Mutating OWLs: semantic mutation testing for ontologies

Cesare Bartolini[1]
[1] *Interdisciplinary Centre for Security, Reliability and Trust (SnT)*
Université du Luxembourg, Luxembourg
cesare.bartolini@uni.lu

Keywords:
Mutation testing, Ontology, OWL, Mutant generation.

Abstract:
Ontologies are an essential component of semantic knowledge bases and applications, and nowadays they are used in a plethora of domains. Despite the maturity of ontology languages, support tools and engineering techniques, the testing and validation of ontologies is a field which still lacks consolidated approaches and tools. This paper attempts at partly bridging that gap, taking a first step towards the extension of mutation testing techniques to ontologies expressed in a widely-used format. Mutation testing techniques, revisited in the light of the peculiar features of the ontology language and structure, can help in the engineering and refinement of ontologies and software based on them.

1 Introduction

The use of semantics in information technology is greatly enhancing the expressiveness of knowledge bases, especially with respect to information representation and retrieval. Information is classified according to domain-specific structures which describe the concepts and the relations between them, and this organization allows an efficient access to such information. Cross-domain organization is also made possible through the use of formal languages to describe the domains. Nowadays, knowledge bases structured according to description logic (Quillian, 1967) are popular, and they can also be generated using Natural Language Processing (NLP) techniques to classify unstructured documents.

Semantic knowledge is a wide field of research and application, and it is based on a multi-layered framework of components and technologies. However, at the very basic level, there is the need to describe the domains. This result is achieved by means of *ontologies*. Ontologies are a general concept to denote the definition of a domain, describing it at various level of abstraction.

Of course, to be used in computer systems, ontologies need to be described according to some formal language. Early attempts at defining a language to structure knowledge resulted in the Resource Description Framework (RDF) language (World Wide Web Consortium (W3C), 2014a). However, the purpose of RDF is mainly to describe resources by means of metadata, and it is too low level to provide an efficient means of describing an ontology. For that purpose, the Web Ontology Language (OWL) specification (World Wide Web Consortium (W3C), 2012) has been defined.

OWL, that was developed starting from another ontology language (Antoniou and van Harmelen, 2004) called DAML+OIL (Horrocks, 2002), is a family of abstract languages which are expressed in several different syntaxes, some of which are based on eXtensible Markup Language (XML). The primary syntax is RDF/XML, which easily maps onto RDF concepts and integrates with other XML languages.

It is widely known that there is no "right" way of defining an ontology. Its definition really depends on the domain, the desired level of abstraction, the purpose for which the ontology is intended, and a number of choices by the developer. In other words, the same domain could be represented by several totally different ontologies, which would result in different structures of the respective knowledge bases (and consequently, with different results when classifying and querying information). However, for

ontology-based applications to be integrated, it is necessary that they are based on the same ontology.

Ontologies have a number of uses, primarily that of describing some domain of knowledge from a specific perspective. In this sense, they act much like a vocabulary, similarly to a database. They have found their place as the basis of knowledge representation in many application fields, from web searches to the medical and legal domains (Horrocks, 2013).

Ontologies are also used for decision support (Rospocher and Serafini, 2013), therefore it is important that they are as complete as possible (within their domain and purpose), and also that they do not contain errors. Previous experiences (Kershenbaum et al., 2006) have highlighted the risks of using an incorrect ontology as a structure for a knowledge base. However, despite the acknowledged importance of the correctness of ontologies, few methodologies and tools exist for the testing and validation of ontologies.

This paper aims at partly filling this void by proposing a mutation testing methodology for OWL ontologies. Mutation testing is a well-known testing method that assesses the validity of a test suite by generating *mutants*, i.e., incorrect versions of the System Under Test (SUT), by introducing single errors in the trustworthy version. The ontology-based software could then be linked to the mutants generated in this way, and run against the test suite. The mutants thus killed can provide important information about the ontology and the program using it, including coverage details and fault detection.

The paper is organized as follows. Section 2 provides a survey of existing literature in ontology testing and mutation testing. Section 3 offers a high-level description of mutation testing. Section 3.1 describes the proposed methodology, explaining the various operators used for the mutation of an OWL ontology. It also contains a high-level description of the implementation of the mutation tool. Section 4 shows the methodology in action, applying the mutation operators to various ontologies in different domains. Finally, Section 5 summarizes the results and envisions some directions for future research.

2 Related work

Although knowledge bases and semantic applications are a very consolidated domain nowadays, it appears that there has been little attention to the validation of ontologies (Blomqvist et al., 2012).

