Themis: a tool for validating ontologies through requirements

Alba Fernández-Izquierdo
Ontology Engineering Group
Universidad Politécnica de Madrid
albafernandez@fi.upm.es

Raúl García-Castro
Ontology Engineering Group
Universidad Politécnica de Madrid
rgarcia@fi.upm.es

***Abstract*— The validation of ontologies, whose aim is to check whether an ontology matches the conceptualization it is meant to specify, is a key activity for guaranteeing the quality of ontologies. This work is focused on the validation through requirements, with the aim of assuring, both the domain experts and ontology developers, that the ontologies they are building or using are complete regarding their needs. Inspired by software engineering testing processes, this work proposes a web-based tool called Themis, independent of any ontology development environment, for validating ontologies by means of the application of test expressions which, following lexico-syntactic patterns, represent the desired behaviour that will present an ontology if a requirement is satisfied.**

I. Introduction

In software engineering it is inconceivable to deliver a software product without its pertinent tests which guarantee that it fulfills all its requirements. Besides, there are several approaches integrated into the software development process whose aim is to test the software. Unit testing [1], which validates that each unit of the software performs as designed, and behaviour-driven development [2], which focuses on the behaviour the software product is implementing, are examples of these approaches. Moreover, there are specific syntaxes, such as Gherkin,[1] which generate unambiguous specifications of software to automate the testing process.

However, in ontology engineering there is a lack of clearly defined testing processes in order to be able to ascertain whether an ontology satisfies its functional requirements [3], which state the particular knowledge that should be represented. Such ontological requirements used to be written in form of competency questions [4] or natural language sentences. The main issue when performing testing processes in the ontology engineering field is the ambiguity of the ontological requirements, which sometimes are difficult to formalize into tests and to translate into axioms. Therefore, inspired by software engineering and its specific syntax for the definition of tests, we propose Themis,[2] a tool which provides a set of test expressions based on lexico-syntactic patterns (LSPs) related to ontological requirements. These LSPs allows to relate different types of requirements with the axioms needed to implement them in an ontology, and such implementations are used by Themis to identify whether a requirement is satisfied.

DOI reference number: 10.18293/SEKE2019-117

[1]https://docs.cucumber.io/gherkin

[2]http://themis.linkeddata.es

Themis can be used by both domain experts and ontology developers to validate ontologies regarding their functional requirements. Other type of requirements, such as non-functional ones (e.g., "the ontology URIs must be in English") are not considered in this work as they cannot be formalized into axioms and, therefore, they cannot be automatically checked.[3] Moreover, the proposed tool allows to execute tests on multiple ontologies simultaneously, in order to check ontological commitments between them.

The paper is organized as follows. Section 2 presents the state of the art on ontology testing tools. Section 3 presents Themis and Section 4 describes a use case in which Themis was integrated into a research project. Finally, Section Section 5 presents the conclusions we obtained and gives an overview of future work.

II. State of the art

Currently, there are several methodologies and tools that support executing tests on an ontology, in order to validate that the ontology satisfies the pertinent requirements.

Regarding methodologies for testing ontologies, Vrandevcic and Gangemi [5] introduced the idea of testing ontologies by borrowing ideas from software engineering, proposing techniques such as testing with axioms and negations or formalizing competency questions. Another work presented by Peroni is SAMOD [6], an ontology development methodology that uses tests for validation. These two approaches are focused on methodological aspects but do not mention how to implement the tests or how to maintain traceability.

Another approach to implement testing is the one presented by Ren et al. [7]; in this work the authors use natural language processing to analyse competency questions written in controlled natural language from where they create competency question patterns that could be automatically tested in the ontology. Additionally, Keet and Lawrynowicz proposed a test-driven development of ontologies [8] in which the competency questions are formalized into axioms and added to the ontology if they are not present.

Concerning testing tools, the OntologyTest tool [9] allows a user to define and execute a set of tests to check the functional requirements of an ontology; these tests are stored in an XML file for future reuse. It is worth mentioning that

[3]For the sake of clarity, from now on we are going to refer to functional requirements simply as requirements.

these tests are focused on the ontology data, rather than on the ontology model itself.

