# An Aspect-Oriented Extension to the OWL API
## *Specifying and Composing Views of OWL Ontologies using Ontology Aspects and Java Annotations*

Ralph Schäfermeier, Lidia Krus and Adrian Paschke

*Corporate Semantic Web Group, Institute of Computer Science, Freie Universität Berlin, Berlin, Germany*

Keywords: Modular Ontology Development, Aspect-Oriented Programming, Ontology APIs.

Abstract: Aspect-Oriented Programming (AOP) is a technology for the decomposition of software systems based on cross-cutting concerns. As shown in our previous work, cross-cutting concerns are also present in ontologies, and Aspect-Oriented Ontology Development (AOOD) can be used for flexible and dynamic ontology modularization based on functional and non-functional requirements. When ontologies are used in applications, application and ontology-related requirements often coincide. In this paper, we show that aspects in ontologies can be expressed as software aspects and directly referred to from software code using the well-known AspectJ language and Java annotations. We present an extension of the well-known OWL API with aspect-oriented means that allow transparent access to and manipulation of ontology modules that are based on requirements.[a]

## 1 INTRODUCTION

Ontologies provide a formal representation of shared knowledge for use in information systems. Recent years have witnessed a significant adoption of ontologies in IT, not least due to the standardization activities by the W3C leading, among others, to the inception of the Web Ontology Language *OWL*, which can be used to describe ontologies and knowledge bases[1] using Description Logics (DL).

Still, the authoring of ontologies is a potentially complex and resource intensive process. Therefore, reusability of existing ontologies is generally desired. However, reuse of ontologies is often not trivial, as many of the existing ontologies designed with this purpose in mind are extremely large, and only small parts of them are actually relevant in a particular reuse scenario (Noy and Musen, 2004).

Ontology modularization tackles the problem of creating meaningful reusable ontology modules or views of large monolithic ontologies that can be easier explored or extracted. Beyond reuse, ontology modularization may serve a number of further goals, such as the improvement of reasoning and query result retrieval performance, scalability for ontology evolution and maintenance, complexity management, amelioration of understandability, context-awareness, and personalization (Parent and Spaccapietra, 2009).

A significant number of approaches to the problem of ontology or knowledge base modularization therefore exist, each of them tackling one or several of the above-mentioned problems.

In our previous work (Schäfermeier and Paschke, 2014), we have presented Aspect-Oriented Ontology Development (AOOD) as a unified, goal independent approach to defining syntactic ontology modules, which was inspired by the Aspect-Oriented Programming (AOP) or Aspect-Oriented Software Development (AOSD) paradigm. A commonly accepted notion of an ontology module is that defined by the con-

[a]Source code and documentation are available at https://github.com/ag-csw/aspect-owlapi

[1]There is a dissent between different communities on the exact definition of the term *ontology*. Some communities define an ontology as only the conceptualization of a specific discourse domain, i.e., a statemtent of a logical theory about concepts and their relations. A knowledge base, by this definition, is a set of assertions about concrete objects (instance data), grounded in the conceptualization given by the ontology. In this paper, we commit to the definition of ontologies by Guarino and Giaretta (Guarino and Giaretta, 1995), according to which a knowledge base is a special kind of ontology. Therefore, whenever we speak of ontologies, we also include what others would refer to as knowledge bases. We have made this choice because in this work, we exclusively talk about OWL 2 ontologies, and the OWL 2 specification does not make this terminological distinction either[2].

[2]http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

Schäfermeier, R., Krus, L. and Paschke, A..
An Aspect-Oriented Extension to the OWL API - Specifying and Composing Views of OWL Ontologies using Ontology Aspects and Java Annotations.
In *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015) - Volume 2: KEOD*, pages 187-194
ISBN: 978-989-758-158-8
Copyright © 2015 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

cept of conservative extensions as proposed by (Grau et al., 2008) and (Konev et al., 2009), which guarantee that a module is self-contained in the sense of being semantically equivalent to its parent ontology wrt. a particular signature. A syntactic module, on the other hand, is defined less rigidly as a set of axioms that is a subset of all axioms of the parent ontology.

