International Journal of Software Engineering
and Knowledge Engineering
Vol. 28, Nos. 11&12 (2018) 1775–1794
© World Scientific Publishing Company
DOI: 10.1142/S0218194018400284

World Scientific
www.worldscientific.com

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

# Object-Oriented Modeling with Ontologies Around: A Survey of Existing Approaches

Selena Baset* and Kilian Stoffel†

*Information Management Institute, University of Neuchâtel*
*A.L.Breguet 2*
*CH-2000 Neuchâtel, Switzerland*
**selena.baset@unine.ch; selena.baset@gmail.com*
*†kilian.stoffel@unine.ch*

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 their technical characteristics and their resulting artifacts. We discuss further potential reasons beyond what have been addressed in the literature before finally providing our own reflection and outlook.

*Keywords*: Knowledge representation; object-oriented programming; software modeling; ontologies; OWL; language transformation.

## 1. 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

*Corresponding author.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

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 Ontology-Driven Software Development (ODSD) [6] or Ontology-Oriented Programming (OOP) [7]. However, shifting a bit from the state-of-the-art research 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 development communities. We start with a brief background section on ontological modeling languages and standards. We then survey existing integration approaches and we deduce some of their common characteristics with the goal of providing a classification framework that can be used by researchers interested in the topic. We further proceed to discuss some of the common challenges and implicit reasons beyond what have been addressed in the literature before finally concluding with our own reflection on the current state of OWL-to-OOP integration and our involvement in the topic.

## 2. Background

In order to lay the ground for the technical notions used throughout the survey, we provide a brief background on ontologies. For a rigorous analysis on ontologies and models and the relationship between them, we refer the reader to the work done by Atkinson *et al.* [8]. Their work tackles the confusion caused by the variety of different interpretations of ontologies and models in enterprise software engineering and proposes a framework to differentiate between them. Here, we rather briefly cover the basics of ontological modeling focusing mainly on the languages and standards adopted for representing and sharing ontologies.

Ontologies, in plural form, is an engineering term derived from the ancient philosophical Greek term "ontology". Ontologies in the context of knowledge representation are formal abstract models used by computer systems to describe and share knowledge about the real world. This is achieved by explicitly defining the concepts relevant to the domain, and the relationships between these concepts, using a formal computer language to avoid ambiguity or incomplete specifications. These key aspects of ontologies are put together in a concise definition by Gruber in 1993: "An ontology is a formal and explicit specification of a shared conceptualization" [1].

What differentiates ontological modeling from other modeling paradigms such as UML or Entity–Relationship diagrams is that ontologies are intended *a priori* to be shared and they need therefore to have formal precise semantics. This calls for a standard ontological language to be used when building and sharing ontologies. In response to this call, many ontology languages were proposed during the last two

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

decades. Most notably we have the frame-based FLogic by Kifer *et al.* [9] and DAML+OIL by Horrocks and McGuinness *et al.* [3, 10].

Nowadays, ontology modeling is largely dominated by the Web Ontology Language OWL, the W3C standard language for the semantic web. OWL has two versions OWL and OWL 2 and both versions have many sublanguages that are varying in expressiveness at an increasing complexity overhead [5]. The most restricted sublanguage is OWL Lite and the most expressive one is OWL Full which has a very expressive vocabulary but is not anymore decidable. In between OWL Lite and OWL Full, we find OWL DL, a language based on Description Logics (DL) [11] that offers the right balance between expressiveness and decidability for most Knowledge Representation (KR) applications [12]. Ontologies defined in OWL consist of classes, properties and individuals (instances of classes) all of which are designated by axioms of class, data ranges, data types and object properties expressions [5]. An OWL ontology $\mathcal{O}$ is a finite set of axioms that describe intentional knowledge, i.e. *TBox* and *RBox* axioms in DL terms, or existential knowledge, i.e. *ABox* assertions [11].

Another important standard for building and sharing ontologies is the Resource Description Framework (RDF) [13], the specification adopted by W3C for describing and interchanging metadata on the Web [14]. Throughout its history, RDF has undergone many transformations that resulted in RDF possessing a variety of syntax notations and data serialization formats. The role of RDF, however, remained always the same. One can relate the role of RDF in describing a resource information to the role of HTML in visually rendering the content of a web page. The main difference is that HTML renders content intended for human receptors whereas RDF is concerned with annotating web content making it accessible for machines. Basically, all RDF notations rely on triplets. A triplet takes the form of a (subject, predicate, object) tuple where the subject — a Uniform Resource Indicator (URI) — denotes the resource and the predicate denotes a relationship between the subject and the object. The object, in turn, can be denoted either via a URI or some literal of a primitive data type. An RDF document is then simply a collection of triplets encoding statements and/or assertions about the resource being described. These collections can be easily fitted into a labeled, directed multigraph making RDF a suitable data model for certain kinds of knowledge representation tasks. Ontological extensions to the RDF model, such as RDF Schema (RDFS) [15], are also available. RDFS provides an expressive vocabulary (e.g. `rdfs:domain`, `rdfs:range` or `rdfs:subclassof`) to structure and enrich the semantics of RDF documents.

In parallel to the fast-paced development of languages and standards in knowledge engineering and semantic web communities, the neighboring software engineering community was also putting similar efforts into crystallizing a standard language for ontology modeling. Already in 2001, Cranefield [16] proposed using UML for representing ontologies, a proposition backed by the wide acceptance of

UML among software engineers. In 2003, the Object Modeling Group (OMG) released its first version of a dedicated Ontology Definition Metamodel (ODM) specification.[a] The ODM specification defines the mapping between the metamodels of OWL and RDFS and the Common Logics Metamodel [17]. ODM is now in its 1.1 version which was released in September 2014. A use case[b] from IBM provided a proof-of-concept implementation of the ODM specification. This use case is probably the most notable example of UML ontologies. Apart from the IBM use case, comparing the sparse results one gets digging the literature for UML ontologies to the number of readily available OWL ontologies, one can safely assume a limited usage of UML profiles for constructing ontologies compared to other options such as OWL or RDF(S).