Another work related to ontology testing tools is Scone,[4] a tool for scenario-based ontology evaluation, which is based on Cucumber[5] and uses controlled natural language to define ontology scenarios which create mock individuals. Additionally, Blomqvist et al. [10] presented an agile approach and tool available as an Eclipse plugin. This tool supports three types of test, namely: (1) verification example, (2) inference verification and (3) error provocation. The first two are concerned with verifying the correct implementation of a requirement and the third is intended to expose faults. All the tests are stored in a different OWL ontology with information about the requirement, e.g., type of test or expected output. By saving these test suites in an ontology it allows the user to reuse them and to maintain traceability between the requirements and all the associated information. However, these tools neither explains how to implement these tests nor how to use them.

To conclude, a more recent work on testing tools is the TDDonto tool [11], proposed by Lawrynowicz and Keet, which follows a test-driven development of ontologies approach [8] in which the competency questions are formalized into axioms and added to the ontology if they are not present. This tool, which is implemented as a Protégé[6] plugin, allows to check the presence of a particular axiom in the ontology.

Even if all these works proposed solutions for testing through requirements, several of these works do not allow the reuse of the tests, limiting the testing process only to a single ontology. Moreover, there is a lack of information about how to translate a requirement into a test.

III. Themis in the testing process

Themis is a web-based tool which implements the testing process described in [12] which describe a framework to validate an ontology regarding its requirements. Such testing framework focuses on analysing the behaviour of the ontology in different situations to verify that certain knowledge is modelled in the ontology. This testing process implemented by Themis is divided into three activities, namely: (1) Test design, (2) Test implementation and (3) Test execution. Figure 1 summarizes the inputs and outputs of each step in the testing process; as shown in the figure, Themis supports and automates the test implementation and execution activities.

A. Test design

During the test design activity the desired behaviour of each requirement, i.e., the expected knowledge that should be added to the ontology, is manually extracted. This desired behaviour is formalized into *test expressions* in a formal language based on the OWL Manchester Syntax.[7] These test expressions do not include any URIs related to the ontology

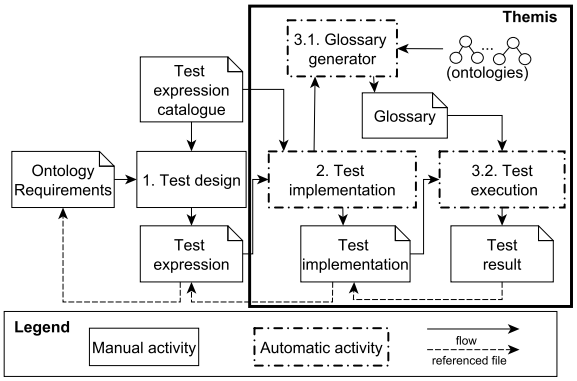

Fig. 1. Testing process proposed in [12] and Themis

in which the test cases are going to be executed. As an example, if a developer wants to check the cardinality of a relation between two concepts called Device and Service in an ontology, the test expression should be "*Device subClassOf Property hasService max 1 Service*". The absence of URIs allows the reuse of the same test expressions in other ontologies, or the reuse of tests in the same ontology even if the naming changes, without being restricted to the ontology from which the tests were generated.

Themis supports a list of possible tests expressions which are extracted from LSPs and that can be executed on an ontology. LSPs are understood as "formalized linguistic schemas or constructions derived from regular expressions in natural language that consist of certain linguistic and paralinguistic elements, following a specific syntactic order, and that permit to extract some conclusions about the meaning they express" [13]. The LSPs used by Themis were extracted from the CORAL corpus[8] which, based on the NeOn modelling components [14], analyses 834 ontology requirements in order to identify LSPs based on the goal that each requirement has regarding its implementation in an ontology, e.g., a relation between two concepts. Therefore, these LSPs indicate the implementation in the ontology associated with a particular requirement template, which can be use to generate the corresponding tests.

