Kukulcan: Semantic Web Framework for Knowledge Management in the Domain of Digital Circuits

F. Edgar Castillo-Barrera[1], **R. Carolina Medina-Ramírez**[2],
J. Emilio Labra Gayo[3], **and S. Masoud Sadjadi**[4]
[1]School of Engineering, Universidad Autónoma de San Luis Potosí, San Luis Potosí, México
[2]Department of Electrical Engineering, Universidad Autónoma Metropolitana, Distrito Federal, México
[3]Department of Computer Science, Universidad de Oviedo, España
[4]School of Computing and Information Sciences, Florida International University (FIU), Miami, USA

Abstract

In recent years Ontologies have boomed as artifacts to represent a domain and they are considered an important key to the success of the Semantic Web. Thus, Humans and Machines would be able to understand and share information on the Web which are also important in the context of Knowledge Management. Although the study of the relation between Ontologies and Knowledge Management is not new and this is applied in Knowledge Engineering, Semantic Web Techniques such as Reasoners and Ontology queries have been recently studied and applied. A Framework based on Semantic Web Techniques can give more options for sharing, increasing, reusing, and capitalizing the knowledge in organizations and companies. In digital circuits domain, a Semantic Web Framework can be employed for teaching logic gates (and, or, not, xor, etc.), and this approach has been deemed as an effective way for capturing and using the knowledge of the logic gates on assembling circuit systems. This knowledge can be reused by new developers gaining time and reducing circuits manufacturing costs. In addition, the correct assembling among logic gates and the right output of a circuit can be validated by using semantic techniques. In this paper, we describe a semantic web framework based on a core ontology, a Pellet reasoner and SPARQL queries for Knowledge Management based on the domain of digital circuits. We use an example and a prototype called Kukulcan to explain our approach.

1 INTRODUCTION

New methods for verifying and validating logic circuits are neccesary during the design phase. These methods have to ensure the funcionality expected by the circuits designer before building it, and at the same time, simulating its behavior. This helps to prevent economic losses. Another important factor to consider is the circuit models reusability. The time for developing a complex circuit using a circuit repository decrease the cost of the project and reduce the learning curve of new people in the project. In this context, semantic technologies seem relevant. We can use Ontologies[41] in order to represent a logic circuit base on logic gates (and, or, not, etc.) and to verify a circuit design. Each connection of the circuit can be validated by means of ontology properties [41] and reasoners [40]. The new knowledge obtained for each part of the circuit assembled, can be stored in an ontology [32] by means of metadata, in this way the knowledge is capitalized [39][33]. This knowledge can be used by new developers or new members of the project to reduce manufacture time. In consequence, the company decreases costs. The circuits behavoir can be modelled by SPARQL queries. In fact, a complex circuit could be represented by one SPARQL query. The Ontology written in OWL-DL [36][26] (is stored as an XML file) can be exchanged among different systems and can be shared by all people in the company by means of the company intranet website.

The rest of the paper is structured as follows. In Section 2 we give the related work of ontologies based on logic circuits domain. In Section 3 we briefly explain concepts about Semantic Web, Ontologies, Core Ontologies, Reasoners, SPARQL queries and Semantic Web Techniques. Section 4 describes our approach for the Verification and Validation of logic circuits in a Semantic Factory Framework. In Section 5 we show the feasibility of our technique by describing an example and a prototype called Kukulcan. Finally, in Section 6 we conclude our work.