## 3. Method and Scope

Given the qualitative nature of the literature and the diversity of the technical spaces in which ontology integration is implemented, it is difficult to establish a unified criterion for automatic classification of existing mapping approaches. In order to define a concise and coherent scope, we will limit our review to the question of integrating ontologies into an object-oriented paradigm and not vice versa, i.e. excluding papers that described approaches to generate ontologies from code and excluding papers on integrating ontologies into a programming language that is not object oriented such as LOOM or Prolog.

The question we are concerned with in this survey was first addressed in three of the earliest papers in the domain: Eberhart 2002 [18], Kalyanpur *et al.* 2004 [19] and Knublauch 2004 [6]. While the mentioned papers are relatively old, adopting these three papers as seeds for our search allows for a more inclusive time window when collecting other related papers. Using Google Scholar, we harvested all papers that cited at least one of the seeds. This resulted in around 311 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 dropped irrelevant papers such as papers accentuating the application domain rather than the integration approach. We then used Google citations metrics (i.e. the number of citations per paper) as an indicative measure of the impact of a paper to pivot our manual inspection of more papers to include. Using the high-impact papers as seeds, we collected — manually this second round — some more relevant papers that were not automatically harvested. At the end of the process we retained 25 approach papers that we are surveying in this paper besides three surveys and position papers. While by no means an exclusive list of all papers in the field, the retained papers give a good indicator on the current state of research.

[a] OMG ODM: https://www.omg.org/spec/ODM/.
[b] IBM Semantics Toolkit: http://freshmeat.sourceforge.net/projects/ibmsemanticstoolkit.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

When reviewing papers, we focused on certain aspects like the extent of integration and the challenges the authors faced rather than focusing on the motivation behind each contribution which was, more or less, common behind most of the reviewed papers.

## 4. A Classification of Ontology Integration Approaches

Ontologies, by design, are not intended as standalone software units [20]. 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 identify some kind of a classification of the possible embeddings of an ontology in a programming environment. In [7], Goldman opposes generic application development using ontologies to ontology-specific programming. Generic programming requires no knowledge of the semantics of specific ontologies (like in query answering engines and consistency checkers) whereas "ontology-specific programming deals with models based on one or more specific ontologies". Puleston *et al.* [21] also differentiate between distinct approaches: A direct approach where ontologies are transformed to code and an indirect one where access to ontologies is provided via APIs and yet another hybrid approach that is proposed by the authors themselves. Many other authors proposed their vision on integrating ontologies into conventional software development with different acronyms and terms to denote this integration; as examples we have the afore mentioned ODSD by Knublauch [6] and OOP by Clark and McCabe [22] and many others. In general, we can differentiate between two main camps: Domain-aware versus domain-neutral application development. The more we move towards domain-aware application development the more it becomes important to abandon generic access and to provide the developer with a transparent means to access and manipulate the content of the ontology.

In this section we propose a classification of the surveyed approaches for ontology integration. We build on top of what is proposed in the literature in order to provide a comprehensive taxonomy of the existing tools and approaches based on their technical characteristics and their resulting artifacts. 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. Table 1 summarizes the surveyed tools/approaches. The proposed taxonomy, as shown in Fig. 1, is constructed using this list of tools. The inner levels of the tree are based on the "Mode" column whereas leaf nodes represent the surveyed tools.

In the following subsections we traverse the taxonomy explaining the logic behind it and providing an overview of leave nodes in each branch.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

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 [18, 23] | RDF(S) | Java | Active/Static | None |
| 2003 | Goldman [7] | OWL | C# | Active/Static | None |
| 2003 | OWL API [24] | OWL | Java | Passive | External |
| 2004 | Knublauch [6] | OWL | Java | — | — |
| 2004 | HarmonIA [19] | OWL | Java | Active/Static | None |
| 2004 | Jena [25] | OWL | Java | Passive | External |
| 2005 | SWCLOS [26] | OWL | CommonLisp | Active/Dynamic | Limited |
| 2005 | RDFReactor [27] | RDF/RDF(S) | Java | Active/Static | None |
| 2006 | Atkinson *et al.* [8] | OWL | UML | — | — |
| 2006 | Babik and Hluchy [28] | OWL | Python | Active/Dynamic | Limited |
| 2006 | Clark and McCabe [22] | — | Go! | — | Supported |
| 2007 | ActiveRDF [29] | RDF(S) | Ruby | Active/Static | None |
| 2007 | Athanasiadis *et al.* [30] | RDF/OWL | JavaBeans | Active/Static | — |
| 2007 | Owlet [31] | OWL | Java | Passive | Supported |
| 2008 | Puleston *et al.* [21] | OWL | OWL/Java | — | — |
| 2009 | OWL2Java [32] | OWL | Java | Active/Static | None |
| 2009 | O3L [33] | OWL | Java | Passive | Supported |
| 2009 | SWOBE [34] | OWL | Java | Active/Dynamic | Limited |
| 2011 | Sapphire [35] | OWL | Java | Active/Dynamic | Limited |
| 2011 | Paar and Vrandeǒić [36] | OWL | Zhi# | Active/Dynamic | Supported |
| 2011 | KOMMA [37] | RDF | Java | Passive | — |
| 2014 | LITEQ [38] | RDF(S) | F# | Active/Dynamic | Limited |
| 2016 | OntoJIT [39] | OWL | C# | Active/Dynamic | Limited |
| 2017 | Owlready [40] | OWL | Python | Active/Dynamic | Indirect |
| 2017 | Leinberger *et al.* [41] | OWL | $\lambda$ DL | Active/Dynamic | Supported |