Applications using ontologies come with a set of requirements. A subset of these requirements are related to how the ontology is going to be used and what the application developers expect to get back from the ontology. These requirements are in fact related to the ontology itself, still they concern the application's mode of operation. As those particular requirements affect both the application and the ontologies in the same manner, it appears natural to have them reflected in a unified way in the application code as opposed to have them scattered across the application. For this reason, we have developed a formalism that allows for a direct mapping of software aspects to ontology aspects and a system that permits to control the use of ontology aspects from within the application code in a declarative fashion.

The contribution of this paper consists of a software artifact that makes ontology aspects accessible to developers of semantic web applications. We have implemented it in the form of an aspect-oriented extension to the well-known OWL API[3], a Java API for OWL 2 ontologies, using AspectJ[4] as a Java aspect language and Java annotations as a way to declaratively control ontology aspects from Java code.

The remainder of this paper is structured as follows. In section 2, we decribe the formalism of aspect-oriented ontologies for OWL. In Section 3, we demonstrate how we make this formalism available in Java code using Java annotations. In Section 4, we discuss the results on the basis of a an experimental evaluation using an ontology from the arts domain that contains AOOD aspects for temporal attribution, provenance information and reasoning complexity. Section 5 gives an overview of related work. Section 6 concludes and points out future work.

## 2 ASPECT-ORIENTED ONTOLOGY DEVELOPMENT

In (Schäfermeier and Paschke, 2014), we describe a formalism for Aspect-Oriented Ontology Development (AOOD) which we derived from the basic principles of Aspect-Oriented Programming (AOP). Aspects in software are self-contained modules that comprise a well-defined fragment of a system's functionality and instructions on how to combine this functionality with the main system, the former being referred to as *advice* and the latter being referred to as *pointcuts*.

Advice is regular software code, in the majority of cases written in the same programming language as the main system. A pointcut is a collection of so called *join points*, points in the code of the main system where the advice of the aspect is supposed to be executed. A pointcut can be an exhaustive list of join points or an abstract description thereof by the means of *quantification*.

Thereby, advice adds a well-defined piece of functionality to an existing system, while the pointcut contains all the necessary information on where exactly to add it. Each piece of information reflects a requirement or business concern. As defined by the IEEE standard 1471 of software architecture (Group, 2000), *"concerns are those interests which pertain to the systems development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders"*. The functionality in the advice normally reflects a cross-cutting concern, i.e. the implementation of a requirement that cross-cuts with other requirements of the system.

As in software development, cross-cutting concerns are linked to requirements. Requirements can be functional, i.e., directly related to the business goals the system or ontology is supposed to accomplish. Non-functional requirements, in contrast, are related to goals concerning the system or ontology itself. Functional ontology requirements are generally directly related to the competency questions the ontology is supposed to answer, while examples for non-functional requirements include provenance information, multilinguality and reasoning complexity.

Each of these requirements is mapped to one ore multiple sets of axioms in an ontology. If there exists a relationship between an aspect and an axiom, it means that this axioms belongs to the aspect in question. This way, an ontology can be modularized into (possibly overlapping) modules, each module representing a requirement, and each module-requirement pair being encapsulated in an aspect.

Aspects can, in turn, be formal descriptions themselves, i.e., just as the ontology module they are mapped to, they can be ontological statements as well. For example, a set of facts in an OWL ontology can be related to a temporal aspect, e.g. *Bonn is capital of West Germany*, which was only valid from 1949 to 1990, where the temporal aspect is an OWL individual that comes with relations and properties formal-

[3] http://owlapi.sourceforge.net/
[4] https://eclipse.org/aspectj/

ized using the W3C time ontology and representing the time interval *1949-1990*.

The individual representing the aspect may be directly referenced by its IRI (if it is a named individual). Beyond that, it is possible to define super aspects by using some sort of query (e.g. *all the things that happened during the 20th century* which is equivalent to all facts in the ontology that are related to temporal aspects which are valid between 1900 and 1999).

## 2.1 An Aspect Vocabulary for Ontologies

We define a vocabulary that is suitable for a sound and complete description of aspects in ontologies on an abstract level. Software aspects apply to lines of code where the execution flow is diverted due to a call to a method, function or routine (or whichever callable units the paradigm of the programming language at hand defines), i.e., each call is a candidate for a join point, and the system can be changed by advising the join points.