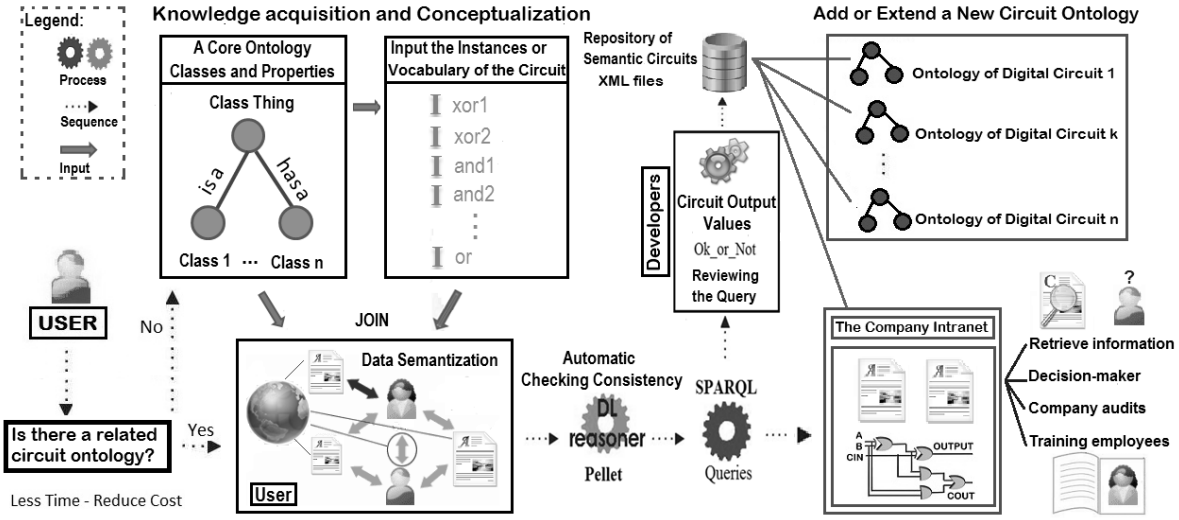


Figure 1. Semantic Web Techniques for Knowledge Managment in the Domain of Logic Circuits

2 RELATED WORK

There are several works about Ontologies and its relation with Knowledge Management and Knowledge Management Systems [1][6][34][15][31]. In the case of ontologies based on digital circuits for teaching is mostly represented by work of Robal et al.[38] who wrote an ontology-based intelligent learning object for teaching the basics of digital logic. Robal's ontology is oriented for teaching the basics of digital logic, our ontology can be used for teaching, validating and verifying logic circuits based on logic gates. An important method for verifying logic circuits is found in the work of J.N. Hooker and H. Yan [25]. The authors propose a new tautology checking algorithm for determining the correct boolean function in a circuit. This algorithm is non-numeric and equivalent to a numeric algorithm obtained by applying Benders descomposition. This proposal is similar to an integer programming problem, which requires calculations and computational resources. Although in our proposal the designer of the circuit does not apply formal verification methods, ontologies are based on formal logic (description logic [5][4]). In contrast to Banders descomposition method, our proposal is a semi-automatic verification method.

3 SEMANTIC WEB TECHNIQUES

The *Semantic Web* [12][11][37] is an extension of the World Wide created by the british scientist Tim Berners-Lee who defines it as *"a web of data that can be processed directly and indirectly by machines"* [9]. This is a collection of standards, a set of tools [14], and a community that shares data. *Semantic Technology* is a concept in computer science which goal is to give semantics to data[20]. Supported by semantic tools that provides semantic information about the meaning of words (RDF, SPARQL, OWL, and SKOS). The Web is a key focal. Semantic Web Techniques are methods and techniques based on semantic tools which allow us to manipulate information also. Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data [24].

3.1 Ontologies and Knowledge Management

Ontologies are the key for Semantic Web goals and they are an important block of the semantic web stack [9]. An Ontology [21][9][23][11][41] is defined by Gruber as "*a specification of a conceptualization*" [21]. An Ontology defines the basic terms used to describe and represent an area of knowledge, as well as the rules for combining terms and relations used to define extensions to the vocabulary. Thus, defines the vocabulary and the meaning of that vocabulary, are used by people and applications that need to share domain information. More specifically, an ontology is a formal representation of knowledge with semantic content which allows the companies and organizations to obtain information[17]. Such information can be retrieved by performing SPARQL queries or using a rule-based inference engine [42]. In our case, the logic circuits are the domain area. *Knowledge management* was defined by Alavi and Leidner [1] as "a systemic and organizationally specified process for acquiring, organizing and communicating both tacit and explicit knowledge of employees so that other em-