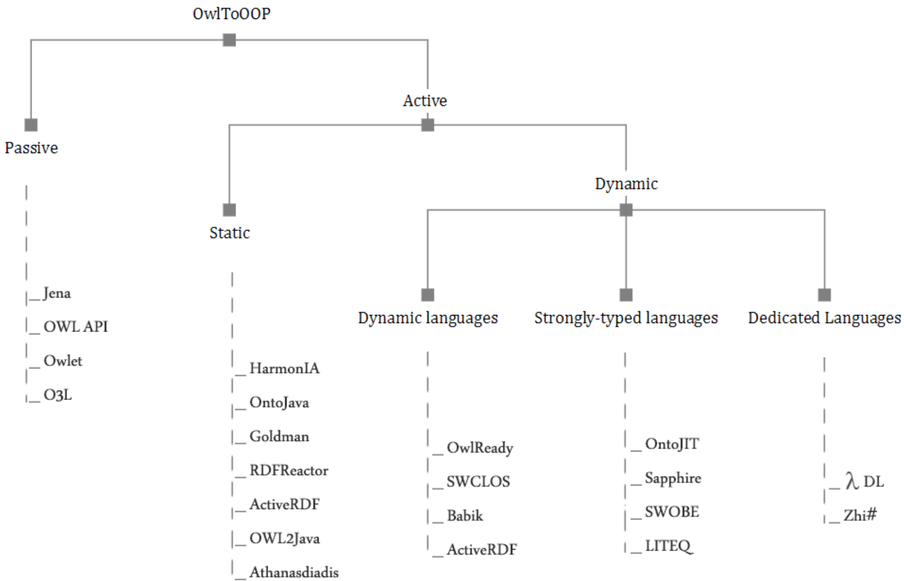


Fig. 1. A taxonomy of existing OWL-to-OOP mapping.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

### 4.1. *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 [42, 24] or an RDF triple-based structure like the one used in Jena [25]. 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 of providing a generic access to the content of ontologies is 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. On the other hand, it forces the developer to write verbose lengthy code even for simple programming tasks. Moreover, it does not provide any kind of static typing or error checking. Under this approach we classify the well-established Jena and OWL API as well as the more recent Owlet system. The following paragraphs briefly describe each of them.

*Jena* [25] is a Java-based and RDF-centric system, it can read RDF serializations of an ontology to produce an in-memory RDF graph representation. It is then possible to use Jena's rules-based reasoner to perform entailment reasoning on the RDF graphs. If we put the problems of verbose code and lack of compile-time error checking aside, Jena offers the most complete, and flexible, system for semantic application development. Besides parsing ontology languages, it provides support for querying, database persistence as well as some visualization tools.

*OWL API* [24] is OWL-centric and, compared to Jena, is less RDF-friendly. It also reads the serialized form of an ontology to produce an in-memory representation but instead of RDF graphs, it produces ASTs as the output of the ontology loading routine. Protégé API can be seen as an extension of OWL API. The OWL API code is open source and is available under either the LGPL or Apache Licenses.

*Owlet* [31] is an object-oriented software system for building and managing OWL ontologies in a heterogeneous distributed environment. The author describes Owlet as having a capability-aware load balancing feature where reasoning tasks are assigned to different nodes based on their computation power but no details on how this feature is realized are provided. From the same author we also have the O3L [33] framework that is derived from Owlet. In both, the ontology is mapped to a set of instances of generic and predefined Java classes.

### 4.2. *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 belongs 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. Besides providing the developer with a more transparent interface into domain objects, active ontology mapping has the advantage of compile-time error checking. The developer can rely on the compiler for detecting type errors that would have otherwise gone undetected until runtime using the generic access provided by the passive approach. Active OWL-to-OOP mapping can itself be further classified into static or dynamic.

#### 4.2.1. *Static mapping*

In static mapping, the translation, i.e. code generation from OWL concepts, properties and individual axioms, 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. Many of the surveyed tools in this paper are classified under this approach. We dedicate the following paragraphs for a brief summary of each.

*HarmonIA* by Kalyanpur *et al.* [19] is one of the oldest but most important attempts at bridging the gap between ontologies and object-oriented languages. While most other approaches merely aimed at providing programmable access to ontologies, the work of Kalyanpur *et al.* [19] attempts at translating OWL DL ontologies into Java classes all while maintaining their semantic profile (except for some cases like OWL inverse functional properties and "`rdfs:sameAs`" axioms). This is achieved by (1) utilizing interfaces with shadow classes for multiple inheritance, (2) custom-tailored design patterns for expressing OWL logical axioms and (3) special listeners on property accessors to enforce property restrictions. As for reasoning support, ABox reasoning was left for future work. Still, the mechanism in which the property listeners check for property changes and throw relevant exceptions can be seen as a limited form of reasoning. While this approach was shown to maintain the semantics of the source ontology, we believe that it suffers from the side effect of being overly restrictive in some cases. HarmonIA does not seem to be maintained at present day but many subsequent research and software tools are based directly on its ideas. Among these are the Sapphire tool surveyed in this paper as well as Jastor[c]: An open-source tool for providing type-safe, ontology-driven RDF access from Java.