In the case of ontologies, the application of an aspect would mean the modification of factual (Tbox or Abox) knowledge. Hence, elements that can be used as join points in ontologies must be expressions that represent canonical units of factual knowledge. What these simple expressions exactly are is defined by the knowledge representation model of each specific language. In the case of RDF they are triples, while in the case of OWL, they are axioms. Once the simple expressions of a language have been identified, we can map the relevant concepts from the AOSD domain to ontologies and come up with an abstract definition of ontology aspects:

*Pointcut:* A pointcut in an ontology is the set of simple expressions that will be affected by the facts that are contained in the aspect (and stem from a cross-cutting requirement).

*Advice:* An advice in an ontology is a set of facts that will be applied to the facts in the pointcut and, depending on their nature, extend or restrict the facts in the pointcut. A language aspect might, for example, add language specific information (a translation) to the facts in the pointcut, while a temporal aspect might impose a temporal restriction on them (with the consequence that they are only valid during a specified period in time).

*Aspect:* An aspect in an ontology is a compound entity consisting of a pointcut and an advice.

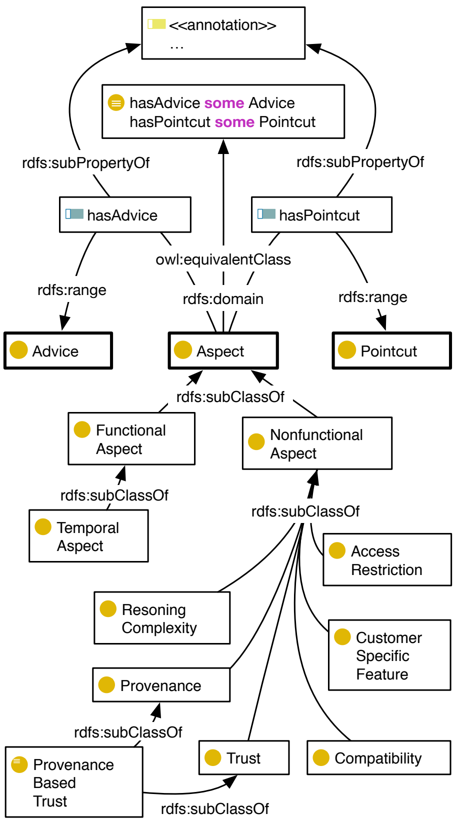

Figure 1: The aspect ontology for OWL 2 ontologies conceptualizing the terms *Aspect*, *Pointcut* and *Advice*. The two annotation properties *hasAdvice* and *hasPointcut* can be applied to axioms (as opposed to only individuals) and are not part of the OWL semantics. The class hierarchy under Aspect is preliminary and designed to be extended by users. The separation of functional and non-functional aspects corresponds to functional vs. non-functional concerns.

## 2.2 Embedding Aspects in OWL Ontologies

The foundation for aspect-oriented ontologies is an ontology that formalizes the concept of an aspect as described in the above Section 2.1. The ontology is depicted in Figure 1. Given this ontology of aspects, we define an aspect-oriented ontology module as follows:

**Definition 1** (Aspect-oriented Ontology Module)**.** *Given an ontology $O$ that consists of a finite set of axioms $\mathsf{Ax}_O$, an aspect ontology $O_A$ with the set $\mathcal{A} \subseteq \mathsf{Sig}(O_A)$ of all aspect individuals in $O_A, \mathcal{A} = \{a \,|\, \exists i \in O_A : \mathsf{hasAdvice}(a,i)\}$ an aspect individual $a_0 \in \mathcal{A}$ and a predicate* hasPointcut, *then $O_{a_0} \subseteq O$, consisting of the set of axioms $\mathsf{Ax}_{O_{a_0}} := \{\mathsf{ax} \in \mathsf{Ax}_O \,|\, \mathsf{hasPointcut}(a_0, \mathsf{ax})\}$ is an ontology module defined by the aspect $a_0$.*

