Using an Inference Engine to Detect Conflicts in Collaborative Design

Moisés Lima Dutra, Catarina Ferreira da Silva, Parisa Ghodous, Ricardo Jardim-Gonçalves

Using an Inference Engine to Detect Conflicts in Collaborative Design

Moisés Dutra[1], Catarina Ferreira da Silva[1], Parisa Ghodous[1], Ricardo Gonçalves[2]

[1]*Laboratory of Computer Graphics, Images and Information Systems (LIRIS) – University of Lyon 1, France, {moises.limadutra, catarina.ferreira.da.silva, ghodous}@liris.cnrs.fr*

[2]*Group for the Research in Interoperability of Systems (GRIS) – Uninova Institute, Portugal, rg@uninova.pt*

Abstract

Integrate multidisciplinary virtual teams in collaborative design process is a hard task to tackle with. Differences of background, culture and expertise usually lead to conflicting situations. Moreover, if we consider to be working on a synchronous environment – where all designers exchange data and share information at the same time that the product design is being done – potential knowledge representation conflicts can also arise from this process. In our work we currently represent knowledge in collaborative design by using Web Ontology Language (OWL). OWL is structured to be a major formalism for the design and dissemination of ontology information. The use of OWL reasoning is intended to be a consistent way to verify the information given by several experts, trying to avoid redundancies, contradictions and misunderstood. This paper presents an approach to detect collaborative design conflicts through the use of an OWL inference engine.

Keywords

Collaborative design, conflicts, ontologies, reasoning.

1 Introduction

The increasing exchanging of knowledge and expertise among virtual teams in collaborative design has brought about a lot of conflicting situations. Time and resources required to solve these situations have increased proportionally to the complexity of modern industrial systems. The collaborative design process uses, even more, geographically distributed knowledge, resources and equipment [Xie et al. 2002]. Virtual team members work in parallel with different tools, languages and time zones. Collaborative engineering proposes to face these problems by reorganizing and integrating the set of development activities, starting from early-stage design [Ghodous, 2002]. An important part of the collaborative work consists of sharing of data, information, and knowledge among individuals with different points of view and specific objectives concerning their particular domains [Sriram 2002]. When different points of view and objectives are considered inside the same scope, the divergences arisen from this process lead frequently to conflicts. This paper proposes an approach to detect collaborative design conflicts, based on an OWL inference engine. All exchanged data and information are represented in OWL, in order to use an OWL reasoner. This proposal takes as scenario a distributed architecture for collaborative design, previously described in [Dutra et al. 2007] and in [Slimani 2006].

2 Collaborative Design Conflicts and OWL

2.1 Conflicts in Collaborative Design

According to [Klein 2000], a conflict is an incompatibility between two design decisions or between two distinct design objectives. [Cointe 1998] defines a conflict as a designers' disagreement about product components or their evolution in time. [Castelfranchi, 2000] says that a conflict must necessarily be a divergence of goals. One can also consider that there is a conflict whenever one or more propositions cannot coexist in the same time at the same design space [Ferreira da Silva et al. 2004]. Design space is a multidimensional representation of the design process parameter set, that is, models but, also, design goals and socio-cultural references

that define the designer's work method. We can say, at last, that a conflict happens whenever one or more designers give incompatible solutions (product representations) to the same problem, or whenever they evaluate negatively a proposed solution.

In our collaborative architecture it is essential to consider the publication of information in public spaces. Publishing a proposition means to put together proposed instances of a product model along with the design space subparts already instantiated. This union is only possible if there is no interference between these two sets, which means, if all their elements are coherent.

Conflict attenuation is another point to be considered. Conflicts can be extremely consumers of resources (development time, budget, materials). That is why it is so important to prevent the conflicts, aiming to detect them already in early-stage design, through the use of identification and categorization. Subsequently, it is important to notify the different involved parts, in order to put the situation under control as soon as possible. These tree steps are called strategies of conflict attenuation [Matta et al. 1996].

