Ontology-Based Spatial Pattern Recognition in Diagrams

Anitta Thomas[1], Aurona J. Gerber[2,3], and Alta van der Merwe[2]

[1] School of Computing, University of South Africa
The Science Campus, Florida Park, South Africa
[2] Department of Informatics, University of Pretoria
Pretoria, South Africa
[3] Center for Artificial Intelligence Research (CAIR), CSIR Meraka
Pretoria, South Africa
thomaa@unisa.ac.za,
{aurona.gerber,alta.vdm}@up.ac.za

Abstract. Diagrams are widely used in our day to day communication. A knowledge of the spatial patterns used in diagrams is essential to *read* and *understand* them. In the context of diagrams, spatial patterns mean accepted spatial arrangements of graphical and textual elements used to represent diagram-specific concepts. In order to assist with the automated understanding of diagrams by computer applications, this paper presents an ontology-based approach to recognise diagram-specific concepts from the spatial patterns in diagrams. Specifically, relevant spatial patterns of diagrams are encoded in an ontology, and the automated instance classification feature of the ontology reasoners is utilised to map spatial patterns to diagram-specific concepts depicted in a diagram. A prototype of this approach to support automated recognition of UML and domain concepts from class diagrams and its performance are also discussed in this paper. This paper concludes with a reflection of the strengths and limitations of the proposed approach.

Keywords: Spatial Patterns, Diagrams, Ontology, Ontology Reasoner, OWL, UML Class Diagrams, SVG

1 Introduction

Diagrams are widely used in our day to day communication, and different types of diagrams are used for various purposes in different domains. Prominent diagrams used in Computing include flow charts, state chart diagrams and Unified Modelling Language (UML) diagrams. A noticeable aspect of these diagrams is that they have generally accepted visual syntax, which are spatial patterns consisting of graphical and textual elements [15, 16] to indicate diagram-specific concepts in the diagrams. This study only considers diagrams that have established visual syntax irrespective of whether it is formally or informally specified. The knowledge of these visual patterns is essential to *read* and *understand* diagrams.

The final publication is available at Springer
via http://dx.doi.org/10.1007/978-3-319-92007-8_6

Numerous approaches have been used to generate declarative specifications of various aspects of diagrams including their spatial patterns, and these specifications have been used to implement applications to either recognise or generate diagrams. The majority of these approaches are grammar-based approaches, and the prominence of these approaches stems from their success in natural language specifications [15]. However, the number of diagram specification techniques that have sufficient supporting literature, freely-available tools and literature that indicates their current use remains small.

Ontologies is a prominent knowledge representation technique, which has been successfully used to realise knowledge-based applications. An ontology, in general, is a model of domain knowledge designed for a specific purpose. Ontologies are used, in general, for two main purposes; as a formal vocabulary of concepts in a domain or as a knowledge source from which additional knowledge can be derived. Logic-based ontologies have the benefit of being able to use the automated reasoning capabilities of existing ontology reasoners to verify and infer additional information from them [5]. The Web Ontology Language (OWL), a logic-based ontology language, has a rich set of current literature, proven value in numerous application domains and freely-available tools [7, 11].

In this paper, we present the results of a study that developed and evaluated an ontology-based approach to recognise spatial patterns in diagrams. In this approach, relevant spatial patterns of diagrams are encoded in an OWL ontology, and the automated instance classification feature of the ontology reasoners is utilised to map spatial patterns to diagram-specific concepts depicted in a diagram. The proposed approach is then instantiated in a prototype to recognise concepts from Scalable Vector Graphics (SVG) files of UML class diagrams using an ontology of relevant spatial patterns. Furthermore, the performance of the prototype in terms of its time usage and correctness is tested with a sample set of class diagrams. Automated recognition of concepts from diagrams can be a useful building block to generate automated summaries of diagrams, which can, for example, be useful for visually impaired users in reading the information depicted in diagrams.

The novel contributions of this study are three-fold; firstly a generic ontology-based approach to recognise diagram-specific concepts from diagrams is presented. Secondly an implementation, albeit a prototype, demonstrates the feasibility of the proposed approach, and thirdly, automated semantic interpretation of limited scope of class diagrams in SVG format using an ontology of spatial patterns is demonstrated.