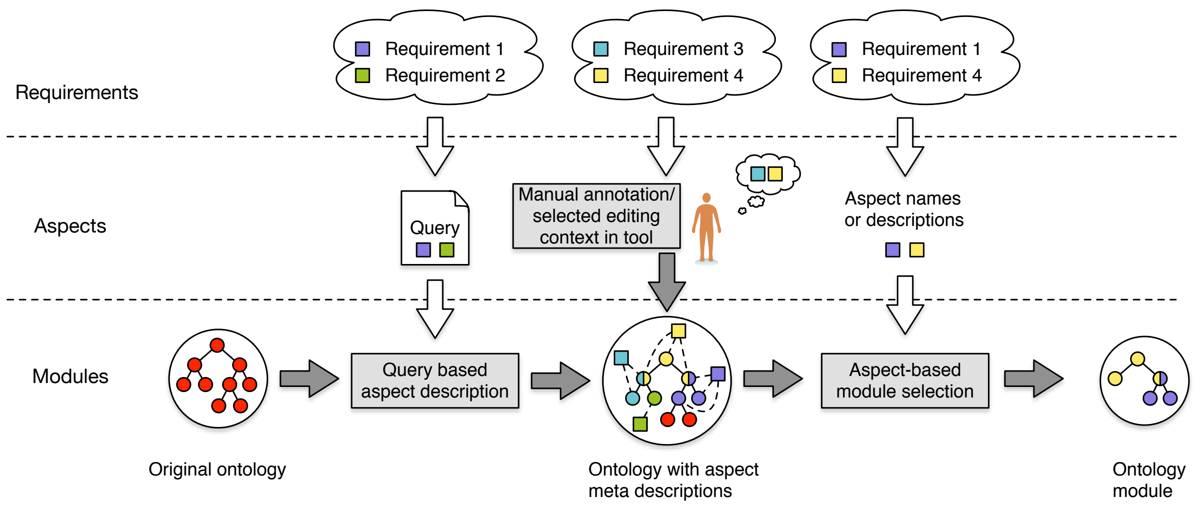
Requirements
Requirement 1
Requirement 2
Requirement 3
Requirement 4
Requirement 1
Requirement 4
Aspects
Query
Manual annotation/ selected editing context in tool
Aspect names or descriptions
Modules
Query based aspect description
Aspect-based module selection
Original ontology
Ontology with aspect meta descriptions
Ontology module

Figure 2: The approach of Aspect-Oriented Ontology Development (AOOD): Parts of an ontology are annotated with aspects by some query (left) or manually (center). Later, parts of the ontology may be extracted by giving a name or specifying a query (right). The aspect descriptions provide a view on ontology modules (bottom). On a higher level, each aspect represents a requirement (top).

Aspect individuals can either be named or anonymously defined by some query (in, for example, SPARQL or SPARQL DL), which would then correspond to the quantification principle mentioned above.

The connnection of an aspect individual to an ontology axiom is a kind of reified statement. While reifcation is defined syntactically on the data exchange level (namely for RDF), OWL does not define semantics for it (and therefore, there exists no direct syntactic category for reification in OWL). The Aspect-Oriented Ontology Development approach uses reifacation as the infrastructure, with a particular semantics described above. Technically, instead of introducing a dedicated syntactic category, for the sake of backwards compatibility, we use the OWL 2 annotation mechanism for making the connection between aspects and their advices and pointcuts.

## 3 OWL ASPECTS IMPLEMENTED AS GENERIC ASPECTJ ASPECTS SUPPLEMENTING THE OWL API

The most notable APIs for web ontologies are Apache Jena[5] and the OWL API[6]. With both being Java APIs, Jena's scope are RDF graphs while the OWL API is tailored to the OWL language[7], providing an object model for OWL entities and axioms and interfaces for DL-based reasoning and explanation support. In what follows, we describe an extension of the OWL API by programmatic access to OWL aspects as defined in the previous sections.

### 3.1 Requirements

We defined the functionality of our approach in terms of functional requirements, which can be seen in Table 1.

We designed an aspect-oriented extension of the OWL API using the Java aspect language AspectJ[8]. We use Java aspects in order to intercept read/write access to the OWL API's Java object model of a loaded OWL ontology and advice the calls by code responsible for extracting ontology modules affected by an OWL aspect.

With regard to requirement 2 we decided to use Java annotations as a means to convey the selected aspects from the Java side.