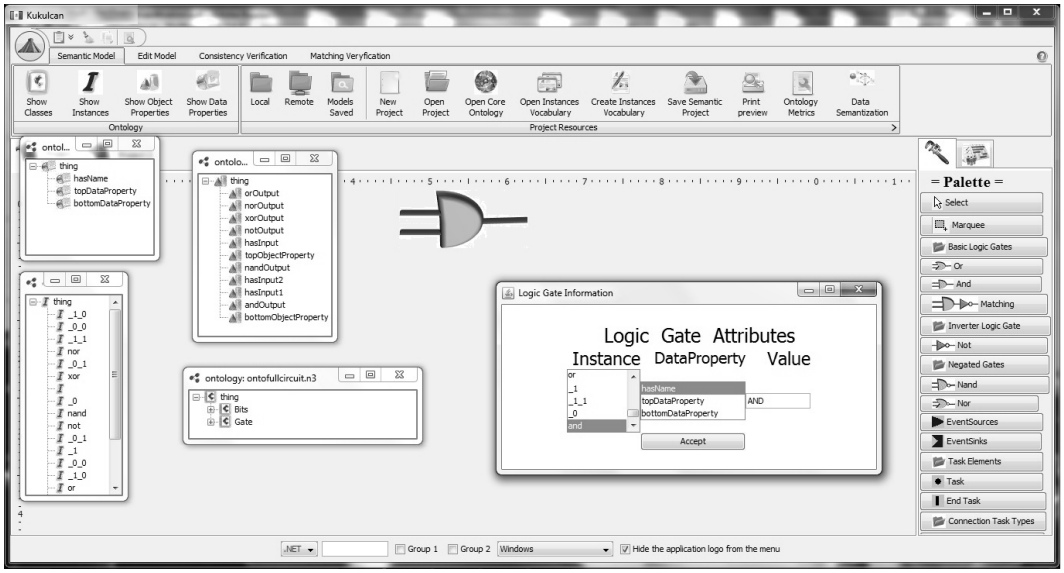


Figure 2. Classes, Instances, Properties and Data Semantization screen

ployees may make use of it to be more effective and productive in their work". A Semantic Web Framework can comply with the above definition. For that reason, we have selected this definition to support our approach in the field of Knowledge Management.

3.1.1 Core Ontologies

In philosophy, a **Core Ontology** [13] is a basic and minimal ontology consisting only of the minimal concepts required to understand the other concepts. It must be based on a core glossary that humans can understand. A **Core Ontology** is a complete and extensible ontology that expresses the basic concepts in a certain domain. In this work we have built a core ontology which consists of a logic gates glossary which developers of circuits understand well. We consider that these kind of ontologies can be reused. The ontology classes have been defined using **n3** notation. Ontologists of these kind of ontologies do not require a complex methodology [17] to do it, in fact, following the Ontology Development 101 [16] or An eXtreme method for developing lightweight ontologies [27] are enough.

3.2 SPARQL Query Language

SPARQL is a query language for the Resource Description Framework (RDF) which is a W3C Recommendation [43]. RDF Schema (RDFS) is extending RDF vocabulary for describing taxonomies of classes and properties. It also extends definitions for some of the elements of RDF, for example it sets the domain and range of properties and relates the RDF classes and properties into taxonomies using the RDFS vocabulary. We use Web Ontology Language OWL which extends RDF and RDFS. Its primary aim is to bring the expressive and reasoning power of description logic to the semantic web. Querying language is necessary to retrieve information [28] from input model (instances of the ontology and its relations). Unfortunately, not everything from RDF can be expressed in Description Logics (DL) [5][4]. For example, the classes of classes are not permitted, and some of the triplet expressions would make no sense in DL. To partially overcome this problem, and also to allow layering within OWL, three types of OWL are defined (FULL, Lite and DL). At this moment, we only have decided to explore semantic queries in SPARQL instead of applying another action such as: production rules [42].