The World Wide Web Consortium (W3C) provides a set of test cases for evaluating the OWL ontology from a structural point of vies (World Wide Web Consortium (W3C), 2004b). (Wang et al., 2005) defines an algorithm to "debug" ontologies in search of inconsistent classes. (García-Ramos et al., 2009) offer a means of ontology validation through user-defined test cases, whereas (McGuinness et al., 2000) defines an approach to merge large ontologies and find inconsistencies.

A lot of research addresses metrics and benchmarks for ontologies. The work proposed by (Gangemi et al., 2006) defines some measures for assessing an ontology, and evaluates these measures by means of a *meta-ontology* against which the ontology under validation is compared. This work does not seem to address the semantic correctness of the ontology but mainly its structure and engineering methodology. A similar approach, but with a greater attention to semantics, is proposed by (Burton-Jones et al., 2005). (Ma et al., 2006) defines a benchmark for the analysis of ontologies based on two different semantics, OWL Lite and OWL DL.

In (Blomqvist et al., 2012), the authors propose a methodology and tool for testing an ontology. The methodology addresses three main perspectives: verification of the Competency Questions (CQs) to which the ontology is supposed to provide an answer, verification of the inferences by means of an OWL reasoner, and provocation of errors. The last perspective differs significantly from the current work because it does not modify the ontology structure, but rather introduces test data that are inconsistent with the ontology.

An interesting approach is described in (Poveda-Villalón et al., 2012). The authors have built a testing tool which tries to search for potential pitfalls in ontology development. The list pitfalls has been introduced by the authors in (Poveda-Villalón et al., 2010). Although different from the idea of introducing errors in the ontology, their work can provide interesting suggestions for the definition of mutation operators.

An approach that combines ontology evaluation with software engineering techniques is described by (Denny Vrandečić, 2006), which introduces a proposal to adapt unit testing to OWL ontologies. In the past, several tools have been

developed for ontology unit testing, although it does not appear to be a mainstream testing approach for ontologies. Another interesting approach is presented in (Granitzer et al., 2007): instances are generated from an ontology, and hypotheses are formulates on these instances. The validation of the generated hypotheses is then fed as an input to refine the ontology.

Some previous work concerning mutation testing in the OWL language can be found in (Lee et al., 2008). The methodology does not apply to the general ontology language OWL, but rather to a specific ontology called OWL-S (World Wide Web Consortium (W3C), 2004a) which can be used as a semantic descriptor web services, and it applies mutation to classes, conditions, control flows and data flows. The purpose of that paper is not to improve an ontology and its related test suite, but rather to detect errors in the web service specification. However, some of the concepts introduced in that work are similar to those introduced in the current work.

3 Mutation testing

Mutation testing is a testing technique originally proposed in (DeMillo et al., 1978; Hamlet, 1977), although allegedly the initial idea can be traced back to a few years earlier (Lipton, 1971). It is classified either among the syntax-based testing techniques (Ammann and Offutt, 2008), or among the error-based or fault-based testing techniques (Howden, 1982; Jia and Harman, 2011). It is normally, but not exclusively, meant for unit testing (Offutt, 1994).

In its essence, it is a methodology in which small parts of a software code are changed. Its main purpose is not to test the SUT proper, but the quality of its test suite. However, it has an indirect benefit on the SUT, because the detection of faults in the test suite can often also lead to detecting errors in the SUT.

According to the description provided by (Ammann and Offutt, 2008), mutation is carried out by applying a set of *mutation operators* to a *ground string*. The ground string is expressed in the grammar, and a mutation operator is "[a] rule that specifies syntactic variations of strings generated from a grammar". These operators can also be applied directly to the grammar if no ground string exists. Mutation can be used to generate both invalid strings and strings that are valid but different from the ground string. In both cases, the strings thus generated are called *mutants*.

The mutants generated from the SUT are then executed on the test suite, and the test results are compared against those of the original code. Those mutants which behave differently with respect to the test suite are *killed* by the test suite. An ideal test suite would kill n out of n generated mutants. The whole process is generally automated by means of batch scripts, because the generation of a high number of mutants and the execution of the test suite on each is a complex and tedious process which is well-suited for automatization. Mutation can be also carried out by introducing simplifications that reduce the number of mutants (Offutt and Untch, 2001; Bartolini et al., 2008) to lower the complexity of the testing process.

Mutation testing has generally been applied to software code, particularly to Java (Ma et al., 2005; Ma et al., 2005). Previous research (Offutt et al., 1996; Ammann and Offutt, 2008) has identified a set of operators for mutation.

Traditional mutation testing operates at the syntax level, by introducing errors in the code. However, semantic mutation testing has also been defined (Offutt and Hayes, 1996; Mottu et al., 2006; Clark et al., 2010), in which mutation operators affect the semantics of the code. In other words, the code is still syntactically correct, but its functionality is different from the intended one.

