crowd: A Tool for Conceptual Modelling assisted by Automated Reasoning - Preliminary Report

Christian Gimenez*[1], Germán Braun[1,2,3], Laura Cecchi[1], and Pablo Fillottrani[3,4]

[1] *Grupo de Investigación en Lenguajes e Inteligencia Artificial*
Departamento de Teoría de la Computación - Facultad de Informática
Universidad Nacional del Comahue
[2] *Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)*
[3] *Laboratorio de I&D en Ingeniería de Software y Sistemas de Información*
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
[4] *Comisión de Investigaciones Científicas de la Prov. Bs.As (CIC)*

Abstract There is an increment on the complexity of the information systems derived from new paradigms, for example Semantic Web, Big Data, e-government, etc. which require high quality solutions to tackle complex problems such as information integration. This quality is widely determined by the conceptual level. In this work, we present *crowd* as a novel tool for designing both conceptual models and ontologies based on visual representations with assistance of logic-based reasoning services. The challenge and the intention behind this work is to define graphical-logical methodologies as effective solutions for the description of interest domains at conceptual level. We detail the tool and demonstrate the usage of an initial prototype with some simple examples. Moreover, we identify limitations and potential issues about modelling and propose some partial solutions to tackle them. Currently, we are working to release the first beta version of *crowd*.

1 Introduction and Motivation

There is an increment on the complexity of the information systems derived from new paradigms, for example Semantic Web [1], Big Data [2], e-government [3], etc., which require high quality solutions to tackle complex problems such as information integration [4, 5]. This quality is widely determined by the conceptual level, therefore, conceptual modelling is key for the later implementation and maintenance of such systems. To this end, this level requires new methodologies and tools that enable domain experts to capture relevant features in the universe of discourse and a good comprehension of the implicit knowledge, which is difficult even for IT experts. Moreover, it is key to establish clear and measurable quality criteria along the process of creation and evolution of models. However,

* Scientific Vocation Scholarship. Consejo Interuniversitario Nacional (CIN). Argentina.

conceptual models also can lead to lack of insights or consequences that can be hidden to users in complex diagrams, causing inconsistencies and anomalies. Hence, to equip tools with capabilities to automatically explore and check models would be highly desirable. This can be generally done first translating models into some logic-based formal system and later handing them to an off-the-shelf reasoner that can be then queried about their properties. Consequently, automatic and graphical tools that assist modellers in obtaining sound conceptual models and ontologies are essential for a successful integration between the modellers' intentions and the formal semantics of them.

Zooming on the integration of visual diagrams with reasoning, ICOM [6] appears as a standalone tool for designing and linking multiple UML [7] and EER [8] models, making easier their evolution. The main objective of ICOM is to validate the effectiveness of class diagrams to express ontologies and retrieve new conclusions through them, even in the same graphical language. However, its communication between the ontology editor and back-end reasoners is by means of the DIG protocol [9], which is not being actively developed nor supported by new versions of reasoning systems. The set of graphical primitives is also limited since it is provided by a deprecated graphical library affecting the scalability of the tool, the visualisation aspects and models constraints considering reasoners expressiveness. On the other hand, OntoUML is a pattern-based and ontologically well-founded version of UML, whose meta-model has been designed in compliance with the ontological distinctions of a well-grounded theory, named Unified Foundational Ontology (UFO). Currently, OntoUML is supported by Menthor Editor[1], which provides a simple and integrated set of features to assist users such as syntactical verification, visual simulation, automatic semantic-anti-patterns detection and correction, validation of parthood relations and ontology patterns. Particularly, logic-based validations are supported by Alloy [10], allowing simulations on specifications. However, reasoning is not integrated to OntoUML's graphical language. Lastly, ontology tools such as the well-known Protégé [11, 12] allows editing and visualising ontologies, however, both the graphical support and the logic-based reasoning are weakly integrated. It offers a wide set of structures for modelling but inferences from external reasoners are only limited to IsA relations in graphical plug-ins [13, 14], while this integration is missing in its Web version, mainly accomplishing with the objective of being an ontology repository and a collaborative editing platform.

