Towards a Unified Approach to Modular Ontology Development Using the Aspect-Oriented Paradigm

Ralph Schäfermeier and Adrian Paschke

Freie Universität Berlin,
Königin-Luise-Str. 24-26, 14195 Berlin, Germany
{schaef,paschke}@inf.fu-berlin.de
http://www.corporate-semantic-web.de/

Abstract. In this paper, we describe our ongoing work on the application of the Aspect-Oriented Programming paradigm to the problem of ontology modularization driven by overlapping modularization requirements. We examine commonalities between ontology modules and software aspects and propose an approach to applying the latter to the problem of a priori construction of modular ontologies and a posteriori ontology modularization.

Keywords: ontology modularization, aspect-oriented development, crosscutting concerns

1 Introduction

The majority of existing modularization approaches are specialized solutions, relying on semantic (cf., for example, [1–5]) or structural relatedness (e.g., [6–8]) and are only parametrizable to a limited degree. Parameters often reflect the internal operational mode of the modularization algorithm rather than requirements concerning the expected outcome of the modularization process from a user's point of view. Moreover, requirements may be related to different dimensions of the problem space. They can comprise functional requirements, i.e., requirements directly related to the problem domain or the task an ontology or ontology module is supposed to fulfill, and non-functional requirements, such as provenance information, multilingualism, or tractability of reasoning tasks.

The aspect-oriented programming paradigm allows for the separation of multidimensional requirements into dedicated software code modules (aspects), leading to better code modularity and therefore reusability. Using AOP, modules can be recombined (interwoven) either extensionally, by explicitly marking the points in the code where the execution flow should be diverted to a different module, or intensionally, by specifying a set of properties such code points are expected to have in common.

In this paper, we examine commonalities between ontology modules and software aspects and describe our ongoing work towards the application of the above mentioned principles of the AOP paradigm to ontologies. We argue that the latter enables (a) straightforward development of modular ontologies from scratch, and (b) flexible a-posteriori modularization, driven by user requirements.

2 The Aspect-Oriented Paradigm Applied to Ontologies

Aspect-oriented software programming (AOP) languages provide syntactic means for the separation of so called *cross-cutting concerns* in software code into dedicated modules. As defined by the ISO/IEC/IEEE standard 42010 of software architecture[1], *"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"*. Cross-cutting concerns are concerns which emerge from requirements on different levels, concern the entire or a significant part of the system and are thus scattered across the system, diminishing code locality and hindering system evolution and reusability [9]. See Figure 1 for an explanatory example of cross-cutting concerns.

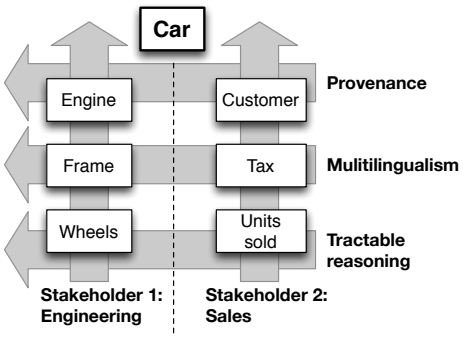

Fig. 1: Cross-cutting concerns by the example of a car ontology. Different stakeholders are interested in different aspects of the core concept (car). The different interests (concerns) reflect requirements formulated by each of the stakeholders. At the same time, stakeholder-independent requirements cross-cut the ontology. Each of theses requirements has a different dimension.

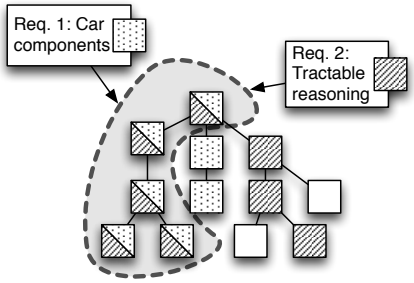

Fig. 2: Selection of an ontology module that satisfies two cross-cutting requirements: It should only contain concepts of the subdomain "car components" of the car domain ("Engineering" aspect, dotted), and it should only contain constructs that allow for tractable reasoning ("Tractability" aspect, dashed). The resulting module (grey) only contains those constructs that are concerned by both aspects.

In AOP terminology, the encapsulation of a single concern in a dedicated module is referred to as *aspect*. An aspect consists of two components: the actual implementation of the functionality, referred to as *advice*, and information about all points in the application's control flow at which the advice should be executed, referred to as *join points*. In this manner, we use the notion of aspects in order to relate ontological constructs to different requirements (see Figure 2).

Listing 1.1 shows an example of a concern "authentication" that cross-cuts with the actual business logic of a bank application and leads to tangled code. Listing 1.2 shows how the authentication concern is encapsulated in an aspect "Authentication". The authentication handling code has been centralized in the advice of the aspect.