This paper is structured into seven sections. In section 2, the proposed ontology-based approach and a generic application design that uses this approach are presented. A prototype that follows this ontology-based approach to recognise UML and domain concepts in class diagrams of SVG format is presented in section 3. Section 4 presents a discussion on the performance of the prototype. In Section 5, a discussion of the pros and cons of the proposed ontology-based approach is included. Section 6 includes a brief summary of related work and a conclusion is provided in section 7.

2 Ontology-based Spatial Pattern Recognition

Diagrams with established visual syntax consist of well-defined spatial patterns for diagram-specific concepts. For example, Figure 1 depicts (a) a *decision* (b) a *state* (c) a *class* used in flow charts, state diagrams and UML class diagrams respectively. In this example, *decision*, *state* and *class* are domain-specific concepts, which are represented using established spatial patterns, which consist of spatial configurations of graphical and textual elements. For example, a *state* in state diagrams can be represented using a circle and a text, where the text is contained in the circle. Additionally the text in the circle is generally considered as the description of the *state*, so in this case the string *A* will be read as *state A*.

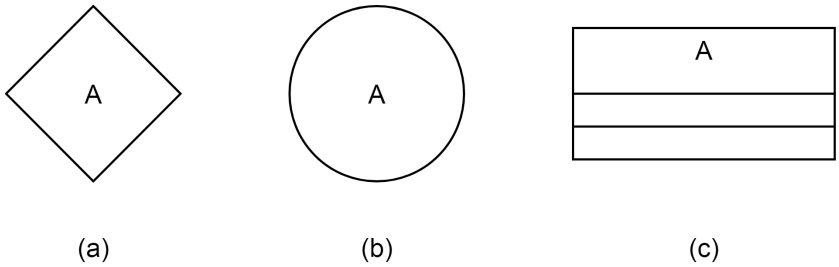

Fig. 1. Notations of (a) a *decision* (b) a *state* (c) a *class* used in flow charts, state diagrams and UML class diagrams respectively

In our proposed approach, recognising domain-concepts in diagrams is achieved using an ontology of visual patterns. The first step in this approach is to create a ontology of visual patterns of diagrams. The model of visual patterns in the ontology should be aligned with the goals of the applications in which it will be used. An application for diagram interpretation that uses this ontology can then be realised using the generic design depicted in Figure 2. In the generic design, the component *Diagram processor* will process the diagram to extract graphical and textual primitives as well as the spatial relationships between these primitives, which is then amended to the ontology of visual patterns. The *ontology reasoner* can then process, using instance classification, this amended ontology to indicate the diagram-specific concepts depicted in the diagram. In Section 3, the generic design given in Figure 2 is modified to implement a prototype that extracts UML and domain concepts from SVG files of class diagrams.

3 Prototype: Automated Extraction of UML concepts from class diagrams in SVG format

The aim of the prototype is to realise automated interpretation of a limited form of class diagrams using a relevant spatial patterns ontology. The file format of

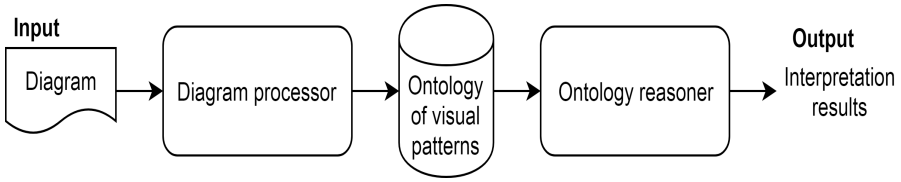

Fig. 2. The components of a generic diagram interpretation application that uses the spatial patterns ontology

the class diagrams is restricted to SVG. The scope of interpretation is limited to extract UML and domain concepts depicted in a given diagram. The concepts of interest are the nodes and links in the diagrams. Moreover, when links are identified in the diagram, the prototype should also extract the connecting nodes for each link. Examples of UML concepts that represent nodes are *class*es and *interface*s. On the other hand, *inheritance* and *dependency* are examples of concepts that are considered links in class diagrams.