*ActiveRDF* [29] uses stateless transparent proxy objects to represent and manipulate RDF resources in Ruby programming environment. The choice of Ruby is motivated by the dynamic features it offers. The authors argue that in order to resolve the impedance mismatch between RDF and OOP paradigms, one needs to

[c]http://jastor.sourceforge.net/.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

remove some restrictions in the object-oriented language, moreover, the mismatch between RDF and the object-oriented paradigm is smaller in scripting object-oriented language than with the standard compiled languages. ActiveRDF makes use of dynamic typing, runtime introspection and Ruby metaprogramming capabilities but the mapping is still done statically prior to runtime and thus the classification of ActiveRDF under this category.

*RDFReactor* aims at delegating the tasks of insuring type safety and providing auto-completion to the Java compiler and the IDE programming environment. Its first working prototype was presented in a paper by Völkel and Sure [27]. The prototype depicts a two-step mapping approach: Ontology is first mapped into an intermediate representation, a JModel, that provides a one-to-one mapping to the source ontlology. The JModel is still more expressive than ordinary Java constructs and may pose problems when generating code. Some algorithms are thus applied to make it more strict for "proper" Java code generation. Obviously, the applied transformations come at a price of lost semantics. For example, multiple inheritance is avoided by simply considering the superclass that has itself the most superclasses as the only parent. According to the authors, this strategy entails no loss in functionality when it comes to conventional software development. Finally, queries to the original RDF source are answered by using handwritten methods to translate method calls into triple-centric methods.

*OWL2Java* by Zimmerman [32] is a tool that aims at hiding the peculiarities of working with an ontology in a strongly-typed environment by automatically generating Java wrapper classes for OWL classes previously imported by Jena. The generated code uses Jena underneath; enabling the developer to exploit other Jena features such as SPARQL[d] queries. To help overcome incomplete specifications in the source ontology such as missing domain and range axioms, OWL2Java provides the developer with some "structure-finding tools". These tools employ some kind of entailment to deduce missing information but they are still not regarded as reasoning tools in the formal sense of OWL reasoning as neither completeness nor soundness of the induced information is guaranteed.

*OntoJava* [18, 23] maps ontologies written in basic RDF, RDFS or DAML+OIL into Java code. Even though RDFS is less expressive than OWL, it is still more expressive than Java and handling issues like multiple inheritance in the RDFS source is considered problematic. On the other hand, unlike OWL, the RDFS specification resembles object-oriented languages in allowing a property to have multiple domains but only one single range. This can greatly facilitate the task of translating RDFS properties into Java in comparison to translating OWL multiple-range properties. It can also detect RDFS multiple-range modeling errors as the Java compiler automatically enforces this constraint. Along with OntoJava is its sister

[d]SPARQL: W3C recommended language for querying RDF data sources. SPARQL is a recursive term that stands for SPARQL Protocol And Query Language.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

tool OntoSQL that translates rules in RuleML[e] into SQL in order to make them available for top–down evaluation by an SQL engine.

*Goldman's* approach [7] is concerned with automatically generating software class libraries for ontology-specific application development in the .Net programming environment. It relies on the .Net abstract code model for generating assemblies. The proposed automatic mapping exploits .Net generic types, uses interfaces for multiple inheritance and proxy wrappers for multiple-class object instantiation. Like it is the case with the other surveyed tools that are .Net-based, integrating ontologies into .Net enjoys a certain degree of portability thanks to the support provided by the .Net Common Language Runtime (CLR) environment that allows for libraries written in one language to be loaded and used by other .Net languages.

#### 4.2.2. *Dynamic mapping*

In this category of active OWL-to-OOP mapping, the translation from the ontological source into code statements in the target programming language is done dynamically at runtime. Besides the challenge of lack of expressiveness of programming languages that is present in active mapping approaches, this approach is even more challenging to implement as it requires a certain degree of flexibility for performing inference tasks on the ontology in its executable form, e.g. 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. Under this branch we have: SWCLOS [26], Owlready [40] and the approach proposed by Babik and Hluchy [28].
(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 [35] and the C# OntoJIT [39].
(3) And finally, some of the surveyed approaches have gone to the extent of proposing a dedicated programming language; like for example the Go! language [22] or the more recent $\lambda$ DL language [41].

*Owlready* [40] is intended as a solution to answer specific practical needs of software projects in the biomedical domain; mainly the need to treat the ontology classes as objects and to support local closed-world reasoning [43]. To address the first need, Owlready relies on Python metaclasses to map ontological entities into Python objects. OWL classes are then treated as instances of Python metaclasses. Owlready adapts ontologies to the Python object model using "special methods" that are defined at metaclass level in case of mapping OWL classes or at class level if they are to be applied on instances. One example of these methods is the

[e]RuleML: A unifying system of families of languages for Web rules. More under: http://wiki.ruleml.org.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

"Close_world()" method that can be called at different levels of the type tree to provide the necessary missing axioms to answer queries under a locally Closed-World Assumption (CWA). Automatic classification of classes and individuals can be performed by connecting to the HermiT reasoner [44] as an external reasoning component. Although very recent, Owlready is a promising proposition given the growing community of Python developers working on building software for the biomedical and bioinformatic domains.

*OntoJIT* [39] offers two possibilities for expressing ontologies as executables. First one is by directly writing C# code and the second is by translating serializations of existing ontologies. This second translation option is dynamically realized at runtime using the .Net abstract model for code together with the Just-In-Time compile units from the .Net CLR environment. To bridge the semantic expressiveness gap between OWL and C#, OntoJIT uses a layer of metaproperties. These metaproperties are defined at the top class level and inherited, or masked, by all subsequent concepts. Metaproperties corresponding to terminological axioms are static (i.e. shared between all instances) whereas assertional metaproperties are nonstatic. OntoJIT supports blank RDF nodes that are usually present in OWL and are necessary for entailment reasoning. Entailment and query answering are then left to the C# developer who can make use of the C# Language Integrated Query (LINQ[f]) that he/she is already familiar with. The use of C# as the only language for both representing and querying ontologies provides a higher level of transparency while the layer of metaproperties is there to maintain the semantics of the source ontology and to be used for advanced scenarios of semantic query answering.