[1] `http://www.iso-architecture.org/42010/`

Listing 1.1: Example of the cross-cutting concern "authentication" affecting different parts of the code.

```
withdraw (Account a, float amount) {
  AuthService as = getAuthService();
  if (!as.authenticated(a.user))
    as.authenticate(a.user);
  a.balance -= amount;
}
deposit(Account a, float amount) {
  AuthService as = getAuthService();
  if (!as.authenticated(a.user))
    as.authenticate(a.user);
  a.balance +=amount;
}
```

Listing 1.2: The concern "authentication" is encapsulated in an aspect "Authentication".

```
public Aspect Authentication() {
  AuthService as = getAuthService();
  if (!as.authenticated(a.user))
    as.authenticate(a.user);
}
@aspect Authentication
withdraw (Account a, float amount) {
  a.balance -= amount;
}
@aspect Authentication
deposit(Account a, float amount) {
  a.balance +=amount;
}
```

Note that this example demonstrates the explicit (extensional) variant of join points, in the form of tags assigned to each part of the code where an aspect is applicable. In order to achieve complete detangling of cross-cutting concerns, AOP introduces two principles: quantification and obliviousness.

2.1 Quantification

AOP allows for an intensional definition of join points by using quantified statements in the form

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

where M is the set of all methods defined within the software product, s a predicate specifying the join point properties, $m(p_1,\ldots,p_n) \in M$ a method adhering to the signature $m(p_1,\ldots,p_n)$, and $a(p_1,\ldots,p_n)$ the execution of the advice with all the parameters of each method, respectively [10]. The set of all join points defined by s is called a *pointcut*.

In the case of the authentication example, an instantiation of this formula would be:

$$\forall m(p_1,\ldots,p_n) \in M : sig(m(p_1,\ldots,p_n)) = m(Account\ acc, float\ amount) \\ \rightarrow (m(acc, amount) \rightarrow Authentication(acc, amount)).$$

In order to select ontology axioms in the same fashion, a means of quantification over such axioms is necessary.

$$\forall ax(p_1,\ldots p_n) \in \mathcal{O} : s(ax(p_1,\ldots p_n)) \rightarrow (ax(p_1,\ldots p_n) \rightarrow a(ax(p_1,\ldots p_n))),$$

with $\mathcal{O}$ being an ontology, $ax(p_1,\ldots p_n)$ axioms (in the form of n-ary predicates the domain of which is the union of the vocabulary of the ontology language

in question and the vocabulary of the problem domain, such as class, property and individual names), and $a(ax(p_1, \ldots p_n))$ a function that applies the aspect to $ax(p_1, \ldots p_n)$, whereupon we define the application of an aspect to an axiom as the inclusion of that particular axiom in the module represented by the aspect. In this manner, the aspect "tractability" from the example scenario could be implemented by building a query that selects all axioms which are compatible with the OWL-EL profile.

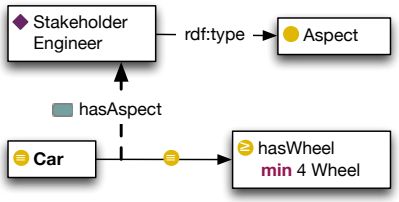

Fig. 3: Extensional join point definition using an OWL annotation.

As mentioned in section 1, join points can also be specified extensionally, for example, by manually annotating axioms which cover a particular aspect with a dedicated OWL Annotation (see Figure 3). This is of particular use for a priori modular ontology development if used, e. g., by an ontology editor with switchable contexts, each context representing an aspect. A user could then extensionally define an aspect-oriented ontology module by switching aspects.

2.2 Obliviousness and Harmlessness

The fact that in aspect-oriented software systems control flow is handed over from the main module to the aspects, making the main module (and any other modules) unaware of the (quantified or extensionally specified) assertions made about it by external aspects that might possibly be applied to it, is termed *obliviousness* [9]. The practical consequence of obliviousness is that the developer of a module is not required to have knowledge about or spend additional effort in anticipation of a possible application of an aspect to her module.

Danters et al. allude to the problem that obliviousness is only guaranteed on a syntactic level while external aspects have in fact the potential to alter the semantics of the target module, making them potentially dangerous [11]. They propose the adaption of the *harmlessness* property for aspect-oriented systems, defining a *harmless aspect* as an aspect which, when applied to a target module, does not alter the semantics of the target.

While it is obvious that the obliviousness property naturally holds for ontology modules, the harmlessness property does not. Nevertheless, it is agreed upon in the recent literature that it is desirable if an ontology module is uninvasive, i.e., its addition to an ontology has no side effects. Whether uninvasiveness of a module applied by the means of an aspect is a required feature or not depends on the specific use case.