A class diagram, in general, is a UML model that models a system or some aspect within some domain. The metamodel of class diagrams is defined in UML [18]. In the developed prototype, the concepts extracted automatically from class diagrams include elements from both the domain model and UML metamodel. For example, referring to Figure 1(C), the notation can be seen as the UML concept (metaclass) *Class*, and the string *A*, representing the class name, can be seen as a concept from the domain model. The proposed approach of automated extraction of concepts is relevant when class diagrams do not contain explicit representations of models, for example, in the XML metadata interchange format XMI [24]. Any non-UML complaint tool, for example a general-purpose free sketching application, can generate class diagrams without explicit model information. One can find numerous class diagrams on web pages that are only available as images without explicit model information.

3.1 An Ontology of Spatial Patterns for Class Diagrams

The central component of the prototype is a dedicated ontology of spatial patterns to support recognition of concepts from class diagrams. In order to develop the ontology, decisions had to be made about how to model the spatial patterns, how to indicate which spatial patterns correspond to which UML concepts and which relevant spatial patterns need to be encoded in the ontology.

UML class diagrams are specified in the Classes package of the UML 2.4.1 specification [17]. This package includes fifty six concepts that can be used to represent an object oriented model in class, package and object diagrams. Moreover, a concept can have more than one spatial pattern. In other words, a concept can be illustrated in different ways. As the developed ontology is to be tested with a prototype, a finite set of UML concepts and spatial patterns was selected to be included in the ontology. This selection was based on which concepts and

notations are typically used in class diagrams and also discussed in UML 2.4.1 specification [17].

The spatial patterns were modelled as spatial configurations of primitive elements and spatial relationships between these elements. This modelling approach was used because it aligns with the OWL constructs, classes and properties [9], where an OWL class represents a set of objects and a property describes a possible relationship between objects [8]. The developed ontology consists of numerous OWL classes and they are used to represent either a primitive element or a UML concept. For example, the graphical object rectangle is a graphical primitive used in class diagrams, so an OWL class named Rectangle is added in the ontology to represent all instances of rectangles in class diagrams. A set of spatial relationships were also selected, which were then added as properties in the developed ontology. The spatial pattern for a UML concept was then modelled as the class definition of the corresponding OWL class. If the selected UML concept had more than one spatial pattern, the OWL class corresponding to the concept encoded multiple spatial patterns. The OWL class (corresponding to UML concepts) definitions encoded spatial patterns in terms of existing OWL classes (corresponding to UML concepts and primitive elements) and properties (corresponding to spatial relationships).

Due to space limitations, the entire ontology cannot be included in this paper. Interested readers may refer to [21, 22] for details about the ontology. For illustrative purposes, consider the UML concept *class*, which has numerous spatial patterns - Figure 3 depicts a few of them. In the developed ontology an OWL class, named *UMLClass* is created with its class definition encoding the spatial patterns of the UML concept *class*. The OWL class definition of *UMLClass* that contains the spatial pattern is listed below:

```
Class: UMLClass
EquivalentTo:
(Rectangle and (hasNonTangentialProperPartOf some String))
or
(Rectangle and (hasNonTangentialProperPartOf some String)
           and (hasTangentialProperPartOf some Line))
```

In the OWL code above, *Rectangle*, *Line* and *String* are the OWL classes in the ontology to represent graphical primitives rectangle and line, and textual primitive (one line of text) respectively. *hasNonTangentialProperPartOf* and *hasTangentialProperPartOf* are properties (corresponding to spatial relationships) encoded in the ontology, and they correspond to the spatial relationships $NTPP^{-1}$ and TPP^{-1} of RCC-8 [2] respectively. The spatial pattern encoded in the ontology for the concept class takes into account its different spatial patterns, especially those depicted in Figure 3.

3.2 Prototype Design and Implementation

The prototype achieves automated interpretation of class diagrams in several steps. The first step involves extracting primitives and spatial relationships be-

tween the primitives from an input diagram, which is used to generate OWL code specific for the input diagram. In the second step, the generated OWL code is added to the ontology described in Section 3.1. In the third step, the ontology reasoner performs instance classification on the updated ontology. In the fourth step, the results of the instance classification and the reasoner justifications for the results of instance classification are processed to extract the desired interpretation for the given diagram.