3.1 Mutation testing applied to OWL

To apply the mutation testing methodology to an ontology, some premises are in order.

First off, the mutation operators will be applied to the ontology. However, the testing can be carried out in two different ways: either by viewing the ontology as the SUT, independently of what it is used for; or when the SUT is the knowledge base or software that relies upon the ontology. Choosing either perspective has significant consequences in the testing and the test suite that is used.

The mutation proposed in this paper is a kind of semantic mutation. The syntax of an ontology is managed satisfactorily by the various parsers and editors available, so unless the SUT is a new OWL editor or parser there would be little need for a syntactic mutation testing. What is significantly more interesting is the evaluation of the

ontology definition. Additionally, using OWL as the underlying specification, there is no point in working at the syntax level because OWL does not have a syntax *per se*, but can be built according to different syntaxes. In fact, the proposed methodology has been executed using the OWL/RDF, OWL/XML and Manchester (Horridge et al., 2006) syntaxes with identical results.

The mutation operators have therefore been defined as a set of operations that conceptually modify the ontology. An ontology refers to *entities*, which are the main building blocks used to represent real-world objects. The ontology does not define the entities, which are defined by the domain itself. For the purposes of this work, the following entity types have been used as the ground string for mutation:

classes represent the core concepts in the ontology. A class is the abstraction which subsumes all individuals of a given type;

individuals are the real-world objects, single instances of a class;

object properties describe the relationships between individuals;

data properties are used to associate information data to classes.

In addition to entity-specific mutation operators, it is also possible to define some general operators. In particular, some static information can be added to any entity by means of *annotations*. Typical annotations include *label* and *comment*, which are part of RDF Schema (RDFS) and are language specific.

All mutation operators affect some *axiom*, which is the base expression in the ontology. Axioms are connections between entities, and some examples of axioms are:

- a *subclass* relationship between two classes;
- the belonging of an individual to a class;
- the *domain* or the *range* of an object or data property;
- association of an annotation with its entity.

3.2 Mutation operators

This section describes the various classes of mutation operators defined for OWL mutation testing. Entities in OWL can be declared using either a human-readable Internationalized Resource Identifier (IRI), or an auto-generated one. When using the latter naming convention, which is recommended by the Protégé software, the domain-specific names must be referred to by means of label annotations. This solution is very versatile, because it does not force a naming, but an entity can have a number of names, also in different languages. However, when referring to entities using labels, the absence of a label can cause errors.

Table 1 offers an overview of all the mutation operators.

Some of the mutation operators produce identical mutants: for instance, the ORI operator, when applied to a class and to its inverse, generates two identical mutants.

3.2.1 Entities

Some mutation operators are general and can be applied to any entity:

ERE Remove entity. This operator deletes the declaration of an entity from the ontology, be it a class, property, or individual. All axioms concerning the deleted entity are removed as well.

ERL Remove label. This operator removes a label annotation from an entity.

ECL Change label language. A label annotation is composed by the actual label and a language attribute. This operator removes the language attribute, setting it to a meaningless value.

While it is possible to also apply mutation operators to comment annotations, comments are generally not meant for processing purposes, but only to provide a description to the human user. Therefore, no mutation on comment annotations has been introduced in this work. Similarly, no mutation operators have been defined for other annotations such as *versionInfo* or *seeAlso*.

3.2.2 Classes

Classes are entities which describe the conceptual abstraction of real-world objects. Class relations can be described in hierarchical terms, from the general to the particular. In other words, a class can be defined as the subclass of another class, by means of an "is a" relationship. Classes can be subclasses of more than one superclass. If a class is not defined as a subclass, then it is implicitly a subclass of the top-level class, *Thing*. A class can also be the subclass of an anonymous class, i.e., a class defined "on the fly" using properties.

Entity	Operator	Effect
Any entity	ERE	Remove the entity and all its axioms
	ERL	Remove entity labels
	ECL	Change label language
Class	CRS	Remove a single subclass axiom
	CSC	Swap the class with its superclass
	CRD	Remove disjoint class
	CRE	Remove equivalent class
Object property	OND	Remove a property domain
	ONR	Remove a property range
	ODR	Change property domain to range
	ORD	Change property range to domain
	ODP	Assign domain to superclass
	ODC	Assign domain to subclass
	ORP	Assign range to superclass
	ORC	Assign range to subclass
	ORI	Remove inverse property
Data property	DAP	Assign property to superclass
	DAC	Assign property to subclass
	DRT	Remove data type
Individual	IAP	Assign to superclasses
	IAC	Assign to subclasses
	IRT	Remove data type

Table 1: List of mutation operators

In addition to the mutation operators applicable to all entities, the following operators have been defined for class entities:

CRS Remove subclass axiom. This operator removes a subclass axiom, thus changing the hierarchical structure of the ontology. If the class has a single superclass, then it will become a subclass of the top-level class.

