# Towards a Visual SPARQL-DL Query Builder

Christian Gimenez†, Germán Braun†, Laura Cecchi†, and Pablo Fillottrani‡

†Universidad Nacional del Comahue
Buenos Aires 1400, Neuquén Capital, Argentina
‡Universidad Nacional del Sur
Avenida Alem 1253, Bahía Blanca, Buenos Aires, Argentina
{christian.gimenez,german.braun,lcecchi,}@fi.uncoma.edu.ar
prf@cs.uns.edu.ar

**Abstract.** Querying ontologies is an every-day activity that users need. This interaction will improve when the query is more expressive and easier to develop. For this purpose, a visual query language is an ideal mean for users and ontology engineers for creating queries taking advantage of the easy-to-understand and low time and cost characteristics, specially, for users which does not know textual query languages. On the other side, SPARQL-DL is a powerful and expressive textual query language for OWL-DL based ontologies that can combine TBox/ABox/RBox queries. Considering the advantage of both, we present in this work a visual query language that can be interpreted as SPARQL-DL sentences and thus being used for querying ontologies for its structure and/or instance information. Altogether, we use this idea to create a modified version of *crowd*, a Web modelling tool with reasoning support, that enables to implement and tests the presented graphical language along with the needed SPARQL-DL support for solving queries with the user's provided OWL 2 ontologies in any of its linearisations.

**Keywords:** Ontology, VQL, Semantic Web, SPARQL-DL, crowd

## 1 Introduction

The Semantic Web [1, 2] includes numerous ontologies that can be mixed semantically. These ontologies provide support for data and meaning for each knowledge base that an organisation can provide in its Web page. The usability of this information is key for these organisations which queries these data with no limitation in expressiveness. Specially, using both levels of DL: structural and assertional. Once created the ontologies, ontology engineers require a query formulation tool that easily express the user's information needs in terms of queries over ontologies. Examples of this are OptiqueVQS [3], OntoVQL [4], etc.

Not less important is to provide tools to users and domain experts for creating and accessing such information they need in a proper way. Querying using visual techniques provides a layer of abstractions that helps to understand easily the entire ontology. *crowd*[1] [5] is a Web tool for graphical ontology modelling created

[1] http://crowd.fi.uncoma.edu.ar

for achieving that purpose. It has been designed for being extendable to other visual language and for supporting reasoning services such as consistency checking. Nowadays, it provides support for standard conceptual modelling languages like UML and EER. Currently, ORM [6] and OVM [7] is under development. Also, another version of *crowd* has been released with support for the KF-metamodel [8, 9] which allows the user to create a model in one language, and through an intermediate representation which summarises common elements, creates a representation of the same model in EER, ORM or UML trying to preserve the semantics of the original.

SPARQL-DL is a query language (QL) for OWL-DL [10] ontologies significantly more expressive than existing DL QL by allowing combined ABox, RBox and TBox queries. It is aligned with SPARQL [11] to improve interoperability of applications on the Semantic Web and to allow to be covered by the standard reasoning services OWL-DL reasoners provide [12].

Aiming at integrating SPARQL-DL in an ontology engineering environment, this paper introduces a graphical language that preserves similarities to UML for creating visual SPARQL-DL queries. Furthermore, it presents a modification of the *crowd* architecture that allows visual SPARQL-DL queries. Implementing a query language in an already developed tool for creating ontologies via visual languages is not only the next logical step, but also a relevant one as its approach to ontology engineers mean to understand and study their ontologies and also help to create queries that their users require for satisfying their information needs. Moreover, it is possible to provide more reasoning services like anti-pattern based searches, in order to look for modelling issues in the ontology by obtaining a subset of the input concepts and rules and displaying them visually. However, a tool that supports both scenarios, design and query in a graphical way, and an implementation of a SPARQL-DL-based visual language is not available to our best of our knowledge.

The rest of the work is structured as follows: Section 2 introduces some well-known textual and visual query languages that has been researched through this work. Section 3 describes the UML-like language we propose for representing SPARQL-DL queries, the mapping rules and an example of query. Later, section 4 describes *crowd* architecture and the implementation needed for providing support to the proposed visual language and the reasoner for executing the queries. Section 5 shows possible applications of for this language. Finally, in section 6 summarises the paper and discuss future works.

