all sub-microtheories. Microtheories differ from our approach in that they provide a static context, while ontology aspects have a more dynamic characteristic. They also differ in that each assertion in the

Cyc knowledge base can only exist under one microtheory (and its subtheories), while assertions can have an arbitrary number of aspects.

5 Conclusion and Outlook

In this paper we presented our approach to weaving context knowledge represented by the means of metamodelling into OWL 2 ontologies, accomplishing our aspect-oriented approach to modular ontology development presented in previous works. Aspects represent modal context, and ontologies may be partitioned into subsets of axioms, according to which contexts they belong to. While we have shown earlier that aspect-oriented ontology development facilitates modular (contextualized) ontology development and modular reuse, the extension of the syntax and semantics of the OWL language hinders use of aspect-oriented ontologies in situations where access to the entire ontology along with the context knowledge is desired.

In this paper, we could show that the approach of an aspect-weaving facility, as employed in aspect-oriented software development, is appliccable to the problem. The aspect-weaver presented here is able to identify particular kind of aspects, using the OPPL selection language and transforming them into valid OWL 2 constructs, preserving the context knowledge by incorporating it into the ontology following a selected set of Ontology Design Patterns.

One shortcoming of the approach is its being restricted to structural patterns, which might not catch all possible varieties modal context representing aspects. For example, the context slices pattern simply relies on the structure of the networks of possible worlds and the semantic characteristics of the accessibility relation, which, in this case, must not be reflexive. By considering these characteristics only, it is not possible to distinguish between an intended doxastic interpretation and, for example, a deontic interpretation, where frames are not reflexive either.

One benefit of the approach using ODPs, however, lies in the fact that it may be extended with additional patterns. It is also possible to extend the patterns catalog beyond structural patterns and add, for example, content ODPs in situations where the types used for expressing aspects are known. This way, a classification of aspect types (as also presented in our earlier work) may provide additional knowledge about the kind of intended meaning and therefore provide more fine-grained control over the transformation.

Future work will include extending the list of used patterns and a formal evaluation.

Acknowledgments

This work has been partially supported by the “InnoProfile-Transfer Corporate Smart Content" project funded by the German Federal Ministry of Education and Research (BMBF) and the BMBF Innovation Initiative for the New German Länder - Entrepreneurial Regions.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, USA (2003)
2. Coskun, G., Rothe, M., Teymourian, K., Paschke, A.: Applying community detection algorithms on ontologies for indentifying concept groups. In: Proceedings of the 5th International Workshop on Modular Ontologies. Ljubljana, Slovenia (September 2011)
3. Cuenca Grau, B., Parsia, B., Sirin, E.: Combining OWL ontologies using $\mathcal{E}$-Connections. Web Semantics: Science, Services and Agents on the World Wide Web 4(1), 40–59 (Jan 2006)
4. d'Aquin, M.: Modularizing Ontologies. In: Suárez-Figueroa, M.C., Gómez-Pérez, A., Motta, E., Gangemi, A. (eds.) Ontology Engineering in a Networked World, pp. 213–233. Springer, Berlin, Heidelberg (2012)
5. d'Aquin, M., Doran, P., Motta, E., Tamma, V.A.M.: Towards a parametric ontology modularization framework based on graph transformation. In: Grau, B.C., Honavar, V., Schlicht, A., Wolter, F. (eds.) WoMO. CEUR Workshop Proceedings, vol. 315. CEUR-WS.org (2007)
6. Doran, P., Palmisano, I., Tamma, V.A.M.: Somet: Algorithm and tool for sparql based ontology module extraction. In: Sattler, U., Tamilin, A. (eds.) WoMO. CEUR Workshop Proceedings, vol. 348. CEUR-WS.org (2008)
7. Filman, R., Friedman, D.: Aspect-Oriented Programming Is Quantification and Obliviousness. Workshop on Advanced Separation of Concerns, OOPSLA (2000)
8. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) The Semantic Web – ISWC 2005, pp. 262–276. No. 3729 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (Jan 2005)
9. Gangemi, A., Presutti, V.: Ontology Design Patterns. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 221–243. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
10. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Extracting Modules from Ontologies: A Logic-Based Approach. In: Stuckenschmidt et al. [23], pp. 159–186
11. Grau, B.C., Parsia, B., Sirin, E., Kalyanpur, A.: Automatic Partitioning of OWL Ontologies Using $\mathcal{E}$-Connections (2005)
12. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP'97 — Object-Oriented Programming, Lecture Notes in Computer Science, vol. 1241, pp. 220–242. Springer Berlin / Heidelberg (1997)
13. Konev, B., Lutz, C., Walther, D., Wolter, F.: Semantic Modularity and Module Extraction in Description Logics. In: Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence. pp. 55–59. IOS Press, Amsterdam, The Netherlands, The Netherlands (2008)
14. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Logic-based ontology comparison and module extraction, with an application to DL-Lite. Artificial Intelligence 174(15), 1093–1141 (Oct 2010)
15. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: E-connections of description logics. In: Calvanese, D., Giacomo, G.D., Franconi, E. (eds.) Proceedings of the 2003 International Workshop on Description Logics (DL2003), Rome, Italy September 5-7, 2003. CEUR Workshop Proceedings,

vol. 81. CEUR-WS.org (2003), http://SunSITE.Informatik.RWTH-Aachen.de/Publications/CEUR-WS/Vol-81/wolter-1.pdf
16. Lenat, D.B., Guha, R.V.: The Evolution of CycL, the Cyc Representation Language. ACM SIGART Bulletin - Special issue on implemented knowledge representation and reasoning systems 2(3), 84–87 (Jun 1991)
17. Parent, C., Spaccapietra, S.: An Overview of Modularity. In: Stuckenschmidt et al. [23], pp. 5–23
18. Schäfermeier, R.: Aspect-Oriented Ontology Development. In: Abramowicz, W. (ed.) Business Information Systems Workshops, pp. 208–219. No. 160 in Lecture Notes in Business Information Processing, Springer Berlin Heidelberg (2013)
19. Schäfermeier, R., Paschke, A.: Aspect-Oriented Ontologies: Dynamic Modularization Using Ontological Metamodeling. In: Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS 2014). pp. 199 – 212. IOS Press (2014)
20. Schild, K.: A correspondence theory for terminological logics: Preliminary report. In: Mylopoulos, J., Reiter, R. (eds.) Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991. pp. 466–471. Morgan Kaufmann (1991)
21. Schlicht, A., Stuckenschmidt, H.: A Flexible Partitioning Tool for Large Ontologies. In: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01. pp. 482—488. WI-IAT '08, IEEE Computer Society, Washington, DC, USA (2008)
22. Steimann, F.: Domain Models Are Aspect Free. In: Briand, L., Williams, C. (eds.) Model Driven Engineering Languages and Systems, pp. 171–185. No. 3713 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (Jan 2005)
23. Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.): Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization. Lecture Notes in Computer Science, Springer Berlin / Heidelberg (2009)
24. Suntisrivaraporn, B.: Module Extraction and Incremental Classification: A Pragmatic Approach for $\mathcal{EL}$ Ontologies. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) The Semantic Web: Research and Applications, pp. 230–244. No. 5021 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (Jan 2008)
25. Thakker, D., Dimitrova, V., Lau, L., Denaux, R., Karanasios, S., Yang-Turner, F.: A priori ontology modularisation in ill-defined domains. In: Proceedings of the 7th International Conference on Semantic Systems. pp. 167–170. I-Semantics '11, ACM, New York, NY, USA (2011)