CSC Swap subclass axiom. This operator exchanges a class with one of its superclasses. Simply put, it reverses part of the hierarchical structure.

CRD Remove disjoint class. A class can be declared as being disjoint from other classes. This operator erases a disjoint declaration, so the two classes are no longer disjoint.

CRE Remove equivalent class. A class can be declared as being equivalent to other classes. This operator erases an equivalent declaration, so the two classes are no longer equivalent.

3.2.3 Object properties

Object properties represent relations between classes which cannot be in hierarchical terms. All relations except "is a" must be defined in terms of object properties.

An object property normally has at least one *domain* and one *range*. The domain represents the classes (which can also be anonymous classes, defined for example using set operations) to which the object property applies. A range represents the possible values that the property can have. In other words, domain and ranges are limitations to the individuals to which the property can be applied and to the individuals that it can have as its values, respectively.

The following mutation operators specific to object properties have been defined:

OND Remove domain. One domain (set of entities to which the property can apply) is removed from the object property. Since the actual domain is the intersection of all ranges, this operator actually widens the possible entities to which the property can apply.

ONR Remove range. One range is removed from the object property. Since the actual range is the intersection of all ranges, this operator actually widens the possible values that the property can have.

ODR Change domain to range. One of the domains of the property is changed to a range, actually restricting its possible values but increasing the classes it can apply to.

ORD Change range to domain. One of the ranges of the property is changed to a domain.

ODP Assign to superclass. One of the domains of the property is replaced with one of the superclasses of that domain. This operator cannot be applied to anonymous domains or to domains which are only subclass of the top-level class.

ODC Assign to subclass. One of the domains of the property is replaced with one of the subclasses of that domain. This operator cannot be applied to anonymous domains.

ORP Set range to superclass. One of the ranges of the property is replaced with one of the superclasses of that range. This operator cannot be applied to anonymous ranges or to range which are only subclass of the top-level class.

ORC Set range to subclass. One of the ranges of the property is replaced with one of the subclasses of that range. This operator cannot be applied to anonymous ranges.

ORI Remove inverse property. The property can be declared as being inverse to another one. This operator removes the inverse declaration, but it does not remove the other property.

3.2.4 Data properties

Data properties are used to describe additional features of an entity. Technically, they represent a connection between entities and literals (such as XML strings and integers). Data properties have a *domain* which limits the entities it can be applied to, and a range which limits the set of possible literals it can have as values.

In addition to the general operators, the following operators have been defined for data properties:

DAP Assign to superclass. One of the domains of the property is replaced with one of the superclasses of that domain. This operator cannot be applied to anonymous domains or to domains which are only subclass of the top-level class.

DAC Assign to subclass. One of the domains of the property is replaced with one of the subclasses of that domain. This operator cannot be applied to anonymous domains.

DRT Remove data range. One of the data ranges of the property is removed, and it is implicitly replaced with the top-level literal *rdfs:Literal*, actually increasing the set of possible literals that this property can have.

3.2.5 Individuals

Individuals represent single instances of a class (including anonymous classes). Individuals are very similar to classes, but they represent a single object and not an abstract generalization. Therefore, they can be defined as belonging to one or more classes.

The following specific operators have been defined for individuals.

IAP Assign to superclass. One of the types of the individual is replaced with one of its superclasses. This operators can be applied only to those types which have a superclass different from the top-level class.

IAC Assign to subclass. One of the types of the individual is replaced with one of its subclasses.

IRT Remove type. One of the types to which the individual belongs is removed (both named and anonymous classes). If the individual is of a single type, then it becomes an individual of the top-level class.

4 Experiments

The proposed mutation testing methodology has been implemented and executed on several existing ontologies. This section describes the test platform, the reference ontologies and the results of the application of the methodology.

4.1 Experimental setup

The implementation of the proposed mutation testing approach was done using Eclipse 4.5 (Mars) as a development environment. The programming language used is Java (Sun Java 1.8). The setup is platform-independent and has been successfully tested on Windows 7, Ubuntu Linux 14.04 and Mac OS X 10.10 machines, both at 32 and 64 bit.

The implementation is lightweight and only requires the OWL API libraries[1], managed through Maven[2].

The mutation testing tool, called Mutating OWLs, is available as a Git repository[3]. The

[1] `http://owlapi.sourceforge.net/`.

[2] `https://maven.apache.org/`.