In this work, we present *crowd* as a novel client-server tool, for graphical conceptual modelling and ontology design with reasoning support to assist users in these processes. The features of *crowd* comprise graphical modelling, the formalisation of models into a logic-based formal system and the communication with off-the-shelf reasoners. This enables a satisfiability checking process and the inference of implicit constraints, which will be expressed in diagrams by using the same graphical language. Moreover, users will visualise their models in an online manner while are being modelled and edited. In this way, we provide means to stimulate interest in this kind of tools for designing both conceptual models and

[1] `http://www.menthor.net/`

ontologies based on the effectiveness of graphical modelling languages for expressing them. Furthermore, we intend to encourage researchers and companies to develop these semantics-aware technologies. Lastly, another aim of *crowd* is to be the base of new and more expressive methodologies assisted by reasoning and oriented to cover ontology engineering needs previously identified in [15, 16].

A first prototype of this tool already runs on the client-server architecture. It supports the OWLlink communication standard protocol [17] and a satisfiability checking process on simple UML graphical diagrams encoded in $\mathcal{ALCQI}$ Description Logics (DL) [18], as demonstrated in [19]. The tool continues being developed with updated and scalable graphical libraries and technologies, such as JavaScript and PHP, installed in an Apache server and by means of quality-oriented programming techniques as Test-Driven Development [20].

This work is structured as follows. Section 2 gives an overview of the main features of *crowd* and details its architecture. Section 3 presents the first prototype developed together with some simple examples of use. A preliminary evaluation and discussions are presented in section 4. To conclude the paper, section 5 details some related works, while section 6 elaborates on final considerations and directions for future works.

2 crowd Tool Overview

crowd is a graphical modelling tool being supported by both Universidad Nacional del Comahue and Universidad Nacional del Sur of Argentina. The intention behind the tool is to allow users to design conceptual models and ontologies adopting standard modelling languages. Complete logical reasoning is employed by *crowd* to verify the satisfiability of specifications, infer implicit constraints and suggest new ones. The leverage of automated reasoning is enabled by a precise semantic definition of all the elements of the class diagrams. Hence, diagrams constraints are internally translated into a logic-based formalism capturing typical features of models. To this end, the tool is fully integrated with a powerful logic-based reasoning server acting as a background inference engine. Moreover, since *crowd* is based on a deduction-complete notion of reasoning support relative to the diagram graphical syntax, users will see the original model graphically completed with all the deductions and expressed in the graphical language itself. This includes checking class and relationship consistency, discovering implied class and cardinality constraints. Other non-graphical constraints can be modelled in the tool, but they should be defined in a textual way. Lastly, *crowd* only focuses on graphical modelling of schemes, while it does not consider individuals.

Additionally, users' preferences and usages have been also considered as part of methodological processes allowing a better understanding of their behaviours and enabling a community of users on the *crowd's* modelling approaches. Some preliminary results about this issue have been also published in [15]. In that work, we propose to integrate a set of pattern-based extension rules that identify elements from a diagram and suggest any possible consistent evolution. These rules guide the modelling process by identifying graphical elements and refactor-

ing from them. Each rule has been defined and analysed by considering different theories of design patterns in [16].

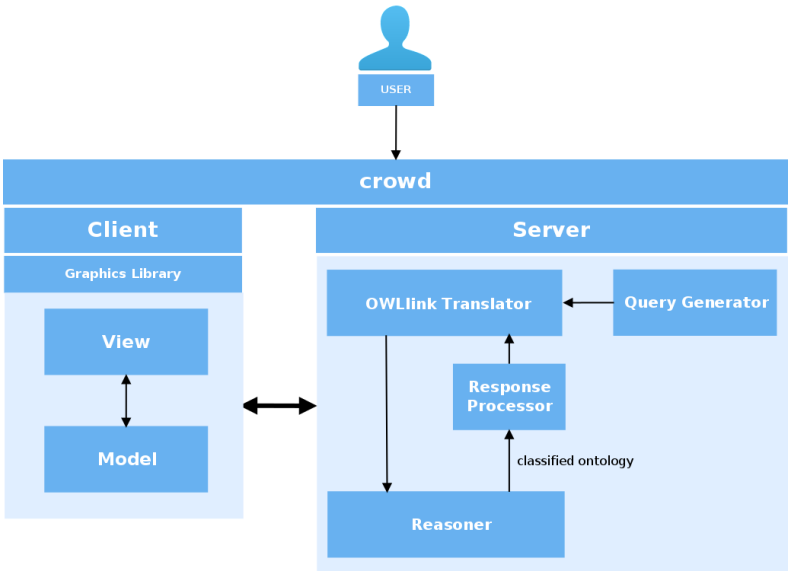


Figure 1. An overview of the *crowd* client-server architecture.