3.3 Reasoners

A reasoner [40] is a program which its main task is checking the ontology consistency. It verifies if the ontology contains contradictory facts, axioms or wrong properties among concepts. Besides, new knowledge can be inferred after applyed it. The most popular reasoners are Cerebra, FACT++, KAON2, Pellet, Racer, Ontobroker, OWLIM. Pellet [40] is an open-source Java based OWL-DL reasoner. In our verification process we use Pellet for checking the consistency of the logic circuit ontology and classify the taxonomy. We select the Pellet reasoner, because it gives an explanation when an inconsistency was detected.

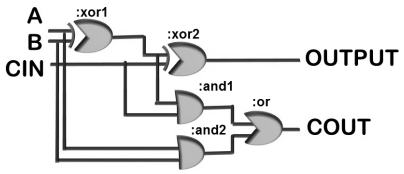


Figure 3. 1-bit Full Adder

4 A SEMANTIC WEB FRAMEWORK

Kukulcan is factory framework of semantic models which focuses on maximising the level of reuse in two dimensions: architecture design and logic circuits. One of the most important features of this framework is enabling knowledge reuse in logic circuits modelling using Semantic web techniques [19]. The aim of this framework is to allow to develop logic circuits using a friendly interface and a graphical arquitectural description language. Our main contributions are twofold. First, we define a framework that allows us to reusing logic circuit. Second, our approach supports the validation of the output values obtained from the logic circuit during the design phase. A prototype of the framework involves a visual editor. Figure 4. The tool makes use of the library Flamingo and the Ribbon component [29] implemented in Java. We have used Jena API [30][35] and Java language [18] for programming that and NetBeans IDE 7.0 [10]. The process of verification, within the Kukulcan framework, is done at very high level, using the ontologies information among logic gates and circuits to be assembled. Each logic gate is represented in a graphic way. That information, introduced in the ontology during the **Data Semantization process**, is evaluated and after that the reasoner verify if it is correct. In Addition, we capture the new knowledge in this new logic circuit, called "Capsule". In our framework, a Capsule has a graphical representation which is stored as a new logic circuit with its own characteristics. The process to verify the assembling among logic gates is easy for an user who building circuits. He introduce his model into the framework by means of a file or by the editor (the *option Create Instances Vocabulary*). Kukulcan transforms his vocabulary (logic gates that the user needs for building his circuit) from a text file into an ontology instances. Then, the user only has to establish its relations using the ontology properties (object and datatype) and he has to associate the logic gates instances created with classes defined in the logic circuit Ontology. This process is called **Data Semantization** See Figure 2.

4.1 A Core Ontology for Logic Circuits

We propose a core ontology called **OntoCircuit** which has the minimum concepts (logic gates) neccesary to represent the 1-bit Full Adder circuit. And, Or, Xor, Not, Nand, Nor and Xnor are universal gates and they do not require to be validate by experts. Besides, we only need 3 or 5 competency questions to validate the ontology [22]. These are adventages in this kind of ontologies which foment the reuse of them. Core Ontology is built by means of classes and relations among concepts. The Ontology is showed in Figure 2. A Logic Gates Ontology was created for capturing and verifying information about the input logic circuit models. This ontology consists of 3 classes (Circuit, Bits and Gate), 10 Object Properties (hasInput1, hasInput2, hasInput3, isTypeGate, andOutput, orOutput, notOutput, nandOutput, norOutput, xorOutput), 1 Datatype Property (hasName) and 25 instances. The notation n3 is used by the ontology, because is a valid RDFS and OWL-DL notation. The Ontology use RDFS and OWL-DL language [2][36]. They are fundamentally based on descriptive logic languages. OWL-DL is a recommendation of the W3C [43]. The OWL-DL ontologies have the ability of: Automatic reasoning, Easy to be distributed through many systems, Compatibility with web standards for accessibility, Opening and extensibility.