[3] `https://bitbucket.org/guerret/lu.uni.owl.mutatingowls`.

repository also contains the test ontologies described below.

4.2 Reference ontologies

The proposed methodology has been executed on three different ontologies.

4.2.1 Data protection

The data protection ontology has been introduced in (Bartolini and Muthuri, 2015; Bartolini et al., 2015). The European Union is currently undergoing a reform of the protection of personal data. The main legislative document of the reform is the General Data Protection Regulation (GDPR), which will introduce significant changes in the duties of the controller (Reding, 2010). The ontology has been defined to describe the upcoming reform; however, it does not aim at modeling the whole domain of data protection in the European Union, but only focuses on the requirements of the data controller.

The ontology is preliminary and subject to change, especially given that the reform is not finalized yet. It is mainly made up of hierarchical relations, and contains a number of object properties that relate the duties of the controller with the corresponding rights of the data subject.

Entities in the ontology are named using an auto-generated IRI, and labels contain the human-readable names.

4.2.2 Passenger rights

The second ontology used as an experimental base has been introduced in (Rodríguez-Doncel et al., 2014a; Rodríguez-Doncel et al., 2014b) to describe the legal framework for flight incidents. In particular, the ontology addresses the perspective of the rights of the passenger.

This ontology has a more complex structure, and is split into three files. Since the import links were actually broken, some changes had to be made to the ontology to allow the OWL API to access local files. Specifically, the ontology had to be converted from Turtle syntax (World Wide Web Consortium (W3C), 2014b) to an XML serialization because of some limitations of OWL API in parsing non-XML syntaxes.

The naming convention differs from the previous ontology in that the IRIs are human-readable terms in English language, and no labels are used throughout the ontology.

4.2.3 Pizza

Finally, the proposed methodology has been run against the well-known pizza ontology[4], which is the one provided as a standard example for OWL and Protégé tutorials. The naming convention used in this ontology is based on English-language identifiers for the entities, but entities also feature label annotations in Portuguese.

4.2.4 Summary

Table 2 displays a summary of the features of the three ontologies used.

4.3 Experimental results

The mutation operators defined in Section 3.2 have been applied to the three test ontologies, generating mutants for each. The total number of mutants per mutation operator is displayed in Table 3.

Some considerations are offered by the very structure of the three ontologies. For example, the data protection ontology, as mentioned earlier, uses auto-generated IRIs as identifiers, and labels for descriptive purposes. The pizza ontology uses English terms as identifiers, but entities also have Portuguese labels. Finally, the passenger rights ontology does not use label annotations. For this reason, the ERL and ECL operators do not generate any mutant in the latter. Similarly, no mutant is generated by the IAP, IAC and IRT operators in the passenger rights ontology because the individuals are not assigned to any class.

The data protection ontology makes a very limited use of data properties, so very few mutants are generated from the data property entity; the same is not true for the passenger rights entity, which has a significant number of data properties but less object properties. The pizza ontology does not have any data properties at all, and few object properties. However, the classes that make up the domain and range of some of the object properties have a large number of subclasses, hence many mutants from the ODC and ORC operators.

[4] `http://protege.stanford.edu/ontologies/pizza/pizza.owl`.

	Data protection	Passenger rights	Pizza
Total number of axioms	680	541	940
Classes	87	89	100
Object properties	41	26	8
Data properties	4	31	0
Individuals	12	14	5
Subclass axioms	114	83	259

Table 2: Summary of the test ontologies.

Operator	Data protection	Passenger rights	Pizza
ERE	142	67	113
ERL	142	0	96
ECL	142	0	96
CRS	114	33	259
CSC	101	33	84
CRD	18	0	796
CRE	28	0	41
OND	39	10	6
ONR	34	8	7
ODR	39	10	6
ORD	34	8	7
ODP	29	8	6
ODC	239	54	254
ORP	29	5	7
ORC	119	22	257
ORI	0	0	0
DAP	2	29	0
DAC	5	3	0
DRT	3	13	0
IAP	12	0	0
IAC	30	0	0
IRT	12	0	10

Table 3: Mutants by mutation operator.

4.4 Validation

The proposed approach was validated by testing an ontology itself and not an application running on top of it. For the SUT to be an ontology, the simplest approach to test it is to have a set of SPARQL Protocol and RDF Query Language (SPARQL) queries (World Wide Web Consortium (W3C), 2008) which retrieve data from the ontology.

Unfortunately, none of the ontologies used provide a SPARQL test suite. A set of queries for the pizza ontology exists as the test suite for an alternative query language[5]. As minimal as this test suite is, it has been used as a starting point to assess the validity of the approach. The queries in that test suite were thus converted back to SPARQL. However, two more queries were added to the test suite, because the existing queries only search for very small parts of the ontology[6].