The prototype has a design with three main components namely *SVG processor*, *OWL code generator and writer* and *Interpretation processor* to realise automated interpretation of class diagrams. The inputs to the prototype are class diagrams in SVG format and the ontology described in Section 3.1, and the final output is the interpretation, which in this case is a list of nodes and links in the diagram. The overall design of the prototype is given in Figure 4. The functionality of each component is described below.

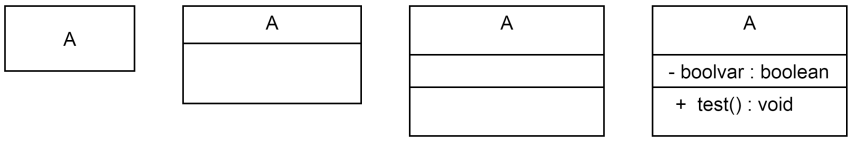

Fig. 3. Different depictions of the UML concept *class*

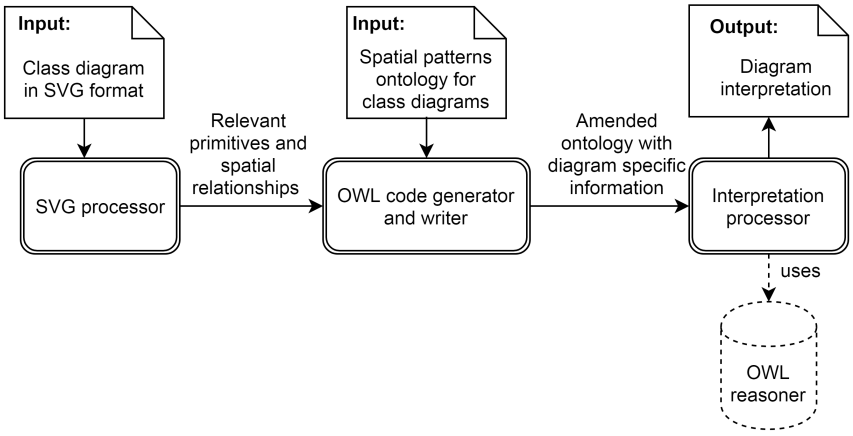

Fig. 4. Prototype design

SVG processor: The aim of this component is to extract relevant graphical and textual primitives, and the spatial relationships between these primitives in the input class diagram. Relevant primitives are graphical and textual elements visible to the user and also specified in the ontology. When extracting relevant graphical and textual primitives, their relevant geometric features are also extracted in order to calculate relevant spatial relationships specified in the ontology.

Different SVG files can generate visibly same image, which is attributed to different ways to encode graphics in SVG. Due to differences in SVG modelling, the models of SVG were restricted for this prototype. However, the chosen models require processing, both XML and geometrical, of SVG elements to determine visible graphical primitives and the spatial relationships between the primitives.

OWL code generator and writer: The *OWL code generator and writer* component consists of two sub-components; one to generate OWL code relating to the input diagram and the second one to write this generated OWL code to the ontology described in Section 3.1. The output from *SVG processor* is used to create the OWL code, which includes unique names, types of all the primitives and spatial relationships between them. The generated OWL code is then written to a copy of the .owl file containing the visual patterns ontology, which means each input diagram has a unique ontology file.

Interpretation processor: *Interpretation processor* invokes the ontology reasoner with the visual patterns ontology output of *OWL code generator and writer* to produce the interpretation. There are two steps taken by *Interpretation processor* to obtain the desired interpretation.

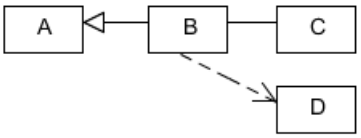

Fig. 5. A simple class diagram

The first step is to use the instance classification feature of the reasoner to obtain the instances of OWL classes that represent relevant UML concepts in the ontology. Specifically, lists of all primitives encoded in the ontology that are classified as instances of OWL classes of interest are obtained. For example, *Interpretation processor* obtains the list of all rectangles that are classified as instances of the OWL class *UMLClass* meaning that these rectangles form part of spatial patterns that represent the UML concept *class*.