The heart of the extension is a set of pointcut definitions that intercept all calls to the OWL API that either return or manipulate OWL entities or axioms. Client code using the OWL API can use the `@OWLAspect` java annotation type to specify one ore more aspect IRIs either on the method or class level. If one or more annotations of that type are encountered, then all operations on the OWL API within

[5] http://jena.apache.org/
[6] http://owlapi.sourceforge.net/

[7] At the time of the writing of this report, the target language of the OWL API was OWL 2.
[8] https://eclipse.org/aspectj/

Table 1: Functional requirements for the OWL API extension with ontology aspects.

| ID | Name | Description |
|---|---|---|
| F-1 | aspects | The system should allow for mapping aspects to ontology modules. |
| F-1.1 | aspect declaration | The system should permit the mapping of declarative aspect descriptions to OWL aspects in the set of loaded ontologies. |
| F-1.1.1 | aspect identification | Aspects should be ontological entities identified by IRIs, as defined in the AspectOWL ontology. |
| F-1.1.2 | aspect combination | The declarative selection of aspects should allow for a combination of multiple aspects using logical AND/OR operations. |
| F-2 | ontology change | If an aspect declaration is associated with code manipulating an ontology, then the manipulation must only affect this aspect |
| F-2.1 | axiom addition | If an axiom is added, then it will be annotated with the aspect(s) present in the declaration. |
| F-2.2 | axiom deletion | If an axiom is deleted, and an aspect declaration is present in the Java code, then only the corresponding aspect associations must be deleted in the ontology (the axiom is only removed from this aspect, not from the ontology). |
| F-3 | module extraction | If an aspect declaration is associated with code reading from an ontology, then only that part of the ontology is returned, which is associated with the given aspects. The rest of the ontology is hidden. |

Listing 1: Example client code using the OWL API. Note that aspect references are added transparently and uninvasively as method annotations. Client code itself does not need to be changed.

```
@OWLAspect({"http://www.fu-berlin.
    de/csw/ontologies/aood/
    ontologies/aspect123", "http://
    www.fu-berlin.de/csw/ontologies/
    aood/ontologies/myAspect456"})
public void doSomething () {
  Set<OWLAxiom> allAxioms =
      myOntology.getAxioms();
  ...}
```

the context of the annotation will only be executed on the subset of the ontology that corresponds to the aspect(s) specified by the annotations. For example, a call to `OWLOntology.getAxioms()` from a client method with the annotations from Listing 1 would only return those axioms that belong to the modules specified by the two IRIs used in this example. Accordingly, write operations would also only be performed on the subset.

## 3.2 Conjunctive and Disjunctive Combination of Aspects

In order to fulfill requirement F-1.1.2, the annotation type system had to be extended by some kind of boolean arithmetics that allows for conjunctive and/or disjunctive combination of multiple aspects. The Java annotations system does not permit relations between single annotations in the same place, but nesting of annotations is possible.

In order to provide a syntax for boolean operations, we subclassed the OWLAspect annotation type by OWLAspectAnd and OWLAspectOr.

Unfortunatley, nesting of annotations is restricted to non-cyclic nesting levels, disallowing arbitrary nesting (and thereby arbitrary boolean combinations). However, using the distributivity property of boolean operations, it is possible to bring every boolean formula into a non-nested form. As a consequence, this approach is feasible for arbitrary boolean combinations of aspect declarations conveyed using Java annotations.

An example of a combination would be:

```
@OWLAspectOr({
  @OWLAspectAnd({"http://...#Aspect1",
    "http://...#Aspect2"}),
  @OWLAspectAnd({"http://...#Aspect2",
    "http://...#Aspect3"})
})
```

## 3.3 Syntactic vs. Semantic Modules

The above approach allows for the selection of subsets of axioms, also referrred to syntactic modules as mentioned in Section 1. Depending on the use case, this might be sufficient, or it might be necessary to extend the selected set of axioms to a proper semantic module, such that the parent ontology is a conservative extension of the module. The system we present here may extend the selected subset to a proper module by using syntactic locality (Del Vescovo et al., 2012) as an approximation.