## 2 Textual and Visual Query Languages

Textual query languages for ontologies provides a mean to retrieve information and structure for the user. The most common of these languages is OWLlink [13], the successor of DIG [14]. OWLlink provides a neutral mechanism for accessing OWL and OWL 2 reasoner functionality through a declarative interface whose protocol can be bound to a concrete transport mechanism like HTTP/1.1 and XML. OWLlink provides a wide spectrum of queries we can make, but they can

only affect the ABox, the TBox or the RBox separately. On the other hand, nRQL[15] is another DL query interface focused on retrieving ABox individuals that satisfy specific conditions which extends the expressivity of Racer [16] query language. However we cannot use it to retrieve structural definition from the ontology. Finally, SPARQL-DL is a powerful and expressive query language for OWL-DL that can combine ABox/TBox/RBox queries. It aligns with SPARQL and is simple enough to be easily built on top of existing OWL-DL reasoners [12].

In regard to visual query languages (VQL), we can find a variety of languages focused on querying OWL-DL ontologies. They present different ways of querying an ontology, but they does not provide a SPARQL-DL representation nor match its expressiveness. To our knowledge, there are no visual language created in base of SPARQL-DL publicly available.

In OntoVQL [4], queries are modelled by a graph connected style where concepts and individuals are nodes and roles are drawn by edges. Operators as "and" and "or" can be used for connecting more than one sub-query. Similarly to others visual languages, like SEWASIE, GRQL and OZONE, all these provides expressiveness enough for querying about the ABox only through a database or a nRQL query mapping.

GrOWL-Query [17] allows us to query about an ontology by representing the underlying DL semantics of OWL ontologies and replacing individuals by variables. Knowledge about OWL and DL is needed for using this language augmenting the learning curve.

A visual query system like OptiqueVQS also provides a visual language along with different techniques that facilitates the query formulation. The ability to query the ontology through SPARQL only allows OptiqueVQS to query about the ABox.

## 3 A UML-like Graphical Language for SPARQL-DL

A UML-like language is selected for preserving the objective of being accessible to the user and easy to understand. This section describes a UML-like language and a mapping from some of its graphical primitives to a SPARQL-DL scheme, which allows us to compute the query.

### 3.1 A UML-Like Visual Query Language

We define a UML-like visual language in which classes, associations, generalisations are primitives that represents concepts, roles and inclusion in DL, respectively. Objects in UML are depicted similarly as classes, where the name is combined with the instance name, a colon (":") and the Classifier name, all of them underlined [18]. This primitive is for representing ABox instances in DL, providing a way for querying about both TBox and ABox.

Associations can have names which we can use for representing roles names but multiplicity is not considered in this preliminary version. Class or instances

classifier names are used as concepts names in the DL representation. All names can be replaced with variables for requesting a set of answers that matches any element in the TBox and/or ABox that meets the UML diagram representation. Variables are written prefixed with a question mark. Depicting an UML primitive named without a variable is considering as asking for existence of that element in the input ontology. For example, an UML class named "Project" will represent the concept Project in the ontology and the output will be asserted as true if founded in the input TBox or false otherwise.

