Object-oriented Software Modeling with Ontologies Around

A Survey of Existing Approaches (*SESE*)[1]

Sohaila Baset
Information Management Institute
University of Neuchatel
Neuchatel, Switzerland
sohaila.baset@unine.ch

Kilian Stoffel
Information Management Institute
University of Neuchatel
Neuchatel, Switzerland
kilian.stoffel@unine.ch

Abstract—**Despite the many integration tools proposed for mapping between OWL ontologies and the object-oriented paradigm, developers are still reluctant to incorporate ontologies into their code repositories. In this paper we survey existing approaches for OWL to OOP mapping trying to identify reasons for this shy adoption of ontologies among conventional software developers. We present a classification of the surveyed approaches and tools based on the characteristics of their resulting artifacts. We finally provide our own reflection for other potential reasons beyond those addressed in the literature.**

Keywords- Knowledge representation, Object Oriented Programming, Software modeling, Ontologies; OWL; Language transformation.

I. MOTIVATION

In software development, like in other engineering disciplines, model sharing is always an encouraged practice. It explains the industry's constant pursuit of open standards for modeling languages that allow for seamless incorporation of models pertaining to a certain modeling school into another. Ontological modeling is no exception. After a few predecessors, Ontolingua [1] [2] and DAML+OIL [3], the Web Ontology Language OWL [4] [5] is now the standard language for developing and sharing ontologies in the semantic web as well as many other fields such as the biomedical domain. In the literature, there exist many attempts at integrating ontologies into mainstream development. These attempts vary from loose integration, i.e. accessing ontologies from a programming language, to a more solid transformation from OWL ontologies into software models.

The synergies between ontologies and software models might seem so evident that in many cases an effortless mapping between the two paradigms is taken for granted. This assumption is further supported by a considerable number of proposed development frameworks such as the Ontology Driven Software Development ODSD [6] or the Ontology Oriented Programming [7]. However, shifting a bit from the research state-of-the-art into the circles of conventional software development, we observe a different image than the one painted in research papers. Despite the many integration tools proposed, developers are still reluctant to incorporate ontologies into their code repositories.

In this paper, we try to examine the different reasons behind the modest adoption of ontologies in software modeling. We survey existing integration approaches and we deduce some of their common characteristics with the goal of providing a classification framework that researchers interested in the topic can refer to. We then discuss some of the common challenges reported in the literature before concluding with our own reflection on the current state of OWL to OOP integration.

II. METHOD AND SCOPE

Given the qualitative nature of literature and the particular topic in question, it is difficult to establish a procedure for automatic classification of existing approaches. When selecting papers to review, we started with three of the earliest papers as seeds for collecting other related papers: OntoJava 2002 [8], Kalynapur 2004 [9] and Knublauch 2004 [6]. Using Google scholar, we harvested all papers that cited at least one of the seed papers. This resulted in around 309 papers. We first grouped similar papers in sets (e.g. papers originating from the same author and/or proposing the same tool) and we chose a representative paper of each group. We then used Google citations metrics as an indicative measure of the impact of a paper (especially for early published papers) to pivot our manual inspection of papers. We dropped irrelevant papers such as papers accentuating the application domain rather than the integration approach. At the end of this process we retained 24 papers that we are surveying in this article. While by no means an exclusive list of all papers in the field, the retained 24 papers give a good indicator on the current state of research.

When reviewing papers, we focused on certain aspects like the extent of integration and the challenges the authors faced