## 4 CASE STUDY

We conducted an experimental evaluation of our approach in the for of a case study using an ontology from the arts domain, containing information about paintings as well as context information in the form of

Table 2: Information encoded in the ontology from the case study: Famous paintings and the paintings' creators and locations. OWL aspects comprise (a) time, (b) provenance, and (c) reasoning complexity (not shown in the table).

| Painter | Painting | Location | Time period | Provenance |
|---|---|---|---|---|
| Raphael | Sistine Madonna | Italy | 1513-1754 | Wikipedia |
| | | Germany | 1754-1945 | Wikipedia |
| | | | since 1955 | Wikipedia |
| | | Russia | 1945-1955 | Wikipedia |
| Dürer | Felsenschloss | Germany | 1494-1945 | Berliner Zeitung |
| | | | since 2000 | Berliner Zeitung |
| | | Russia | 1945-2000 | Berliner Zeitung |
| | Frauenbad | Germany | 1496-1945 | The New York Times |
| | | | since 2001 | Spiegel |
| | | Azerbaijan | 1993 | The New York Times |
| | | USA | 1997-2001 | Spiegel, The New York Times |
| | Johannes der Täufer | Germany | 1503-1945 | Die Welt |
| | | | since 2004 | Die Welt |
| | | Unknown | 1945-2003 | Die Welt |
| | | Estonia | 2003-2004 | Die Welt |

aspects. The particular case stems from a consortium partner in the realm of digital curation of content from the arts domain. The ontology we used is an extended version of the publicly available painting ontology[9]. The ontology was used to model geolocations of contemporary artworks.

The additional aspects that were modeled were

- location changes over time (for which period in time was painting A in location X), using temporal aspects,
- provenance information about each location/time statement (which source stated that painting A was in location X during the time interval T) using provenance aspects, and
- the belonging of each axiom to one or several of the OWL 2 profiles OWL2-EL, OWL2-RL or OWL2-QL, using reasoning complexity aspects.

Table 2 provides an example of the information encoded in the ontology.

For the temporal aspects, the W3C time ontology[10] was used. For the provenance aspects, we used the W3C PROV ontology[11].

The ontology contains 1,378 axioms. We used the OWL API in combination with the Pellet reasoner in order to extract the subsets of the ontology that conform to the OWL-EL, QL, and RL profiles, respectively, and annotated the axioms with the corresponding reasoning complexity aspect.

In order to test the functional requirements, we implemented unit tests in which the ontology described above is loaded and a variety of read and write operations is performed, each using different combinations of aspect annotations.

The results show that the relevant use cases can be captured by the implementation. Competency questions, such as *Where was an artpiece located at a particular point in time?* or a reduced view on the ontology with only axioms belonging to a less expressive but tractable OWL profile for, e.g., only browsing the class hierachy, could all be represented with ontology aspects and corresponding java annotations.

Details on the cases and the different combinations thereof we coverered can may be examined in the github repository.

## 5 RELATED WORK

Aspect-oriented Ontology Development for ontology module definition shares similarities with the work of Doran et al. (Doran et al., 2008) and d'Aquin et al. (d'Aquin et al., 2007).(Doran et al., 2008) proposes SPARQL queries in order to define ontology modules. The authors show that a set of specialized approaches, such as (d'Aquin et al., 2006), (Seidenberg and Rector, 2006), (Doran et al., 2007) and (Noy and Musen, 2000) may be reformulated in the form of SPARQL queries. (d'Aquin et al., 2007) proposes a similar approach using graph transformations and relying on user-defined graph-based extraction rules.

(Grau et al., 2005) presents a partitioning approach using $\mathcal{E}$-Connections (Cuenca Grau et al., 2006). The criterion for the partitioning process is semantic relatedness. This is determined by checking the $\mathcal{E}$-safe property a structural constraint which avoids the separation of semantically dependent ax-

[9] http://spraakbanken.gu.se/rdf/owl/painting.owl
[10] http://www.w3.org/TR/owl-time/
[11] http://www.w3.org/TR/prov-o/