In the second step, *Interpretation processor* processes the *justifications* of the results of instance classification from the first step to obtain relevant *node*

pairs that satisfy relevant *links*. In particular, *justifications* of the results of instance classification of OWL classes that correspond to only UML concepts that represent *links* are selected for further processing. This selection is because UML concepts that represent *links* are modelled in terms of *nodes* in the class diagrams. For example, the spatial pattern encoded for the UML concept inheritance consists of graphical primitives, triangle and line(s), as well as *node*(s) (examples of nodes include classes and interfaces). If the reasoner inferred that an instance of OWL class *triangle* is an instance of the OWL class *inheritance*, the *justifications* for this instance classification would contain the information about the *node*(s) that led to this inference. Thus by processing these *justifications* one can extract the node pairs that satisfy relationship links.

Input and output: The inputs to the prototype are class diagrams in SVG format, and the spatial patterns ontology for class diagrams. An intermediate output of the prototype is a modified spatial patterns ontology that contains input diagram-specific information in addition to the spatial patterns. The final output is a textual summary of the interpretation, i.e. lists of all relevant nodes and links in the input diagram.

To illustrate the output produced by the prototype, consider the simple class diagram depicted in Figure 5. The textual interpretation produced for this diagram is $Class[A, B, C, D]$, $Inheritance[(A, B)]$, $Association[(B, C)]$ and $Dependency[(D, B)]$. Note that the textual interpretation contains a summary of the domain model in terms of UML concepts; A, B, C and D refer to domain concepts and $Class$, $Inheritance$, $Association$ and $Dependency$ refer to UML concepts.

Implementation: *SVG processor* and *OWL code generator and writer* are written in C++ without the use of any external libraries. *Interpretation processor* is written in Java because the OWL API is a Java API [7], and it makes use of the OWL API version 3.5.0 and the OWL reasoner HermiT [6]. Additionally, *Interpretation processor* makes use of the clarkparsia library to obtain *justifications* from the OWL reasoner.

4 Prototype Performance

An initial evaluation of the prototype was conducted using fifteen class diagrams. The SVG file sizes of the class diagrams varied from 2 kB to 59 kB. As expected the larger files have larger number of SVG elements and graphical representations in the diagrams than smaller files. The number of XML elements varied from 15 to 640 in the sample SVG files and the number of actual graphical and textual primitives visible in the diagrams varied from 2 to 205.

For the initial evaluation, precision, recall and processing times of the prototype for the sample class diagrams were noted. The average precision and recall of the final interpretation results were 0.95 and 0.98 respectively. However, the

lowest precision of 0.78 and recall of 0.76 were recorded for certain class diagrams. The main cause for mistakes in diagram interpretation was incorrect modelling of spatial patterns for certain UML concepts, which led to certain UML concepts identified incorrectly as well as certain UML concepts not identified in the diagrams.

The average time used by the prototype was under a minute for the sample class diagrams. However, the longest time taken was approximately 7 minutes for the sample set. Based on the times recorded for each process in the prototype, it was noted that accessing justifications from the reasoner using the clarkparsia library was the most time-intensive process, taking up to an average of 97% of the total time used by the prototype.

The prototype demonstrates high precision and recall, which is highly desirable even though the sample set was relatively small. However, the time taken for accessing justifications of reasoner seems unreasonably high in comparison to the times taken by other processes in the prototype. Overall, the performance of the prototype is deemed satisfactory and it encourages further evaluation with larger number of diagrams, and improvements to the prototype. Improvements to the prototype will include modifications to the spatial patterns ontology to improve precision and recall, as well as the use of other available libraries to obtain justifications from the ontology reasoner to improve its time usage.

5 Discussion

The ontology-based approach presented in this paper can be applied to any type of diagrams with well-established spatial patterns, and thus the design in Figure 2 can be reused in other diagram interpretation applications. This paper also showed an example of how the general design in Figure 2 was adapted and instantiated in an interpretation application for class diagrams with promising results.

It should be noted that the scope of interpretation of diagrams using this approach is dependant on how the spatial patterns are modelled in the ontology. Similarly, the content of the spatial patterns ontology influences all the processes in the design in Figure 2. For example, which primitives should be extracted in *Diagram processor* is directed by the model encoded in the ontology. This tight coupling between the spatial patterns ontology and all the components in the general design may limit the reusability of program codes from one diagram interpretation application to another.

Diagram processor tries to extract graphical and textual primitives from the input diagram, which is indeed a form of mapping from raster or scalar data to semantic concepts of graphical shapes and text. Thus it may be possible to use a different type of ontology to support the functionality of *Diagram processor*.