2.2 Web Ontology Language (OWL)

The Web Ontology Language (OWL) [OWL 2004], released as a W3C recommendation in February 2004, is an expressive ontology language that is layered on top of RDF and RDFS. OWL can be used to define classes and properties, like RDFS does, but in addition, it provides a rich set of constructs to create new class descriptions as logical combinations (intersections, unions, or complements) of other classes, to define value and cardinality restrictions on properties (e.g., a restriction on a class to have only one value for a particular property) and so on.

OWL is the first ontology language whose design is based on the Web architecture, i.e., it is open (non-proprietary); it uses Universal Resource Identifiers (URIs) to unambiguously identify resources on the Web (similar to RDF and RDFS); it supports the linking of terms across ontologies making it possible to cross-reference and reuse information; and it has an XML syntax (RDF/XML) for easy data exchange.

Semantically speaking, OWL is placed right above RDFS in the web stack layer (Figure 1).

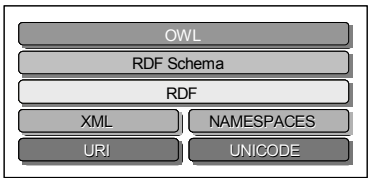

Figure 1 – Semantic web stack.

One of the main benefits of OWL is the support for automated reasoning and, to this effect, it has a formal semantics based on Description Logics (DL). DLs are typically a decidable subset of First Order Logic. They are suitable for representing structured information about concepts, concept hierarchies and relationships between concepts. The decidability of the logic ensures DL reasoners can be built to check an OWL ontology consistency, i.e., verify whether there are any logical contradictions in two or more ontology axioms. Furthermore, reasoners can be used to derive inferences from the asserted information, e.g., to infer whether a particular ontology concept is a subconcept of another, or whether a particular individual in an ontology belongs to a specific class [OWL 2004].

A practical OWL reasoner provides at least the standard set of Description Logic inference services, namely [Sirin et al. 2007]:

- ***Consistency checking:*** ensures that the ontology does not contain any contradictory facts. The *OWL Abstract Syntax & Semantics* document [OWL Semantics 2004] provides a formal definition of ontology consistency.
- ***Concept satisfiability:*** checks if it is possible for a class to have any instances. If class is unsatisfiable, then defining an instance of the class will cause the whole ontology to be inconsistent.
- ***Classification:*** computes the subclass relations between every named class to create the complete class hierarchy. The class hierarchy can be used to answer queries such as getting all or only the direct subclasses of a class.
- ***Realization:*** finds the most specific classes that an individual belongs to or, in other words, computes the direct types for each of the individuals. Realization can only be performed after classification since direct types are defined with respect to a class hierarchy. Using the classification hierarchy, it is also possible to get all the types for that individual.

Examples of OWL reasoners [Mei et al. 2004][Description Logics Reasoners 2008]:

- ***Pellet:*** is a free open-source Java-based reasoner with simple datatypes (i.e., for OWL 1.1). It implements a tableau-based decision procedure for general TBoxes (subsumption, satisfiability, classification) and ABoxes (retrieval, conjunctive query answering). It supports the OWL-API, the DIG-API, and Jena interface.

- ***F-OWL***: is an ontology inference engine for OWL based on Flora-2 (an object-oriented knowledge base language).

- ***FaCT++***: is a free open-source C++-based reasoner with simple datatypes (i.e., for OWL-DL with qualifying cardinality restrictions). It implements a tableau-based decision procedure for general TBoxes (subsumption, satisfiability, classification) and incomplete support of ABoxes (retrieval). It supports the lisp-API and the DIG-API.

- ***Euler***: is an inference engine supporting logic-based proofs. It is a backward chaining reasoner and will tell us whether a given set of facts and rules supports a given conclusion.

- ***Hoolet***: is an OWL DL reasoner that uses a First Order Prover to reason about OWL ontologies.

- ***fuzzyDL:*** is a free Java/C++ based reasoner with concrete fuzzy concepts (explicit definition of fuzzy sets plus modifiers). It implements a tableau plus Mixed Integer Linear Programming optimization decision procedure to compute the maximal degree of subsumption and instance checking with regard to a general TBox and Abox. It supports Zadeh semantics, Lukasiewicz semantics and is backward compatible with classical description logic reasoning.