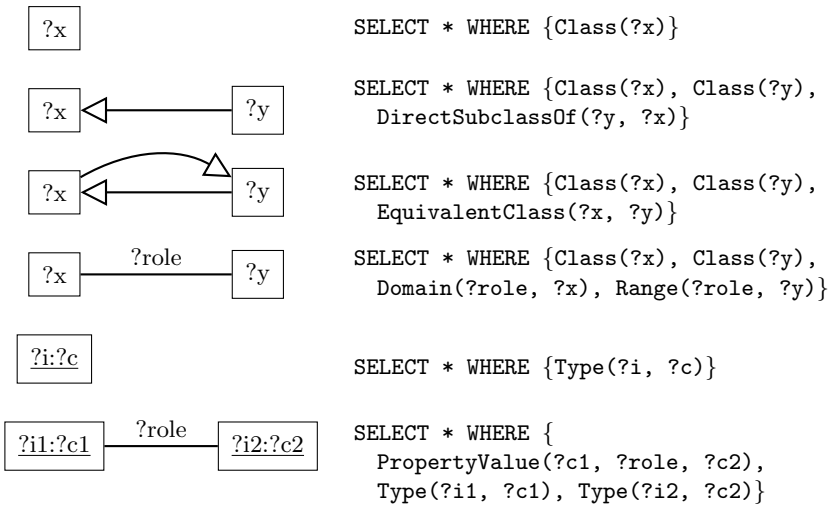


**Fig. 1.** Summary of the UML primitives mapping to SPARQL-DL.

### 3.2 SPARQL-DL Encoding

SPARQL-DL is used as a textual language for representing queries in *crowd* VQL. This language posses two types of queries: ASK sentence for querying about existence or assertions, which returns a true/false answer and SELECT queries for retrieving a set of elements that matches the provided condition.

On Figure 1, a summary of UML primitives with variables are depicted with their respective SPARQL-DL encoding. Using variables in the graphical query is considered as a query for retrieving a set of elements that matches the user's condition, and thus, we use a SELECT sentence. The conditions are determined by the query atoms, which are constructed depending on the primitives used in the diagram. Classes are interpreted as `Class()` in SPARQL-DL, allowing the engine to search for concepts in the input. A generalisation requires the representation in SPARQL-DL of the two classes or objects involved and a `DirectSubclassOf()` sentence for searching a direct subclass relation between these two elements. For

equivalence, the sentence `EquivalentClass()` is used. Associations are mapped into a `Domain()` and `Range()` query atoms for requesting the classes involved in the role.

Instances in the ABox can be queried by the objects UML representation, where we can use the name of the instance and class in a `Type(?i, ?c)` SPARQL-DL sentence which requests the instance "?i" of a concept "?c". Roles also can be mapped by a combination of three query atoms: a `PropertyValue()` and two `Type()`, the first one for searching the classes involved in the role, and the rest for retrieving the instance that can participate on it.

ASK queries are used when the graphical query provides names without variables. In this case, they are represented similarly as the SELECT queries but providing the name string.

The SPARQL-DL encoding order which we currently support emphasise the primitives related to querying the TBox. First, classes are mapped, then generalisations and finally associations. Next, primitives related to the ABox are considered: objects, generalisations between objects and roles (associations between objects).

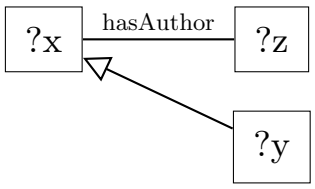


**Fig. 2.** A query expressed in UML class diagram model.

### 3.3 Example UML to SPARQL-DL

In the example presented on Figure 2 the user creates a model which requests all classes involved in the model, where a class "X" is in relation named `hasAuthor` with another class "Z" and is a parent of class "Y". The conversion of this model to a SPARQL-DL sentence will start by creating a query atom for each class and then for each relation. Then, it will detect the use of variables and create a SELECT statement with all of them. The sentence generated will be as follows:

```
SELECT ?x,?z,?y WHERE
{
  Class(?x), Class(?z), Class(?y), DirectSubclassOf(?y,?x),
  Domain(?x,:hasAuthor), Range(?z,:hasAuthor)
}
```

Afterwards, the SPARQL-DL reasoner will be used for processing the query with the user ontology. All the results will be displayed on the user interface.

## 4 *crowd* for VQL

The *crowd* architecture, depicted in Figure 3, consists in a client-server style where the client provides a Web interface following the view-model architecture provided by the BackboneJS and JointJS libraries. This allows the Javascript interface to generate a JSON representation of the user's model for sending to the server. The server side provides reasoning services while creating a DL encod-