Use of *justifications* from the ontology reasoner seems to be a feasible way to obtain necessary information for the diagram interpretation scope of the developed prototype. However, it is possible that using *justifications* may not work for other diagram interpretation scopes. Although processing *justifications* was

a bottle neck in the prototype, there are other ways to access *justifications* from the reasoners. One possibility is to use the Protégé [19] API instead of clarkparsia library to access the *justifications*.

Currently the prototype provides a textual summary of the class diagram. However, it is possible to format the summary in XMI since it is the recommended format for model exchange among UML tools [18].

6 Related Work

In [13], an ontology-based approach to recognise design patterns in program codes is proposed, where the ontology consists of models of design patterns and the reasoner is used to identify design patterns from an input program. The approaches in [13] and in this paper are similar but the difference is that the former is applied to the analysis of program codes in text and the latter is applied to the analysis of diagrams.

In [20], a framework to realise domain-based Resource Description Framework (RDF) annotation of SVG images is presented. In this framework, the SVG image is initially processed to extract relevant graphical features, which is used to create a layer of RDF data. In order to provide semantic annotation, the application uses a RDF database relevant for the domain in order to compute similarities between the given image and the semantic concepts in the database. The semantic annotation generated by the application is also checked by a domain expert, which is fed back into the RDF domain database. Comparing the prototype discussed in this paper and the annotation framework presented in [20], it is evident that both approaches use declarative domain knowledge to assist with the interpretation of SVG images. However, the approach in this work is focussed on spatial patterns in diagrams, and how an ontology of such patterns can be used for diagram interpretation.

Ontology-based approaches have been successfully used in the field of image analysis - see examples of such studies in [3, 10, 14]. In the field of image analysis, single to multiple ontologies are used in an application, and what is modelled in the ontology differs from domain knowledge to generic knowledge relevant for describing images in general. The ontology-based approaches used in image analysis may be adapted for semantic interpretation of diagrams, or generic ontologies used in image processing may be applied in *Diagram processor* or *SVG processor* in Figures 2 and 4 respectively. The primary focus of ontologies in the current work is to model spatial patterns in diagrams, and to reuse the automated reasoning features of ontology reasoners for diagram interpretation.

There are numerous works that combine UML and OWL, which can be roughly classified into two categories. The first category deals with model translations from UML to OWL or RDF (examples of such studies include [4, 23, 25]) and the second category focuses on reasoning about the models represented in UML diagrams (examples of such studies include [1, 12]). These two categories of works assume that the models represented in diagrams are easily accessible, so that they can focus on the translation of UML models into OWL. However, the

prototype presented in this work is useful when the models have to be *read* from the class diagrams. In other words, the developed prototype *reads* the models in the class diagrams using a visual syntax ontology. Thus an OWL ontology is used to infer the model specified in the class diagram. Once the models in the class diagrams have been *read*, then the approaches specified in [1, 4, 12, 23, 25] can be used to translate these models into OWL, and to reason about class diagrams, if required.

7 Conclusion

This paper presented an approach to represent spatial patterns of diagrams in an ontology, which is then used for diagram interpretation using instance classification feature of ontology reasoners. This ontology-based approach was used to realise a prototype to interpret UML class diagrams in SVG format. The performance of the prototype was evaluated with a small set of sample class diagrams, which showed promising results that support the need for further improvements to and evaluations of the prototype.