- ***RacerPro:*** is a commercial (free trials and research licenses are available) lisp-based reasoner with simple data types (i.e., for OWL-DL with qualified number restrictions, but without nominals). It implements a tableau-based decision procedure for general TBoxes (subsumption, satisfiability, classification) and ABoxes (retrieval, nRQL query answering). It supports the OWL-API and the DIG-API and comes with numerous other features.

- ***Cerebra***: is a commercial C++-based reasoner. It implements a tableau-based decision procedure for general TBoxes (subsumption, satisfiability, classification) and ABoxes (retrieval, tree-conjunctive query answering using a XQuery-like syntax). It supports the OWL-API and comes with numerous other features.

In this work we chose to use Pellet, because it is a Java-based open source platform and because it supports OWL-API and Jena interface.

3 Applying an Inference Engine to Detect Conflicts

We state knowledge representation conflicts may be solved by applying services of an inference engine. This inference engine is based on description logics [Schmidt-Schauß et al. 1991] [Baader et al. 2003] [Horrocks et al. 2007]. Our system includes a consistency verification process and a functionality for ontologies debugging which apply inference engines services.

The consistency verification process calls the consistency service of the inference engine. This one checks the ontologies, looking for contradictions. We follow the formal definition of ontology consistency, given by [OWL Semantics 2004]. If the inference engine founds unsatisfiable concepts [Baader et al. 2003] then a conflicting situation is detected and notified to the concerned parts. Usually these parts are those who have published the conflicting ontologies into the public space (cf. Figure 2).

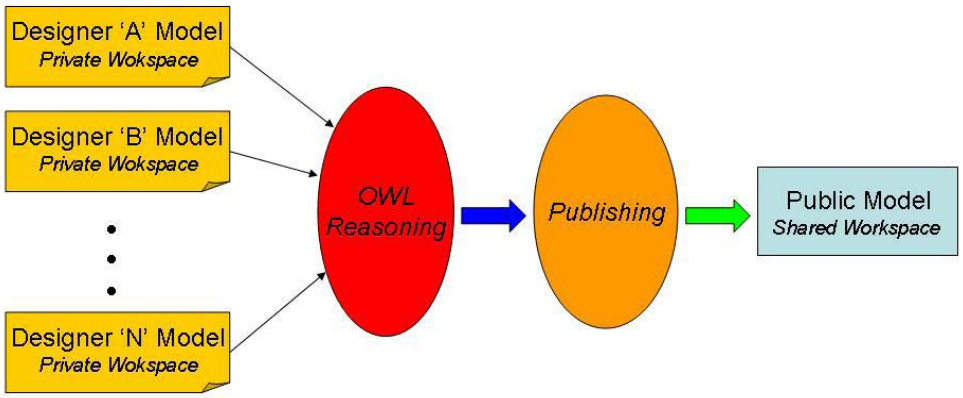

Figure 2 – Publication of ontologies into the public space.

The ontologies debugging functionality tries to detect the axioms that have caused the inconsistency. It lets the designers know the source of the representation conflict and sets the bases to start solving it.

Our application uses the Java libraries of Pellet [Sirin et al. 2007] and those of OWL API [Horridge et al. 2007] to implement the consistency verification process and the ontologies debugging functionality.

3.1 Modelling an Electrical Connector

To illustrate our approach, a collaborative design of an electrical connector was taken as example. We consider that two different points of view of the same product are up to be published in the public space. Firstly, a designer publishes his ontology in the shared workspace. After this publication, all design project members are able to access this published ontology. Figure 3 shows the main ontological classes and non-hierarchical properties, while Figure 4 shows the definition of the "Connector" concept. Protégé ontology editor tool [Noy et al. 2001] has been used to display the connector representation.