*LITEQ* [38] stands for Language Integrated Types, Extensions and Queries. It represents a paradigm for integrating RDF triples into the context of F# programming environment. LITEQ provides a mechanism for querying the RDF data source then mapping and strongly typing the imported triples as variables of code types created in F#. Querying RDF sources is achieved using a special variable-free language that is called Node Path Query Language (NPQL). NPQL is syntactic sugar for querying RDF graphs for developers who are less familiar with SPARQL. NPQL allows for RDF graph traversal via three main operators: subtype navigation, property navigation and property restriction operators. NPQL makes no distinction between schema and data, both can be explored using query expression composed of these three operators. LITEQ is supported by Microsoft and is supposed to be packed under the FSharp.Data .Net library. It is also open source and available on GitHub[g] but the project seems to be frozen for a few years now.

$\lambda$ *DL* [41] is a hybrid language resulting from integrating the *ALOCI* fragment of Description Logics into $\lambda$-calculus. It can be considered as a dedicated type system for semantic data that treats concept expressions as types. Reasoning is supported by providing a query mechanism based on description logics. In the design principles of

[f] LINQ: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/.
[g] https://github.com/Institute-Web-Science-and-Technologies/Liteq.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

$\lambda$ DL, accent is made on type-safe usage of queries meaning that query results must be properly typed. The authors provide a formal proof of soundness in the paper all while acknowledging certain limitations at the current stage of their work. The $\lambda$ DL language is an elegant way of insuring type checking over semantic data. However, it presumes that the developer is already familiar with at least $\lambda$-calculus or Description Logics and from the perspective of conventional software development, this presumption puts into question the usability of the proposed language.

*Sapphire* [35] builds on the works of Kalyanpur *et al.* (HarmonIA) [19] and Zimmerman (OWL2Java) [32] as established techniques for mapping OWL classes to Java interfaces but it differs from them in relying on dynamic proxies to provide a runtime approximate mapping for OWL type system. Each time an OWL type is added or removed, a corresponding wrapping or unwrapping operation is performed on the proxy object. Sapphire provides an option to support open-world logic within Java by replacing Boolean values by a three-valued enumeration. A limited form of reasoning is realized using methods that interpret OWL axioms such as `owl:sameAs` or `owl:DifferentFrom`. These methods belong to the top interface (`Thing`) and are propagated into all extending interfaces. OWL entailment rules are not translated into production code but stored in an underlying quad store where a reasoner can be applied. The argument the authors provide for utilizing such a hybrid approach even if it imposes a certain performance overhead is that it keeps production code isolated from classifications that do not cleanly map into Java.

*SWCLOS* [26] is a semantic web processor built on top of Common Lisp Object System (CLOS). CLOS shares the same class semantics with most OOP languages but the mechanism of its class–subclass and class–instance relations is unique in the sense that they are based on transitivity and subsumption relations just like in RDFS. This resemblance between CLOS and RDFS semantics explains the authors' motivation for using CLOS. SWCLOS maps RDF(S) and OWL classes into metaobjects in CLOS. It uses dynamic programming, by means of metaclasses, to modify the behavior of metaobjects at runtime. Reasoning in SWCLOS is supported by implementing an extended version of the structural subsumption algorithm [11]. Like its original version, the extended algorithm is incomplete but is shown to be useful for most practical cases of OWL reasoning.

Babik and Hluchy [28] proposed an integration approach that aims at making the most out of the dynamic features of Python; mainly python metaclasses, i.e. classes of classes, to reflect the set-theoretic semantics of OWL while preserving the classic object notations in object-oriented languages. The instance of a metaclass is itself a class and it represents the mapping of the OWL class to the intentional set. Babik and Hluchy propose an interface to a Java-based OWL reasoner. Reasoning under open-world semantics is not supported in this approach due to the binary nature of Python Boolean values. The use of a Java-based reasoner in Python and the need to convert between virtual machines impose a certain performance penalty.

Athanasiadis *et al.* [30] propose mapping ontologies into a three-layered semantic-rich development architecture that uses conventional software development tools

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

such as JavaBeans and Hibernate. In this architecture, OWL models are used to express rich semantics and to connect to an external reasoner for logical inferences, JavaBeans are used for end-user application development while Hibernate object-relational mapping is used for content persistence. To overcome the expressiveness gap between OWL and Java, the proposed mapping only considers nonanonymous OWL classes and ignores other OWL classes such as restrictions and union classes as they do not add more attributive specifications to the relational model.

Clark and McCabe [22] have introduced ontology-oriented programming in Go! language. Their work does not constitute an integration approach but rather a new custom-made language to represent ontological knowledge in a more general purpose programming environment. The motivation the authors present for creating a whole new language is "cleanly integrating logic, functional, object-oriented and imperative programming styles", all in a multi-threaded agents environment. Knowledge in Go! is represented as "sets of labeled theories incrementally constructed using multiple inheritance". The expressiveness of Go! is compared to that of OWL Lite but in general other OWL profiles are all more declarative than Go! language. Go! was first introduced in 2006 and had good potentials as a general-purpose programming language that uses declarative style for functions and relations and imperative style for procedural actions. Reviewing the literature, however, we could not assert that the language has finally gained momentum.