To conclude, our purpose is not to supply to the scientific community a robust tool potentially replacing the other ones available. We do not claim that is more usable than any of the existing tools either, as will be detailed after. *crowd* is a meant to evaluate the integration of graphical languages with reasoning systems and thus assist users by means of methodologies for conceptual modelling and ontology design. Furthermore, we intend to stimulate interest in these knowledge representation based technologies to be considered as tools for researchers and companies.

2.1 A Description of the *crowd's* Architecture

Figure 1 shows a high level overview of the client-server architecture. Users interact with a front-end running in a Web browser, which provides a graphical environment with the necessary functionalities for creating and editing models together with a set of graphical primitives. Moreover, it shows the reasoner responses using the very same graphical syntax in addition to a textual manner. The back-end running in the server side comprises modules to translate visual diagrams, generate queries to the reasoning systems, reason over models specifications and process its output to be sent back to the front-end. Both sides interact via asynchronous HTTP requests in order to allow users to visualise their

diagrams, while are being modelled and edited. We explore each component in turn.

Client. *crowd's* client features are supported by a JavaScript Graphical Library, called JointJS[2]. The main objective is to give easier instructions to draw the primitives and currently, is being used in related tools as [21] for data visualisation. A preliminary review about different graphical libraries and their features shows that this kind of libraries use Canvas or SVG drawing technologies through basic commands to draw diagrams like UML. Also, they handle users' events by default such as drag and drop, between others. In particular, JointJS presents some key advantages: it focus on drawing diagrams primitives, provides plug-ins for UML, EER , etc. and is expandable by creating your own. Furthermore, it also use the Backbone.js library to give structure to Web applications. Backbone.js is a JavaScript library that assists in the development of Web applications by providing a model-view-controller architecture for the client.

Similar libraries, such as Raphael[3], Processing[4], p5js[5], provide simpler interface for drawing primitives like curves and geometric figures. However, they do not directly give developers an API of dedicated functions for drawing primitives from our considered modelling languages. Lastly, jsPlumb[6] does follow another direction towards the assistance for drawing different kinds of connections between elements, but does not provide diagrams primitives yet.

Server. Firstly, the module `OWLlink Translator` provides the OWL 2 representation from an input graphical model to be sent to the back-end reasoner. It takes a JSON representation of an user's diagram and generates an equivalent OWLlink-syntax model. The `Query Generator` module supplies a set of queries to hand them to the reasoner over the initial model. These queries retrieves models properties by using built-in reasoning services. Different sets of queries can be configured in this component according to invoked methodologies or services from the *crowd's* front-end. Hence, the reasoning over models is composed by the initial user's input together with the set of queries supplied by the previous module. They are executed by an off-the-shelf inference engine, which is represented through the `Reasoner` module. The reasoning results are the input of the `Response Processor` module, where are finally processed and returned to the front-end displaying new insights or consequences possibly hidden.

3 *crowd* First Prototype and Examples of Use

crowd's prototype front-end is developed in JavaScript and running in a Web browser, while its back-end runs in an Apache server and is developed in PHP.

[2] `http://www.jointjs.com/`
[3] `http://raphaeljs.com/`
[4] `http://processingjs.org`
[5] `http://p5js.org`
[6] `https://jsplumbtoolkit.com/`

Currently, the prototype allows graphically creating and editing simple UML diagrams, although more expressive OWL 2 [22] constraints can be appended to models by inserting OWLlink statements. *crowd's* prototype is integrated with the Racer DL reasoning server [23].

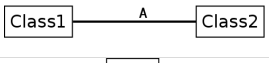
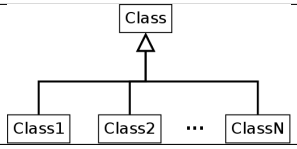
Expression	***crowd* syntax**	**Description Logic syntax**
Class		$Class_1$
Binary Association *0..n* *0..n*		$\top \sqsubseteq \forall A.Class_2 \sqcap \forall A^{-}.Class_1$
IsA Relationship		$Class_1 \sqsubseteq Class$ $Class_2 \sqsubseteq Class$ ... $Class_n \sqsubseteq Class$

Table 1. *crowd* representation and description logic formalisation for concepts, binary associations and IsA relationships.