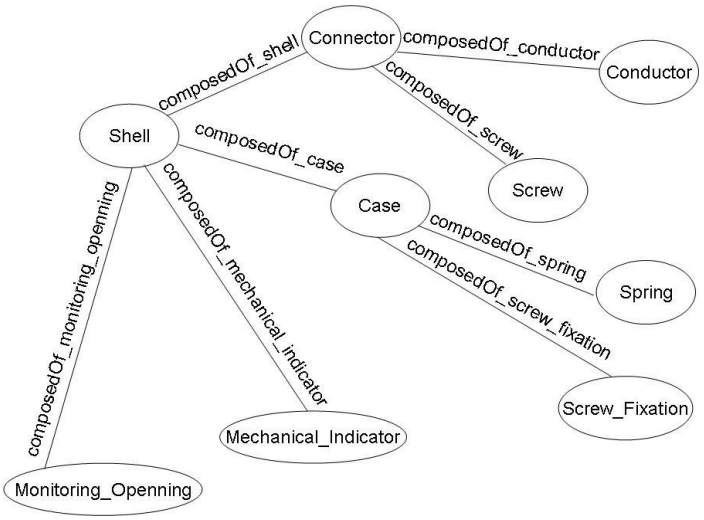

Figure 3 - Graphical representation of the first connector, arcs represent non-hierarchical properties.

According to this ontology, the "Connector" concept comprises other product's concepts, namely: "Conductor", "Screw" and "Shell". In other words, the "Connector" concept is necessarily and sufficiently defined using existential and universal restrictions by the following parts (1) to (3):

1. $\forall composedOf\_conductor.Conductor \sqcap \exists composedOf\_conductor.Conductor$
2. $\forall composedOf\_screw.Screw \sqcap \exists composedOf\_screw.Screw$
3. $\forall composedOf\_shell.Shell \sqcap \exists composedOf\_shell.Shell$

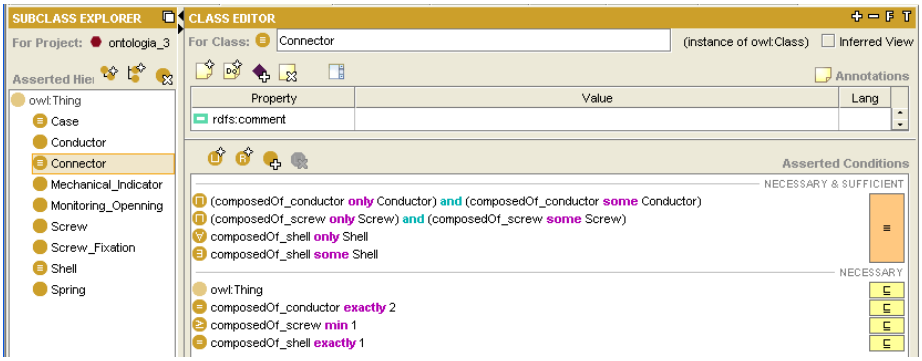

Figure 4 – The first hierarchical representation of the Connector classes and its definition.

Figure 5 shows the object properties hierarchy of this ontology.

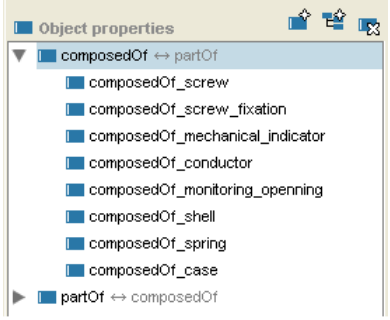

Figure 5 - List of properties defined for the Connector product by the first designer.

Latter on, another designer publishes his own ontology for the same product (cf. Figure 6). In that one, the "Connector" concept is composed by the concepts "Cable", "Screw" and "Body".