References

1. Berardi, D., Cali, A., Calvanese, D., Giacomo, G.D.: Reasoning on UML Class Diagrams. Artificial Intelligence 168, 70–118 (2005)
2. Cohn, A., Bennett, B., Gooday, J., Gotts, N.: Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. GeoInformatica 1(3), 275–316 (1997)
3. Fiorini, S.R., Abel, M., Scherer, C.M.: Semantic image interpretation of gamma ray profiles in petroleum exploration. Expert Systems with Applications 38(4), 3724 – 3734 (2011)
4. Gašević, D., Djurić, D., Devedžić, V., Damjanović, V.: Converting UML to OWL Ontologies. In: Proceedings of the 13^{th} International World Wide Web Conference on Alternate Track Papers & Posters. pp. 488–489. WWW Alt. '04, ACM, New York, NY, USA (2004)
5. Halland, K., Britz, K., Gerber, A.: Investigations into the use of SNOMED CT to enhance an OpenMRS health information system. South African Computer Jounal 47, 33–46 (2011)
6. HermiT OWL Reasoner. URL : http://www.hermit-reasoner.com/, accessed October 16, 2017
7. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web. CRC Press, Taylor & Francis Group (2009)
8. Horridge, M., Drummond, N., Jupp, S., Moulton, G., Stevens, R.: A Practical Guide to Building OWL Ontologies using Protégé 4 and CO-ODE Tools. University of Manchester (2009)
9. Horrocks, I., Parsia, B., Sattler, U.: OWL 2 Web Ontology Language: Direct Semantics (Second Edition). World Wide Web Consortium (2012)
10. Hudelot, C.: Towards a Cognitive Platform for Semantic Image Interpretation; Application to the Recognition of Biological Organisms. Ph.D. thesis, University of Nice - Sophia Antipolis (2005)

11. Kashyap, V., Bussler, C., Moran, M.: The Semantic Web: Semantics for Data and Services on the Web. Springer-Verlag Berlin Heidelberg (2008)
12. Khan, A.H., Porres, I.: Consistency of UML Class, Object and Statechart Diagrams using Ontology Reasoners. Journal of Visual Languages & Computing 26, 42 – 65 (2015)
13. Kirasić, D., Basch, D.: Ontology-Based Design Pattern Recognition, pp. 384–393. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
14. Maillot, N., Thonnat, M., Boucher, A.: Towards ontology-based cognitive vision. Machine Vision and Applications 16(1), 33–40 (Dec 2004)
15. Marriott, K., Meyer, B., Wittenburg, K.B.: In: Marriott, K., Meyer, B. (eds.) Visual Language Theory, chap. A Survey of Visual Language Specification and Recognition, pp. 5–85. Springer-Verlag New York, Inc., New York, NY, USA (1998)
16. Minas, M.: Syntax Definition with Graphs. Electronic Notes in Theoretical Computer Science 148(1), 19–40 (Feb 2006)
17. Object Management Group: Information technology - Object Management Group Unified Modeling Language (OMG UML), Superstructure (apr 2012)
18. Object Management Group: Object Management Group Unified Modeling Language (OMG UML) Version 2.5 (March 2015)
19. Protégé. URL : https://protege.stanford.edu/, accessed March 16, 2018
20. Salameh, K., Tekli, J., Chbeir, R.: SVG-to-RDF Image Semantization. In: Traina, A.J.M., Traina, C., Cordeiro, R.L.F. (eds.) Similarity Search and Applications: 7th International Conference, SISAP 2014, Los Cabos, Mexico, October 29-31, 2014. Proceedings. pp. 214–228. Springer International Publishing (2014)
21. Thomas, A., Gerber, A.J., van der Merwe, A.: Visual Syntax of UML Class and Package Diagram Constructs as an Ontology. In: Fred, A., Dietz, J., Aveiro, D., Liu, K., Filipe, J. (eds.) Proceedings of the 7^{th} International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015). vol. 2, pp. 17–28. SCITEPRESS (2015)
22. Thomas, A., Gerber, A.J., van der Merwe, A.: An Investigation into OWL for Concrete Syntax Specification Using UML Notations. In: Jamnik, M., Uesaka, Y., Schwartz, S.E. (eds.) Diagrammatic Representation and Inference: 9th International Conference, Diagrams 2016 Philadelphia, PA, USA, August 7-10, 2016 Proceedings. vol. LNAI 9781, pp. 197–211. Springer (2016)
23. Tong, Q., Zhang, F., Cheng, J.: Construction of RDF(S) from UML Class Diagrams. Journal of Computing and Information Technology 22(4), 237–250 (2014)
24. About the XML Metadata Interchange Specification Version 2.5.1. URL : http://www.omg.org/spec/XMI, accessed March 15, 2018
25. Zedlitz, J., Jorke, J., Luttenberger, N.: From UML to OWL 2. In: Lukose, D., Ahmad, A., Suliman, A. (eds.) Knowledge Technology, Communications in Computer and Information Science, vol. 295, pp. 154–163. Springer Berlin Heidelberg (2012)