*SWOBE* [34] proposes embedding semantic web languages: RDF/XMAL as data language, SPARQL as query language and SPARUL[h] as update language into Java to provide compile-time type checking and detect syntactic and semantic errors earlier. This approach results in a heterogeneous Java code snippets that the authors call SWOBE programs. The developer would write a SWOBE program containing a query in SPARQL and the SWOBE precompiler then checks — via a separate static program analysis step — and reports any syntactic or semantic errors detected at compile time. The precompiler finally transforms the SWOBE program along with the embedded languages into standard Java classes: main, iterator and helper classes.

*Zhi#* [36] is a programming language that integrates OWL and XML Schema Definition (XSD) into C#. Zhi# programs are interoperable with .Net assemblies because they are compiled into C#. In fact, Zhi# is itself a superset of C# version 1.0. The syntactical extensions take the form of external types that can be imported using nonrecursive import directives. This approach does not rely on code generation but rather on semantic-aware access to the underlying ontology management system. In order to support OWL subsumption, Zhi# extends the explicit subtyping system of OOP languages to permit value space-based subtyping where a type is a subtype of another type if its value space (set of given values) is a subset of the value space of the other type. The main difference Zhi# brings compared to other active dynamic approaches with a dedicated language is its "pay-as-you-go" compile strategy. When

[h] SPARUL: SPARQL Update Language. https://www.w3.org/wiki/SparqlUpdateLanguage.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

it comes to external OWL types, the programmer has access to all types in the source ontology but only the types eventually consumed by the developer will be mapped into C# code during compilation. On the other hand, runtime dynamic checking is required each time an instance of an external type is used and this could, at least marginally, penalize runtime performance. Finally, the issue of providing the developer with a transparent access to the ontology is only partially solved as the .Net developer still has to go through a certain syntactic twist to access the underlying source via Zhi# code.

## 5. Semantic Gap

The most prominent challenge that is present in OWL-to-OOP mapping, particularly in active mapping approaches, is the semantic gap between the ontological and object-oriented paradigms. This gap is most prominently present when considering the different assumptions on which modeling in the two paradigms is based or in the cases where formal programming languages lack the expressive means to represent certain ontological constructs.

### 5.1. *Expressiveness and different interpretations*

The semantic richness of ontological languages makes it very difficult to find an OOP counterpart for each OWL semantic construct. One of the most obvious examples is perhaps the different interpretation of class inheritance. OWL, or Description Logics, 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 a class cannot be a subclass of two different disjoint parent classes and consequently multiple inheritance is, generally speaking, not supported. Multiple inheritance is not the only example of the missing semantic equivalence. A similar argument goes for other OWL axioms such as `owl:equivalentClass`, `owl:sameAs` or `owl:disjointWith`. In native Java or C# semantics, there is no equivalent options to express these axioms.

Many of the approaches we surveyed did not attempt to bridge 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 metalayer of code to substitute for the missing semantics in programming languages [39], to more sophisticated approaches of stretching the expressiveness of modeling in Java to that of OWL DL by enforcing some constraints and design patterns like in HarmonIA [19] or even extending the subtyping system of the host programming language to support Description Logics subsumption as in Zhi#.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

### 5.2. *The open-world assumption*

Most existing ontologies in the semantic web are OWL DL ontologies. That is, they are based on the Open-World Assumption (OWA) of Description Logics [11]. The OWA argues that since it is not possible for an agent to have complete knowledge, then it is not possible, via deductive reasoning, for this agent to infer the truth value of a fact that is not, explicitly or implicitly, present in its base; irrespective of whether or not it is known to be true.

The OWA concerns reasoning rather than modeling yet it has great impact on how models are written in the first place. OWA makes perfect sense in the context of Description Logics and other logic programming languages but it is still rather counterintuitive and a major source of confusion for most conventional software developers; especially when contrasted to CWA present in other common modeling paradigms where the absence of a statement in the database allows the agent to infer its falsity.

Earlier in the survey, we have seen that Owlready proposed closing the world locally by providing missing axioms at different levels of the type tree, or globally if provided starting from the root node. There exist many other calls in the literature for supporting closed-world ontological reasoning but covering them more in details falls outside the scope of this paper. We forward interested readers into the works done in [45–47].

### 5.3. *Unique name assumption*

Another basic assumption that can make a fundamental difference between modeling in object-oriented and ontological languages is the Unique Name Assumption (UNA). This assumption is a simplifying one that requires different names to always refer to different entities. It is the default assumption when it comes to programming languages: Different class names yield different classes and different variable names yield different variables. In ontological languages, however, this assumption is not made. Two OWL classes or individuals with different unrelated names may very well refer to the same entity. This difference explains the presence (or absence in OOP languages) of explicit OWL axioms to express the uniqueness of an entity: `owl:equivalentClass`, `owl:sameAs` or `owl:disjointWith`.

## 6. Discussion

One of the main motivations behind most of the works surveyed in this paper was the difficulty of manipulating ontologies in mainstream software development and the scarcity of options for an ontology programming interface that provides static typing and error checking. Nevertheless, as we can see from earlier sections, a retrospective scan on works done in this area revealed a different story. Besides the tools and

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

approaches covered, there exist several other projects around the same topic: OntologyBeanGenerator,[i] Alibaba,[j] KOMMA [37], API a gogo [48] and others. Some of these tools are already listed in Table 1, but we abstain from covering them in more details as they largely intersect with what has been explained in other sections.

In other words, there exist already many options both for accessing and integrating ontologies in an OOP paradigm, yet we still did not witness ontologies spanning new development territories and the question about the usability of mapping ontologies into the landscape of conventional software development remains open despite all the proposed integration approaches. Below we try to look beyond what has been proposed in the literature and we list some of the potential reasons for the developers' shy adoption of ontologies:

**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 languageand Stanford Protégé is the editor, when it comes to integrating ontologies as conventional 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 cumbersome and not even affordable in most of today's agile software projects.