[1] DOI reference number: 10.18293/SEKE2018-198

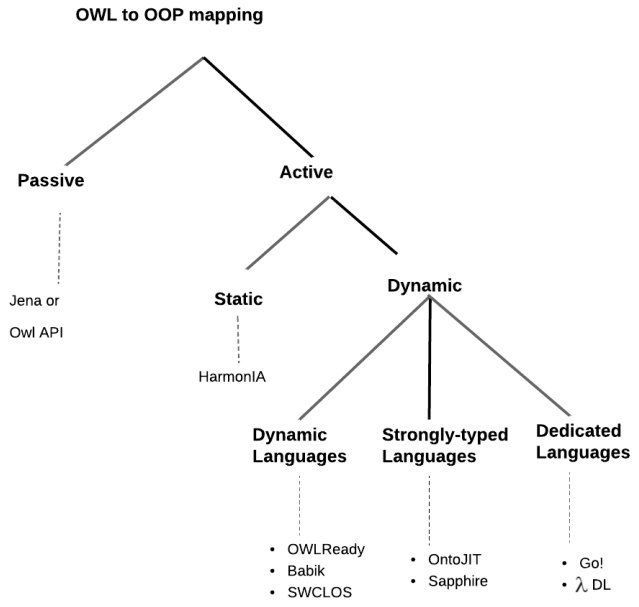

Figure 1. A taxonomy of existing OWL to OOP mapping

rather than focusing on the motivation behind each contribution which was, more or less, common behind most of the reviewed papers.

III. ONTOLOGY MAPPING

Ontologies, by design, are not intended as standalone software units [10]. They need to be considered in the context of an application that is responsible for accessing and manipulating the concepts they represent. For that reason, it is essential to provide some mapping between the content of an ontology and the application environment in which it resides. Scanning the literature, we can find different terms and expressions denoting this sort of mapping. In this paper, we will be using the term OWL to OOP mapping as an umbrella term for integrating OWL ontologies to the object-oriented paradigm in general. This includes both OOP modeling and programming languages.

We can differentiate between two main approaches for OWL to OOP mapping. In the following sections we define these two approaches, their main characteristics as well as their sub-categories.

A. Passive OWL to OOP mapping

Passive OWL to OOP mapping depicts a generic and a rather loose mapping between the content of an ontology and its programming environment. Nevertheless, this approach is more dominant for most applications on the semantic web. In this approach, ontologies are integrated into the mainstream OOP language simply by loading them into memory. Loading is achieved by an ontology loader that transforms the ontology from its syntactic form, e.g. RDF/XML, into an in-memory representation. This in-memory representation can be an Abstract Syntax Tree (AST) like in the case of OWL API loader [11] [12], or an RDF triple-based structure like the one used in Jena [13]. In either case, the loaded ontology is treated as data and will reside in the data segment of the program allocated memory, hence the name passive approach. This approach is also less challenging as there are no constrains imposed by the target programming language on the kind of data structures used to encapsulate the concepts of the source ontology.

B. Active OWL to OOP mapping

In contrast to passive ontology mapping, the active mapping approach will transform an ontology from its serializable format into code statements in the target programming language. The resulting ontology is then executable and belong to the code segment of the allocated memory. This approach is more challenging as it requires finding a native equivalent in the target programming language for each axiom in the source ontology; a requirement that happens to be problematic in many cases as we will demonstrate in the following sections.

Active OWL to OOP mapping can itself be further classified into static or dynamic [14]. In static mapping, the translation, i.e. code generation from owl axioms (concepts, properties and individuals) is done in one shot as a separated prior step. In this case and depending on the target language support for dynamic typing, type checking is mostly limited to compile time. Under this category, we can classify the work done by Kalynapur [9] and Zimmerman [15] to translate OWL ontologies into Java or the .Net ontology compiler by Goldman [7].

Dynamic mapping, on the other hand, will also consider reasoning possibilities about the executable ontology. By dynamically translating OWL axioms at runtime, active OWL to OOP mapping permits certain inference tasks like for example entailing the class of an individual and assigning its type at runtime. Under this approach we can find many tools proposed in the literature and they all provide different degrees of reasoning support. Here again we differentiate between:

1) Tools that have a dynamic language as output and will rely on its dynamic typing features. As examples we can name SWCLOS [16], an OWL processor built on top of Common Lisp Object System (CLOS) and Owlready [14], a python module that beside relying on python interpreter for dynamic typing, can also make use of an external reasoning component (HermiT reasoner [17]) for further reasoning tasks. Under this category as well, we classify the approach proposed by Babik and Hluchy [18], an approach that uses python metaclasses to represent OWL concepts and perform type checking dynamically.