Moreover, some of these LSPs are related to one or more ontology design patterns (ODPs) [15] which indicate how, following good practices, the LSP should be implemented in an ontology. Table I summarizes the list of test expressions supported by Themis, together with the test expression syntax, an example of a test expression and an example of a requirement from which the test could be extracted. More examples of test expressions with their associated requirement templates are available in the Themis portal.[9]

In the Themis web user interface the user enters the test expression or set of test expressions that represent the desired behaviour of a requirement. Such translation between requirements and test expressions should be done manually

[4]https://bitbucket.org/malefort/scone
[5]https://docs.cucumber.io
[6]https://protege.stanford.edu
[7]https://www.w3.org/TR/owl2-manchester-syntax
[8]http://coralcorpus.linkeddata.es
[9]http://themis.linkeddata.es/examples.html

TABLE I

TEST EXPRESSION CATALOGUE

Test goal	Test expression syntax	Test expression example	Example of requirement associated
T1 Equivalence	A EquivalentTo B	SecuritySchema equivalentTo Security	Security schema is equivalent to security
T2 Subsumption	A subClassOf B	Sensor subClassOf Device	A sensor is a type of device
T3 Disjointness	A disjointWith B	Sensor disjointWith Actuator	A sensor cannot be an actuator
T4 Property between two concepts	A subClassOf P some B	Bird subClassOf build some Nest	Birds build nests
T5 Universal restriction	A subClassOf P only B	User subClassOf interactsWith only Application	A user can only interact with the systems by means of an application
T6 Multiple inheritance	A subClassOf B and C	Sensor subClassOf System and Device	A sensor must be a system and a device
T7 Symmetry	A Symmetric(P) B	Partnership subClassOf symmetricProperty(hasParntershipWith) some Organization	There is a partnership between two organizations
T8 Maximum cardinality	A subClassOf P max [num] B	Person subClassOf hasAddress max 1 Address	A person has at most 1 address
T9 Minimum cardinality	A subClassOf P min [num] B	Device subClassOf hasManufacturer min 1 string	A device has at least 1 manufacturer
T10 Cardinality	A subClassOf P max [num]B	Device subClassOf hasManufacturer exactly 1 Brand	A device has exactly 1 brand
T11 The ontology contains the individual	I type A	StriatedPardalote type Bird	A Striated Pardalote is an example of bird
T12 Subsumption and relation between classes	A subClassOf [ClassB] that [PropertyP] some C	Price subClassOf UnitOfMeasure that isCharacterizedBy some Currency	The price, which is a type of unit of measure, is characterized by a value using currency
T13 Minimum cardinality and relation between classes	A subClassOf [PropertyP] min [num]B and B subClassOf [PropertyP] some C	Device subClassOf performs min 1 Function and Function subClassOf accomplish some Task	A device performs at least 1 function and each function accomplishes some task
T14 Minimum cardinaly and universal restriction	A subClassOf [PropertyP] min [num]B and B subClassOf [PropertyP] only C	Person subClassOf hasTask min 1 Task and Task subClassOf hasGoal only Goal	A person performs at least one task and each task has a goal
T15 Definition of a disjoint set of classes	A subClassOf B and C subClassOf B that disjointWith A	Carnivora subClassOf MarineMammals and Sirenia subClassOf MarineMammals that disjointWith Carinvora	Marine mammals are divided into two different types: Carnivora and Sirenia

using the guidelines provided in Themis website.[10] As an example, the user can enter the test expression *Event subClassOf InteractionPattern* in order to check a subsumption relation in the ontology expected from the requirement "An event is a type of Interaction pattern".

Additionally, Themis allows the users to export a set of test expressions, i.e., a test suite, as an RDF file in order to be able to reuse it. Listing 1 shows an example of the test expressions *Action subClassOf InteractionPattern* and *Event subClassOf InteractionPattern* in RDF syntax. Such test expressions aims to check subsumption relations in the ontology. This RDF file uses the ontology Verification Test Case[11] ontology to describe each test case.

This RDF file can be improved by adding the requirements associated to the tests, in order to provide a link between the test case and the requirements from which it is extracted. Listing 1 shows the test expressions *Action subClassOf InteractionPattern* and *Event subClassOf InteractionPattern* with the URIs of requirements associated, together with the description of the requirements which specifies what is an action and what is an event.