Reasoning service	**OWLlink query**	**number of queries**
project status	`<IsKBSatisfiable kb="http://<server>/kb"/>`	1

class satisfiability	`<IsClassSatisfiable kb="http://<server>/kb">`
	`  <owl:Class IRI="Class"/>`
	`</IsClassSatisfiable>`
		c

Table 2. OWLlink requests and number of queries in the first *crowd* prototype. The value of c represents the number of classes in the model.

Table 1 outlines the graphical UML primitives being supported by the prototype and their corresponding $\mathcal{ALCQI}$ expressions. In this respect, description logic concepts represent graphical UML classes and binary associations assume *0..n* cardinality by default. Similarly, Table 2 summarises the reasoning services to be executed, their OWLlink syntax and the number of these queries, where c is the number of classes. Hence, when reasoning is invoked, the whole model and each class in it is checked for satisfiability (i.e. non-emptiness) through these queries.

The current look-and-feel of the tool is shown in Fig. 2. Its interface includes the `Tools` and `Details` menus. The first presents users' options to add UML classes, enable the translation of graphical models to OWLlink syntax and invoke

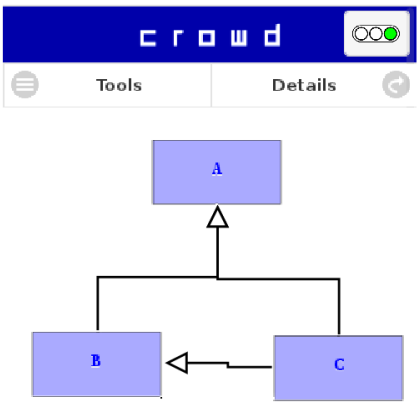


Figure 2. *crowd's* diagram for satisfiable example. The Graphical User Interface (GUI) includes a `Tools` menu to add classes and a `Details` menu to check the reasoning outputs in a textual way.

the reasoning services by means of a traffic light icon. On the other hand, the `Details` menu shows text area elements to insert OWL 2 expressions in textual way and visualise the reasoner's details.

```
<owl:SubClassOf>
  <owl:Class IRI="A"/>
  <owl:Class abbreviatedIRI="owl:Thing"/>
</owl:SubClassOf>
<owl:SubClassOf>
  <owl:Class IRI="B"/>
  <owl:Class abbreviatedIRI="owl:Thing"/>
</owl:SubClassOf>
<owl:SubClassOf>
  <owl:Class IRI="C"/>
  <owl:Class abbreviatedIRI="owl:Thing"/>
</owl:SubClassOf>
```

```
<owl:SubClassOf>
  <owl:Class IRI="B"/>
  <owl:Class IRI="A"/>
</owl:SubClassOf>
<owl:SubClassOf>
  <owl:Class IRI="C"/>
  <owl:Class IRI="A"/>
</owl:SubClassOf>
<owl:SubClassOf>
  <owl:Class IRI="C"/>
  <owl:Class IRI="B"/>
</owl:SubClassOf>
```

Figure 3. OWL 2 statements generated by *crowd* from the UML initial model in Fig. 2. This task is done by the `OWLlink Translator` module.

Lastly, the following two examples will illustrate how the models satisfiability checking is done. In both cases, we instruct *crowd* to send the OWL 2 representation of diagrams to the server for invoking reasoning services. After this, the tool will inform the details of the reasoning process and update the user interface according to its responses through the traffic light icon in such front-end.

```
<IsKBSatisfiable kb="http://kb1"/>
<IsClassSatisfiable kb="http://kb1">
  <owl:Class IRI="A"/>
</IsClassSatisfiable>                    <BooleanResponse result="true"/>
<IsClassSatisfiable kb="http://kb1">     <BooleanResponse result="true"/>
  <owl:Class IRI="B"/>                   <BooleanResponse result="true"/>
</IsClassSatisfiable>                    <BooleanResponse result="true"/>
<IsClassSatisfiable kb="http://kb1">
  <owl:Class IRI="C"/>
</IsClassSatisfiable>
```

Figure 4. Set of queries of the OWLlink request message generated in the `Query Generator` to be submitted to the reasoning system. This picture also describes the responses received for each query in the same OWLlink syntax. Notice that both classes are satisfiable as well as the complete knowledge base.