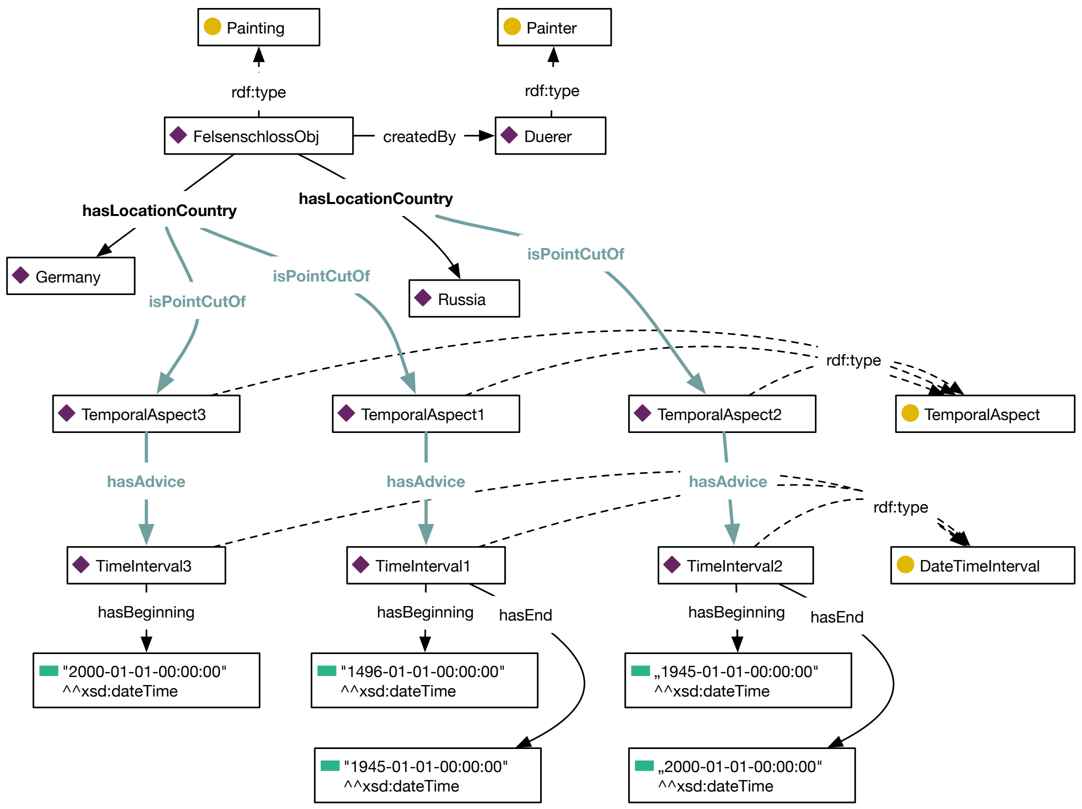

Figure 3: Example of how a temporal aspect (of locations of Dürer's paintings Feldschloss) is concretely expressed in the knowledge base.

ioms in order to achieve semantically consistent modules. The relatedness of the different modules is retained by the $\mathcal{E}$-Connections. Concept subsumption or the use of roles across different modules is not possible. $\mathcal{E}$-Connections apply to ontology entities (classes, individuals and properties), in contrast, our approach applies to axioms.

Schlicht et al. propose a semi-automated approach to ontology partitioning based on application-imposed requirements (Schlicht and Stuckenschmidt, 2008). 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. Then, the ontology is partitioned retaining strongly related groups in the same module. The method is parametrizable by specifying the features taken into account for constructing the dependency graph and the size a module should have in terms of the number of axioms.

While the latter approaches work with *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 (Thakker et al., 2011), 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.

## 6 CONCLUSION AND OUTLOOK

This paper describes an approach to declarative syntactic or semantic ontology module selection for developers of semantic web applications, based on our approach to aspect-oriented ontology modularization. We presented a demo implementation based on the OWL API, which we extended with AspectJ aspects and an annotation system, which is used to specify ontology aspects (referenced by IRIs), which are mapped to ontology modules. In this fashion, operations performed using the OWL API are restricted to the submodule(s) of the ontology specified using the annotations.

The approach is transparent in that developers need not change their application code. As mentioned in section 4, a weakness of the approach is that the pointcut definitions need to be exhaustive in order to capture all relevant method calls in the OWL API. Furthermore, they may potentially break in case of future changes of the OWL API.