Listing 1. Example of improved test suite in RDF file

```
@prefix dc:      <http://purl.org/dc/terms/> .
@prefix vtc: <http://w3id.org/def/vtc#> .
@prefix xsd: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix vicinity: <http://vicinity.iot.linkeddata.es/vici-
                  nity/requirements/report-wot.html#> .
@prefix   :      <> .

:test-case-3 a vtc:TestCaseDesign;
  vtc:isRelatedToRequirement vicinity:wot16;
  dc:description "What is an action? The Action interaction
  pattern is an interaction pattern that targets changes or
  processes on a Thing that take a certain time to com-
  plete"
  vtc:desiredBehaviour "Action subClassOf Interaction-
  Pattern"^^xsd:string .

:test-case-4 a vtc:TestCaseDesign;
  vtc:isRelatedToRequirement vicinity:wot17;
  dc:description "What is an event? The Event interaction
  pattern  is an interaction pattern that enables a
  mechanism to  be notified by a Thing on a certain
  condition.";
  vtc:desiredBehaviour "Event subClassOf Interaction-
  Pattern"^^xsd:string .
```

These RDF files can be also loaded in Themis, which will extract the test expressions that in the ontology are stored as *desiredBehaviour* data properties. The test suites can be modified and adapted to a particular ontology by the user. This functionality allows to reuse test suites already defined and published on the Web, e.g., test suites already defined for the ontology to be analysed, or test suites defined for other ontologies that could be used to check alignment.

B. Test implementation

During the test implementation activity each test expression is formalized into a precondition, a set of auxiliary term declarations and a set of assertions to check the behaviour. The precondition is a SPARQL query which checks whether the terms involved in the ontology requirement are defined in the ontology. The axioms to declare auxiliary terms are a set of temporary axioms added to the ontology to declare the auxiliary terms needed to carry out the assertions. Finally, the assertions to check the behaviour are a set of pairs of axioms and expected results that represent different ontology scenarios. For each pair, the axiom is temporary

[10] http://themis.linkeddata.es/tests-info.html

[11] http://w3id.org/def/vtc

added to the ontology to force a scenario, after which the reasoner is executed. The expected result determines if the ontology status (i.e., the ontology is inconsistent, there is an unsatisfiable class or the ontology is consistent) after the addition is the expected one in the case the requirement was satisfied. If all the status concurs with the expected status, then the requirement is satisfied. Figure 2 summarizes the flow of this test implementation.

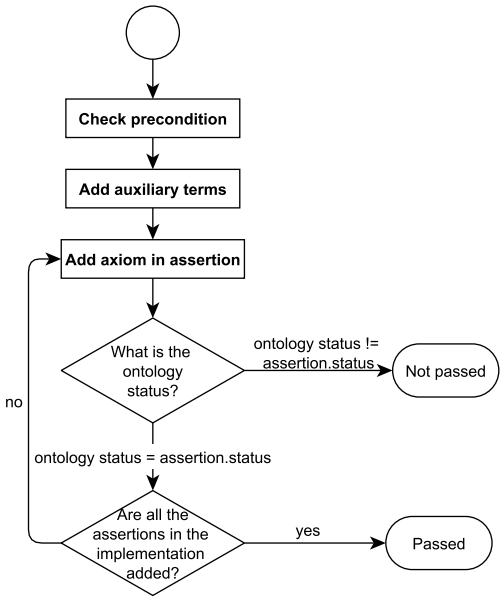

Fig. 2. Test implementation flow

Themis creates the implementation of each test expression listed in Table I. Once the user enters the test expression to be checked, Themis identifies the type of test and generates the correct implementation. As an example, if the test expression added by the user is *Action subClassOf InteractionPattern*, Themis will identify that it is of type *T2 Subsumption* and that the implementation includes[12]:

- **Preconditions**: Class Action and class InteractionPattern exist
- **Axioms to declare auxiliary terms**: Declaration of ¬Action and ¬InteractionPattern
- **Assertion 1**:
  - **Axiom**: Action' $\sqsubseteq$ ¬Action $\sqcap$ InteractionPattern
  - **Expected status after adding the axiom**: Consistent
- **Assertion 2**:
  - **Axiom**: Action' $\sqsubseteq$ Action $\sqcap$ ¬InteractionPattern
  - **Expected status after adding the axiom**: Unsatisfiable class
- **Assertion 3**:
  - **Axiom**: Action' $\sqsubseteq$ Action $\sqcap$ InteractionPattern
  - **Expected status after adding the axiom**: Consistent ontology

[12]This implementation is described by means of Description Logics symbols.

C. Test execution

Finally, during the test execution activity, the implementation of the test is executed in the ontology to be tested. At this point, the ontology URI needs to be indicated, as well as the URIs of each term in the test expression in order to be able to execute it, e.g., the terms Action and InteractionPattern in the ontology with URI *http://iot.linkeddata.es/def/wot#* will be *http://iot.linkeddata.es/def/wot#Action* and *http://iot.linkeddata.es/def/wot#InteractionPattern*, respectively. These terms URIs needs to be extracted from the glossary of terms of the analysed ontology.