Satisfiable Example. Figure 2 shows a simple UML class diagram composed by three classes `A`, `B` and `C`. Classes `B` and `C` are subclasses of `A` and also `B` subsumes `C` Both relationships are incomplete and overlapping. The current colours in the traffic light mean that no satisfiability checking event has been required yet. Such event will be triggered when clicking on this icon enabling the diagram translation. Fig. 3 depicts the OWL 2 statements generated from the initial graphical diagram. These statements are encapsulated in an OWLlink request message together with the set of queries shown in Fig. 4. As soon as the queries answers are received and processed, they are sent back to the front-end and the green colour is highlighted in the traffic light indicating that our model is satisfiable. Reasoner output is also shown in Fig. 4.

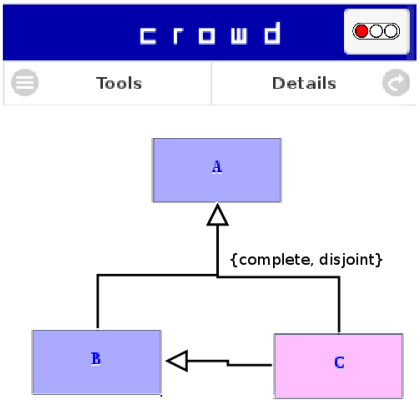


Figure 5. Conceptual diagram for unsatisfiable example. *crowd* highlights the inconsistent class in a different colour and the traffic light in red.

```
<owl:DisjointUnion>
  <owl:Class IRI="A"/>
  <owl:Class IRI="B"/>
  <owl:Class IRI="C"/>
</owl:DisjointUnion>
```

Figure 6. OWL 2 for complete and disjoint IsA relationship in Fig. 5. The OWL 2 `DisjointUnion` is a syntatic shortcut for the axioms `EquivalentClasses`, `ObjectUnionOf` and `DisjointClasses`.

Unsatisfiable Example. In this example, user's initial model is similar to the previous one, but we have modified by adding a complete and disjoint IsA relationship between the parent class `A` and its children `B` and `C`. OWL 2 statements from this disjoint union of class are depicted in Fig. 6 to be appended to the code already shown in Fig. 3. Despite the fact that this feature is still under development, we expect that this modification is to be displayed as depicted in Fig. 5. After invoking the reasoning services from the application asking about diagram and classes satisfiability, the background reasoner returns the responses messages shown in Fig. 7. This result is to be displayed in the front-end as a red traffic light indicating inconsistencies in the current diagram. Particularly, class `C` is coloured in red highlighting its inconsistency.

```
<BooleanResponse result="true" warning="Unsatisfiable classes:
                            (*BOTTOM* BOTTOM file:<OWLlink file>C)"/>
<BooleanResponse result="true"/>
<BooleanResponse result="true"/>
<BooleanResponse result="false"/>
```

Figure 7. Reasoner output returning "false" for the unsatisfiable class `C` and "true" for `A` and `B` class. Notice that the complete model still continues being satisfiable.

4 Preliminary Evaluation

crowd has been designed as a scalable and maintainable architecture for adapting new graphic engines, design methodologies and background reasoners. In this respect, we have selected expansible graphical libraries and Web technologies to achieve properly this objective. As introduced in section 1, related tools were designed by following other aims and generally, they only visualise models and ontologies in a limited way. Furthermore, the graphical and reasoning integration is weak or no reasoner assistance is supplied to modellers. Consequently, these limitations require a lot of cognitive effort from users along the modelling process.

crowd has been conceived from scratch as a graphical-centric tool supporting standard modelling graphical language and considering the possibility to expand its graphical primitives for more expressiveness. We realise that such languages are not enough expressive if we take into account the underlying expressiveness of state-of-the-art reasoning systems. For example, users could not visually represent disjoint and equivalence constraints, between others logical axioms. Because of this, we have considered two possible solutions: to allow users to define logical expressions, which requires knowledge about the formal systems, as $\mathcal{DLR}$ [5] or provide new graphical primitives with a precise semantic definition. In this sense, *crowd* implements a text area for introducing OWL 2 statements as a temporary solution.

From a graphical point of view, *crowd's* prototype does not yet implement special visual techniques for handling large diagrams. However, this has been taken into account during the development process and the selection of graphical libraries. Although some graphical primitives are still missing, we have tested the JointJS library by creating more than 50 UML classes keeping the prototype user interface browseable. These 50 classes represent 50 concepts in a DL knowledge base, which is perfectly supported by the current off-the-shelf reasoners. Nevertheless, a trade-off between the size of the diagrams and their readability should be considered as part of a visualisation process [24]. In such processes, cognitive activities are proposed and related to how visual models are known, understood and learnt.