The results of the validation is shown in Table 4, and a summary of killed mutants is shown in Figure 1.

A brief analysis of the results elicits some interesting considerations. First, it is clear that the test suite mainly addresses classes, with little tests covering the properties. Thus, additional tests, especially for the object properties, are required. Also, concerning the classes, the tests mostly cover a particular branch of the hierarchy,

[5] `https://code.google.com/p/twouse/wiki/SPARQLASExamples`.

[6] The complete setup is available in the repository (see footnote 3).

Operator	Mutants killed	Total mutants
ERE	108	112
ERL	95	95
ECL	95	95
CRS	255	255
CSC	83	83
CRD	471	753
CRE	41	41
OND	0	6
ONR	0	7
ODR	0	6
ORD	0	7
ODP	0	6
ODC	1	250
ORP	0	7
ORC	1	253
IRT	0	10

Table 4: Results of the mutation testing.

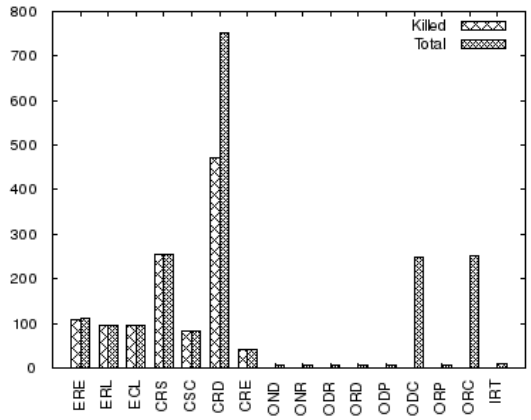

Figure 1: Overview of mutants killed.

while almost no tests search other branches of the ontology. Finally, some considerations can be done on the ontology itself. For example, by examining the live mutants in the ERE operator, it emerges that some object properties are not used anywhere. Depending on the purposes of the ontology, this might suggest that those properties are irrelevant. Of course, such a small test suite does not yield a lot of results, but a richer test suite would allow a more significant analysis.

5 Conclusions and future work

The work presented in this paper extends and adapts mutation testing techniques to ontologies defined using the OWL language. The essentials of OWL ontologies are described, to introduce a methodology and operators for mutation testing. The work then presents an implementation of the mutation testing technique and some basic experiments on previously-defined ontologies.

The benefits of mutation testing are manifold: by analyzing the patterns of killed and alive mutants, testers can detect errors in the SUT and in the test suite. Equivalent mutants can help detect redundancies in the ontology, which may not be errors but still facilitate errors, for example when creating instances of the ontology.

More in general, the extension of software engineering and testing approaches to ontologies and semantic knowledge bases can pave the way to the formalization of integrated design and testing patterns in for semantics-based applications.

This work is at its initial stages, with many opportunities for future development. First off, the proposed methodology needs to be expanded to support a full test suite: a significant set of SPARQL queries, if the SUT is the ontology itself; or, if the SUT is an ontology-based software, testing it with its own test suite. The purpose would be to compare the outputs of the test suite when executed against the original ontology and against the mutants. In this phase, it is possible that the complexity of the mutation testing is excessive and causes performance problems, and it might be necessary to apply or develop algorithms designed to reduce the number of mutants.

Second, the mutation methodology can be improved, by extending it with additional mutation operators. For example, the mutation operators do not currently address annotations other than labels, or the value and cardinality constraints. Some of these features of the OWL language can have a significant effect in the ontology definition, and mutants thus created might be useful in assessing the ontology.

Third, the mutation testing should take into account the peculiarities of ontology engineering. In particular, while the domain certainly imposes some constraints on the ontology developer, many decisions are based on discretionary choices, balancing different aspects such as human readability and efficiency of the ontology. Traditional mutation testing techniques might be extended to embrace these features, for example by separating those mutant operators that are likely to introduce errors in the domain (for example swapping a class with its parent) from those that simply change the ontology structure without making it inconsistent with the domain. If such a partition

were possible, then mutation testing techniques could be used not only to detect errors in the design, but also to suggest different ontology architectures that the designer might overlook.

Finally, stretching along the line of the previous point, an extended mutation technique could be designed which alters the structure of the ontology. For example, there might be circumstances where using a hierarchical relationship (subclass axiom) might be an alternative to using an object property. An extended mutation technique that generates mutants based on a different structure of the ontology might offer a fast way to compare a wide number of ontology designs.

REFERENCES

Ammann, P. and Offutt, A. J. (2008). Syntax-based testing. In *Introduction to Software Testing*, chapter 5, pages 170–212. Cambridge University Press.