Themis proposes a glossary of terms where the terms are extracted from the fragments of the URIs of each concept. Table II shows an excerpt of a glossary of terms created by Themis for the ontology VICINITY Web Of Things.[13] This preliminary glossary of terms can be modified by the users if needed. Even though in this case the terms coincide with the URIs' fragments, that is not required.

TABLE II
EXCERPT OF GLOSSARY OF TERMS

Term	URI
Event	http://iot.linkeddata.es/def/wot#Event
Action	http://iot.linkeddata.es/def/wot#Action
InteractionPattern	http://iot.linkeddata.es/def/wot#InteractionPattern
CommunicationProtocol	http://iot.linkeddata.es/def/wot#CommunicationProtocol
isProvidedOverProtocol	http://iot.linkeddata.es/def/wot#isProvidedOverProtocol

The test execution activity consists of three parts: the execution of the query which represents the preconditions, the addition of the axioms which declare the auxiliary terms, and the addition of the assertions. After the addition of each axiom, the reasoner is executed to report the status of the ontology. The addition of the auxiliary axioms needs to always lead to a consistent ontology. In the case of the assertions, the agreement between the reasoner status after the addition of all the axioms and the status indicated in the test implementation determines whether the ontology satisfies the desired behaviour. All the results of each step are stored in RDF files, in order to enable traceability between them and the requirements.

Themis uses OWL API[14] to load the ontologies and to add axioms, and uses Pellet[15] as the reasoner to check the ontologies consistency. Themis can execute a test expression on several ontologies simultaneously, allowing the users to identify common knowledge and commitments between a collection of ontologies regarding their requirements.

Four possible results can be returned for each test and each ontology:

1) *Undefined terms*: if the ontology does not pass the preconditions, i.e., the terms in the test expression are not defined in the ontology to be analysed.
2) *Passed*: if the ontology passes the preconditions and the results of the assertions are the expected ones.

[13]http://iot.linkeddata.es/def/wot
[14]http://owlapi.sourceforge.net/documentation.html
[15]https://www.w3.org/2001/sw/wiki/Pellet

3) *Absent relation*: if the ontology passes the preconditions and the results of the assertion are not the expected ones but there are no conflicts in the ontology.
4) *Conflict*: if the ontology passes the preconditions and the results of the assertion are not the expected ones, and the addition of the axioms related to the test expression leads to a conflict in the ontology.

As an example, Themis can determine that the test expression *Action subClassOf InteractionPattern* is passed in the VICINITY Web Of Things ontology[16] but has undefined terms in the oneM2M ontology[17], due to the fact that the latter ontology does not consider the modelling of such concepts related to actions or interaction patterns.

IV. Application of Themis

Themis has been used during the ontology development process of the VICINITY European h2020 project,[18] where five ontologies are currently under development.

The five ontologies to be analysed in the project, i.e., the VICINITY Core (Core), the Web of Things (WoT), the WoT mappings (Mappings), the VICINITY Adapters (Adapters), and the Datatypes (Datatypes) ontologies, belong to the VICINITY ontology network and aim to provide interoperability in the IoT domain. The Core ontology represents the information needed to exchange IoT descriptor data between peers through the VICINITY platform; this ontology is being created by following a cross-domain approach and implements requirements from different domain experts. The WoT ontology aims to model the Web of Things domain according to the W3C WoT Interest Group[19] descriptions. The Mappings ontology represents the mechanism for accessing the values provided by web things in the VICINITY platform. The Adapters ontology aims to model all the different types of devices and properties that can be defined in the VICINITY platform. Finally, the Datatypes ontology aims to model the required and provided datatypes that are used in the interaction patterns of the platform.

Table III summarizes the number of requirements defined for each ontology, as well as the number of test cases extracted from them. These requirements represent the needs asked by the domain experts in order to model the VICINITY platform. More information about these ontologies is available in the VICINITY ontology network portal.[20]

The test cases were extracted by using the test expression catalogue provided in Table I. Each requirement is translated to one or more test expressions, selecting the most appropriate one from the test catalogue. Several test cases can be related to the same requirement. The test suite associated to each ontology, where all the test expressions are stored, was exported to an RDF file and uploaded to the VICINITY ontology portal,[21] with the aim of reusing them in future releases of the ontology to assure that all the previous requirements are still satisfied.