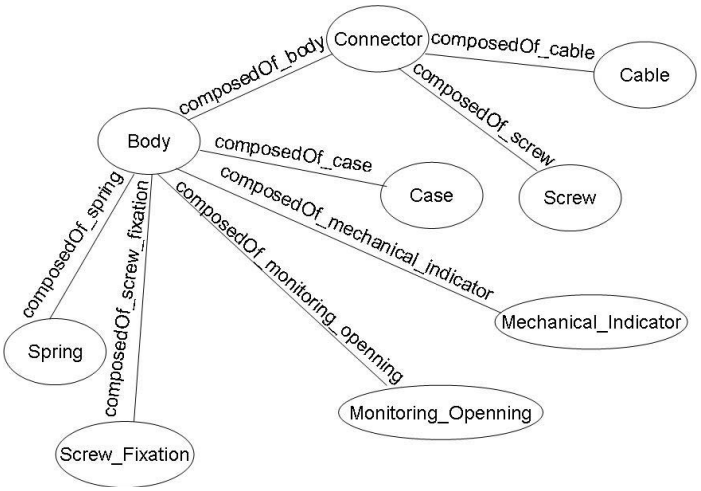

Figure 6 - Graphical representation of the second connector, arcs represent non-hierarchical properties.

Figure 7 shows the necessary and sufficient definition, as shown in Protégé tool.

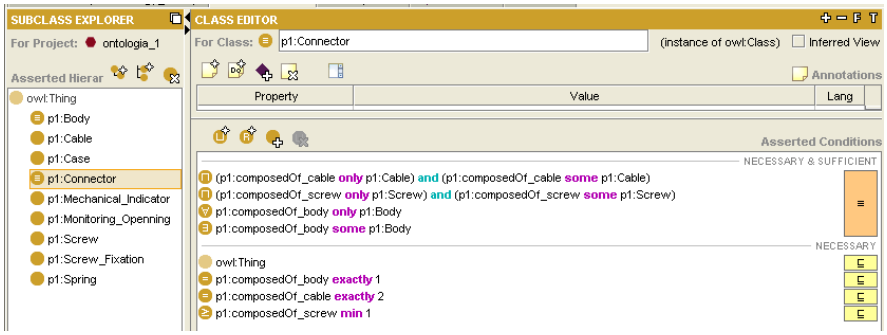

Figure 7 - The second hierarchical representation of the Connector classes and their definition.

The second designer also publishes information assigning that “Cable” concept is equivalent to “Conductor” concept on the ontology previously published by the other designer. The existence of two ontologies about the same product design in the shared workspace triggers the consistency verification process that calls the consistency checking service of the inference engine. In this case, both ontologies are inconsistent (cf. Figure 8).

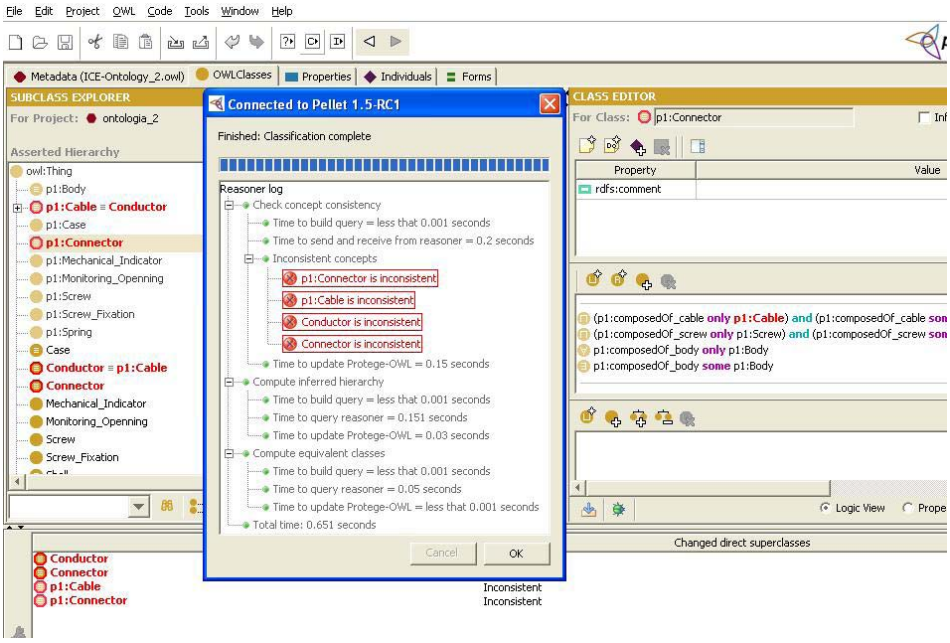

Figure 8 - Display of inconsistencies between ontologies through Protégé and Pellet.