Lastly, interoperability with other tools is another important aspect so that modules to support importing and exporting should be implemented for standard languages.

5 Comparison with other Tools

With reference to existing tools, we have surveyed some of them covering creating, editing and visualising of conceptual models and ontologies by using different standard languages or graphs. However, either they are weakly integrated or no integration exists with automatic reasoning systems.

Protégé [11] offers a wide and rich set of modelling structures, but deductions obtained by external OWL reasoners are limited to Isa relationships by using graphical node-link-based plug-ins as OntoGraf[7], OWLViz [8] or SOVA [9]. This limitation is also presented in the Web version of this tool, called WebProtégé [12], where the interaction with reasoners is missing and no graphical support is provided, being mainly an ontology repository for collaborative edition. TopBraid Composer [25] and NeOn toolkit [26] also show inferences graphically limited to Isa relationships. The first one, however, also provides another visualisation format by means of RDF [27] graphs while visualisation support in NeOn toolkit is by ad-hoc plug-ins. OWLGrEd [28] does provide UML-like graphical

[7] `http://protegewiki.stanford.edu/wiki/OntoGraf`
[8] `http://protegewiki.stanford.edu/wiki/OWLViz`
[9] `http://protegewiki.stanford.edu/wiki/SOVA`

syntax and works as an external visualiser of Protégé since it allows importing ontologies from there to be rendered. Nevertheless, inferred axioms in Protégé are not visualised in OWLGrEd when exporting.

Unlike the previous tools, NORMA [29], Graphol [30] and VOWL [31] are mainly graphical tools although no interaction with reasoning is supported. The novel of these implementations is that the first two present a new graphical languages named ORM and a diagrammatic representation closer to DL ontologies, respectively. In this same direction, VOWL tool exploits node-link diagrams to visualise ontologies by focusing on schemes and some recommendations on how to depict individuals and data values.

Finally, the more related to ours work are ICOM and OntoUML. The former concerns the foundational concepts behind *crowd*. However, the old-fashioned DIG protocol and the limitations to increase models expressiveness, in addition to deprecated graphical technologies, are critical aspects. The latter is also a graphic-centric tool with support for satisfiability checking by simulating specifications, but it is not integrated with visual diagrams. On the other hand, both ICOM and OntoUML provide partial assistance to the graphical modelling process by integrating with reasoners and checking for anti-patterns, respectively. To sum up, none of the surveyed tools completely implement the integration of visual representations with automatic reasoning. Hence, quality aspects become difficult to assess making more relevant the objectives of our tool.

6 Conclusions and Future Works

We have presented *crowd* as a client-server Web architecture to support conceptual modelling and ontology design with DL-based reasoning assistance for deductions. We have identified this need after surveying and analysing the state-of-the-art tools. In that way, we focus on the graphical representation of diagrams and the well-known updated technologies for Web applications. We reach the first prototype of *crowd* that allows to model very simple UML diagrams and invokes reasoning services to check for satisfiability. This has been demonstrated in two examples of use along this work. Finally, we have evaluated our experience developing the tool in a preliminary manner, identifying its current limitations and proposing solutions for each one.

In the future, we plan to release the first beta version supporting a complete subset of UML primitives and continue the evaluation of the tool. We also plan to integrate methodologies from [15, 16] and evaluate results of these proposals in real domains. Finally, we will work in a metamodel to enable coordinated conceptual modelling by showing linkable ORM, UML, and EER diagrams according to [32].

Acknowledgements

The authors would like to thank the anonymous referees for their comments and suggestions. This work is based upon research partially supported by the Univer-

sidad Nacional del Comahue (Project ID: 04/F006), the Universidad Nacional del Sur (Project ID: 24/N038), the Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), the Consejo Interuniversitario Nacional (CIN) and the Comisión de Investigaciones Científicas de la prov. de Buenos Aires (CIC).