4.2 Logic Circuits Verification: a Semantic Approach

Semantic verification and validation is the process which uses an Ontology and Semantic Technologies (SPARQL queries) to guarantee the correct construction of logic circuits with specific connections and outputs. The semantics of assembling the logic gates are described with object properties. An important aspect of the logic gates to consider during the assembling is the Input and Output connections. A logic gate has one output, but different number of input connections. The logic gate connections are based on the output of one of them using as input in the others.

5 BUILDING A 1-BIT FULL ADDER IN KUKULCAN FRAMEWORK

A 1-bit full adder is a logic circuit with 3 bit binary inputs (A, B, CIN) and two single bit binary outputs (OUTPUT, COUT). Having both carry in and carry out capabilities, the full adder is highly scalable and found in many cascaded circuit implementations. For that reason, we have chosen this circuit in this work. The truth table using the instance notation is showed in 4. This circuit is built with 5 logic gates (2 xor, 2 and, 1 or), as showed in Figure 3. The logic circuit model used for describe an 1-bit Full Adder circuit was made in Kukulcan Framework using its graphical interface of logic gates, and is shown in figure 4. The input model is created by the user who selects classes and

relation among concepts and he creates the logic gates instances (:and1, :and2, :xor1, :xor2 and :or). In this case the input model only has 5 logic gates and we can create its instances and relations among them using the Kukulcan's menus (create instances vocabulary).

5.1 Assembling Verfication using The Pellet Reasoner

The Core Ontology written in OWL-DL, allow us to define restrictions which Pellet can verify during the consistency checking process. For instance, the following code establishes that the *and* gate has only 1 output, because a FunctionalProperty is defined for *:andOutput* Object Property.

```
:andOutput a owl:ObjectProperty ;
                  rdfs:domain   :Gate ;
                  rdfs:range    :Bits ;
    rdf:type  owl:FunctionalProperty .
```

An interesting property of the ontology used in this work is a blank node. It is a node in an RDF graph representing a resource without URI or literal. We used it as variable. If we put the same blank node, the result for this node has to be the same. In our example below, _:c1 and _:c2 are blank nodes (working as variables). The example shows how to *:xor1* and *:and2* gates are forced to have the same input (_:c2).

```
:xor1 :isTypeGate _:c1. # :xor1 is a member
                        #  of xor gates
_:c1  :hasInput2  _:c2. # :xor1 requires 2
                        #  input values
```

A difference with Logic Programming Paradigm, we can check our types using ontologies. In particular when we create a new logic gate, for example *:and2*, we do not have to introduce all input and output values. In this case, it is only neccesary to establish the property relation *:and2* **:isTypeGate** *:and* . Besides, the ontology allow us to see circuits and gates saving in the ontology at the same time because the *Gate* class is a subclass of *Circuit*.

```
:Circuit         a owl:Class .
:Gate rdfs:subClassOf :Circuit .
:isTypeGate a owl:ObjectProperty ;
             rdfs:domain   :Gate ;
             rdfs:range    :Gate .
```

The *disjointWith* property allow to verify restrictions in the input model. For example a logic gate is not a bit, these two classes are different. Defining disjoint classes is also possible [3].

```
 :Gate rdfs:subClassOf   :Thing ;
        owl:disjointWith :Bits .
```