This happens by the fact that both concepts are disjoint. Moreover, because "Cable" and "Conductor" have been assigned as equivalent, incoherence is detected. Figure 9 shows how the co-existence of axioms 2 and 3 produces incoherence. As “Conductor” concept is part of “Connector” concept definition, the previous incoherence brings about the next one, signalled by axiom 1.

The SWOOP editor tool [Kalyanpur et al. 2005] displays the ontologies debugging process. As so, Figure 9 shows the ontologies axioms causing the inconsistency. From this moment on, it must be started the conflict resolution process.

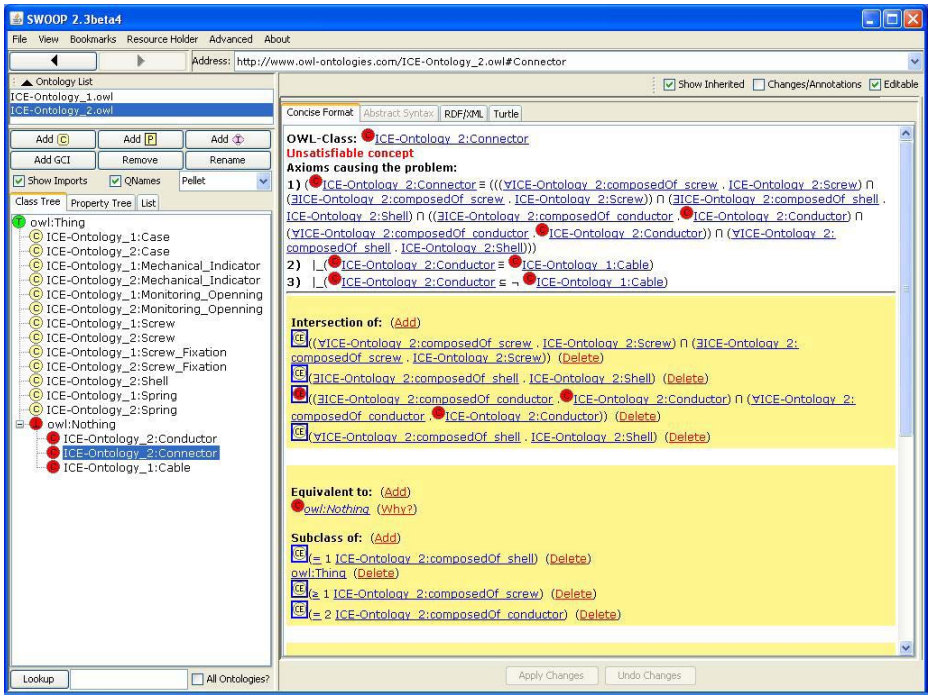

Figure 9 – The ontologies axioms causing the inconsistency.

4 Closing Remarks

The use of ontology modelling in collaborative design has proved to be a prominent approach to detect conflicts in early-stage design. Modelling expertises as OWL-DL classes permits us to identify conflicting axioms through ontology description logics reasoning. Besides, representing knowledge in OWL-DL offers a reasonable trade-off between expressibility and decidability, witch when used to verify product specifications in collaborative design may fit as an efficient conflict attenuator.

We also think the use of an inference engine based on description logics is very appropriate to detect representation conflicts. Thus, this kind of tool permits us to automate most of the conflict detection process and, hence, to diminish the remaining situations to be treated in an eventual negotiation process among designers.

Next steps of this work comprise the study of other kinds of incoherence that can also be detected by an inference engine, such as the study of the influence of particular combinations of functional properties and disjoint classes in conflict detection. The result of this work will be used to set up and undertake a conflict resolution process, important step to enhance the performance of a collaborative design project.

References

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; Patel-Schneider, P. (2003). The Description Logic Handbook - Theory, Implementation and Applications. Cambridge University Press. ISNB 0-521-78176-0.

Castelfranchi, C. (2000). Conflict Ontology. Computational Conflicts, Conflict Modelling for Distributed Intelligent Systems, pages 21-40, 2000.