TABLE III
Details of the ontologies

Ontology	Numer of requirements	Number of tests
Core ontology	50	50
WoT ontology	24	24
Mappings ontology	15	15
Adapters ontology	127	154
Dataypes ontology	11	12

Once all the tests were defined, Themis was executed in order to identify if there are tests that are not passed by the ontology. Figure IV summarizes the obtained results of such execution. As the table shows, the majority of the requirements are passed by the ontology. However, there are several tests whose results are *absent relation*, which means that the ontology passes the preconditions, the results of the assertion are not the expected ones but there are no conflicts in the ontology. These results warn the ontology developers that there are some tests that, even though they do not cause any conflict in the ontology, they are not implemented, at least completely, in the ontology.

TABLE IV
Testing results of the VICINITY ontology network

Ontology	Number of tests	Undefined terms	Passed	Absent relation	Conflict
Core	50	0	34	16	0
WoT	24	0	22	2	0
Mappings	15	0	13	2	0
Adapters	154	0	154	0	0
Dataypes	12	0	12	0	0

Additionally, the VICINITY ontology network should be aligned with the requirements of several standards in the IoT field in order to reuse concepts and patterns of well-known resources, namely: (1) the ETSI SAREF ontology[22] [16], the W3C SSN ontology[23] , (3) OCF standards[24], (4) the oneM2M ontology [16] and (5) ISO/IEC 30141:2017 [17]. Therefore, the requirements related to these standards were collected, and the associated tests were defined by using the test expression catalogue. Table V summarizes the number of requirements and test cases associated.

TABLE V
Details of the external ontologies

Ontology	Number of requirements	Number of tests
ETSI SAREF	70	70
W3C SSN	24	24
OCF	27	27
oneM2M	33	33
ISO/IEC 30141:2017	36	36

Themis was also used to check if the VICINITY ontology network satisfies the test expressions defined for the requirements of these ontologies and standards. From the execution

[16]http://iot.linkeddata.es/def/wot
[17]https://git.onem2m.org/MAS/BaseOntology/raw/master/base_ontology.owl
[18]https://vicinity2020.eu/vicinity/
[19]http://w3c.github.io/wot/
[20]http://vicinity.iot.linkeddata.es/
[21]http://vicinity.iot.linkeddata.es/vicinity/testing.html
[22]https://w3id.org/saref
[23]http://www.w3.org/ns/ssn/
[24]https://openconnectivity.org/

of Themis, it could be deduced that the VICINITY ontology network satisfies several tests, but did not take into consideration some concepts related to these ontologies and standards, as the developers did not consider it necessary for the project domain definition. However, it could also be concluded that there were no conflicts between the VICINITY ontology network and these ontologies and standards. Therefore, it can be determined that the VICINITY ontologies, even though they do not satisfy all the requirements related to the anlysed IoT standards, are not incompatible with them. Table IV summarizes the results obtained after Themis execution. More information related to the results is available in the VICINITY ontology portal.[25]

TABLE VI
TESTING RESULTS OF THE EXECUTION OF THE STANDARDS' REQUIREMENTS IN VICINITY ONTOLOGY NETWORK

Standard	Number of tests	Undefined terms	Passed	Absent relation	Conflict
ETSI SAREF	70	66	4	0	0
W3C SSN	24	9	13	0	0
OCF	27	21	6	0	0
oneM2M	33	31	2	0	0
ISO/IEC 30141:2017	36	22	14	0	0

V. CONCLUSIONS AND FUTURE WORK

In this work we present Themis, a web-based tool to validate ontologies regarding their functional ontology requirements. To validate an ontology, Themis supports a set of different types of tests expressions, which are extracted from a collection of lexico-syntactic patterns with the aim of easing the formalization of requirements into tests cases. These lexico-syntactic patterns link requirements templates with possible implementations in an ontology and the test expressions check that these implementations are satisfied by a given ontology. Moreover, lexico-syntactic patterns use ontology design patterns to identify possible implementations. Therefore, Themis can also check if an ontology is satisfying a particular test following the ontology design patterns associated to the lexico-syntactic pattern.