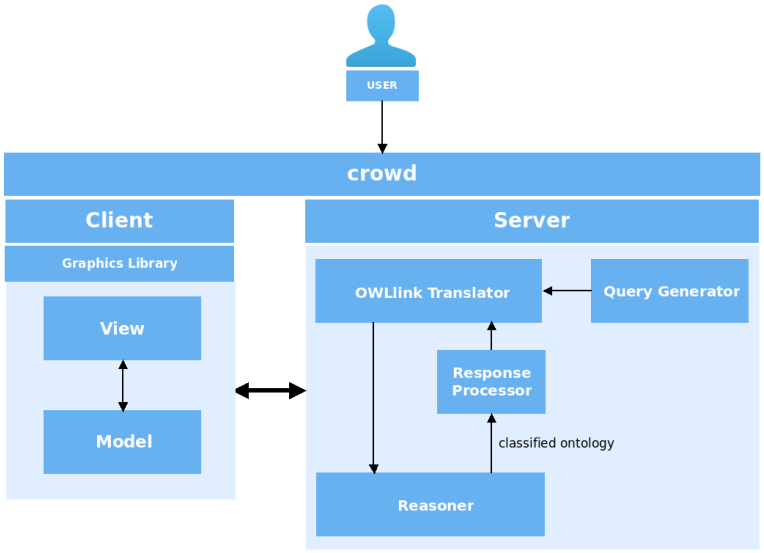


**Fig. 3.** *crowd* architecture.

ing from the JSON input. The module responsible for this is called "OWLlink Translator", which first creates an ontology in OWL 2 according to the method described by [19, 20]. For providing consistency checking services, a "Query Generator" module defines the necessary queries for the reasoner depending on the user's model and appends them to the OWLlink [13] document with the OWL 2 ontology already generated. The "Reasoner" module executes and maintains the program which use the OWLlink document as input, elaborating the answer which will be interpreted by the "Response Processor" creating a JSON understandable by the interface.

In Figure 4, the overall visual query processing is depicted. The visual query model designed by the user and the ontology are imported from the front-end interface. Both elements are sent to the back-end, the first in JSON format for creating the SPARQL-DL encoding in the SPARQL-DL Processor. Finally, the Reasoner receives both inputs, the ontology in OWL 2 syntax and the query, processes them and generates the output results in JSON or XML which can be used for sending back to the interface for displaying to the user.

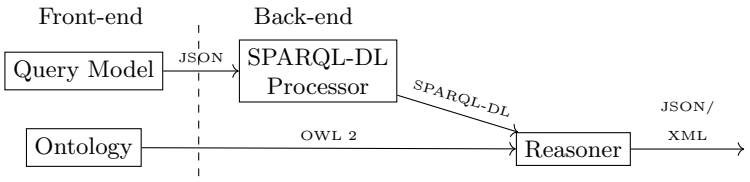


**Fig. 4.** The back-end steps for processing the query modelled by the user.

For implementing this process, some features has to be added to the initial version of *crowd*. The mapping rules described before were implemented for creating the SPARQL-DL encoding. The SPARQL-DL reasoner is called similarly as the current one with its own parameters and input files. Finally, the output is captured and processed by an extended version of the Response Processor for sending to the interface for the user consumption.

Currently, the graphical expressiveness of *crowd* VQL is constrained to UML classes, instances, generalisations and binary associations. Remains the rest of UML class diagram representation and SPARQL-DL encoding for future development.

## 5 Applications

Visual techniques for accessing data facilitate the design of queries for users and developers, reducing the learning curve and also abstract the complexity of the underlying logical query language. Moreover, it is possible to use the UML-like syntax for searching ontology patterns [21] and anti-patterns [22, 23] represented in a visual context and thus helping the engineer to raise the quality of the ontology design. In the same direction, other applications are related to the bottom-up construction of ontologies by means of non-standard services such as last common subsumers and most specific concepts [24]. Lastly, reasoners can also search for a possible pattern in a top-down way by querying the existence of part of the pattern into the input ontology, and thus suggesting the missing elements which completes them [25, 26]. For example, consider the pattern "List 2" presented in Figure 5. Part of the pattern is queried by using the UML class diagram depicted into the input ontology. The missing elements of the pattern can be suggested to the user for appending to the input ontology according to the results of the query.

Searching for anti-patterns represented in a graphical context can avoid situations that normally result in inconsistencies. This anti-patterns can be imported or drawn as a query in a UML-like diagram, making it easier to the user to remember and to understand. For example, searching for the logical anti-pattern called "Synonym or Equivalence" (SOE) explained in [28] requires to look into the ontology for the expression $C1 \equiv C2$, which latter should be treated by the user removing C2 according to the pattern solution. The Figure 6 displays a UML representation for the expression.