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (May 2001)
2. Zikopoulos, P.C., Eaton, C., deRoos, D., Deutsch, T., Lapis, G.: Understanding Big Data - Analytics for Enterprise Class Hadoop and Streaming Data. (2012)
3. Misra, C.: Defining E-government: A Citizen-centric Criteria-based Approach (2006)
4. Calvanese, D., Giacomo, G.D., Lenzerini, M., Nardi, D., Rosati, R.: Information Integration: Conceptual Modeling and Reasoning Support. In: CoopIS, IEEE Computer Society (1998) 280–291
5. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Description Logic Framework for Information Integration. (1998)
6. Fillottrani, P., Franconi, E., Tessaris, S.: The ICOM 3.0 intelligent conceptual modelling tool and methodology. Semantic Web (2012)
7. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language User Guide. Addison-Wesley Professional (2005)
8. Gogolla, M.: Extended Entity-Relationship Model: Fundamentals and Pragmatics. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1994)
9. Bechhofer, S., Moller, R., Crowther, P.: The DIG Description Logic Interface. In: In Proc. of International Workshop on Description Logics (DL2003). (2003)
10. Jackson, D.: Alloy: A Lightweight Object Modelling Notation. ACM Trans. Softw. Eng. Methodol. **11**(2) (April 2002)
11. Knublauch, H., Fergerson, R., Noy, N., Musen, M.: The Protégé OWL plugin: An open development environment for semantic web applications. (2004)
12. Tudorache, T., Nyulas, C., Noy, N.F., Musen, M.A.: WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the Web. Semantic Web **4**(1) (2013) 89–99
13. Horridge, M.: OWLViz `http://protegewiki.stanford.edu/wiki/OWLViz` accessed July 2015.
14. Falconer, S.: OntoGraf `http://protegewiki.stanford.edu/wiki/OntoGraf` accessed July 2015.
15. Braun, G., Cecchi, L., Fillottrani, P.: Integrating Graphical Support with Reasoning in a Methodology for Ontology Evolution. In: Proc. of the 9th Int. Workshop on Modular Ontologies WoMO 15 IJCAI 15. CEUR Workshop Proceedings (2015)
16. Braun, G., Cecchi, L.: Extension Rules for Ontology Evolution within a Conceptual Modelling Tool. In: Proc. of the 1st Simposio Argentino de Ontologías y sus Aplicaciones SAOA 15 JAIIO 15. CEUR Workshop Proceedings (2015)
17. Liebig, T., Luther, M., Noppens, O., Wessel, M.: Owllink. Semantic Web **2**(1) (2011) 23–32
18. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, USA (2003)

19. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artif. Intell. **168**(1-2) (2005) 70–118
20. Beck, K.: Test Driven Development: By Example. Addison-Wesley Longman Publishing Co., Inc. (2002)
21. Mouromtsev, D., Pavlov, D., Emelyanov, Y., Morozov, A., Razdyakonov, D., Galkin, M.: The Simple Web-based Tool for Visualization and Sharing of Semantic Data and Ontologies. In: Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015. (2015)
22. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S., eds.: OWL 2 Web Ontology Language: Primer. W3C Recommendation (27 October 2009) Available at `http://www.w3.org/TR/owl2-primer/`.
23. Haarslev, V., Möller, R.: Racer system description. In Goré, R., Leitsch, A., Nipkow, T., eds.: International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy, Springer-Verlag (2001) 701–705
24. Ware, C.: Information Visualization: Perception for Design. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)
25. TopQuadrant: TopQuadrant — Products — TopBraid Composer (2011)
26. Hasse, P., Lewen, H., Studer, R., Erdmann, M.: The NeOn Ontology Engineering Toolkit. (2008)
27. World Wide Web Consortium: RDF 1.1 Primer. Available at `https://www.w3.org/TR/rdf11-primer/`, last accessed May, 2016.
28. Cerans, K., Ovcinnikova, J., Liepins, R., Sprogis, A.: Advanced OWL 2.0 Ontology Visualization in OWLGrEd. In: DB&IS. Frontiers in Artificial Intelligence and Applications, IOS Press (2012)
29. Curland, M., Halpin, T.A.: The NORMA Software Tool for ORM 2. In: CAiSE Forum. Lecture Notes in Business Information Processing, Springer (2010)
30. Console, M., Lembo, D., Santarelli, V., Savo, D.F.: Graphol: Ontology representation through diagrams. In: Informal Proceedings of the 27th International DL'14
31. Lohmann, S., Negru, S., Bold, D.: The ProtégéVOWL Plugin: Ontology Visualization for Everyone. In: Proceedings of ESWC 2014 Satellite Events, Springer (2014)
32. Keet, C.M., Fillottrani, P.R.: An ontology-driven unifying metamodel of UML Class Diagrams, EER, and ORM2. Data Knowl. Eng. (2015)