2) Tools that have a strongly-typed language as output but can still offer some degree of flexibility for type changes at runtime. Examples are the Sapphire tool [19] that relies on the concept of cascade wrapping, or un-wrapping, of proxy objects to handle type changes at runtime and the C# OntoJIT parsing component [20] that exploits a mix of metaprogramming techniques, namely C# reflection, with the dynamic compilation support of CLR, the common language runtime of .Net languages.

3) And finally, some of the proposed tools for dynamic OWL to OOP mapping have gone to the extent of proposing a

Table 1- List of main tools and approaches for OWL to OOP mapping

Year	Tool/Approach	Source language	Target language	Mode	Reasoning support
2002	OntoJava [23]	RDF(s)	Java	Active/Static	None
2003	Goldman [7]	OWl	C#	Active/Static	None
2003	OWL API [11]	OWL	Java	passive	External
2004	Knublauch [6]	OWL	Java	___	___
2004	HarmonIA [9]	OWL	Java	Active/Static	None
2004	Jena [13]	OWL	Java	Passive	External
2005	SWCLOS [16]	OWL	CommonLisp	Active/Dynamic	Limited
2005	RDFReactor [24]	RDF/RDF(S)	Java	Active/Static	None
2006	Atkinson et al. [25]	OWL	UML	___	___
2006	Babik [18]	OWL	Python	Active/Dynamic	Limited
2006	Clark et al. [21]	OWL	Go!	Active/Dynamic	Supported
2007	ActiveRDF [26]	RDF(s)	Ruby on Rails	Active/Static	None
2007	Athanasiadis [27]	RDF / OWL	JavaBeans	Active/Static	___
2007	Owlet [28]	OWL	Java	passive	Supported
2008	Puleston et al. [29]	OWL	Hybrid OWL/Java	___	___
2009	OWL2Java [15]	OWL	Java	Active/Static	None
2011	Sapphire [19]	OWL	Java	Active/Dynamic	Limited
2014	LITEQ [30]	RDF(s)	F#	Active/Static	Limited
2016	OntoJIT [20]	OWL	C#	Active/Dynamic	Limited
2017	Owlready [14]	OWL	Python	Active/Dynamic	Indirect
2017	Leinberger et al. [22]	OWL	λ*DL*	Active/Dynamic	Supported

dedicated programming language for that purpose such as Go! [21] or the more recent language λ*DL* [22]. Figure 1. provides a treelike representation of existing OWL to OOP mapping approaches while table 1. provides a somewhat extended list of main tools and approaches.

IV. SEMANTIC GAP

The most prominent challenge that is present in active OWL to OOP mapping approaches is the semantic gap between the ontological and object-oriented paradigms. The semantic richness of ontological languages makes it very difficult to find an OOP counterpart to express OWL semantic constructs. One of the most obvious examples is perhaps the different interpretation of class inheritance. OWL, or Description Logics DL in general, has a looser interpretation of a class being the subclass of another. In OWL, the term "rdfs:subclassOf" is the manifestation of the subsumption operator of DL. An OWL class is allowed to have many parent classes (named or anonymous) as long as it is subsumed by all these parents. On the other hand, pure OOP languages like Java or C# have a stricter definition of class inheritance, OOP classes are disjoint by design and that is why a class cannot be a subclass of two different parent classes and multiple inheritance is, generally speaking, not supported. Multiple inheritance is not the only example of the missing semantic equivalence. A similar argument goes for OWL axioms such as "owl:equivalentClass", "owl:sameAs" or "owl:disjointWith".

Many of the approaches we surveyed did not attempt at bridging this gap and have instead limited the mapping scope to what is expressible in the target programming languages. On the other hand, some of the active approaches proposed interesting solutions ranging from a "Keep it simple, stupid" approach of adding a meta-layer of code as a substitute for the missing semantics in formal programming languages [20] to a more sophisticated approach of stretching the expressiveness of modeling in Java to that of OWL DL by enforcing some constraints and design patterns [9]: Interfaces with shadow classes for multiple inheritance, special listeners on property accessors, type checking for domain and range properties, etc.