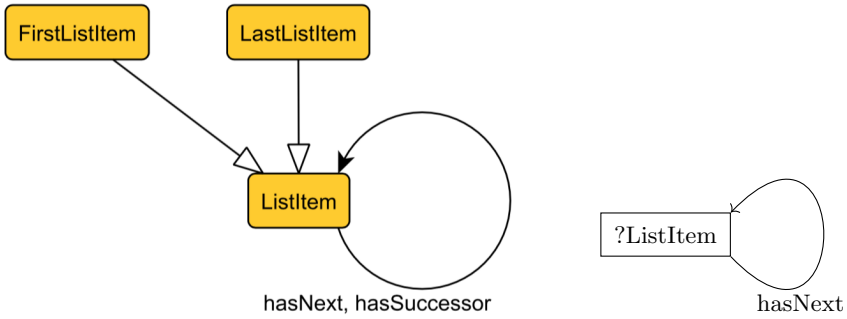


**Fig. 5.** List pattern obtained from [27] and the UML-like query used for searching the pattern.

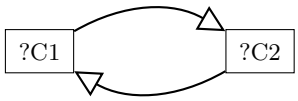


**Fig. 6.** UML-like diagram for querying the expression $C1 \equiv C2$.

## 6 Conclusions and Future Works

In this work, we proposed a graphical visual query language using an UML-like notation in order to keep the consistency with the standard conceptual modelling languages and reduce the curve of learning drastically. As an intermediate language, we propose the use of SPARQL-DL a textual language, whose expressiveness is greater than the current implemented, like OWLlink, nRQL and DIG, by mixing the ABox, RBox and TBox in the same query. Moreover, we modified our Web tool named *crowd* for supporting this graphical language and the import of ontologies in OWL-DL, creating an environment suitable for designing the query and testing it with an ontology. The back-end has been adapted for supporting the encoding of the graphical models with the SPARQL-DL queries and for executing it with the provided OWL 2 ontology. A running prototype is available following URL `http://crowd.fi.uncoma.edu.ar/sparqldl/`.

In future works, we expect to extend *crowd* with an interface for displaying the results of the reasoning system in a tabular and graphical view instead of the textual output. The rest of the UML class diagrams primitives not presented in this work are considered for next releases along with their SPARQL-DL encoding. Also, an expansion of this conceptual modelling language for supporting more SPARQL-DL query atoms are under study for future implementations. Finally,

the inclusion of Object Constraint Language (OCL) [29] for augmenting the expressiveness of the query is considered for future study.

## References


1. Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer, 2Nd Edition (Cooperative Information Systems)*. The MIT Press, 2 edition, 2008.
2. Dean Allemang and James A. Hendler. *Semantic Web for the working ontologist effective modeling in RDFS and OWL*. Morgan Kaufmann/Elsevier, Waltham, MA, 2011.
3. Ahmet Soylu, Evgeny Kharlamov, Dimitry Zheleznyakov, Ernesto Jimenez Ruiz, Martin Giese, Martin G. Skjaeveland, Dag Hovland, Rudolf Schlatte, Sebastian Brandt, Hallstein Lie, and Ian Horrocks. OptiqueVQS: a visual query system over ontologies for industry. *Semantic Web*, (in press), 2017.
4. Amineh Fadhil. Ontovql: Ontology visual query language. Master's thesis, Concordia University Montreal, Quebec, Canada, September 2008.
5. Christian Gimenez, Germán Braun, Laura Cecchi, and Pablo Fillottrani. crowd: A Tool for Conceptual Modelling assisted by Automated Reasoning - Preliminary Report. In *the 2nd Simposio Argentino de Ontologías y sus Aplicaciones SAOA '16 JAIIO '16*, 2016.
6. Federico Solorza, Germán Braun, Laura Cecchi, and Pablo Fillottrani. Hacia la Formalización de un Lenguaje Visual Unificador de UML, EER y ORM2. In *Workshop de Investigadores en Ciencias de la Computación 2018*, 2018.
7. Ángela Oyarzun, Germán Braun, Laura Cecchi, and Pablo Fillottrani. Una Herramienta Gráfica con Razonamiento basado en DL para el Análisis de Modelos OVM. In *Workshop de Investigadores en Ciencias de la Computación 2018*, 2018.
8. C. Maria Keet and Pablo Rubén Fillottrani. An ontology-driven unifying metamodel of UML Class Diagrams, EER, and ORM2. *Data & Knowledge Engineering*, 2015.
9. Pablo R. Fillottrani and C. Maria Keet. KF metamodel formalization. *CoRR*, abs/1412.6545, 2014.
10. Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, December 2012. Available at `http://www.w3.org/TR/owl2-primer/` last visited July 2018 (Internet archive `http://archive.today/H7G0H`).
11. W3C OWL Working Group. *SPARQL 1.1 Query Language*. W3C Recommendation, 21 March 2013. Available at `http://www.w3.org/TR/sparql11-query/`.
12. Ilianna Kollia, Birte Glimm, and Ian Horrocks. SPARQL query answering over OWL ontologies. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *The Semantic Web: Research and Applications*, pages 382–396, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
13. Thorsten Liebig, Marko Luther, Olaf Noppens, and Michael Wessel. Owllink. *Semantic Web*, 2(1):23–32, 2011.
14. S. Bechhofer, R. Moller, and P. Crowther. The DIG Description Logic Interface. In *In Proc. of International Workshop on Description Logics (DL2003)*, 2003.
15. Volker Haarslev, Ralf Möller, and Michael Wessel. Querying the semantic web with Racer + nRQL. In *In Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL'04*, 2004.