Cointe, C. (1998). Aide à la gestion de conits en conception concourante dans un système distribué [in french]. PhD thesis, University of Montpellier II, France, 1998.

Description Logics Reasoners (2008). WWW page. Available at: http://www.cs.man.ac.uk/~sattler/reasoners.html . Accessed on: March 2008.

Dutra, M.; Ghodous, P. (2007). A Reasoning Approach for Conflict Dealing in Collaborative Design. In proceedings of the 14th International Conference in Concurrent Engineering (CE2007) – Springer Verlag. São José dos Campos, Brazil, July 2007.

Ferreira da Silva, C. ; Médini, L. ; Ghodous, P. (2004). Atténuation de conflits en conception coopérative [in french]. In 15èmes journées francophones d'Ingénierie des Connaissances (IC'2004), Nada Matta ed. Lyon. pp. 127-138. University Press of Grenoble, Grenoble, France, ISBN 2 7061 1221 2, 2004.

Ghodous, P. (2002). Modèles et Architectures pour l'Ingénierie Coopérative [in french]. Habilitation Thesis, University of Lyon 1, Lyon, France, 2002.

Horridge, M.; Bechhofer, S.; Noppens, O. (2007). Igniting the OWL 1.1 Touch Paper: The OWL API. OWLED 2007, 3rd OWL Experienced and Directions Workshop, Innsbruck, Austria, June 2007.

Horrocks, I.; Sattler, U. (2007). A Tableaux Decision Procedure for SHOIQ. Journal of Automated Reasoning, Springer Verlag.

Kalyanpur, A.; Parsia, B.; Sirin, E.; Cuenca-Grau, B.; Hendler, J. (2005). Swoop: A Web Ontology Editing Browser, Journal of Web Semantics Vol 4(2), 2005.

Klein, M. (2000). Towards a systematic repository of knowledge about managing collaborative design conflicts. In Gero J (ed.), Artificial Intelligence in Design '00, Boston. Dordrecht: Kluwer Academic Publishers, pp. 129-146, 2000.

Matta, N.; Corby, O. (1996). Conflict Management in Concurrent Engineering: Modelling Guides. Proceedings of the European Conference in Artificial Intelligence, Workshop on Conflict Management, Budapest, Hungary, 1996.

Mei, J.; Bontas, EP. (2004). Reasoning Paradigms for OWL Ontologies. Technical Report B-04-12, Institut für Informatik, Freue Universität Berlin, Germany, November 2004.

Noy, N. F.; McGuinness, D. L. (2001). Ontology Development 101: A Guide to Creating Your First Ontology. Semantic Web Working Symposium 2001.

OWL Web Ontology Language Overview (2004). WWW page. Available at: http://www.w3.org/TR/owl-features/. Accessed on: March 2008.

OWL Semantics and Abstract Syntax (2004). WWW page. Available at: http://www.w3.org/TR/owl-semantics/ . Accessed on: March 2008.

Sirin, E,; Parsia, B.; Cuenca Grau, B.; Kalyanpur, A.; Katz, Y. (2007). Pellet: A practical OWL-DL reasoner, Journal of Web Semantics: Science, Services and Agents on the World Wide Web. In E. Wallace (Ed.) Software Engineering and the Semantic Web, 5(2), pp. 51-53.

Schmidt-Schauß, M.; Smolka, G. (1991). Attributive concept descriptions with complements. Artificial Intelligence, 48(1):1-26, 1991.

Slimani, K. (2006). Système d'échange et de partage de connaissances pour l'aide à la Conception Collaborative [in french]. PhD Thesis, University of Lyon 1, Lyon, France, September 2006.

Sriram, RD. (2002). Distributed and Integrated Collaborative Engineering Design. Savren, 2002. ISBN 0-9725064-0-3.

Xie, H.; Neelamkavil, J.; Wang, L.; Shen, W.; Pardasani, A. (2002). Collaborative conceptual design - state of the art and future trends. Proceedings of Computer-Aided Design, vol. 34, pp. 981-996, 2002.