Grau et al. [12] as well as Konev et al. [13] propose the notion of conservative extensions which guarantee that a module added to an ontology does not alter the meaning of the original ontology:

Let $\mathcal{O}_1 \subseteq \mathcal{O}$ ontologies, S a signature and L a logic. $\mathcal{O}$ is a conservative extension of $\mathcal{O}_1$ wrt. L, if for every axiom ax with $sig(ax) \subseteq \mathcal{S}$: $\mathcal{O} \models ax$ iff $\mathcal{O}_1 \models ax$.

In the same vein, we define a harmless aspect of an ontology $\mathcal{O}$ as an aspect that yields the selection of a module $\mathcal{O}_1$ that is the conservative extension of $\mathcal{O}$ with respect to L.

It has been noted that determining whether a module is a conservative extension of an ontology is a highly complex problem and even undecidable for expressive ontologies [12, 13]. However, semantic locality is a sufficient condition for a conservative extension [4], and [14] suggests that the less complex syntactic locality constitutes a practically acceptable approximation.

3 Conclusion and Future Work

In this paper, we pointed out that ontology modularization and aspect-oriented programming share interesting commonalities and that the aspect-oriented paradigm can be applied to a priori modular ontology development as well as a posteriori module extraction. The next step will consist in providing a proof-of-concept system that dynamically interweaves aspects defined in the above manner.

Further work is necessary in order to achieve a functional meta description of ontology axioms for the purpose of pointcut definition. The formalism described in section 2.1 works in terms of meta predicates with the domain consisting of vocabulary of the ontology language, reifing axioms contained in the ontology.

The research question raised in this paper is how the application of the aspect-oriented paradigm affects the quality of ontology modularizations. Our hypothesis is that aspect-oriented ontology development yields useful ontology modules wrt. to cross-cutting modularization requirements, such as dynamic access, understandability, maintenance, and re-use. We expect that the intensional specification of ontology modules with pointcuts adds dynamicity and flexibility to modular development, making it easier to evolve modular ontologies in situations where evolution implies modularization requirement changes.

To evaluate our approach and test our hypothesis, we will apply the approach to different modularization use-cases in the context of ontology development projects. Aspects considered in these use cases will comprise project affiliation, temporal attribution, workflow affiliation, re-use, and module understandability. We then use quality metrics in order to assess the quality of the modularizations gained using our approach and compare it with existing approaches.

Acknowledgements

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. 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) (January 2006) 40–59
2. 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, Amsterdam, The Netherlands, The Netherlands, IOS Press (2008) 55–59
3. 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. Number 5021 in Lecture Notes in Computer Science. Springer Berlin Heidelberg (January 2008) 230–244
4. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Extracting Modules from Ontologies: A Logic-Based Approach. [15] 159–186 DOI: 10.1007/978-3-642-01907-4.
5. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Logic-based ontology comparison and module extraction, with an application to DL-Lite. Artificial Intelligence **174**(15) (October 2010) 1093–1141
6. 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.: Proceedings of the 2nd International Workshop on Modular Ontologies, WoMO 2007. Volume 315 of CEUR Workshop Proceedings., CEUR-WS.org (2007)
7. 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. WI-IAT '08, Washington, DC, USA, IEEE Computer Society (2008) 482—488
8. 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)
9. Filman, R., Friedman, D.: Aspect-Oriented Programming Is Quantification and Obliviousness. Workshop on Advanced Separation of Concerns, OOPSLA (2000)
10. Steimann, F.: Domain Models Are Aspect Free. In Briand, L., Williams, C., eds.: Model Driven Engineering Languages and Systems. Number 3713 in Lecture Notes in Computer Science. Springer Berlin Heidelberg (January 2005) 171–185
11. Dantas, D.S., Walker, D.: Harmless Advice. In: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. POPL '06, New York, NY, USA, ACM (2006) 383–396
12. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular Reuse of Ontologies: Theory and Practice. Journal of Artificial Intelligence Research **31** (February 2008) 273–318 ACM ID: 1622664.
13. Konev, B., Lutz, C., Walther, D., Wolter, F.: Formal Properties of Modularisation. [15] 25–66 DOI: 10.1007/978-3-642-01907-4.
14. Del Vescovo, C., Klinov, P., Parsia, B., Sattler, U., Schneider, T., Tsarkov, D.: Syntactic vs. Semantic Locality: How Good Is a Cheap Approximation? In Schneider, T., Walther, D., eds.: Workshop on Modular Ontologies (WoMO) 2012. (2012) 40–50
15. 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) DOI: 10.1007/978-3-642-01907-4.