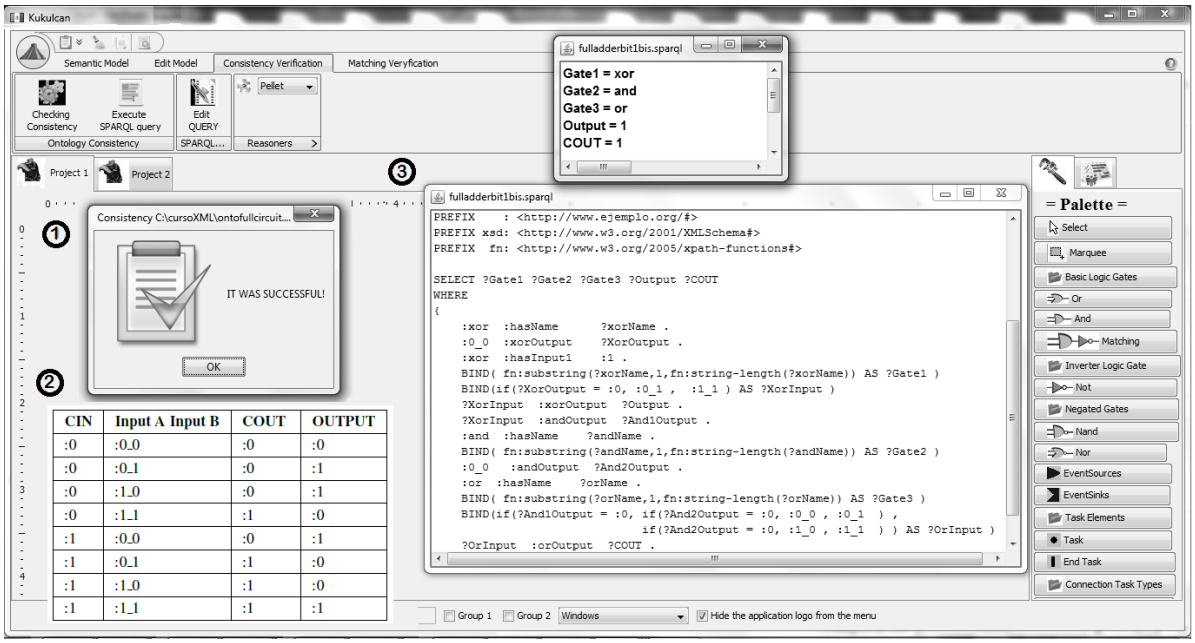
All instances created, properties (object and datatype) established among instances, and blank nodes in the Ontology are checked by the reasoner Pellet during the consistency verification process.

5.2 Output Validation using a SPARQL Query

The second step after the reasoner have checked the ontology circuit consistency is to apply a SPARQL query for validating the correct output of 1-bit fadder circuit. In our case, we have defined a query which describes the circuit and obtain the output for given input values. Of course, all this process is transparent, for the user. He does not need to know nothing about ontologies, reasoners or SPARQL queries, only the manager of the ontology system has to know about that. We can think that SPARQL is the version of SQL for ontologies. Besides, we can use variables in the queries, constraints, filtering information, logic operators, if statements and more. Each triples (each line after) are linking by variables which begin with a question mark. In this code *?type1* and *?AB* are examples of variables. The same name of variable imply the same value to look for in the query. We can execute and edit queries in Kukulcan framework because the Jena API allowed us to use SPARQL queries in our framework programmed in Java language. The las step, when the logic circuit has been verified and validated, consists on storing the project independent of the ontology or include it in the core ontology. It is important to note that these challenges increase the reuse of this ontology and decrease the time in the development of future circuits. Benefiting the economy of companies (Knowledge Capitalization [33][39]). In our example, part of the code included in the core ontology was:

```
:fulladder :hasName "1-bit full
                adder"^^xsd:string .
:fulladder :hasInput3       :0_0_0 .
:fulladder :hasInput3       :0_0_1 .
:fulladder :hasInput3       :1_1_1 .
:0_0_0 :fullAdderOutput :0 .
:0_0_0 :fullAdderOutput :1 .
:1_1_0 :fullAdderOutput :0 .
                :
```

In the code above, there are mainly three properties: *hasName*, *hasInput3* and *fullAdderOutput*. The meaning of the *hasName* property is an string with the name of the logic circuit. But the most interesting properties are *hasInput* and *fullAdderOutput*. The first is formed by a circuit instance (in this case called *fulladder*), second the name of the property *hasInput3* where the number 3 means that this gate receive 3 input values and finally with 3 bits values ending with a period. The second property begins with the 3 bits values following the *fullAdderOuput* property and finish with the bit output value with a period. The colon before each element and the ending period are only *n3* notation [7][8].