16. V. Haarslev and R. Möller. Racer system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy*, pages 701–705. Springer-Verlag, 2001.
17. Sergey Krivov, Richard Williams, and Ferdinando Villa. Growl: A tool for visualization and editing of OWL ontologies. *J. Web Sem.*, 5(2):54–57, 2007.
18. Object Management Group (OMG). OMG unified modelling language version 2.5, 2015.
19. Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams. *Artif. Intell.*, 168(1-2):70–118, 2005.
20. Germán Alejandro Braun, Laura Andrea Cecchi, and Pablo R. Fillottrani. Integrating graphical support with reasoning in a methodology for ontology evolution. In *Proceedings of the Joint Ontology Workshops 2015 Episode 1: The Argentine Winter of Ontology co-located with the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, July 25-27, 2015.*, 2015.
21. Steffen Staab and Rudi Studer. *Handbook on Ontologies*. Springer Publishing Company, Incorporated, 2nd edition, 2009.
22. Tiago Prince Sales and Giancarlo Guizzardi. Ontological anti-patterns. *Data Knowl. Eng.*, 99(C):72–104, September 2015.
23. Catherine Roussey, Oscar Corcho, and Luis Manuel Vilches-Blázquez. A catalogue of owl ontology antipatterns. In *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP '09, pages 205–206, New York, NY, USA, 2009. ACM.
24. Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, 2003.
25. Pablo Rubén Fillottrani and C. Maria Keet. Patterns for heterogeneous TBox mappings to bridge different modelling decisions. In Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rinke Hoekstra, Pascal Hitzler, and Olaf Hartig, editors, *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I*, volume 10249 of *Lecture Notes in Computer Science*, pages 371–386, 2017.
26. Germán Alejandro Braun and Laura Andrea Cecchi. Extension rules for ontology evolution within a conceptual modelling tool. In *Proceedings of the 1st Argentine Symposium on Ontologies and their Applications co-located with 44 Jornadas Argentinas de Informática (44JAIIO), Rosario, Argentina, September 2-3, 2015.*, 2015.
27. David Carral, Pascal Hitzler, Hilmar Lapp, and Sebastian Rudolph. List 2. `http://ontologydesignpatterns.org/wiki/Submissions:List_2` visited on July 2018 (Internet archive `http://archive.today/vOHit`).
28. Catherine Roussey and Oscar Corcho. Synonym or equivalence. `http://ontologydesignpatterns.org/wiki/Submissions:SynonymOrEquivalence_(SOE)` visited on July 2018 (Internet archive `http://archive.today/OO8AI`).
29. Object Management Group. *Object Constraint Language (Version 2.4)*. Pearson Higher Education, 2014.