Antoniou, G. and van Harmelen, F. (2004). Web Ontology Language: OWL. In Staab, S. and Studer, R., editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 67–92. Springer Berlin Heidelberg.

Bartolini, C., Bertolino, A., Marchetti, E., and Parissis, I. (2008). Data flow-based validation of web services compositions: Perspectives and examples. In de Lemos, R., Di Giandomenico, F., Gacek, C., Muccini, H., and Vieira, M., editors, *Architecting Dependable Systems V*, volume 5135 of *Lecture Notes in Computer Science*, pages 298–325. Springer Berlin / Heidelberg.

Bartolini, C. and Muthuri, R. (2015). Reconciling data protection rights and obligations: An ontology of the forthcoming EU regulation. In *Proceedings of the Workshop on Language and Semantic Technology for Legal Domain (LST4LD), Recent Advances in Natural Language Processing (RANLP)*.

Bartolini, C., Muthuri, R., and Santos, C. (2015). Using ontologies to model data protection requirements in workflows. In *Proceedings of the Ninth International Workshop on Juris-informatics (JURISIN)*, pages 27–40.

Blomqvist, E., Seil Sepour, A., and Presutti, V. (2012). Ontology testing - methodology and tool. In ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Acquin, M., Nikolov, A., Aussenac-Gilles, N., and Hernandez, N., editors, *Knowledge Engineering and Knowledge Management*, volume 7603 of *Lecture Notes in Computer Science*, pages 216–226. Springer Berlin Heidelberg.

Burton-Jones, A., Storey, V. C., Sugumaran, V., and Ahluwalia, P. (2005). A semiotic metrics suite for assessing the quality of ontologies. *Data & Knowledge Engineering*, 55(1):84–102.

Clark, J. A., Dan, H., and Hierons, R. M. (2010). Semantic mutation testing. In *Proceedings of the 3rd IEEE International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*, pages 100–109. IEEE.

DeMillo, R. A., Lipton, R. J., and Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41.

Denny Vrandečić, A. G. (2006). Unit tests for ontologies. In Meersman, R., Tari, Z., and Herrero, P., editors, *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, volume 4278 of *Lecture Notes in Computer Science*, pages 1012–1020. Springer Berlin Heidelberg.

Gangemi, A., Catenacci, C., Ciaramita, M., and Lehmann, J. (2006). Modelling ontology evaluation and validation. In Sure, Y. and Domingue, J., editors, *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science*, pages 140–154. Springer Berlin Heidelberg.

García-Ramos, S., Otero, A., and Fernández-López, M. (2009). OntologyTest: A tool to evaluate ontologies through tests defined by the user. In Omatu, S., Rocha, M. P., Bravo, J., Fernández, F., Corchado, E., Bustillo, A., and Corchado, J. M., editors, *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, volume 5518 of *Lecture Notes in Computer Science*, pages 91–98. Springer Berlin Heidelberg.

Granitzer, M., Scharl, A., Weichselbraun, A., Neidhart, T., Juffinger, A., and Wohlgenannt, G. (2007). Automated ontology learning and validation using hypothesis testing. In Wegrzyn-Wolska, K. M. and Szczepaniak, P. S., editors, *Advances in Intelligent Web Mastering*, volume 43 of *Advances in Soft Computing*, pages 130–135. Springer Berlin Heidelberg.

Hamlet, R. G. (1977). Testing programs with the aid of a compiler. *IEEE Transactions on Software Engineering*, SE-3(4):279–290.

Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., and Wang, H. H. (2006). The Manchester OWL syntax. In *OWL: Experiences and Directions Workshop (OWLED)*.

Horrocks, I. (2002). DAML+OIL: a description logic for the semantic web. *Bulletin of the Technical Committee on Data Engineering*, 25(1):4–9.

Horrocks, I. (2013). What are ontologies good for? In Küppers, B.-O., Hahn, U., and Artmann, S., editors, *Evolution of Semantic Systems*, pages 175–188. Springer Berlin Heidelberg.

Howden, W. E. (1982). Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, SE-8(4):371–379.

Jia, Y. and Harman, M. (2011). An analysis and survey of the development of mutation test-

ing. *IEEE Transactions on Software Engineering*, 37(5):649–678.

Kershenbaum, A., Fokoue, A., Patel, C., Welty, C., Schonberg, E., Cimino, J., Ma, L., Srinivas, K., Schloss, R., and Murdock, J. W. (2006). A view of OWL from the field: Use cases and experiences. In Cuenca Grau, B., Hitzler, P., Shankey, C., and Wallace, E., editors, *Proceedings of the Second Workshop on OWL: Experiences and Directions (OWLED)*, volume 216 of *CEUR Workshop Proceedings*.