Additionally, Themis allows to execute tests on a collection of ontologies simultaneously, which can be used to analyse the overlap of ontological commitment between them. This could be relevant if a developer requires to analyse whether a particular ontology is compliant with a given ontology or standard. The execution of tests on a collection of ontologies can also be used to ease the reuse of ontologies based on their requirements, due to the fact that the user is able to check which ontology satisfies the set of requirements he or she needs to cover.

As part of the continuous process of improving Themis, new tests can be added to the tool if new types of requirements or lexico-syntactic patterns are found. Furthermore, it is also planned to provide more types of results to the users, in addition to the four types that are provided so far. This addition of possible results aims to help the users to detect more accurately which is the problem when a test is not passed and, therefore, to ease the repairing task. Future work will be also directed to the development of a REST service to be integrated in other ontology engineering tools, such as OnToology.[26]

[25]http://vicinity.iot.linkeddata.es/vicinity/testing.html

[26]http://ontoology.linkeddata.es

VI. ACKNOWLEDGMENTS.

This work is partially supported by the H2020 project VICINITY: Open virtual neighbourhood network to connect intelligent buildings and smart objects (H2020-688467) and by a Predoctoral grant from the I+D+i program of the Universidad Politécnica de Madrid.

REFERENCES

[1] P. Hamill, *Unit test frameworks: tools for high-quality software development*. O'Reilly Media, Inc., 2004.

[2] M. Wynne, A. Hellesoy, and S. Tooke, *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2017.

[3] M. C. Suárez-Figueroa, A. Gómez-Pérez, and B. Villazón-Terrazas, "How to write and use the ontology requirements specification document," in *Proceedings of the International Conference on On the Move to Meaningful Internet Systems*, 2009, pp. 966–982.

[4] M. Grüninger and M. S. Fox, "Methodology for the Design and Evaluation of Ontologies," 1995.

[5] D. Vrandečić and A. Gangemi, "Unit tests for ontologies," in *Proceedings of the 2006 International Conference on On the Move to Meaningful Internet Systems*, 2006, pp. 1012–1020.

[6] S. Peroni, "A simplified agile methodology for ontology development," in *Proceedings of the OWL: Experiences and Directions Workshop and OWL reasoner evaluation workshop*, 2017, vol. 10161, p. 55.

[7] Y. Ren, A. Parvizi, C. Mellish, J. Z. Pan, K. Van Deemter, and R. Stevens, "Towards competency question-driven ontology authoring," in *Proceedings of the European Semantic Web Conference*, 2014, pp. 752–767.

[8] C. M. Keet and A. Ławrynowicz, "Test-Driven Development of ontologies," in *Proceedings of the International Semantic Web Conference*, 2016, pp. 642–657.

[9] S. García-Ramos, A. Otero, and M. Fernández-López, "OntologyTest: A tool to evaluate ontologies through tests defined by the user," in *Proceedings of the International Work-Conference on Artificial Neural Networks 2009*, 2009, pp. 91–98.

[10] E. Blomqvist, A. S. Sepour, and V. Presutti, "Ontology testing-methodology and tool," in *Proceedings of the Knowledge Engineering and Knowledge Management*, 2012, pp. 216–226.

[11] A. Lawrynowicz and C. M. Keet, "The TDDonto Tool for Test-Driven Development of DL Knowledge bases." in *Description Logics*, 2016.

[12] A. Fernández-Izquierdo and R. García-Castro, "Requirements behaviour analysis for ontology testing," in *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2018, pp. 114–130.

[13] G. Aguado De Cea, A. Gómez-Pérez, E. Montiel-Ponsoda, and M. C. Suárez-Figueroa, "Natural language-based approach for helping in the reuse of ontology design patterns," in *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2008, pp. 32–47.

[14] M. C. Suárez-Figueroa, S. Brockmans, A. Gangemi, A. Gómez-Pérez, J. Lehmann, H. Lewen, V. Presutti, and M. Sabou, "NeOn D5. 1.1: NeOn Modelling Components," 2007.

[15] A. Gangemi and V. Presutti, "Ontology design patterns," in *Handbook on ontologies*, 2009, pp. 221–243.

[16] SmartM2M, "SAREF extension investigation Technical Report (TR 103 411)."

[17] "ISO/IEC 30141:2017:Internet of Things (IoT) - Reference Architectures." International Organization for Standardization, Geneva, Switzerland, 2017. [Online]. Available: https://www.iso.org/standard/65695.html