CIN	Input A Input B	COUT	OUTPUT
:0	:0_0	:0	:0
:0	:0_1	:0	:1
:0	:1_0	:0	:1
:0	:1_1	:1	:0
:1	:0_0	:0	:1
:1	:0_1	:1	:0
:1	:1_0	:1	:0
:1	:1_1	:1	:1

Figure 4. Consistency checking (1), Full Adder Truth Table (2) and SPARQL query execution (3)

6 CONCLUSIONS

Knowledge Management using Semantic Web Techniques, in organizations and companies based on Digital Circuits, is possible by means of core ontologies, reasoners, and SPARQL queries. Ontologies are usually expressed in a logic-based language (Description-Logic), enabling detailed, sound, meaningful distinctions to be made among the classes, properties and relations. Core Ontologies give more expressive meaning, maintains computability, do not require the validation of experts or apply a complex methodology for its construction. This core ontology for logic circuits increase the reuse of it and decrease the time in the development of future circuits. The use of an core ontology of logic circuits allowed us to validate the output of the 1-bit Full Adder and verify the correct assembling of its gates using the Pellet reasoner and a SPARQL query with semantics in comparison with a classic SQL query. The queries on the ontology are simple and easy to do for all users whereas a classic SQL query in a database requires computational knowledge. In this paper we have presented a Semantic Web framework called Kukulcan and described Semantic Web Techniques used for Knowledge Managment on the Digital Circuits domain.

7 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. OISE-0730065.

References

[1] M. Alavi and D. Leider. Knowledge management systems: emerging views and practices from the field. In *System Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*, pages 8–pp. IEEE, 1999.

[2] D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann, 2011.

[3] F. E. Antoniou Grigoris and V. H. Frank. Introduction to semantic web ontology languages. 2005.

[4] F. Baader. *The description logic handbook: theory, implementation, and applications*. Cambridge Univ Pr, 2003.

[5] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. *Mechanizing Mathematical Reasoning*, pages 228–248, 2005.

[6] R. Benjamins, D. Fensel, and A. Gómez-Pérez. Knowledge management through ontologies. CEUR Workshop Proceedings (CEUR-WS. org), 1998.

[7] T. Berners-Lee. N3 notation: http://www.w3.org/designissues/notation3.html.

[8] T. Berners-Lee, D. Connolly, and S. Hawke. Semantic web tutorial using n3. In *Twelfth International World Wide Web Conference*, 2003.

[9] T. Berners-Lee, J. Hendler, O. Lassila, and Others. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[10] T. Boudreau. *NetBeans: the definitive guide*. O'Reilly Media, 2002.

[11] K. Breitman, M. A. Casanova, and W. Truszkowski. *Semantic Web: Concepts, Technologies and Applications (NASA Monographs in Systems and Software Engineering)*. Springer-Verlag London, 2006.

[12] W. P. Davies John, Stunder Rudi. Semantic web technologies trens and research in ontology-based systems. 2006.

[13] M. Doerr, J. Hunter, and C. Lagoze. Towards a core ontology for information integration. *Journal of Digital information*, 4(1), 2011.

[14] W. M. K. B. Duineveld A.J., Stoter R. and B. V.R. Wondertools? a comparative study of ontological engineering tools. 2000.

[15] D. Fensel, F. Van Harmelen, M. Klein, H. Akkermans, J. Broekstra, C. Fluit, J. van der Meer, H. Schnurr, R. Studer, J. Hughes, et al. On-to-knowledge: Ontology-based tools for knowledge management. In *Proceedings of the eBusiness and eWork*, pages 18–20, 2000.