Lee, S., Bai, X., and Chen, Y. (2008). Automatic mutation testing and simulation on OWL-S specified web services. In *Proceedings of the 41st Annual Simulation Symposium (ANSS)*, pages 149–156. IEEE.

Lipton, R. (1971). Fault diagnosis of computer programs. Technical report, Carnegie Mellon University.

Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., and Liu, S. (2006). Towards a complete OWL ontology benchmark. In Sure, Y. and Domingue, J., editors, *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science*, pages 125–139. Springer Berlin Heidelberg.

Ma, Y.-S., Offutt, A. J., and Kwong, Y. R. (2005). Mujava: An automated class mutation system. *Software Testing, Verification and Reliability*, 15(2):97–133.

McGuinness, D. L., Fikes, R., Rice, J., and Wilder, S. (2000). An environment for merging and testing large ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, pages 483–493.

Mottu, J.-M., Baudry, B., and Le Traon, Y. (2006). Mutation analysis testing for model transformations. In Rensink, A. and Warmer, J., editors, *Model Driven Architecture - Foundations and Applications*, volume 4066 of *Lecture Notes in Computer Science*, pages 376–390. Springer Berlin Heidelberg.

Offutt, A. J. (1994). A practical system for mutation testing: help for the common programmer. In *Proceedings of the International Test Conference (ITC)*, pages 824–830. IEEE Computer Society.

Offutt, A. J. and Hayes, J. H. (1996). A semantic model of program faults. *SIGSOFT Software Engineering Notes*, 21(3):195–200.

Offutt, A. J., Lee, A., Rothermel, G., Untch, R. H., and Zapf, C. (1996). An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(2):99–118.

Offutt, A. J. and Untch, R. H. (2001). Mutation 2000: Uniting the orthogonal. In Wong, W. E., editor, *Mutation Testing for the New Century*, volume 24 of *The Springer International Series on Advances in Database Systems*, pages 34–44. Springer US.

Poveda-Villalón, M., Suárez-Figueroa, M. C., and Gómez-Pérez, A. (2010). Common pitfalls in ontology development. In Meseguer, P., Mandow, L., and Gasca, R. M., editors, *Current Topics in Artificial Intelligence*, volume 5988 of *Lecture Notes in Computer Science*, pages 91–100. Springer Berlin Heidelberg.

Poveda-Villalón, M., Suárez-Figueroa, M. C., and Gómez-Pérez, A. (2012). Validating ontologies with OOPS! In ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Acquin, M., Nikolov, A., Aussenac-Gilles, N., and Hernandez, N., editors, *Knowledge Engineering and Knowledge Management*, volume 7603 of *Lecture Notes in Computer Science*, pages 267–281. Springer Berlin Heidelberg.

Quillian, M. R. (1967). Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12(5):410–430.

Reding, V. (2010). The upcoming data protection reform for the European Union. *International Data Privacy Law*.

Rodríguez-Doncel, V., Santos, C., and Casanovas, P. (2014a). A model of air transport passenger incidents and rights. In *Proceedings of the 27th International Conference on Legal Knowledge and Information Systems (JURIX)*, pages 55–60. IOS Press.

Rodríguez-Doncel, V., Santos, C., and Casanovas, P. (2014b). Ontology-driven legal support-system in the air transport passenger domain. In *Proceedings of the International Workshop on Semantic Web for the Law (SW4Law)*.

Rospocher, M. and Serafini, L. (2013). An ontological framework for decision support. In Takeda, H., Qu, Y., Mizoguchi, R., and Kitamura, Y., editors, *Semantic Technology*, volume 7774 of *Lecture Notes in Computer Science*, pages 239–254. Springer Berlin Heidelberg.

Wang, H., Horridge, M., Rector, A., Drummond, N., and Seidenberg, J. (2005). Debugging OWL-DL ontologies: A heuristic approach. In Gil, Y., Motta, E., Benjamins, V. R., and Musen, M. A., editors, *The Semantic Web - ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 745–757. Springer Berlin Heidelberg.

World Wide Web Consortium (W3C) (2004a). OWL-S: Semantic markup for web services.

World Wide Web Consortium (W3C) (2004b). OWL Web Ontology Language test cases.

World Wide Web Consortium (W3C) (2008). Sparql query language for rdf.

World Wide Web Consortium (W3C) (2012). OWL 2 Web Ontology Language document overview (second edition).

World Wide Web Consortium (W3C) (2014a). RDF 1.1 concepts and abstract syntax.

World Wide Web Consortium (W3C) (2014b). RDF 1.1 Turtle.