V. DISCUSSION

One of the main motivations behind most of the work surveyed in this paper is the difficulty of manipulating ontologies in mainstream software development and the scarcity of options for an ontology programming interface. Nevertheless, as we can see from earlier sections, a retrospective scan on work done in this area revealed a different story. In fact, there exist many options both for accessing or integrating ontologies in an OOP paradigm, yet we still did not witness ontologies spanning new development territories.

Below we try to list some of the potential reasons for this shy adoption of ontologies beyond what has been proposed in the literature:

1) Too many options: The real problem developers may have with integrating ontologies is not necessarily the lack of ontology programming interfaces – there exist well many – but rather the lack of consensus on a standardized option. Unlike the case of ontological modeling where OWL is "the language" and Stanford protégé is "the editor"; when it comes to integrating ontologies as software models, there exist many options but none of them has reached a good level of maturity to gain community consensus. As a result, the developer has to go through the hassle of sorting them out before being able to judge on the pertinence of any of these options; a task that is not affordable in most of today's agile software projects.

2) Paradigm shift: Although largely addressed in the literature, the paradigm shift the developer has to go through when integrating ontologies is still present. Providing tool support is one thing, but it takes much more to overcome the conceptual switch behind ontological modeling. Translating ontologies into a program does not change the fact that ontological modeling is explicit and most of the time based on an Open World Assumption OWA in contrast to implicit closed-world modeling in UML and OOP languages.

3) Legacy projects: From a pure practical point of view, introducing ontologies to mainstream software projects is especially challenging when there is some legacy code to maintain and respect; which is the case of the majority of software projects in large scale organizations.

4) Resistance to change: For most “right-wing” software engineers, adopting ontological approaches, or generally speacking linked data approaches from the semantic web, means, in a way or another, shifting towards a more volatile domain model. A move that may not be perceived as a positive step in the circles of software engineering with strong preferences for tidy and well-engineered domain models. It provokes a lot of philosophical discussions similar to the dynamic vs. static-style of coding in languages that permits the two possibilities. Eventually, such a change may very well be welcome, but just when the right time comes.

VI. Conclusion

In this paper, we surveyed the literature for existing approaches for mapping between OWL ontologies and object-oriented programming paradigms. We presented a classification of the surveyed mapping tools based on the characteristics of their resulting artifacts. We highlighted some of the common challenges encountered in the literature before finally providing our own reflection on why software engineers are still reluctant to incorporate ontologies into their code repositories. Unfortunately, as we mentioned before, most of the tools and prototypes, especially the early ones, did not yield a concrete body of use cases in software industry. It would be interesting therefore to see how the more recent propositions will evolve given the re-awaken interest of the semantic web in the last few years.

VII. References

[1] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, pp. 199-220, 1993.

[2] A. Farquhar, R. Fikes and J. Rice, "The ontolingua server: A tool for collaborative ontology construction," *International journal of human-computer studies*, vol. 46, no. 6, pp. 707-727, 1997.

[3] I. Horrocks and others, "DAML+OIL: A Description Logic for the Semantic Web," *IEEE Data Eng. Bull.*, vol. 25, pp. 4-9, 2002.

[4] I. Horrocks, P. F. Patel-Schneider and F. Van Harmelen, "From SHIQ and RDF to OWL: The making of a web ontology language," *Web semantics: science, services and agents on the World Wide Web*, vol. 1, pp. 7-26, 2003.

[5] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz and others, "Owl 2 web ontology language: Profiles," *W3C recommendation*, vol. 27, p. 61, 2009.

[6] H. Knublauch, "Ontology-driven software development in the context of the semantic web: An example scenario with Protege/OWL," in *1st International Workshop on the Model-Driven Semantic Web (MDSW2004)*, 2004.

[7] N. M. Goldman, "Ontology-oriented programming: static typing for the inconsistent programmer," in *International Semantic Web Conference*, Florida, 2003.

[8] A. Eberhart, "OntoJava--Applying Mainstream Technology to the Semantic Web," in *Workshop on Semantic Web-based E-Commerce and Rules Markup Languages at the ICEC*, Vienna (Austria), 2001.

[9] A. Kalyanpur, D. J. Pastor, S. Battle and J. A. Padget, "Automatic Mapping of OWL Ontologies into Java.," in *SEKE*, 2004.