[16] N. F.Noy and D. L.McGuinness. Ontology development 101:a guide to creating your first ontology. March 2006.

[17] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. Ontological engineering with examples from the areas of knowledge management,e-commerce and the semantic web. 2003.

[18] J. Gosling, B. Joy, G. Steele, and G. Bracha. *Java (TM) Language Specification, The (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.

[19] J. Gracia, J. Liem, E. Lozano, O. Corcho, M. Trna, A. Gómez-Pérez, and B. Bredeweg. Semantic techniques for enabling knowledge reuse in conceptual modelling. *The Semantic Web–ISWC 2010*, pages 82–97, 2010.

[20] T. Gruber. Ontolingua: A mechanism to support portable ontologies. pages KSL 91–66. 1992.

[21] T. Gruber. Toward principles for the design of ontologies used for knowledge sharing. pages 907–928. 1995.

[22] M. Gruninger and M. S. Fox. The role of competency questions in enterprise engineering. In *Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, 1994.

[23] N. Guarino. Formal ontology in information systems. pages 3–15. IOS-Press, June 1998.

[24] S. Heiner and V. H. Frank. Information sharing on the semantic web. 2005.

[25] J. Hooker and H. Yan. Logic circuit verification by benders decomposition. *Principles and Practice of Constraint Programming: The Newport Papers, MIT Press (Cambridge, MA, 1995)*, pages 267–288, 1995.

[26] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. Wang. The manchester owl syntax. *OWL: Experiences and Directions*, pages 10–11, 2006.

[27] M. Hristozova and L. Sterling. An extreme method for developing lightweight ontologies. In *In Workshop on Ontologies in Agent Systems, 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002.

[28] M. Hwang, H. Kong, and P. Kim. The design of the ontology retrieval system on the web. volume 3, pages 1815 –1818, feb. 2006.

[29] Java.net. Flamingo. http://java.net/projects/flamingo/, 2010.

[30] Jena. Jena a semantic web framework for java. 2000.

[31] I. Jurisica, J. Mylopoulos, and E. Yu. Using ontologies for knowledge management: An information systems perspective. In *Proceedings of the Annual Meeting-American Society For Information Science*, volume 36, pages 482–496. Information Today; 1998, 1999.

[32] S. L. and M. B. Ontology evolution within ontology editors. volume 62, pages 53–62. September 2002.

[33] F.-M. Lesaffre and V. Pelletier. A business case of the use of ontologies for knowledge capitalization and exploitation.

[34] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. Ontologies for enterprise knowledge management. *Intelligent Systems, IEEE*, 18(2):26–33, 2003.

[35] B. McBride. Jena: Implementing the rdf model and syntax specification, 2001.

[36] D. McGuinness, F. Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10:2004–03, 2004.

[37] K. T. S. Michael C. Daconta, Leo J. Obrst. *The Semantic Web: A guide to the future of XML, Web Services and Knowledge Management*. Wiley Computer Publishing, Inc., 111 River Street Hoboken, NJ, jun. 2003.

[38] T. Robal, T. Kann, and A. Kalja. An ontology-based intelligent learning object for teaching the basics of digital logic. In *Microelectronic Systems Education (MSE), 2011 IEEE International Conference on*, pages 106–107. IEEE, 2011.

[39] B. D. Rodriguez-Rocha, F. E. Castillo-Barrera, and H. Lopez-Padilla. Knowledge capitalization in the automotive industry using an ontology based on the iso/ts 16949 standard. volume 0, pages 100–106. IEEE Computer Society, Los Alamitos, CA, USA, sep. 2009.

[40] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[41] S. H. Staab S., Studer R. and Y. Sure. Knowledge processes and ontologies. volume 16, pages 26–34. Jan-Feb 2001.

[42] SWRL. Swrl:a semantic web rule language.

[43] W3C. http://www.w3.org/consortium/. 1994.