**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 is most of the time based on an OWA in contrast to implicit closed-world modeling in UML and OOP languages. The same kind of argument also applies for the approaches that went as far as proposing dedicated languages for ontology-oriented programming. While we are sure these languages can easily target an audience with specific fine-tuned profiles, we are less sure whether such propositions would answer the question of democratizing semantic application development or if it would simply lead the developer into a "yet another language" syndrome.

**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.

**Resistance to change.** Finally, for most right-wing software engineers, adopting ontological approaches, or generally speaking 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

[i] OntologyBeanGenerator Protégé plug-in: https://protegewiki.stanford.edu/wiki/OntologyBeanGenerator.
[j] Alibaba: https://bitbucket.org/openrdf/alibaba.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

engineering with strong preferences for tidy and well-engineered domain models. It provokes a lot of philosophical discussions similar to the dynamic versus 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.

## 7. Conclusion and Outlook

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 the software industry. In fact, when trying to further trace the surveyed tools, we were unfortunate to find out that the vast majority have went idle for years. It would be interesting therefore to see how the more recent propositions will evolve given the reawakened interest of semantic application development in the last few years.

Ourselves, we are involved in working on a project for integrating ontologies into conventional C# code (the OntoJIT tool covered in this survey). Having collected many lessons learned surveying the literature, our main objective of developing OntoJIT is to minimize, as much as possible, the syntactic and semantic paradigm twist that the developer has to go through. We try to do so by providing a transparent access to ontologies expressed and queryable in C# all while maintaining the semantics in an easily accessible metalayers for developers working on programming tasks involving more advanced semantic querying. Our prototype is still a work-in-progress but we hope it will contribute to the collective scientific effort in democratizing semantic-aware software development.

## References

1. T. R. Gruber, A translation approach to portable ontology specifications, *Knowl. Acquis.* **5**(2) (1993) 199–220.
2. A. Farquhar, R. Fikes and J. Rice, The Ontolingua server: A tool for collaborative ontology construction, *Int. J. Hum.-Comput. Stud.* **46**(6) (1997) 707–727.
3. I. Horrocks, DAML+OIL: A description logic for the semantic web, *IEEE Data Eng. Bull.* **25**(1) (2002) 4–9.
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 Semant., Sci. Serv. Agents World Wide Web* **1**(1) (2003) 7–26.
5. B. Motik, P. F. Patel-Schneider and B. C. Grau, OWL 2 Web Ontology Language Direct Semantics (2009), W3C Recommendation, https://www.w3.org/TR/2009/CR-owl2-direct-semantics-20090611/all.pdf.

6. H. Knublauch, Ontology-driven software development in the context of the semantic web: An example scenario with Protégé/OWL, in *Proc. 1st Int. Workshop Model-Driven Semantic Web*, 2004, pp. 381–401, http://www.knublauch.com/publications/MDSW2004.
7. N. M. Goldman, Ontology-oriented programming: Static typing for the inconsistent programmer, in *Proc. Int. Semantic Web Conf.*, 2003, pp. 850–865.
8. C. Atkinson, M. Gutheil and K. Kiko, On the relationship of ontologies and models, in *Proc. Workshop Meta-Modeling and Corresponding Tools*, 2006, pp. 47–60.
9. M. Kifer, G. Lausen and J. Wu, Logical foundations of object-oriented and frame-based languages, *J. ACM* **42**(4) (1995) 741–843.
10. D. L. McGuinness, R. Fikes, J. Hendler and L. A. Stein, DAML+ OIL: An ontology language for the semantic web, *IEEE Intell. Syst.* **17**(5) (2002) 72–80.
11. F. Baader, *The Description Logic Handbook: Theory, Implementation And Applications* (Cambridge University Press, 2003).
12. I. Horrocks, OWL: A description logic based ontology language, in *CP 2005: Principles and Practice of Constraint Programming*, LNCS Vol. 3709 (Springer, Berlin, 2005), pp. 5–8.
13. O. Lassila and R. R. Swick, Resource Description Framework (RDF) Model and Syntax Specification (1999), W3C Recommendation, https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.
14. S. Powers, *Practical RDF: Solving Problems with the Resource Description Framework* (O'Reilly Media, 2003).
15. B. McBride, The resource description framework (RDF) and its vocabulary description language RDFS, in *Handbook on Ontologies* (Springer, 2004), pp. 51–65.
16. S. Cranefield, UML and the semantic web (2001), https://pdfs.semanticscholar.org/cf65/bfc0e9113de36c27a16fb59c4fc74ba32ad7.pdf.
17. D. Gašević, N. Kaviani and M. Milanović, Ontologies and software engineering, in *Handbook on Ontologies* (Springer, 2009), pp. 593–615.
18. A. Eberhart, OntoJava: Applying mainstream technology to the semantic web, in *Proc. Int. Conf. Electronic Commerce and Workshop Semantic Web-based E-Commerce and Rules Markup Languages*, 2001.
19. A. Kalyanpur, D. J. Pastor, S. Battle and J. A. Padget, Automatic mapping of OWL ontologies into Java, in *Proc. 16th Int. Conf. Software Engineering and Knowledge Engineering*, Vol. 4, 2004, pp. 98–103.
20. H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann and S. Hübner, Ontology-based integration of information: A survey of existing approaches, in *Proc. IJCAI-01 Workshop: Ontologies and Information Sharing*, Vol. 2001, 2001, pp. 108–117.
21. C. Puleston, B. Parsia, J. Cunningham and A. Rector, Integrating object-oriented and ontological representations: A case study in Java and OWL, in *Proc. Int. Semantic Web Conf.*, 2008, pp. 130–145.
22. K. L. Clark and F. G. McCabe, Ontology oriented programming in Go! *Appl. Intell.* **24**(3) (2006) 189–204.
23. A. Eberhart, Automatic generation of Java/SQL based inference engines from RDF schema and RuleML, in *Proc. Int. Semantic Web Conf.* (Springer, 2002), pp. 102–116.
24. S. K. Bechhofer and J. J. Carroll, Parsing OWL DL: trees or triples? in *Proc. 13th Int. Conf. World Wide Web* (ACM, 2004), pp. 266–275.
25. J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne and K. Wilkinson, Jena: Implementing the semantic web recommendations, in *Proc. 13th Int. World Wide Web Conf. Alternate Track Papers and Posters* (ACM, 2004), pp. 74–83.


Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.


26. S. Koide and H. Takeda, OWL-full reasoning from an object oriented perspective, in *Proc. Asian Semantic Web Conf.* (Springer, 2006), pp. 263–277.
27. M. Völkel and Y. Sure, RDFReactor: From ontologies to programmatic data access, in *Proc. Fourth Int. Semantic Web Conf.*, 2005, p. 55.
28. M. Babik and L. Hluchy, Deep integration of Python with Web Ontology Language, in *Proc. 2nd Workshop Scripting for the Semantic Web*, 2006.
29. E. Oren, R. Delbru, S. Gerke, A. Haller and S. Decker, ActiveRDF: Object-oriented semantic web programming, in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 817–824.
30. I. N. Athanasiadis, F. Villa and A.-E. Rizzoli, Ontologies, JavaBeans and relational databases for enabling semantic programming, in *Proc. 31st Annu. Int. Computer Software and Applications Conf.*, Vol. 2, 2007, pp. 341–346.
31. A. Poggi, Owlet: An object-oriented environment for OWL ontology, in *Proc. 11th WSEAS Int. Conf. Computers* (World Scientific and Engineering Academy and Society, 2007), pp. 44–49.
32. M. Zimmermann, OWL2Java: A Java code generator for OWL (2009), http://www.incunabulum.de/projects/it/owl2java/.
33. A. Poggi, Developing ontology based applications with O3L, *WSEAS Trans. Comput.* **8**(8) (2009) 1286–1295.
34. S. Groppe, J. Neumann and V. Linnemann, SWOBE: Embedding the semantic web languages RDF, SPARQL and SPARUL into Java for guaranteeing type safety, for checking the satisfiability of queries and for the determination of query result types, in *Proc. ACM Symp. Applied Computing*, 2009, pp. 1239–1246.
35. G. Stevenson and S. Dobson, Sapphire: Generating Java runtime artefacts from OWL ontologies, in *Proc. Int. Conf. Advanced Information Systems Engineering*, 2011, pp. 425–436.
36. A. Paar and Denny Vrandečić, Zhi#: OWL aware compilation, in *Proc. Extended Semantic Web Conf.*, 2011, pp. 315–329.
37. K. Wenzel, Ontology-driven application architectures with KOMMA, in *Proc. Workshop Semantic Web Enabled Software Engineering*, 2011.
38. M. Leinberger, S. Scheglmann, R. Lämmel, S. Staab, M. Thimm and E. Viegas, Semantic web application development with LITEQ, in *Proc. Int. Semantic Web Conf.*, 2014, pp. 212–227.
39. S. Baset and K. Stoffel, OntoJIT: Parsing native OWL DL into executable ontologies in an object oriented paradigm, in *OWL: Experiences and Directions — Reasoner Evaluation* (Springer, 2016), pp. 1–14.
40. J.-B. Lamy, Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies, *Artif. Intell. Med.* **80** (2017) 11–28.
41. M. Leinberger, R. Lämmel and S. Staab, The essence of functional programming on semantic data, in *Proc. European Symp. Programming*, 2017, pp. 750–776.
42. S. Bechhofer, R. Volz and P. Lord, Cooking the semantic web with the OWL API, in *Proc. Int. Semantic Web Conf.*, 2003, pp. 659–675.
43. P. Doherty, W. Lukaszewicz and A. Szalas, Efficient reasoning using the local closed-world assumption, in *Proc. Int. Conf. Artificial Intelligence: Methodology, Systems, and Applications*, 2000, pp. 49–58.
44. R. Shearer, B. Motik and I. Horrocks, HermiT: A highly-efficient OWL reasoner, in *Proc. Fifth OWLED Workshop OWL: Experiences and Directions and 7th Int. Semantic Web Conf.*, Vol. 432, 2008, p. 91.
45. R. Reiter, A logic for default reasoning, *Artif. Intell.* **13**(1–2) (1980) 81–132.

46. M. Knorr, J. J. Alferes and P. Hitzler, Local closed-world reasoning with description logics under the well-founded semantics, *Artif. Intell.* **175**(9–10) (2011) 1528–1554.
47. C. Patel, J. Cimino, J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, L. Ma, E. Schonberg and K. Srinivas, Matching patient records to clinical trials using ontologies, in *ASWC 2007: The Semantic Web*, 2007, pp. 816–829.
48. F. S. Parreiras, C. Saathoff, T. Walter, T. Franz and S. Staab, APIs à gogo: Automatic generation of ontology APIs, in *Proc. 2009 IEEE Int. Conf. Semantic Computing*, 2009, pp. 342–348.

Int. J. Soft. Eng. Knowl. Eng. 2018.28:1775-1794. Downloaded from www.worldscientific.com
by UNIVERSITY OF GEORGIA on 08/25/19. Re-use and distribution is strictly not permitted, except for Open Access articles.