[10] Wache, Holger and Voegele, Thomas and Visser, Ubbo and Stuckenschmidt, Heiner and Schuster, Gerhard and Neumann and Holger and Hübner, Sebastian, "Ontology-based integration of information-a survey of existing approaches," in *IJCAI-01 workshop: ontologies and information sharing*, Seattle, USA, 2001.

[11] S. Bechhofer, R. Volz and P. Lord, "Cooking the Semantic Web with the OWL API," in *International Semantic Web Conference*, 2003.

[12] S. K. Bechhofer and J. J. Carroll, "Parsing owl dl: trees or triples?," in *Proceedings of the 13th international conference on World Wide Web*, 2004.

[13] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters*, 2004.

[14] J.-B. Lamy, "Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies," *Artificial intelligence in medicine*, vol. 80, no. 2017, pp. 11-28, 2017.

[15] M. Zimmermann, "OWL2Java: a Java Code Generator for OWL, website," -, -, 2009.

[16] Koide, Seiji and Takeda and Hideaki, "OWL-Full reasoning from an object oriented perspective," in *Asian Semantic Web Conference*, Beijing, 2006.

[17] Shearer, Rob and Motik, Boris and Horrocks and Ian, "HermiT: A Highly-Efficient OWL Reasoner.," in *OWLED*, Washington, DC, 2008.

[18] M. Babik and L. Hluchy, "Deep integration of python with web ontology language," in *In Proceedings of the 2nd Workshop on Scripting for the Semantic Web*, 2006.

[19] Stevenson, Graeme and Dobson and Simon, "Sapphire: Generating Java runtime artefacts from OWL ontologies," in *International Conference on Advanced Information Systems Engineering*, Gdańsk, Poland, 2011.

[20] S. Baset and K. Stoffel, "OntoJIT: Parsing Native OWL DL into Executable Ontologies in an Object Oriented Paradigm," in *International Experiences and Directions Workshop on OWL*, 2016.

[21] Clark, Keith L and McCabe and Frank G, "Ontology oriented programming in Go!," *Applied Intelligence*, vol. 24, no. 3, pp. 189--204, 2006.

[22] M. Leinberger, R. Lämmel and S. Staab, "The essence of functional programming on semantic data," in *European Symposium on Programming*, Uppsala, Sweden, 2017.

[23] A. Eberhart, "Automatic generation of java/sql based inference engines from rdf schema and ruleml," in *International Semantic Web Conference*, Sardinia,Italy, 2002.

[24] M. Völkel and Y. Sure, "RDFReactor-from ontologies to programmatic data access," in *Poster Proceedings of the Fourth International Semantic Web Conference*, Galway,Irland, 2005.

[25] C. Atkinson, M. Gutheil and K. Kiko, "On the Relationship of Ontologies and Models.," *WoMM*, vol. 96, pp. 47-60, 2006.

[26] E. Oren, R. Delbru, S. Gerke, A. Haller and S. Decker, "ActiveRDF: object-oriented semantic web programming," in

Proceedings of the 16th international conference on World Wide Web, 2007.

[27] I. N. Athanasiadis, F. Villa and A.-E. Rizzoli, "Ontologies, JavaBeans and Relational Databases for enabling semantic programming," in *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, 2007.

[28] A. Poggi, "OWLET: an object-oriented environment for OWL ontology," in *Proceedings of the 11th WSEAS International Conference on Computers*, Agios Nikolaos, Crete Island, Greece, 2007.

[29] C. Puleston, B. Parsia, J. Cunningham and A. Rector, "Integrating object-oriented and ontological representations: a case study in Java and OWL," in *International Semantic Web Conference*, 2008.

[30] M. Leinberger, S. Scheglmann, R. Lämmel, S. Staab, M. Thimm and E. Viegas, "Semantic web application development with LITEQ," in *International Semantic Web Conference*, Riva del Garda, Italy, 2014.

[31] C. Atkinson and T. Kuhne, "Model-driven development: a metamodeling foundation," *IEEE software*, vol. 20, pp. 36-41, 2003.