Future work comprises the integration of the declaration of aspects using SPARQL or DL based queries from our previous work into the API extension for more dynamic module selection. Currently, an extension to the well-known ontology editor is developed, which allows focused work on ontology modules based on aspects by hiding parts of the ontology that do not belong to a selected aspect. WebProtégé was built on top of the OWL API, and the aspect-oriented extension to the OWL API described in this paper, allows the implementation of this feature in an uninvasive fashion, i.e., without changing any WebProtégé or OWL API source code.

## REFERENCES

Cuenca Grau, B., Parsia, B., and Sirin, E. (2006). Combining OWL ontologies using $\mathcal{E}$-Connections. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(1):40–59.

d'Aquin, M., Doran, P., Motta, E., and Tamma, V. A. M. (2007). Towards a parametric ontology modularization framework based on graph transformation. In Grau, B. C., Honavar, V., Schlicht, A., and Wolter, F., editors, *WoMO*, volume 315 of *CEUR Workshop Proceedings*. CEUR-WS.org.

d'Aquin, M., Sabou, M., and Motta, E. (2006). Modularization: a key for the dynamic selection of relevant knowledge components. In Haase, P., Honavar, V., Kutz, O., Sure, Y., and Tamilin, A., editors, *WoMO*, volume 232 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Del Vescovo, C., Klinov, P., Parsia, B., Sattler, U., Schneider, T., and Tsarkov, D. (2012). Syntactic vs. Semantic Locality: How Good Is a Cheap Approximation? In Schneider, T. and Walther, D., editors, *Workshop on Modular Ontologies (WoMO) 2012*, pages 40–50.

Doran, P., Palmisano, I., and Tamma, V. A. M. (2008). Somet: Algorithm and tool for sparql based ontology module extraction. In Sattler, U. and Tamilin, A., editors, *WoMO*, volume 348 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Doran, P., Tamma, V., and Iannone, L. (2007). Ontology Module Extraction for Ontology Reuse: An Ontology Engineering Perspective. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 61–70, New York, NY, USA. ACM.

Grau, B. C., Horrocks, I., Kazakov, Y., and Sattler, U. (2008). Modular Reuse of Ontologies: Theory and Practice. *Journal of Artificial Intelligence Research*, 31:273–318.

Grau, B. C., Parsia, B., Sirin, E., and Kalyanpur, A. (2005). Automatic Partitioning of OWL Ontologies Using $\mathcal{E}$-Connections.

Group, I. A. W. (2000). IEEE standard 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE.

Guarino, N. and Giaretta, P. (1995). Ontologies and Knowledge Bases Towards a Terminological Clarification. *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, 25:32.

Konev, B., Lutz, C., Walther, D., and Wolter, F. (2009). Formal Properties of Modularisation. In (Stuckenschmidt et al., 2009), pages 25–66.

Noy, N. and Musen, M. (2004). Specifying ontology views by traversal. In McIlraith, S., Plexousakis, D., and van Harmelen, F., editors, *The Semantic Web – ISWC 2004*, volume 3298 of *Lecture Notes in Computer Science*, pages 713–725. Springer Berlin Heidelberg.

Noy, N. F. and Musen, M. A. (2000). PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 450–455. AAAI Press.

Parent, C. and Spaccapietra, S. (2009). An Overview of Modularity. In (Stuckenschmidt et al., 2009), pages 5–23.

Schäfermeier, R. and Paschke, A. (2014). Aspect-Oriented Ontologies: Dynamic Modularization Using Ontological Metamodeling. In *Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS 2014)*, pages 199 – 212. IOS Press.

Schlicht, A. and Stuckenschmidt, H. (2008). 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*, WI-IAT '08, pages 482—488, Washington, DC, USA. IEEE Computer Society.

Seidenberg, J. and Rector, A. (2006). Web Ontology Segmentation: Analysis, Classification and Use. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 13–22, New York, NY, USA. ACM.

Stuckenschmidt, H., Parent, C., and Spaccapietra, S., editors (2009). *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

Thakker, D., Dimitrova, V., Lau, L., Denaux, R., Karanasios, S., and Yang-Turner, F. (2011). A priori ontology modularisation in ill-defined domains. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 167–170, New York, NY, USA. ACM.